1

MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS
STANDARD REFERENCE MATERIAL 1010a
(ANSI and ISO TEST CHART No. 2)

National Library
of Canada

Acquisitions and
Bibliographic Services Branch

395 Wellington Street
Ottawa, Ontario
K1A 0N4

Bibliothèque nationale
du Canada

Direction des acquisitions et
des services bibliographiques

395, rue Wellington
Ottawa (Ontario)
K1A 0N4

Your file - Votre référence

Our file - Notre référence

NOTICE

The quality of this microform is heavily dependent upon the quality of the original thesis submitted for microfilming. Every effort has been made to ensure the highest quality of reproduction possible.

If pages are missing, contact the university which granted the degree.

Some pages may have indistinct print especially if the original pages were typed with a poor typewriter ribbon or if the university sent us an inferior photocopy.

Reproduction in full or in part of this microform is governed by the Canadian Copyright Act, R.S.C. 1970, c. C-30, and subsequent amendments.

AVIS

La qualité de cette microforme dépend grandement de la qualité de la thèse soumise au microfilmage. Nous avons tout fait pour assurer une qualité supérieure de reproduction.

S'il manque des pages, veuillez communiquer avec l'université qui a conféré le grade.

La qualité d'impression de certaines pages peut laisser à désirer, surtout si les pages originales ont été dactylographiées à l'aide d'un ruban usé ou si l'université nous a fait parvenir une photocopie de qualité inférieure.

La reproduction, même partielle, de cette microforme est soumise à la Loi canadienne sur le droit d'auteur, SRC 1970, c. C-30, et ses amendements subséquents.

Canada

Permutational Power of Priority Queues

by

Murali Thiyagarajah

**A thesis submitted to
the Faculty of Graduate Studies and Research
in partial fulfillment of
the requirements for the degree of
Master of Computer Science**

Ottawa-Carleton Institute for Computer Science

**School of Computer Science
Carleton University
Ottawa, Ontario
December 1993**

**© Copyright
1993, Murali Thiyagarajah**

National Library
of Canada

Acquisitions and
Bibliographic Services Branch

395 Wellington Street
Ottawa, Ontario
K1A 0N4

Bibliothèque nationale
du Canada

Direction des acquisitions et
des services bibliographiques

395, rue Wellington
Ottawa (Ontario)
K1A 0N4

Your file Votre référence

Our file Notre référence

The author has granted an irrevocable non-exclusive licence allowing the National Library of Canada to reproduce, loan, distribute or sell copies of his/her thesis by any means and in any form or format, making this thesis available to interested persons.

L'auteur a accordé une licence irrévocable et non exclusive permettant à la Bibliothèque nationale du Canada de reproduire, prêter, distribuer ou vendre des copies de sa thèse de quelque manière et sous quelque forme que ce soit pour mettre des exemplaires de cette thèse à la disposition des personnes intéressées.

The author retains ownership of the copyright in his/her thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without his/her permission.

L'auteur conserve la propriété du droit d'auteur qui protège sa thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation.

ISBN 0-315-89855-0

Canada

Name MURALI TH. YAGARAJAH

Dissertation Abstracts International is arranged by broad, general subject categories. Please select the one subject which most nearly describes the content of your dissertation. Enter the corresponding four-digit code in the spaces provided.

COMPUTER SCIENCE

SUBJECT TERM

0984

SUBJECT CODE

U·M·I

Subject Categories

THE HUMANITIES AND SOCIAL SCIENCES

COMMUNICATIONS AND THE ARTS

Architecture 0729
Art History 0377
Cinema 0900
Dance 0378
Fine Arts 0357
Information Science 0723
Journalism 0391
Library Science 0399
Mass Communications 0708
Music 0413
Speech Communication 0459
Theater 0465

EDUCATION

General 0515
Administration 0514
Adult and Continuing 0516
Agricultural 0517
Art 0273
Bilingual and Multicultural 0282
Business 0688
Community College 0275
Curriculum and Instruction 0727
Early Childhood 0518
Elementary 0524
Finance 0277
Guidance and Counseling 0519
Health 0680
Higher 0745
History of 0520
Home Economics 0278
Industrial 0521
Language and Literature 0279
Mathematics 0280
Music 0522
Philosophy of 0998
Physical 0523

Psychology 0525
Reading 0535
Religious 0527
Sciences 0714
Secondary 0533
Social Sciences 0534
Sociology of 0340
Special 0529
Teacher Training 0530
Technology 0710
Tests and Measurements 0288
Vocational 0747

LANGUAGE, LITERATURE AND LINGUISTICS

Language 0679
 General 0289
 Ancient 0290
 Linguistics 0291
 Modern 0401
Literature 0294
 General 0295
 Classical 0297
 Comparative 0298
 Medieval 0316
 Modern 0591
 African 0305
 American 0352
 Asian 0355
 Canadian (English) 0593
 Canadian (French) 0311
 English 0312
 Germanic 0315
 Latin American 0313
 Middle Eastern 0314
 Romance 0314
 Slavic and East European

PHILOSOPHY, RELIGION AND THEOLOGY

Philosophy 0422
Religion 0318
 General 0321
 Biblical Studies 0319
 Clergy 0320
 History of 0322
 Philosophy of 0469
Theology

SOCIAL SCIENCES

American Studies 0323
Anthropology 0324
 Archaeology 0326
 Cultural 0327
 Physical 0310
Business Administration 0272
 General 0770
 Accounting 0454
 Banking 0338
 Management 0385
 Marketing 0501
Canadian Studies 0503
Economics 0505
 General 0508
 Agricultural 0509
 Commerce-Business 0510
 Finance 0511
 History 0358
 Labor 0366
 Theory 0351
Folklore 0578
Geography 0366
Gerontology 0351
History 0578
 General

Ancient 0579
Medieval 0581
Modern 0582
Black 0328
African 0331
Asia, Australia and Oceania 0332
Canadian 0334
European 0335
Latin American 0336
Middle Eastern 0333
United States 0337
History of Science 0585
Law 0398
Political Science 0615
 General 0616
 International Law and Relations 0617
 Public Administration 0814
Recreation 0452
Social Work 0626
Sociology 0627
 General 0938
 Criminology and Penology 0631
 Demography 0628
 Ethnic and Racial Studies 0629
 Individual and Family Studies 0630
 Industrial and Labor Relations 0700
 Public and Social Welfare 0344
 Social Structure and Development 0709
 Theory and Methods 0999
Transportation 0453
Urban and Regional Planning 0453
Women's Studies

THE SCIENCES AND ENGINEERING

BIOLOGICAL SCIENCES

Agriculture 0473
 General 0285
 Agronomy 0475
 Animal Culture and Nutrition 0476
 Animal Pathology 0359
 Food Science and Technology 0478
 Forestry and Wildlife 0479
 Plant Culture 0480
 Plant Pathology 0817
 Plant Physiology 0777
 Range Management 0746
 Wood Technology
Biology 0306
 General 0287
 Anatomy 0308
 Biostatistics 0309
 Botany 0379
 Cell 0329
 Ecology 0353
 Entomology 0369
 Genetics 0793
 Limnology 0410
 Microbiology 0307
 Molecular 0317
 Neuroscience 0416
 Oceanography 0433
 Physiology 0821
 Radiation 0778
 Veterinary Science 0472
 Zoology
Biophysics 0786
 General 0760
 Medical
EARTH SCIENCES 0425
Biogeochemistry 0996
Geochemistry

Geodesy 0370
Geology 0372
Geophysics 0373
Hydrology 0388
Mineralogy 0411
Paleobotany 0345
Paleoecology 0426
Paleontology 0418
Paleozoology 0985
Palynology 0427
Physical Geography 0368
Physical Oceanography 0415

HEALTH AND ENVIRONMENTAL SCIENCES

Environmental Sciences 0768
Health Sciences 0566
 General 0300
 Audiology 0992
 Chemotherapy 0567
 Dentistry 0350
 Education 0769
 Hospital Management 0758
 Human Development 0982
 Immunology 0564
 Medicine and Surgery 0347
 Mental Health 0565
 Nursing 0570
 Nutrition 0380
 Obstetrics and Gynecology 0354
 Occupational Health and Therapy 0581
 Ophthalmology 0571
 Pathology 0419
 Pharmacology 0572
 Pharmacy 0382
 Physical Therapy 0573
 Public Health 0574
 Radiology 0575
 Recreation

Speech Pathology 0460
Toxicology 0383
Home Economics 0386

PHYSICAL SCIENCES

Pure Sciences 0485
Chemistry 0749
 General 0466
 Analytical 0487
 Biochemistry 0488
 Inorganic 0738
 Nuclear 0490
 Organic 0491
 Pharmaceutical 0494
 Physical 0495
 Polymer 0754
 Radiation 0405
Mathematics 0605
Physics 0986
 General 0606
 Acoustics 0608
 Astronomy and Astrophysics 0748
 Atmospheric Science 0607
 Atomic 0798
 Electronics and Electricity 0759
 Elementary Particles and High Energy 0609
 Fluid and Plasma 0610
 Molecular 0752
 Nuclear 0756
 Optics 0611
 Radiation 0463
 Solid State
Statistics 0346
Applied Sciences 0984
Applied Mechanics
Computer Science

Engineering 0537
 General 0538
 Aerospace 0539
 Agricultural 0540
 Automotive 0541
 Biomedical 0542
 Chemical 0543
 Civil 0544
 Electronics and Electrical 0348
 Heat and Thermodynamics 0545
 Hydraulic 0546
 Industrial 0547
 Marine 0794
 Materials Science 0548
 Mechanical 0743
 Metallurgy 0551
 Mining 0552
 Nuclear 0549
 Packaging 0765
 Petroleum 0554
 Sanitary and Municipal 0790
 System Science 0428
 Geotechnology 0796
 Operations Research 0795
 Plastics Technology 0994
 Textile Technology

PSYCHOLOGY

General 0621
Behavioral 0384
Clinical 0622
Developmental 0620
Experimental 0623
Industrial 0624
Personality 0625
Physiological 0989
Psychobiology 0349
Psychometrics 0632
Social 0451

The undersigned hereby recommend to
The Faculty of Graduate Studies and Research
acceptance of the thesis,

**Permutational Power of
Priority Queues**

submitted by

Murali Thiagarajah

in partial fulfilment of the requirements
for the degree of Master of Computer Science

A handwritten signature in black ink, appearing to read 'J. R. P. J.', written over a horizontal line.

Director, School of Computer Science

A handwritten signature in black ink, appearing to read 'M. D. Atkinson', written over a horizontal line.

Thesis Supervisor

Carleton University

December 20, 1993

Abstract

Priority queue is an abstract data type which processes the data items that pass through it and outputs them in a certain order. They have a number of important applications ranging from the design of process management in operating systems to *Huffman* coding and external sorting.

This thesis explores some mathematical analysis of the relationships between the input and output permutations and the manner in which the priority queue computations are carried out. We give an interesting result to express the number of allowable input and output numeric permutations in an ordinary priority queue. We also provide some mathematical relationships between the *Insert-DeleteMin* sequences and the number of output sequences for ordinary and binary priority queues.

The results obtained for double ended priority queue, a variation of the ordinary priority queue, though interesting, are more difficult to explain than that of ordinary priority queue. Unlike the ordinary priority queue, we do not have any general formula to express the number of allowable input and output permutations in double ended priority queues. We show that considering the fixed *Insert-Delete* bracket sequences, an analogous result for the ordinary priority queues can be obtained for the double ended priority queues. We also provide expressions to compute the number of output permutations for a given *Insert-Delete* bracket sequence and to compute the average number of outputs for a random priority queue computation. Finally we provide two conjectures based upon the numerical evidences obtained for double ended priority queues.

Acknowledgments

My greatest debt is to my supervisor, Dr. Michael D Atkinson. This thesis would not have seen the light of the day, had it not been for his unstinted support, insightful observations and enthusiasm throughout the entire duration of the work. My sincere thanks to him.

Contents

Abstract	iii
Acknowledgments	iv
1 Introduction	1
1.1 Specification: Ordinary Priority Queue	3
1.2 Implementation of Ordinary Priority Queues Using Heap Ordered Trees	4
1.2.1 Location of Nodes in the Binary Tree	4
1.2.2 Shape of the heap	6
1.2.3 Implicit Representation of a Heap Ordered Tree	9
1.3 Implementation of Ordinary Priority Queues Using Binomial Queues	15
1.3.1 Binomial trees and their properties	15
1.3.2 Binomial Queues	16
1.3.3 Merging of Two Binomial Queues	17
1.3.4 <i>Insert</i> on Binomial Queue Representation	19
1.3.5 <i>DeleteMin</i> on Binomial Queue Representation	19
1.4 Other Priority Queue Implementations	21
1.5 Comparison of Implementation Schemes	22
1.6 An Implementation of Double Ended Priority Queues: MinMax Heaps	23
1.7 Some Applications of Priority Queues	29
1.7.1 Huffman Trees: An Application of Ordinary Priority Queues .	29
1.7.2 External Quicksort: An Application of Double Ended Priority Queues	32

2	Ordinary Priority Queues	33
2.1	Generating Output Sequences	34
2.1.1	Valid Sequences of <i>Insert</i> and <i>DeleteMin</i>	34
2.1.2	Two Methods of Generating Output Sequences	35
2.2	Number of Allowable Pairs (σ, τ)	38
2.3	Number of Output Permutations for each <i>Insert-DeleteMin</i> Sequence	50
2.3.1	Average Number of Outputs for a Random Priority Queue Computation	53
2.4	Binary Priority Queues	56
2.4.1	Average Number of Outputs for a Random Binary Priority Queue Computation	57
3	Double Ended Priority Queues	63
3.1	Constraints on the Input	64
3.2	Generating Output Sequences Using Double Ended Priority Queues .	67
3.2.1	The Total Number of Allowable Pairs	69
3.2.2	The Number of Allowable Pairs with Fixed <i>Insert-Delete</i> Sequence	75
3.2.3	The Number of Allowable Pairs with Fixed <i>DeleteMin-DeleteMax</i> Sequences	82
3.2.4	The Number of Outputs for Fixed Inputs and Fixed <i>Insert-Delete</i> Sequences	84
3.2.5	Number of Outputs for Fixed Inputs and Fixed <i>DeleteMin-DeleteMax</i> Sequences	85
3.2.6	The Number of Outputs for Fixed <i>Insert-Delete</i> and Fixed <i>DeleteMin-DeleteMax</i> Sequences	86
4	Conclusion	87
4.1	Summary of Results	87
4.2	Future Work	89
A	Turbo Pascal Listings of Codes: Ordinary and Double Ended Priority Queues	92

A.1	Listing of Priority Queue Engine	92
A.2	Generating Output Sequences Using Ordinary Priority Queues	103
A.3	Generating Output Sequences Using Double Ended Priority Queues .	104
A.3.1	The Total Number of Allowable Pairs	104
A.3.2	The Number of Allowable Pairs with Fixed <i>Insert-Delete</i> Sequences	106
A.3.3	The Number of Outputs with Fixed Inputs and Fixed <i>Insert-Delete</i> Sequences	108
A.3.4	Output Generation with Fixed <i>DeleteMin-DeleteMax</i> Sequences	109
A.3.5	The Number of Allowable Pairs with Fixed Inputs and Fixed <i>DeleteMin-DeleteMax</i> Sequences	111
A.3.6	Output Generation with Fixed <i>Insert-Delete</i> Bracket Sequences and <i>DeleteMin-DeleteMax</i> Sequences	113
B	Output Listings for Ordinary and Double Ended Priority Queues	115
B.1	Ordinary Priority Queues	115
B.1.1	Output Generation: Method (i)	115
B.1.2	Output Generation: Method (ii)	117
B.2	Double Ended Priority Queues	127
B.2.1	The Total Number of Allowable Pairs	127
B.2.2	The Number of Allowable Pairs with Fixed <i>Insert-Delete</i> Bracket Sequences	129
B.2.3	The Number of Outputs with Fixed Inputs and Fixed <i>Insert-Delete</i> Sequences	131
B.2.4	Output Generation with Fixed <i>DeleteMin-DeleteMax</i> Sequences	141
B.2.5	The Number of Allowable Pairs with Fixed Inputs and Fixed <i>DeleteMin-DeleteMax</i> Sequences	153
B.2.6	Output Generation with Fixed <i>Insert-Delete</i> Bracket Sequences and <i>DeleteMin-DeleteMax</i> Sequences	155

List of Tables

1	Running Times: Operations with Heap Ordered Tree	14
2	Comparison of Major Implementation Schemes	23
3	Comparison of Min Heap and MinMax Heap	29
4	Allowable Pairs (σ, τ) for $n = 3$	38
5	Number of Elements in $S(\tau)$ for $n = 4$	45
6	Number of Elements in $T(\sigma)$ for $n = 4$	46
7	Number of Allowable Pairs (σ, τ)	49
8	Values of $k(\mathcal{B})$ for $n = 3$	51
9	Number of outputs and $\mathcal{B}$ -sequences	52
10	Allowable Pairs (σ, τ) for $n = 3$	56
11	Allowable Pairs (σ, τ) in Binary Priority Queues	57
12	Number of τ for Fixed Insert-Delete Sequences	62
13	Number of Allowable Pairs (σ, τ)	72
14	$\alpha(n, t)$ for $n = 1 \dots 6$	78
15	Number of Allowable Pairs and Bracket Sequences for $n = 4$	79
16	Number of allowable pairs for all <i>Insert-Delete</i> Sequences	81
17	Number of τ and <i>Dd</i> Sequences for $n = 4$	83
18	Number of τ and Bracket Sequences for $n = 4$	84
19	Number of τ for Fixed $\mathcal{D}$ and Fixed $\mathcal{B}$ -sequences	86

List of Figures

1	Sequence of Events in an Ordinary Priority Queue	3
2	A Heap Ordered Binary Tree	5
3	Node with value 35 in Figure 2 changing its value to 14	6
4	Node with value 8 in Figure 2 changing its value to 13	7
5	A Complete Binary Tree with Heap Numbered Nodes	7
6	A Heap Ordered Binary Tree	8
7	Applying <i>DeleteMin</i> from the tree in Figure 6	8
8	Applying <i>Insert</i> into the tree in Figure 6	9
9	Building a Heap-Ordered Tree Using <i>MakeTree</i>	13
10	Construction of Binomial Tree B_3	15
11	Binomial Queue with 13 Nodes	16
12	Merging of two Binomial Queues	17
13	Adding B_0 Trees	17
14	Adding two B_1 Trees	18
15	Adding two B_2 Trees	18
16	Result of Merging Q_1 and Q_2	18
17	Input Queue for the <i>DeleteMin</i> Operation	19
18	Fragments After Deleting the Smallest Root	20
19	Formation of Q_1 and Q_2	20
20	Resultant Binomial Queue after Applying <i>DeleteMin</i>	20
21	A MinMax Heap	25
22	Hasse Diagram for the MinMax Heap of Figure 21	25
23	Encoding Alphabets Using Binary Tree	30

24	Generation of output sequences for $\sigma = 3\ 2\ 1$	36
25	Generation of output sequences for $\mathcal{B} = iididd$	37
26	Computing $s(32154)$ using a labeled binary tree	48
27	Generation of output sequences for $\sigma = 2\ 3\ 1$	70
28	Generation of output sequences for $\mathcal{B} = iirirx$	77

Chapter 1

Introduction

The abstract data type *priority queue* is a data type whose instances are sets containing an arbitrary number of ordered items, all of the same type. It is characterized by the operations *Insert*, *DeleteMin*, *MakeEmpty* and *IsEmpty*. The effect of the operations *Insert* and *DeleteMin* is the principal object of study in this thesis.

The name, *priority queue*, comes from the process management application in the design of an operating system, which involves the maintenance of a queue of processes, to be run according to their priorities. In a multiuser environment, the operating system scheduler must decide which of several processes to run. Generally a process is only allowed to run for a fixed period of time. Jobs are initially placed at the end of the queue. The scheduler will repeatedly take the first job on the queue, run it until either it finishes or its time limit is up, and place it at the end of the queue if it does not finish. This strategy is generally not appropriate, because very short jobs will seem to take a long time because of the wait involved to run. Generally it is important that short jobs finish as fast as possible, so these jobs should have preference over jobs that have already been running. Moreover, some jobs that are not short are still very important and should also have preference in some situations. This particular application requires a special kind of queue, and the priority queue comes in handy to provide the solution. Apart from its important use in the design of

operating systems, priority queues also have many other applications, such as *Huffman* coding, external sorting, discrete event simulation and implementation of *greedy algorithms*, which operate by repeatedly finding a minimum.

In addition to the type of priority queue which uses *Insert* and *DeleteMin* operations, there are also other types of priority queues, namely *double ended* priority queue and *binary* priority queue [1]. Double ended priority queues are similar to the (ordinary) priority queues except that they allow the operation, *DeleteMax*, in addition to *Insert* and *DeleteMin*. This means the double ended priority queue can output the next element currently held in the queue by either applying *DeleteMin* or *DeleteMax* operation. Both ordinary priority queue and double ended priority queue use priorities represented in the form of integers as their input sequences. The notion of binary priority queue is relatively new [1]. Unlike ordinary and double ended priority queues, binary priority queue uses input sequences which are binary sequences and this means that they can process jobs with two levels of priorities, namely 0 and 1.

The research work presented in this thesis discusses some interesting numerical results obtained with ordinary, binary and double ended priority queues and presents some mathematical analysis which establish relationships between the input and output sequences.

In what follows in this first chapter we briefly look at some of the implementations of ordinary and double ended priority queues already discussed in the literature and provide example applications of them. Also in the following sections, when we talk about *priority queue* without a qualifier such as *double ended* or *binary*, we assume it to be an *ordinary priority queue* using the operations *Insert* and *DeleteMin*. It is needless to say that the operation *DeleteMin* can be substituted with the operation *DeleteMax* to produce an analogous ordinary priority queue which always outputs the largest element currently held in the queue instead of the smallest one.

Figure 1: Sequence of Events in an Ordinary Priority Queue

1.1 Specification: Ordinary Priority Queue

Suppose that $m_1 m_2 \dots m_n$ is some stream of input to a priority queue. Each *Insert* operation places the next item of the input stream in the priority queue and each *DeleteMin* operation removes the current smallest element and places it in an output stream. After n *Insert* operations and n *DeleteMin* operations, the input stream will be exhausted, the priority queue will be empty, and the output permutation will contain some permutations of the input stream. The only restriction on a valid sequence of *Insert* and *DeleteMin* operations is that *DeleteMin* must not be applied to an empty priority queue.

The following example illustrates the action of a priority queue on a given input stream. Assume that there are four ordered items, which for convenience sake are taken as the integers 1, 2, 3 and 4, on the input stream. Let us say they are arranged as 3, 1, 2, and 4 before being input into the priority queue. Let us also assume that the sequence of operations performed by the priority queue is *iidiidd*, where *i* and *d* denote the *Insert* and *DeleteMin* operations respectively. The sequence of operations,

ididdd, will first insert the first two elements from the input stream into the priority queue, delete one element from the queue, then insert the remaining two elements followed by three deletions from the priority queue. Figure 1 illustrates the sequence of events.

1.2 Implementation of Ordinary Priority Queues Using Heap Ordered Trees

There are several ways of implementing a priority queue, using a simple linked list to a binary tree. Using a sorted linked list, we could insert the elements in a sorted order at a cost of $O(n)$ time, but delete the minimum at a cost of $O(1)$ time. Similarly an unsorted linked list would require an $O(1)$ to do the insertion and an obvious $O(n)$ to do the deletion.

A better way of implementing priority queue is to use a heap [19], [21]. A heap is a certain type of binary tree (heap ordered and heap shaped, see below) that allows *Insert* and *Delete Min* operations to be accomplished in $O(\log n)$ worst case running time. A further advantage of a heap is that it can be implicitly represented in an array.

1.2.1 Location of Nodes in the Binary Tree

A binary tree is *heap ordered* (or *partially ordered*) if the value at each node in the tree is no larger than the values at its children. Figure 2 below shows an example of a heap ordered binary tree.

Figure 2: A Heap Ordered Binary Tree

The following observations about heap ordered trees suggest why they are useful as a data structure for implementing priority queues.

- The element with the smallest value resides at the root of the tree - this makes it easy to locate the minimum element.
- If the priority of an item is changed, then we can restore the tree to heap order fairly easily.

If the changed element's value is now smaller than the value of its parent, we could restore the heap order by letting the element move up one level at a time in the tree along the path to the root. And its parent moves down to the element's previous location. When the changed element stops moving up, either it is the child of a node whose value is smaller, or it is the root. In either case, when the changed element stops moving up, the entire tree is heap ordered again. This is shown in Figure 3 below. Note that the nodes affected have borders with dotted lines.

On the other hand, if the changed element's value is now larger than the key at one of its children, then we could restore the heap order by moving the element down in the tree. At each step in its descend, the changed element swaps places with the element among its children whose value is minimum. This is shown in Figure 4. Again nodes affected have borders with dotted lines. Hence, both before and after the changed element moves down, the entire tree minus the subtree rooted at the changed element is heap ordered. The changed element descends down the tree until either it becomes

Figure 3: Node with value 35 in Figure 2 changing its value to 14

the parent of nodes whose values are all at least as large as its value, or it becomes an end node or leaf. In either case, the entire tree is heap ordered when the changed element stops descending down.

In order to implement the *Insert* procedure, we append the element as a leaf of the tree, and let it move up toward the root as necessary to restore the heap order of the tree. To implement the *DeleteMin* procedure, we replace the element at the root by the element at any leaf and let this element descend down the tree as necessary to restore the heap order. During the restoration of the heap order, the element either ascends toward the root or descends toward the leaves - i.e: it moves only in one direction. Hence if h is the height of the tree, the changed element cannot move more than h levels away from its original position. Thus if we can restrict the binary tree to be of height $O(\log n)$ (i.e: with n nodes), then we can perform a priority queue operation, *Insert* or *DeleteMin*, and still maintain the heap order in $O(\log n)$ time.

1.2.2 Shape of the heap

A binary tree is heap shaped if every level is complete, except possibly the last level where the nodes are as far to the left as possible. In such a tree the nodes can be numbered by the following rules: (a) the root of the tree is numbered as 1 (b) the left and right children of the node numbered i are $2i$ and $2i + 1$ respectively.

A 15 node-complete binary tree constructed according to the above rules is shown in

Figure 4: Node with value 8 in Figure 2 changing its value to 13

5. Hence it is apparent that the node numbered i would appear in the tree at depth $\lfloor \log_2 i \rfloor$, at a position $i - 2^{\lfloor \log_2 i \rfloor}$ from the left of the tree. By applying the above rules on nodes numbered from 1 to n , we could form a heap shaped subtree of height $\lfloor \log_2 n \rfloor$. A heap ordered binary tree on n nodes that has the heap shape property is simply called a *heap*.

Figure 5: A Complete Binary Tree with Heap Numbered Nodes

The following example illustrates the operations of a priority queue on a heap ordered tree shown in Figure 6.

Examining Figure 6 we could see that one can locate the minimum element in $O(1)$ time by looking at the root of the above heap tree. To apply the procedure *DeleteMin*, we move the element at the node numbered n to the root of the heap and let it fall down the tree until the heap order is restored. Thus, when we apply *DeleteMin* on

the tree given in Figure 6, we put 36 at the root replacing 4, then let it fall down the tree. Figure 7 shows the result after 36 has fallen two levels to a leaf of the tree.

Figure 6: A Heap Ordered Binary Tree

In order to apply the procedure *Insert*, we append it to the location $n + 1$ and let it move up in the tree until the heap order is restored. Figure 8 shows the result of adding 20 to the tree of Figure 6. As can be observed, 20 moves up by only one level in order to regain its heap order. Since the height of the tree is $O(\log n)$, both *DeleteMin* and *Insert* can be achieved in $O(\log n)$ time.

Figure 7: Applying *DeleteMin* from the tree in Figure 6

Figure 8: Applying *Insert* into the tree in Figure 6

1.2.3 Implicit Representation of a Heap Ordered Tree

When applying the procedures *Insert* and *DeleteMin* on the heap, we impose the restriction that the nodes at the bottom level be left-justified. This means, whenever an *Insert* is performed, the new node is added to the right most leaf of the bottom level and whenever a *DeleteMin* is performed, the root is removed and the node from the right most leaf of the bottom level is moved to the root. The advantage of building a left-justified binary tree is that by numbering the nodes from top to bottom and left to right, the children of node i are $2i$ and $2i + 1$ if they exist, and the parent of node i is $\lceil i/2 \rceil$ if it exists. Such a binary tree can be stored in a linear array. In order to move about in the tree, one has to only calculate the indices of parents and children.

For example the tree,

can be represented by the array,

A =	3	5	7	11	16	23
	i=1	i = 2	i = 3	i = 4	i = 5	i = 6

Using the above array implementation of heap, we construct priority queue by means of a record as shown below

```
type
  PriorityQueue = record
    node : array [1 ...MaxSize] of NodeType;
    size : integer;
  end;
```

The above record declaration of heap data structure consists of two fields, an array of **NodeType** holding the elements of the queue and an integer representing the current heap size. For ease of manipulation **NodeType** is substituted by integer in the actual implementations given in the appendices. The algorithms *Insert* and *DeleteMin* are implemented in Pascal below.

```
procedure Insert(x : NodeType; var q : PriorityQueue);
var i : integer;
begin
  if q.size < MaxSize then
    begin
      inc(q.size);
      i := q.size;
      while q.node[i div 2] > x do
        begin
          q.node[i] := q.node[i div 2];
          i := i div 2;
        end;
      q.node[i] := x;
    end;
  end;
```

If the element to be inserted into the heap is the new minimum, the running time of *Insert* will be as much as $O(\log n)$ as the new element has to move all the way to the root. But on average, the moving up terminates early. This has been shown by Porter and Simon [16] by analyzing the average cost of inserting a random element into a

random heap in terms of element exchanges. They proved that 2.607 comparisons are required on average to perform an insert, so the average *Insert* moves an element up by 1.607 levels. In order to make the *while* loop in the above procedure terminate, we place a very small value in position 0 of the array. This value is guaranteed to be smaller or equal to any element in the heap and thus acts as a sentinel which is similar to the header nodes in linked lists.

```

function DeleteMin(var Q : PriorityQueue) : NodeType;
var   i, child : integer;
      LastNode : NodeType;
      done : boolean;
begin
  if Q.size > 0 then
    begin
      DeleteMin := Q.node[1];
      LastNode := Q.node[Q.size];
      dec(Q.size);
      i := 1;
      done := false;
      while (i*2 <= Q.size) and (not done) do
        begin
          child := i*2 { find smaller child }
          if child <> Q.size then
            if Q.node[child+1] < Q.node[child] then
              inc(child);
            if LastNode > Q.node[child] then { move }
              begin
                Q.node[i] := Q.node[child];
                i := child;
              end else done := true
            end;
          Q.node[i] := LastNode;
        end;
      end;
    end;
  end;
end;

```

The worst-case running time of *DeleteMin* is obviously $O(\log n)$. On average, the

element that is placed at the root is pushed down almost to the bottom of the heap (which is the level it came from), hence the average running time is $O(\log n)$.

In order to efficiently manipulate priority queues, in addition to the *Insert* and *DeleteMin* operations, we may also need another operation to take n input elements and place them into an empty heap. Obviously this can be done with n successive *Inserts*. Since each *Insert* takes $O(1)$ average and $O(\log n)$ worst-case time, the total running time of this operation would be $O(n)$ average but $O(n \log n)$ worst-case. The following procedure *BuildTree* provides a better algorithm with linear time bound for the above operation.

```
procedure BuildTree;  
begin  
  for  $i := n \text{ div } 2$  downto 1 do  
    percolate-down( $i$ )  
  end
```

The operation *percolate-down(i)* performs a heap-order restructuring of the sub-tree rooted at node i . We place the n input elements into the tree in any arbitrary order, maintaining the structure property and invoke *BuildTree* on it to create a heap-ordered tree.

The example given in Figure 9 illustrates how the algorithm *BuildTree* creates a heap-ordered tree from an unordered tree. The first tree in Figure 9 is the unordered tree and the seven remaining trees show the result of each of the seven *percolate-downs*. The line connecting two dashed-circles corresponds to two comparisons: one to find the smaller child and one to compare the smaller child with the node i . It is apparent that in order to obtain the bound for the running time of *BuildTree*, we need to bound the number of lines connecting two dashed-circles in the tree. This can be obviously done by computing the sum of the heights of all the nodes in the tree, which is the maximum number of lines connecting two dashed-circles.

Initial Tree

After percolate-down(7)

After percolate-down(6)

After percolate-down(5)

After percolate-down(4)

After percolate-down(3)

After percolate-down(2)

After percolate-down(1)

Figure 9: Building a Heap-Ordered Tree Using *MakeTree*

If the height of the tree is h , then it is easy to see that our tree consists of 1 node at height h , 2 nodes at height $h - 1$, 2^2 nodes at height $h - 2$, and in general 2^i nodes at height $h - i$. Hence the sum of the heights of all the nodes is then,

$$\begin{aligned} s &= \sum_{i=0}^h 2^i (h - i) \\ &= h + 2(h - 1) + 4(h - 2) + 8(h - 3) + \cdots + 2^{h-1}(1) \end{aligned}$$

$$\text{Hence } 2s = 2h + 4(h - 1) + 8(h - 2) + 16(h - 3) + \cdots + 2^h(1)$$

Hence,

$$\begin{aligned} -(s - 2s) &= -h + 2 + 4 + 8 + \cdots + 2^{h-1} + 2^h \\ &= (2^{h+1} - 1) - (h + 1) \end{aligned}$$

This shows that for a complete tree with $n = 2^h$ nodes, the sum of heights has a bound of approximately $2n$ and for a tree with $n = 2^{h+1}$ nodes it has a bound of approximately n , which in turn implies that *BuildTree* has a linear running time.

Hence summarizing the implementation of ordinary priority queue using heap ordered binary trees, we can tabulate average and worst case running times of the major operations as follows:

Table 1: Running Times: Operations with Heap Ordered Tree

Operations	Average Running Time	Worst-case Running Time
Insert	$O(1)$	$O(\log n)$
DeleteMin	$O(\log n)$	$O(\log n)$
BuildTree	$O(n)$	$O(n)$

1.3 Implementation of Ordinary Priority Queues Using Binomial Queues

Using binomial queue as a data structure for implementing ordinary priority queues was originally proposed by Vuillemin in 1978 [18]. Binomial queues support the operations *Insert*, *DeleteMin* and *Merge* in $O(\log n)$ worst case time, but *Insert* takes constant time on average. Binomial queues differ from the previous implementation of priority queues in that a binomial queue is not a heap-ordered tree but rather a *collection* of heap-ordered trees, known as a *forest*. Each of the heap-ordered trees is of a constrained form known as a binomial tree.

1.3.1 Binomial trees and their properties

A binomial tree B_r is a tree structure defined inductively by,

The binomial tree B_r is created by linking together or merging two copies of B_{r-1} . For example the construction of binomial tree B_3 from B_0 is,

Figure 10: Construction of Binomial Tree B_3

By examining the construction of a binomial tree, we could observe that the binomial trees have the following interesting properties:

- n th binomial tree B_n has 2^n nodes.
- The height of the n th binomial tree, $h(B_n) = n + 1$.
- The root of the n th binomial tree, B_n has exactly n children.
- The binomial tree B_n contains exactly $\binom{n}{d}$ nodes of depth d .

As one can expect, the name binomial tree originated from the last property mentioned above. From this property we can also easily show that the internal path length of the n th binary tree, $i(B_n) = n2^{n-1}$.

1.3.2 Binomial Queues

A binomial queue is a forest of heap-ordered binary trees. The sizes of the binomial trees forming the binomial queue are determined by the binary representation of n , the number of nodes in the binomial queue. For example if the number of nodes in the queue is 13 (1101_2), then we could have one B_3 , one B_2 and one B_0 in the forest as shown below.

Figure 11: Binomial Queue with 13 Nodes

1.3.3 Merging of Two Binomial Queues

One of the important operations performed on binomial queues is merging, which, in its trivial form, is how an insertion of a new node is carried out on priority queues using binomial queue as their data structure.

We will illustrate the steps involved in merging with an example. Let us consider the following binomial queues, Q_1 , and Q_2 .

Figure 12: Merging of two Binomial Queues

Merging of two binomial queues is done in a manner quite analogous to the binary addition $0111 + 0011 = 1010$. Hence we would first link the two B_0 trees together, obtaining no B_0 trees and one B_1 tree. This B_1 tree is stored in an intermediate array as shown below.

Figure 13: Adding B_0 Trees

Since we now have three B_1 trees, two of them can be linked together to give a B_2 tree as shown below.

Figure 14: Adding two B_1 Trees

Linking the above B_2 tree with the B_2 tree in Q_1 , we obtain a B_3 tree as given below:

Figure 15: Adding two B_2 Trees

Hence the resulting binomial queue will have the following structure:

Figure 16: Result of Merging Q_1 and Q_2

Since at most one link per array position is carried out, this gives us an $O(\log n)$ worst-case complexity per merge, where n is the number of nodes in the resultant binomial queue. This result follows from the observation that the largest binomial tree, B_i has at most n nodes, and since B_i has 2^i nodes, $i \leq \log_2 n$.

1.3.4 Insert on Binomial Queue Representation

Insertion is a special case of merging: merging of one node tree into another. Hence the worst-case running time of *Insert* will be that of merging which is $O(\log n)$.

1.3.5 DeleteMin on Binomial Queue Representation

A *DeleteMin* operation on the priority queue implemented using binomial queues should perform the following actions:

- Find the smallest of the $O(\log n)$ roots of all the binomial trees.
- Delete the smallest root.
- Merge the resulting binomial queues consisting of the trees rooted at the deleted node's children back into the main queue.

Let us illustrate the *DeleteMin* operation by applying it on the following binomial queue.

Figure 17: Input Queue for the *DeleteMin* Operation

The smallest root is 3, so we delete it and obtain the following:

Figure 18: Fragments After Deleting the Smallest Root

Now we merge the above fragments into two binomial queues Q_1 and Q_2 as shown below.

Figure 19: Formation of Q_1 and Q_2

Finally merging Q_1 and Q_2 together, we obtain the following binomial queue:

Figure 20: Resultant Binomial Queue after Applying *DeleteMin*

It takes $O(\log n)$ time to find the tree containing the smallest root, and to merge the fragments into Q_1 and Q_2 . Merging the two binomial queues Q_1 and Q_2 together also takes $O(\log n)$ time. Thus the entire *DeleteMin* operation takes $O(\log n)$ time.

1.4 Other Priority Queue Implementations

In addition to heaps and binomial queues, many other forms of ordinary priority queue implementations have been discussed in the literature. The following is a brief look at some of them [12].

Linear Linked List

In linear linked list implementations of priority queues, using a sorted linked list, we could insert the elements in a sorted order at a cost of $O(n)$ time, but delete the minimum at a cost of $O(1)$ time. Similarly using an unsorted linked list we need an $O(1)$ time to do the insertion and an obvious $O(n)$ to do the deletion.

Leftist Tree

Leftist trees use the heap structure, but in this implementation, the heap is explicitly represented with pointers from parents to their children. Both the *Insert* and *DeleteMin* operations are done using the merge operation. The *Insert* operation initializes the new element as a one-node tree before merging it with the original tree. The *DeleteMin* operation returns the root of the tree and merges the resulting subtrees.

Pagodas

Pagodas are based on heap ordered binary trees. The primary pointers in a pagoda lead from the leaves of the tree toward the root. A secondary pointer in each element of the tree points downwards from that to its otherwise unreachable left- or rightmost descendant. The root does not have any upward pointer but has pointers to both its left- and rightmost descendants. Hence, all elements in a pagoda are reachable from the root by somewhat complex paths, and all branches of the structure are circularly

linked. The *Insert* and *DeleteMin* operations on pagodas are based on merging the right branch of one pagoda with the left branch of another.

Skew Heaps

Two major variants of skew heaps have been implemented: the bottom-up variant and the top-down variant. The top-down variant uses a simple binary tree analogous to that used with leftist trees. The bottom-up variant is similar in many ways to pagodas. The pointers along the skeleton of the tree structure point from leaves toward the root, and each item has a second pointer pointing down to an otherwise unreachable element. This downward pointer always points to the rightmost descendant of the left child, and the root contains pointers to the rightmost descendants of both children.

Fibonacci Heaps

A Fibonacci heap is a forest of heap-ordered trees, which need not be any particular shape or size. Some nodes may be *marked*, which is done with an asterisk. A root node is never marked. A pointer to a root node containing a minimum entry is maintained. To implement the *Insert* operation, the element is converted into a one-node Fibonacci heap, and merged with the original heap. The *DeleteMin* operation is done by removing the minimum node, thus making its children become roots of trees. Any marked children of the deleted node are made unmarked, in line with the policy of having no marked root nodes. Then these trees are linked together in the same way as was done for binomial queues earlier.

1.5 Comparison of Implementation Schemes

The following table [12] gives the time complexities for the major priority queue operations. These operations are, in addition to *Insert* and *DeleteMin*, *initialize* - create an empty queue, *empty* - empty the queue, *FindMin* - locate the minimum element, *Merge* - meld two queues, *delete* - delete a given element from the queue, and *DecreaseKey* - reduce the priority of an element. Except for Fibonacci heaps where the

time complexities are amortized, all times given are worst-case time complexities.

It should be noted that the heap is the simplest efficient implementation scheme. For the operations, *initialize*, *empty*, *Insert*, and *DeleteMin*, which are all that are required in the majority of applications, it is generally the method of choice. All the ordinary priority queue algorithms listed in the Appendices are based on the heap implementation.

Table 2: Comparison of Major Implementation Schemes

<i>Operations</i>	<i>Unsorted List</i>	<i>Heap</i>	<i>Binomial Queue</i>	<i>Fibonacci Heap</i>
Initialize	$O(1)$	$O(1)$	$O(1)$	$O(1)$
Empty	$O(1)$	$O(1)$	$O(1)$	$O(1)$
Insert	$O(1)$	$O(\log n)$	$O(\log n)$	$O(1)$
DeleteMin	$O(n)$	$O(\log n)$	$O(\log n)$	$O(\log n)$
FindMin	$O(n)$	$O(1)$	$O(\log n)$	$O(1)$
Delete	$O(1)$	$O(\log n)$	$O(\log n)$	$O(\log n)$
Merge	$O(1)$	$O(n)$	$O(\log n)$	$O(1)$
DecreaseKey	$O(1)$	$O(\log n)$	$O(\log n)$	$O(1)$

As can be seen from the above table, Fibonacci heap is the most efficient implementation of the full complement of priority queue operations that is currently known. The complexity bounds shown above for the Fibonacci heap have led to asymptotically efficient implementations of several important graph algorithms, most notably Dijkstra's and Prim's algorithms.

1.6 An Implementation of Double Ended Priority Queues: MinMax Heaps

As discussed earlier, a common way of implementing (ordinary) priority queues is by means of a heap. A number of implementations of double ended priority queues which use heaps or heap-like structures have also been discussed in the literature.

Some examples of this are the *MinMax Heap* proposed by Atkinson *et al* [5], *Deap*, a variant of MinMax Heap, proposed by Carlsson [7] and the structure with two heaps placed 'back-to-back' proposed by Williams [20]. In this section we discuss the implementation of double ended priority queues using the original MinMax heap proposed by Atkinson *et al*.

The traditional heaps can be either min-ordered or max-ordered. A min-ordered heap is a binary tree having the following properties:

- heap-shape: all leaves lie on at most two adjacent levels and the leaves on the last level occupy the leftmost positions; all other levels are complete.
- min-ordering: the value stored at a node is smaller than or equal to the values stored at its children.

It is apparent that a max-ordered heap is like a min-ordered heap except that it is max-ordered, i.e: the value stored at a node in a max-ordered heap is greater than or equal to the values stored at its children.

A *MinMax heap* is a binary tree T with the following properties:

- T has the heap-shape.
- T is *MinMax* ordered: i.e. values stored at nodes on even (odd) levels are smaller (greater) than or equal to the values stored at their descendants (if any) where the root is at level zero.

Hence the smallest value of T is stored at the root of T and the largest value is stored at one of the root's children. This is illustrated by the example shown in Figure 21. Similarly in a MaxMin heap, the largest value of the tree is stored at the root and the smallest value is stored at one of the root's children. An array of n elements can be used to implicitly represent a MinMax heap. In such a representation, the i th location in the array will correspond to a node located on level $\lfloor (\log_2 i) \rfloor$ in the

MinMax heap.

Figure 21: A MinMax Heap

The Hasse diagram for a MinMax heap representing the order relationships implicit within the heap is more complex than the one for the traditional heap. For the traditional heap, the Hasse diagram is the heap itself. The Hasse diagram for the heap shown in Figure 21 is shown in Figure 22.

Figure 22: Hasse Diagram for the MinMax Heap of Figure 21

MinMax heaps are created using a method based on the Floyd's algorithm [9]. We

represent the MinMax heap in an array A of n entries. When the algorithm examines the subtree rooted at node $A[i]$, the both subtrees of $A[i]$ are max-ordered, but the subtree itself may not necessarily be max-ordered. The *TrickleDown* step of Floyd's algorithm exchanges the value at $A[i]$ with the maximum of its children. The *TrickleDown* step is then applied recursively to this maximum child to maintain the max-ordering. From Figure 22 we could easily see that in order to construct a Min-Max heap, we need to establish the correct ordering between an element, its children, and its grandchildren and be able to differentiate between min- and max-levels.

The procedures, *TrickleDown* and *TrickleDownMin* listed below perform the above tasks. It is apparent that the procedure *TrickleDownMax* is similar to *TrickleDownMin* except that the relational operators are reversed.

```
procedure TrickleDown(i)
begin
  if i is on a min level then
    TrickleDownMin(i)
  else
    TrickleDownMax(i)
  endif
end
```

The operations *DeleteMin* and *DeleteMax* are carried out by removing the required element from the heap and filling the empty position with the last element of the heap - a process very similar to the *Delete* operation in the traditional heaps. Subsequent application of the *TrickleDown* procedure restores the MinMax ordering of the heap.

The *Insert* operation is carried out by placing the element to be inserted into the first available leaf position and then re-establishing the ordering on the path from this element to the root. It is apparent from the Hasse diagram that the leaf-positions of the heap correspond to the elements located on the middle row in the Hasse diagram.


```
procedure TrickleDownMin(i)
begin
  if A[i] has children then
    m := index of smallest of the children and
        grandchildren (if any) of A[i]
    if A[m] is a grandchild of A[i] then
      if A[m] < A[i] then
        swap A[i] and A[m]
        if A[m] > A[parent(m)] then
          swap A[m] and A[parent(m)]
        endif
      endif
      TrickleDownMin(m)
    endif
  else A[m] is a child of A[i]
    if A[m] < A[i] then
      swap A[i] and A[m]
    endif
  endif
endif
end
```

In order to restore the MinMax ordering, the new element is placed into the next available leaf position, and must then move up the Hasse diagram toward the top, or down toward the bottom, to ensure that all paths running from top to bottom remain sorted. Hence the *Insert* algorithm must first decide whether the new element should proceed further down the Hasse diagram (i.e: up the heap on max-levels) or up the Hasse diagram (i.e: up the heap on successive min-levels). Then the algorithm needs to examine only the grandparents along the path to the root of the heap. These are the ones which are located on the min-levels or the ones located on the max-levels. The process is given as the following two procedures, *BubbleUp* and *BubbleUpMin*.

The procedure *BubbleUpMax* is similar to *BubbleUpMin* except that the relational operators are reversed.

```
procedure BubbleUp(i)
begin
  if i is on a min-level then
    if i has a parent and  $A[i] > A[\text{parent}(i)]$  then
      swap  $A[i]$  and  $A[\text{parent}(i)]$ 
      BubbleUpMax(parent(i))
    else
      BubbleUpMin(i)
    endif
  else
    if i has a parent and  $A[i] < A[\text{parent}(i)]$  then
      swap  $A[i]$  and  $A[\text{parent}(i)]$ 
      BubbleUpMin(parent(i))
    else
      BubbleUpMax(i)
    endif
  endif
end
```

```
procedure BubbleUpMin(i)
begin
  if  $A[i]$  has grandparent then
    if  $A[i] < A[\text{grandparent}(i)]$  then
      swap  $A[i]$  and  $A[\text{grandparent}(i)]$ 
      BubbleUpMin(grandparent(i))
    endif
  endif
end
```

Reference [4] provides the complexity analysis of all the above operations, *Create*, *DeleteMin*, *DeleteMax*, and *Insert*. Table 3 compares the worst-case time complexities for min-heaps and MinMax heaps.

Table 3: Comparison of Min Heap and MinMax Heap

<i>Operations</i>	<i>Min Heap</i>	<i>MinMax Heap</i>
Create	$2n$	$7n/3$
Insert	$\log(n + 1)$	$0.5 \log(n + 1)$
DeleteMin	$2 \log(n)$	$2.5 \log(n)$
DeleteMax	$0.5n + \log(n)$	$2.5 \log(n)$

1.7 Some Applications of Priority Queues

1.7.1 Huffman Trees: An Application of Ordinary Priority Queues

Most computer architectures represent characters using 7-bit codes such as ASCII (American Standard for Information Interchange) or BCDIC (Binary Coded Decimal Interchange Code). For example in ASCII, the 7-bit binary code 1000001 represents the character *A*, and 1010001 represents *a*. In these character representation schemes, all codes are of the same length. Thus storage and retrieval of codes are easy. But, when all the codes are of the same length, a code can use more bits than actually necessary to represent a sequence of characters. This is an obvious disadvantage in the ASCII and BCDIC schemes.

Huffman coding, [11], [17] using priority queues for its implementation, overcomes this problem. Huffman coding is a compact binary encoding technique which uses binary codes that represent sequences of characters in far fewer bits than the encoding commonly used for variables of type *char* in other schemes.

Figure 23: Encoding Alphabets Using Binary Tree

We could use fewer bits than ASCII or BCDIC schemes by encoding frequent characters in short codes and infrequent characters in long codes. The natural relationship between binary codes and binary trees illustrates the construction of codes with different lengths. One such scheme would keep the characters to be encoded at the leaves of a binary tree. To encode a character located in a leaf of such tree, we traverse the path from the root to the leaf. Each time we go from a node to its left child, our coding scheme writes out a 0 and each time we go to the right tree, it writes out a 1. The string of bits written down as we traverse from root to the leaf is the code word for the character located in the leaf. Figure 23 shows an encoding binary tree used to encode 6 characters, *a*, *b*, *c*, *d*, *e*, and *f*. Recall that a 7-bit encoding scheme such as ASCII or BCDIC would require a binary tree of height 7.

In order to encode frequent characters in fewer bits than infrequent characters, frequent characters should obviously appear at a smaller depth in the encoding tree than the characters that are infrequently used. For this purpose, we could use a generalization of path length to compare trees. We associate a weight with each character to give a measurement of frequency of the characters. If the weight of a character located in a given leaf l_i is w_i , then the weighted external path length of the tree is,

$$\sum_{l_i} w_i \times \text{depth}(l_i)$$

Obviously if two trees have the same set of characters at their leaves, then the tree with

smaller weighted external path length gives a better encoding scheme for the given distribution of weights. A Huffman tree is a tree with the minimum weighted external path length for a given set of characters and their associated weights and Huffman algorithm provides an optimal code in the sense of minimizing $\sum w_i \times \text{depth}(l_i)$. The Huffman algorithm is given below as a simple pseudocode.

```
procedure Huffman-Algorithm;  
begin  
  Forest-Setup Step:  
    Construct a forest of one-node trees  
    Each tree contains a single item.  
    The weight of each tree is the weight of the item it contains.  
  
  Tree-Building Step:  
    Until the forest contains only one tree,  
      remove two trees of smallest weights from the forest,  
      make them children of a new node whose weight is the sum  
      of their weights, and  
      add the new tree to the forest.  
end
```

The forest-setup step builds the pieces out of which the Huffman tree will be constructed by the tree-building step. The tree-building step requires that we be able to find and remove the smallest tree in the forest, and to add a tree to the forest. Hence Huffman algorithm can use a priority queue of binary tree for its implementation.

A large file whose contents are unlikely to change is a good candidate for Huffman coding. Here we could obtain frequency statistics for each characters directly from the data file, then construct a Huffman code that is minimal for the file so that the encoding uses no more bits than any other coding scheme which always uses the same code for each character.

1.7.2 External Quicksort: An Application of Double Ended Priority Queues

External quicksort, used as an ideal sorting routine for direct access files, is one of the important applications of double ended priority queues [10], [14]. The sorting procedure does not require any additional file storage since records are sorted *in place*. The basic algorithm is the same as the internal quicksort algorithm which uses arbitrarily chosen record as a pivot to split the file into two parts containing records with keys smaller and larger than that of the pivot. Thus the original problem is reduced to two simpler problems, namely to sort the subfile to the left and to sort the subfile to the right of the pivot. The procedure is applied to each subfile until the problem is solved. With external quicksort however, instead of using a single pivot, the procedure keeps several pivots in memory which are updated as the splitting phase progresses. The goal is to produce an even split and to place as many records as possible in their final position.

To sort a file, records are read from both ends of the file and these positions may then be written to the memory. The pivots in memory are maintained so that at the end of each pass through an unsorted file, they are written as close to the center of the file as possible. Therefore, as the splitting phase progresses, either the minimum or maximum pivot is written to the file and replaced by a new record which is read in. The reading of records is done as balanced as possible with respect to the writing positions. A double ended priority queue used to store the pivots allows both the minimum and maximum to be found and extracted efficiently. In addition, a new record which is read in can be inserted efficiently into the queue.

Chapter 2

Ordinary Priority Queues

An ordinary priority queue transforms an input permutation σ of some set of size n into an output permutation τ using the operation *DeleteMin* to output symbols from the queue. We can input one or more symbols from the input permutation σ into the queue at each input operation. There is no rule governing the manner in which the symbols are input to the queue, except that they should be input into the queue sequentially treating the input stream as a regular conventional queue holding n symbols. The rule governing the manner in which the symbols are output from the priority queue is simple: only the smallest of the symbols currently held in the queue is output next regardless of the sequence of its arrival into the queue. It is obvious that we cannot remove a symbol from an empty priority queue. We can also see that inserting and outputting symbols into and from the priority queue in different orders may give rise to distinct output sequences for any given input sequence.

The usefulness of priority queue as an abstract data type comes from its ability to generate different ordered patterns of output sequences from a given input sequence. Hence our study of ordinary priority queue in this chapter involves the manipulation of patterns of successive *Insert* and *DeleteMin* operations on the input symbols to generate output sequences and the relationships between the *Insert*, *DeleteMin* operations, number and characteristics of input and output sequences. We shall first look at two different ways of generating output sequences using ordinary priority queues

and then study these schemes in the context of the relationships between the *Insert*, *DeleteMin* operations, number and characteristics of input and output sequences.

2.1 Generating Output Sequences

An input permutation σ of some set of size n can be transformed into an output permutation τ using the operations *Insert* and *DeleteMin*. Since the operation *DeleteMin* cannot be carried out on an empty priority queue it is obvious that any given sequence of *Insert* and *DeleteMin* operations must be well-formed in the sense of matching bracket sequences. In other words, in a valid sequence of *Insert* and *DeleteMin* operations, every *DeleteMin* operation should have a corresponding *Insert* operation somewhere in the subsequence preceding it. Since the number of valid sequences of *Insert* and *DeleteMin* operations is an important quantity in the computation of the output permutations in a priority queue, we will first determine this quantity in the next section. Because the valid sequences of *Insert* and *DeleteMin* operations satisfy the same restrictions as an admissible sequence of n pairs of brackets we shall also refer to them as *bracket sequences*.

2.1.1 Valid Sequences of *Insert* and *DeleteMin*

Lemma 1 *The number of valid sequences of n Insert and n DeleteMin operations on a priority queue is equal to the Catalan number, $c_n = \frac{\binom{2n}{n}}{n+1}$.*

Proof: [13] Let i and d denote *Insert* and *DeleteMin* respectively. A sequence of *Insert* and *DeleteMin* operations can be given in the form of a string consisting of n i 's and n d 's. There are obviously $\binom{2n}{n}$ sequences of i 's and d 's that contain n of each. The number of invalid (inadmissible) sequences of i 's and d 's, which contain equal number of i 's and d 's but as one moves from left to right the string contains more number of d 's than i 's, can be calculated as follows.

In any inadmissible sequence, locate the first d for which d 's outnumber the i 's. Then in the partial sequence leading up to and including this d , replace all d 's by i and all

i 's by d . The result is a sequence with $(n + 1)$ i 's and $(n - 1)$ d 's. Conversely for every sequence of the latter type we can reverse the process and find the inadmissible sequence of the former type which leads to it. For example, the sequence $ddidiiiiddiii$ must have come from $iididddddiii$. This correspondence shows that the number of inadmissible sequences is $\binom{2n}{n-1}$. Hence the number of valid sequences of *Insert* and *DeleteMin* operations is,

$$c_n = \binom{2n}{n} - \binom{2n}{n-1} = \frac{\binom{2n}{n}}{n+1}$$

2.1.2 Two Methods of Generating Output Sequences

When considering the generation of output sequences using an ordinary priority queue there are two natural questions one might ask viz.,

(i) How many different outputs can be generated for each input permutation as the number of bracket sequences (i.e: *Insert*, *DeleteMin* sequences) varies over all possibilities? For example $\sigma = 3\ 2\ 1$ is an input sequence of length 3 made up of three distinct symbols. There are $c_3 = 5$ legal *Insert-DeleteMin* sequences, namely, $iiiddd$, $iiddid$, $iididd$, $idiidd$ and $ididid$ that can be used to generate the output sequences on the above input. Figure 24 illustrates the generation of output sequences. Algorithm PQ1 listed below generates outputs using this method.

(ii) How many different outputs can be generated for each bracket sequence as the number of input sequences varies over all permutations? For example $B = iididd$ is one of the 5 legal *Insert-DeleteMin* sequences. There are $3! = 6$ distinct input sequences, namely, $\sigma = 123, 132, 213, 231, 312$, and 321 that can be used to generate the output sequences on the input. Figure 25 illustrates the generation of output sequences using this method. An algorithm generating outputs using this scheme is also listed below.

Figure 24: Generation of output sequences for $\sigma = 3\ 2\ 1$

```

procedure PQ1;
begin
  NoOf- $\sigma := n!$ ;
  NoOf- $\mathcal{B} := c_n$ ;
  EmptyPriorityQueue( $Q$ );
  for  $i := 1$  to NoOf- $\sigma$  do
    input :=  $\sigma[i]$ ;
    for  $j := 1$  to NoOf- $\mathcal{B}$  do
      for  $k := 1$  to  $n$  do { insert each symbol in  $\sigma[i]$  }
        insert( $\sigma_k[i]$ );
         $\tau_k := DeleteMin(Q)$ ;
      endfor;
    endfor;
    output( $\tau$ );
  endfor;
end { PQ1 }

```

As can be deduced from the algorithms PQ1 and PQ2, calculating the numbers of outputs in (i), (ii) is expensive to do by direct enumeration (running time is equal to $\Omega\left(\frac{(2n)!}{n!}\right)$ in the case of PQ1 and $\Omega(n!)$ in the case of PQ2). We shall discuss some formulae and better algorithms to compute these numbers in the following sections.

Figure 25: Generation of output sequences for $B = iididd$

```

procedure PQ2;
begin
  NoOf- $\sigma := n!$ ;
  NoOf- $B := c_n$ ;
  EmptyPriorityQueue( $Q$ );
  for  $i := 1$  to NoOf- $B$  do
    for  $j := 1$  to NoOf- $\sigma$  do
      input :=  $\sigma[j]$ ;
      for  $k := 1$  to  $n$  do { insert each symbol in  $\sigma[j]$  }
        insert( $\sigma_k[i]$ );
       $\tau_k := DeleteMin(Q)$ ;
      endfor;
    endfor;
    output( $\tau$ );
  endfor;
end { PQ2 }

```

2.2 Number of Allowable Pairs (σ, τ)

Consider the set $\mathcal{S}$ of input permutations and the set $\mathcal{T}$ of output permutations both of which are of size $n = 3$.

$$\mathcal{S} = \{123, 132, 213, 231, 312, 321\}$$

$$\mathcal{T} = \{123, 132, 213, 231, 312, 321\}$$

Using the output generation - method (i), we can list all the output permutations τ for each input permutation σ as listed in Table 4 below.

Table 4: Allowable Pairs (σ, τ) for $n = 3$

σ	τ
123	123
132	123, 132
213	123, 213
231	123, 213, 231
312	123, 132, 312
321	123, 213, 231, 312, 321

From Table 4 it is apparent that we cannot obtain every output permutation $\tau \in \mathcal{T}$ from every input permutation $\sigma \in \mathcal{S}$. Given a pair of input-output permutations (σ, τ) , it is easy to see if the output sequence can be obtained from the input sequence, i.e: if (σ, τ) is allowable. The algorithm *allowable-pair* which has a time complexity of $O(n \log n)$ ¹ determines if a given (σ, τ) pair is allowable [2].

Even though the input streams can consist of any arbitrary values integer or real, for ease of manipulation we consider mainly input streams consisting of distinct elements from the set of positive integers from 1 to n . Let σ be some such input which gives rise to some output sequence τ . It is easy to see that when the input is the sequence $1, 2, \dots, n$ only one output is possible. On the other hand, if the input sequence is

¹Using a heap-based implementation for *Insert* and *DeleteMin*

$n, n-1, \dots, 1$ every legal sequence of *Inserts* and *DeleteMins* gives rise to a different output. For any other input sequence the number of output sequences s_n will be, $1 \leq s_n \leq c_n$.

```

function allowable-pair( $\sigma, \tau$ : string): boolean;
begin
   $Q :=$  empty priority queue;
   $i := 1; j := 1;$ 
  while  $j \leq n$  do
    while  $\tau_j \notin Q$  do
      insert( $\sigma_i$ );
       $i := i + 1$ 
    endwhile
    if ( $\min(Q) \neq \tau_j$ ) then
      return(false) else
        DeleteMin;
         $j := j + 1;$ 
    endif;
  endwhile;
  return(true);
end; { allowable-pair }

```

We wish to find out how many output permutations can be generated for a given input permutation. But the complementary problem of finding out how many inputs can give rise to a given output is an easier one and analyzing this will give us insight into solving the original problem. Hence we will first look at the complementary problem. We can characterize the input sequences which give rise to a given output sequence as the set of linear extensions of a certain poset defined from the output. If π is a sequence we shall let $T(\pi)$ denote the set of all τ such that (π, τ) is allowable. Thus $T(\pi)$ is the set of sequences which can be output by a priority queue if the input stream is π . Let $S(\pi)$ be the set of all σ such that (σ, π) is allowable. This set is the set of input streams capable of generating π on the output stream. Moreover, let $t(\pi) = |T(\pi)|$ and $s(\pi) = |S(\pi)|$.

Lemma 2 Let σ be some input stream expressed in the form $\alpha m \beta$ where m is the maximal symbol. Suppose $\beta = b\gamma$ (i.e: b is the first symbol of β). Then $t(\sigma) = t(\alpha)t(\beta) + t(\alpha b m \gamma)$.

Proof: Clearly, in any sequence of the set $T(\sigma)$, since m is the maximal element in σ and since m follows all the symbols α in σ , m must come after the symbols of α . There are $t(\alpha)t(\beta)$ sequences in $T(\sigma)$ arising from outputting m as the next symbol after all the symbols in α . If we consider an input of the form $\sigma = \alpha b m \gamma$, it is obvious that m , being the maximal symbol, cannot be output immediately after the symbols in α . Hence those sequences in $T(\sigma)$ for which m does not immediately succeed the symbols of α are precisely the outputs that arise if $\alpha b m \gamma$ is processed by a priority queue. Hence the lemma follows.

Corollary 1 If $\beta = b_1 b_2 \cdots b_r$ and $\alpha_i = \alpha b_1 \cdots b_i$, $\beta_i = b_{i+1} \cdots b_r$ then,

$$t(\sigma) = \sum_{i=0}^r t(\alpha_i)t(\beta_i)$$

Proof: $t(\sigma) = t(\alpha_0 m \beta_0)$
 $= t(\alpha_0)t(\beta_0) + t(\alpha_0 b_1 m b_2 \cdots b_r)$ by the previous lemma
 $= t(\alpha_0)t(\beta_0) + t(\alpha_1 m \beta_1)$
 and the corollary follows by repeated application of the lemma

Example 1 Given that the number of output sequences for $\sigma = 32154$ is 10, compute the number of output sequences for $\sigma = 32514$.

Using lemma 2, $t(32514) = t(32)t(14) + t(32154)$. It is a trivial task to find out that $\sigma = 32$ and $\sigma = 14$ will give rise to 2 and 1 distinct output permutations respectively. Hence,

$$\begin{aligned} t(32514) &= 2 \times 1 + 10 \\ &= 12. \end{aligned}$$

Suppose that $\tau = t_1 t_2 \cdots t_n$ is any sequencing of a set C . We define a partially ordered set $P(\tau) = (C, <)$ by the following set of constraints:

1. if $i < j$ and $t_i > t_j$, then $t_i < t_j$,
2. if $k < i < j$ and $t_i > t_j$, then $t_k < t_j$,

The relation $<$ is obviously irreflexive² and transitive³.

Lemma 3 *Let τ be any sequencing of the elements of a set C . Then $S(\tau)$ is the set of linear extensions of the poset $P(\tau)$. Moreover, if τ is expressed in the form $\alpha m \beta$, where m is the maximal element of C and $|\alpha| = k$, then $s(\tau) = s(\alpha)s(\beta)(k+1)$.*

Proof: Let $\sigma \in S(\tau)$. We shall prove that σ is a linear extension of $P(\tau)$. Let $y < z$ be one of the constraints of $P(\tau)$. We must show that y precedes z in its occurrence in σ . There are two cases:

1. y precedes z in τ and $y > z$, or
2. There exists a symbol x such that y, x, z occur in this order in τ and $x > z$

In either case the supposition that z precedes y in σ leads to a contradiction. For, in order for the two symbols y, z to be transposed when processed by a priority queue, z must not be output until y (and all intervening symbols) have been placed in the priority queue. But, in the first case, the priority queue would then output z before y (because it is smaller) and, in the second case, the priority queue would output z before x .

To prove the converse, that every linear extension of $P(\tau)$ belongs to $S(\tau)$, we proceed by induction on $n = |C|$. We express τ as $\tau = \alpha m \beta$ where m is the maximal element of C and let σ be a linear extension of $P(\tau)$. Then, in σ , by the two conditions which

²A relation $\mathcal{R}$ is irreflexive if $x \mathcal{R} x$ is false for all x . For example, the relation ' $<$ ' over the integers is irreflexive: $x < x$ is false for all integers x .

³A relation $\mathcal{R}$ is transitive if $x \mathcal{R} y$ and $y \mathcal{R} z$ implies $x \mathcal{R} z$, for all x, y , and z . For example, ' $\leq$ ' is transitive: $x \leq y$ and $y \leq z$ implies $x \leq z$.

define $P(\tau)$, we can deduce that m precedes every symbol of β and that every symbol of α precedes every symbol of β . Moreover, the symbols of α will be arranged in σ as some linear extension α^* of $P(\alpha)$ and the symbols of β will be arranged in σ as some linear extension β^* of $P(\beta)$ and m will occur somewhere among the symbols of α^* . Thus σ has the form $\alpha_1^* m \alpha_2^* \beta^*$ where $\alpha_1^* \alpha_2^* = \alpha^*$. The inductive hypothesis guarantees that there are sequences of *Insert* and *DeleteMin* operations which transform an input stream α^* into α and transform an input stream β^* into β . It is now easy to see that there is a sequence of *Insert* and *DeleteMin* operations which, with σ as input, produces τ as output.

Finally, the description just given of the linear extensions of the poset $P(\alpha m \beta)$ proves that $s(\tau) = s(\alpha)s(\beta)(k+1)$.

The lemma above suggests a recursive algorithm for computing $s(\tau)$. However it is more illuminating to consider an iterative version of it. Put $\tau_0 = \infty$ and, for each $1 \leq i \leq n$, define

$$b(i) = \max\{j < i, \tau_j > \tau_i\}$$

Lemma 4 Let $\tau = t_1 t_2 \cdots t_\rho t_{\rho+1} \cdots t_n$ where $t_\rho > t_k, k = \rho+1, \rho+2, \dots, n$. Then every input that gives τ as output ends with a permutation of $t_{\rho+1}, t_{\rho+2}, \dots, t_n$.

Proof: We show that in any such input,

1. t_ρ precedes all of $t_{\rho+1}, t_{\rho+2}, \dots, t_n$.
2. Every $t_i, i < \rho$, precedes all of $t_{\rho+1}, t_{\rho+2}, \dots, t_n$.

Assume that t_ρ does not precede all of $t_{\rho+1}, t_{\rho+2}, \dots, t_n$ in the input. Hence in order to output t_ρ before any of the $t_{\rho+1}, t_{\rho+2}, \dots, t_n$ we need to input all the symbols $t_{\rho+1}, t_{\rho+2}, \dots, t_i$, which precede t_ρ in σ before inputting t_ρ into the queue. Now since all of $t_{\rho+1}, t_{\rho+2}, \dots, t_i$ are smaller than t_ρ , obviously t_ρ cannot be output before any of these symbols. Hence by contradiction, t_ρ precedes all of $t_{\rho+1}, t_{\rho+2}, \dots, t_n$.

In order to prove the second part, assume that $t_i, i < \rho$ appears in the $(\rho + j)$ th position in σ . If t_i is input along with t_ρ then, for $t_i < t_\rho$, t_i can be output before t_ρ . On the other hand, if t_ρ is input before t_i , then t_i will succeed t_ρ in τ - a contradiction. If $t_i > t_\rho$, then t_i will succeed t_ρ regardless how it is input into the queue - again a contradiction. Hence every $t_i, i < \rho$, precedes all of $t_{\rho+1}, t_{\rho+2}, \dots, t_n$.

Corollary 2 Every input giving $\tau^* = t_1 t_2 \dots t_{n-1}$ as output ends in a permutation of $t_{\rho+1}, t_{\rho+2}, \dots, t_n$.

Proof: All of $t_{\rho+1}, t_{\rho+2}, \dots, t_n$ are smaller than t_ρ so the result follows from lemma 4.

Lemma 5 The size of the set of all σ such that (σ, τ) is allowable is,

$$s(\tau) = \prod_{i=1}^n \{i - b(i)\}$$

Proof: [6] We prove this by induction on n . The case for $n = 1$ is clear. So assume that the result is true for $\tau^* = t_1 t_2 \dots t_{n-1}$ where $n > 1$. The values $b(1), b(2), \dots, b(n-1)$ for τ^* are the same as they are for τ . So, by the inductive hypothesis,

$$s(\tau^*) = \prod_{i=1}^{n-1} \{i - b(i)\}.$$

Let σ be any input giving τ . Then $\sigma - (t_n)$ is clearly an input that gives τ^* . So every input giving τ arises from inserting t_n into an input giving τ^* .

Now define $\rho = b(n)$. The hypotheses of lemma 4 are fulfilled so t_n can be inserted into σ^* in at most $(n - \rho)$ places. We next show that *each* of these $n - \rho$ places is possible. We know that σ^* ends with a permutation of $t_{\rho+1}, \dots, t_{n-1}$, all of which, by definition of $b(n)$, are less than t_n . Consider the sequence which arises by inserting t_n among one of these places. The algorithm which, from σ^* gives τ^* , can now be adapted. When it meets t_n it inserts it in the priority queue. But then t_n must remain until $t_{\rho+1}, \dots, t_{n-1}$ have all been output as they are smaller than t_n . Thus every input

that gives τ^* gives rise to $n - \rho$ inputs that all give τ . Hence,

$$\begin{aligned} s(\tau) &= (n - \rho) \prod_{i=1}^{n-1} \{i - b(i)\} \\ &= \prod_{i=1}^n \{i - b(i)\} \end{aligned}$$

as required.

Example 2 Compute $s(2143)$, the number of σ such that $(\sigma, 2143)$ is allowable.

τ_0	τ_1	τ_2	τ_3	τ_4
∞	2	1	4	3

Since $b(i)$ is defined as,

$$b(i) = \max_j \{j < i, \tau_j > \tau_i\}, \text{ for each } 1 \leq i \leq n$$

we compute $\{i - b(i)\}$ as follows:

i	$b(i)$	$i - b(i)$
1	0	1
2	1	1
3	0	3
4	3	1

Hence,

$$\begin{aligned} s(\tau) &= \prod_{i=1}^n \{i - b(i)\} \\ &= 3 \end{aligned}$$

Table 5 and table 6 list $|S(\tau)|$ and $|T(\sigma)|$ respectively for an input of length $n = 4$. Note that even though $|S(\tau)| \neq |T(\sigma)|$, $\sum_{\tau} |S(\tau)| = \sum_{\sigma} |T(\sigma)|$.

Lemma 6 There is an algorithm to compute $s(\tau)$ which, for random τ has expected execution time $O(n \log n)$.

Table 5: Number of Elements in $S(\tau)$ for $n = 4$

Output τ	Number of Inputs $s(\tau)$	Output τ	Number of Inputs $s(\tau)$
1234	24	3124	8
1243	6	3142	3
1324	8	3214	4
1342	6	3241	3
1423	4	3412	4
1432	2	3421	2
2134	12	4123	6
2143	3	4132	2
2314	8	4213	3
2341	6	4231	2
2413	4	4312	2
2431	2	4321	1

Proof: [6] Let us define a labeled binary tree $B(\tau)$. The root of $B(\tau)$ is labeled with (p, m) , where m is the maximal element of τ and p is its position in τ , and, if $\tau = \alpha m \beta$, the left and right subtrees are $B(\alpha)$ and $B(\beta)$ respectively. It is clear, again by induction, that if some node is labeled with the pair (i, τ_i) then $i - b(i)$ is the cardinal of the set of nodes in the subtree consisting of (i, τ_i) and all nodes in its left subtree. An example for computing $s(32154)$ using a labeled binary tree is illustrated in Figure 26. The *position* and *value* of each element in τ are written within the corresponding node and the corresponding $i - b(i)$ is written beside each node.

The tree $B(\tau)$ can be constructed as a binary search tree by inserting the *(position, value)* pairs into an initially empty tree. The pairs are inserted in decreasing order of value but are keyed by position.

The pairs (i, τ_i) are first sorted by second component and then inserted into an initially empty binary tree thereby creating $B(\tau)$. Then the sizes of all subtrees are

Table 6: Number of Elements in $T(\sigma)$ for $n = 4$

Input σ	Number of Outputs $s(\sigma)$	Input σ	Number of Outputs $s(\sigma)$
1234	1	3124	3
1243	2	3142	5
1324	2	3214	5
1342	3	3241	7
1423	3	3412	6
1432	5	3421	9
2134	2	4123	4
2143	4	4132	7
2314	3	4213	7
2341	4	4231	9
2413	5	4312	9
2431	6	4321	14

found and the product in the previous lemma is computed. The only part of this procedure which is not of time complexity $O(n \log n)$ is the creation of $B(\tau)$. However, it is well known [13] that, if all input orders are equally likely, the expected height of a binary search tree is $O(\log n)$. Hence the expected time for creating an n -node labeled binary tree $B(\tau)$ is $O(n \log n)$.

The following function *BuildBinaryTree* builds a binary tree $B(\tau)$, taking τ as its argument, and returns its root. We assume *tree* to be a pointer type containing at least two fields, namely *value* and *position* which hold the value of the element and its position in the output string τ respectively. It also has left and right pointers pointing to the left and right subtrees. (We may also need an additional field for holding the cardinal of the left subtree for each node; see below). We also assume that we have access to the functions, *FindMax*, *Extract α* and *Extract β* . *FindMax* returns the maximal element m in τ . Dividing τ into $\alpha m \beta$, *Extract α* and *Extract β* return the substrings α and β respectively. As one can see *BuildBinaryTree* calls itself recursively to process τ . It divides τ into two substrings α and β at the maximal element m . Assigning m to the current node, it repeatedly processes the substrings

α and β until τ becomes a single element.

```

function BuildBinaryTree( $\tau$  : string) : tree;
var
  t : tree;
   $\alpha, \beta$  : string;
  m : integer;
begin
  if  $|\tau| = 1$  then
    begin
      new(t);
      t↑.value := value;
      t↑.position := position;
      t↑.left := NIL;
      t↑.right := NIL;
    end else
      begin
        m := FindMax( $\tau$ );
         $\alpha$  := Extract $\alpha$ ( $\tau, m$ );
         $\beta$  := Extract $\beta$ ( $\tau, m$ );
        new(t);
        t↑.value := m;
        t↑.position := positionm;
        t↑.left := BuildBinaryTree( $\alpha$ ); { build left subtree }
        t↑.right := BuildBinaryTree( $\beta$ ); { build right subtree }
      end;
      BuildBinaryTree := t; { return root of the tree }
    end { BuildBinaryTree }

```

Assuming variable *tree* has an additional field named *CardinalLeft* which holds the cardinal of the set of nodes in the subtree consisting of the node itself and all nodes in its left subtree, the recursive function *ComputeS* computes the number of input permutations that give rise to a given output permutation τ . *ComputeS* takes the binary tree built by the previous function *BuildBinaryTree* as its argument.

Figure 26: Computing $s(32154)$ using a labeled binary tree

```

function ComputeS(t: tree) : integer;
begin
  if t = NIL then ComputeS := 1 else
    ComputeS := t↑.CardinalLeft * ComputeS(t↑.left) * ComputeS(t↑.right);
end { ComputeS }

```

Computing *CardinalLeft* for each node can be accomplished by another recursive function, *ComputeCardinalLeft*, as follows:

```

function ComputeCardinalLeft(N: node) : integer;
begin
  if N↑.left = NIL then ComputeCardinalLeft := 1 else
    ComputeCardinalLeft := 1 + ComputeCardinalLeft(N↑.left)
      + ComputeCardinalLeft(N↑.left↑.right);
end { ComputeCardinalLeft }

```

Table 7 lists the number of allowable pairs (σ, τ) for values of $n = 1 \dots 6$. These values were first discovered by a Turbo Pascal program (see Appendix A.2) which calculated them by direct enumeration. The values suggested the following theorem [6].

Theorem 1 *The number of allowable pairs (σ, τ) on a set of size n is $(n + 1)^{n-1}$.*

Proof: For any set X let $Sym(X)$ denote the set of all permutations of X . In proving the theorem we may suppose with no loss of generality that the n -element set in question is $C = \{1, 2, \dots, n\}$. The theorem is clearly true when $n = 0$ and so we now take $n > 0$ and, as an inductive hypothesis, assume that the theorem is true for sets of size less than n .

The number of allowable pairs is $\sum_{\tau \in Sym(C)} s(\tau)$. We shall express each permutation τ in the form $\tau = \alpha n \beta$ and let A, B denote the sets of symbols occurring in α and β respectively. Our sum can be expressed as a sum over the different possible subsets A (grouped according to size) where, for each possible A we sum over all $\alpha \in Sym(A)$ and all $\beta \in Sym(B)$. It then becomes,

$$\begin{aligned}
 s_n &= \sum_{k=0}^{n-1} \sum_{A, |A|=k} \sum_{\alpha \in Sym(A)} \sum_{\beta \in Sym(B)} s(\alpha n \beta) \\
 &= \sum_{k=0}^{n-1} \sum_{A, |A|=k} \sum_{\alpha \in Sym(A)} \sum_{\beta \in Sym(B)} s(\alpha) s(\beta) (k + 1) \\
 &= \sum_{k=0}^{n-1} \sum_{A, |A|=k} \sum_{\alpha \in Sym(A)} s(\alpha) \sum_{\beta \in Sym(B)} s(\beta) (k + 1) \\
 &= \sum_{k=0}^{n-1} (k + 1) \sum_{A, |A|=k} \sum_{\alpha \in Sym(A)} s(\alpha) \sum_{\beta \in Sym(B)} s(\beta)
 \end{aligned}$$

Table 7: Number of Allowable Pairs (σ, τ)

Length of Input	Number of (σ, τ)
1	1
2	3
3	16
4	125
5	1296
6	16807

Now using the inductive hypothesis, the theorem is true for sets of sizes less than n . Hence we know that $\sum_{\alpha \in \text{Sym}(A)} s(\alpha) = (k+1)^{k-1}$ and $\sum_{\beta \in \text{Sym}(B)} s(\beta) = (n-k)^{n-k-2}$.

Thus,

$$\begin{aligned} s_n &= \sum_{k=0}^{n-1} (k+1) \sum_{A, |A|=k} (k+1)^{k-1} (n-k)^{n-k-2} \\ &= \sum_{k=0}^{n-1} (k+1) \binom{n-1}{k} (k+1)^{k-1} (n-k)^{n-k-2} \end{aligned}$$

And using one of Abel's identities (see [15] §1.5), the above expression is equal to $(n+1)^{n-1}$.

The paper by Atkinson and Beals [2] gives another proof of the above theorem, in which they establish a 1-1 correspondence between allowable pairs and labeled trees. Then by using Cayley's theorem which states that the number of unrooted trees on $n+1$ nodes is $(n+1)^{n-1}$, they complete the proof.

2.3 Number of Output Permutations for each *Insert-DeleteMin* Sequence

In section 2.1 we considered the output permutations when each of the $n!$ input permutations was processed through all of the c_n *Insert-DeleteMin* bracket sequences using *output generation - method (i)*. In this section we will consider the output permutations for fixed *Insert-DeleteMin* bracket sequences using *output generation - method (ii)*. In other words we will compute the output sequences for each of the c_n *Insert-DeleteMin* bracket sequences, keeping each of them fixed and processing all $n!$ input permutations through the priority queue.

Table 8 shows the output permutations for each of the $\mathcal{B}$ -sequences for $n = 3$. Note that in order to find all the output permutations, all $n!$ possible input permutations

were considered for each of the $\mathcal{B}$. Here $k(\mathcal{B})$ denotes the number of output permutations for a given bracket sequence $\mathcal{B}$ and s_n denotes the sum of all $k(\mathcal{B})$ taken over all $\mathcal{B}$. As before i and d in $\mathcal{B}$ denote *Insert* and *DeleteMin* respectively.

It can be observed from table 8 that the maximum value of $k(\mathcal{B})$ occurs when $\mathcal{B}$ contains a strictly alternating pattern of i 's and d 's and the minimum number of $k(\mathcal{B})$ occurs when all the d 's follow all the i 's in $\mathcal{B}$. The reason for this follows easily from the basic *DeleteMin* property of the priority queue. But the results for the other patterns of $\mathcal{B}$ are not so easy to interpret. The following lemma helps us to analyze the relationship between $\mathcal{B}$ and the number of output permutations, $k(\mathcal{B})$.

Lemma 7 *In an ordinary priority queue,*

1. *If $\mathcal{B}_1, \mathcal{B}_2$ are well-formed Insert-Delete bracket sequences of length $2a, 2b$ then*

$$k(\mathcal{B}_1\mathcal{B}_2) = k(\mathcal{B}_1)k(\mathcal{B}_2)\binom{a+b}{a}$$
2. *If $\mathcal{B}$ is a well-formed Insert-Delete bracket sequence then $k(\mathcal{B}) = k(i\mathcal{B}d)$*

Proof: [2] The priority queue in the computation $\mathcal{B}_1\mathcal{B}_2$ is empty just before the $(a + 1)$ th symbol of the input is read. So the first a symbols of the output are a permutation of the first a symbols of the input. The first a symbols may be chosen in $\binom{a+b}{a}$ ways and, once chosen, may be permuted by $\mathcal{B}_1$ in $k(\mathcal{B}_1)$ ways, after which the remaining b symbols may be permuted in $k(\mathcal{B}_2)$ ways. The first part now follows.

Table 8: Values of $k(\mathcal{B})$ for $n = 3$

$\mathcal{B}$	Output Permutations	$k(\mathcal{B})$
ididid	123, 132, 213, 231, 312, 321	6
idiidd	123, 213, 312	3
iiddid	123, 132, 231	3
iididd	123, 213	2
iiiddd	123	1
$s_n =$		15

Table 9: Number of outputs and $\mathcal{B}$ -sequences

Length of Input	$\sum k(\mathcal{B})$
1	1
2	3
3	15
4	105
5	945
6	10395

For the second part note that if τ is an output of the priority queue computation $\mathcal{B}_1$ (arising, say, from the input σ) then τn is an output from the computation $i\mathcal{B}_1 d$ (clearly, it is the result of processing the input $n\sigma$). On the other hand any output from $i\mathcal{B}_1 d$ must have the form τn for some τ . This is because, for the computation $i\mathcal{B}_1 d$, the priority queue cannot become empty at any intermediate step, and so n cannot be deleted from it as smallest element until the end of the computation. To complete the proof it is only necessary to prove that τ is an output of the priority queue computation $\mathcal{B}_1$. Let ρ be an input to the computation $i\mathcal{B}_1 d$ which gives the output τn . If n is not the first symbol of ρ then we can write $\rho = \alpha m \beta n \gamma$ where m is the maximum symbol that precedes n in ρ . When this input is processed by $i\mathcal{B}_1 d$ the symbol m will remain in the priority queue after it is inserted at least until n is inserted (otherwise the priority queue would become empty before the computation terminates). It therefore follows that the input $\alpha n \beta m \gamma$ will result in the same output as ρ . By repetition of this principle we obtain an input of the form $n\sigma$ which generates the output τn . It is apparent that the priority queue computation $\mathcal{B}_1$ transforms σ into τ as required.

Table 9 lists the summation of $k(\mathcal{B})$ taken over all the $\mathcal{B}$ for input size of n , for $n = 1 \dots 6$.

2.3.1 Average Number of Outputs for a Random Priority Queue Computation

[2] Given that there are c_n *Insert-DeleteMin* bracket sequences for an input of length n , we could compute the number of output sequences for a random priority computation $\mathcal{B}$ as an average taken over all c_n bracket sequences. We define the average number of outputs for a random priority queue computation to be

$$k_n = \sum_{\mathcal{B}} \frac{k(\mathcal{B})}{c_n}$$

where $\mathcal{B}$ is a *Insert-DeleteMin* bracket sequence of length $2n$, and $k(\mathcal{B})$ is the number of output sequences generated by $\mathcal{B}$.

Theorem 2 *The average number of output permutations for a random priority queue computation is*

$$k_n = \frac{(n+1)!}{2^n}$$

where n is the length of input permutations.

Proof: [2] Let $l_n = \sum_{\mathcal{B}_1} k(\mathcal{B}_1)$ so that $k_n = l_n/c_n$. Every priority queue computation can be expressed in the form $\mathcal{B}_1 = i\mathcal{B}_2d\mathcal{B}_3$ where $\mathcal{B}_2, \mathcal{B}_3$ themselves are priority queue computations and are uniquely determined by $\mathcal{B}_1$. Since the length of $\mathcal{B}_2$ can be any even integer from 0 to $2n-2$ we have,

$$l_n = \sum_{\mathcal{B}_1} k(\mathcal{B}_1) = \sum_{i=0}^{i=n-1} \sum_{\mathcal{B}_2} \sum_{\mathcal{B}_3} k(i\mathcal{B}_2d\mathcal{B}_3)$$

where the second and third summations are over all priority queue computations $\mathcal{B}_2, \mathcal{B}_3$ which process inputs of length $i, n-i-1$ respectively. Thus, by the lemma 7,

$$l_n = \sum_{i=0}^{i=n-1} \sum_{\mathcal{B}_2} \sum_{\mathcal{B}_3} k(\mathcal{B}_2)k(\mathcal{B}_3) \binom{n}{i+1} = \sum_{i=0}^{i=n-1} l_i l_{n-i-1} \binom{n}{i+1}$$

Put $h_n = l_n/n!$. Then $h_n = \sum_{i=0}^{n-1} h_i h_{n-i-1}/(i+1)$. This recurrence can be solved by introducing the generating function,

$$H(x) = \sum_{r=0}^{\infty} h_r x^r.$$

Integrating the above generating function on both sides and multiplying by $H(x)$ we get,

$$\begin{aligned} H(x) \int H(x) &= H(x) \sum_{r=0}^{\infty} h_r \frac{x^{r+1}}{r+1} \\ &= \left(\sum_{r=0}^{\infty} h_r x^r \right) \left(\sum_{s=0}^{\infty} h_s \frac{x^{s+1}}{s+1} \right) \\ &= \sum_{r=0}^{\infty} \sum_{0 \leq r, s \leq r} h_r h_s \frac{x^{r+s+1}}{s+1} \\ &= \sum_{r=0}^{\infty} \sum_{s=0}^r h_{r-s} h_s \frac{x^{r+1}}{s+1} \\ &= \sum_{r=0}^{\infty} h_{r+1} x^{r+1} \\ &= \sum_{r=0}^{\infty} h_r x^r - 1 \quad (\text{since first element of } \sum_{r=0}^{\infty} h_r x^r = 1) \\ &= H(x) - 1 \end{aligned}$$

Hence the generating function $H(x) = \sum_{r=0}^{\infty} h_r x^r$ satisfies the integral equation,

$$H(x) \int H(x) dx = H(x) - 1$$

from which it follows that $H(x) = 1/\sqrt{1-2x}$ and the result follows by expanding the generating function.

Corollary 3 Total number of output sequences taken over all c_n permissible bracket sequences is, $\frac{(2n)!}{2^n n!}$.

Proof: Average number of output sequences k_n for a random priority queue computation $\mathcal{B}$, from theorem 2 is $k_n = \frac{(n+1)!}{2^n}$. Hence the total number of output sequences for all c_n priority queue computations is,

$$\begin{aligned} s_n &= k_n c_n \\ &= \frac{(n+1)!}{2^n} \frac{\binom{2n}{n}}{n+1} \\ &= \frac{(2n)!}{2^n n!} \end{aligned}$$

The number of allowable pairs when using the output generation - method (i), as can be recalled from section 2.1, is $s_n = (n+1)^{n-1}$. From corollary 3, the number of allowable pairs when using the output generation - method (ii) is $s_n = \frac{(2n)!}{2^n n!}$.

2.4 Binary Priority Queues

Hitherto this thesis has discussed only inputs whose elements are distinct. In this section we allow a different sort of input, namely binary sequences. The priority queues whose inputs are binary sequences have been discussed in detail by Atkinson [1]. In addition to several other interesting results, it was shown in [1] that the number of allowable pairs of binary sequences of length n is the $(n + 1)$ th Catalan number. The output permutations generated by a binary priority queue for inputs of size $n = 3$ are listed in the following table. The number of distinct input sequences of size n in the case of binary priority queues is obviously 2^n as opposed to $n!$ in the general case.

Table 10: Allowable Pairs (σ, τ) for $n = 3$

Input Sequence σ	Output Sequences τ
000	000
001	001
010	001 010
011	011
100	001 010 100
101	011 101
110	011 101 110
111	111

In this section we prove a result similar to the result of section 2.3.1 for the binary case. We first state and prove the following lemma.

Lemma 8 *In a binary priority queue,*

1. *If B_1, B_2 are well-formed Insert-Delete sequences then $k(B_1B_2) = k(B_1)k(B_2)$.*
2. *If B is a well-formed Insert-Delete sequence then $k(iBd) = 1 + k(B)$.*

Proof: 1. This is a specialized case of lemma 7 discussed in section 2.3. Assume that the lengths of B_1 and B_2 are $2a$ and $2b$ respectively. The priority queue in the computation B_1B_2 is empty just before the $(a + 1)$ th symbol of the input is read. So

Table 11: Allowable Pairs (σ, τ) in Binary Priority Queues

Input Length	Number of (σ, τ)
1	2
2	5
3	14
4	42
5	132
6	429

the first a symbols of the output are a permutation of the first a symbols of 1 and 0 of the input. These symbols can be permuted by $\mathcal{B}_1$ in $k(\mathcal{B}_1)$ ways, after which the remaining b symbols can be permuted in $k(\mathcal{B}_2)$ ways. Hence $k(\mathcal{B}_1\mathcal{B}_2) = k(\mathcal{B}_1)k(\mathcal{B}_2)$.

2. For the input $00 \cdots 00$, the output given by $i\mathcal{B}d$ is of course also $00 \cdots 00$. So we now consider inputs which have at least one 1. Such inputs give rise to outputs of the form $\tau 1$ for some τ . This is because, for the computation $i\mathcal{B}d$, the priority queue never becomes empty and once a symbol 1 has been inserted, the priority queue will always contain a 1 until the last element is deleted. Let ρ be an input to the computation $i\mathcal{B}d$ which gives the output $\tau 1$. If 1 is not the first symbol of ρ we can write $\rho = 0^i 1 \theta$. Consider the input $10^i \theta$. It is easy to see that this input also gives the output $\tau 1$. This shows that τ is an output given by $\mathcal{B}$ (with $0^i \theta$ as input). Hence $k(\mathcal{B})$ outputs arise in this way and the total number of outputs is equal to $1 + k(\mathcal{B})$.

2.4.1 Average Number of Outputs for a Random Binary Priority Queue Computation

In this section we investigate the quantity $s_n = \sum_{\mathcal{B}} k(\mathcal{B})$ whose values for $n \leq 6$ have been calculated directly as shown in table 12. The quantity s_n is interesting since s_n/c_n is the average number of output sequences for a fixed *Insert-Delete* sequence.

$$c_n = \frac{\frac{(\frac{1}{2})_n}{(n+1)!} \left\{ F\left(-n-1, -2n-1; -n-\frac{1}{2}; -\frac{1}{4}\right) - \frac{1}{2} \right\}}{4^n}$$
$$F(a, b; c; z) = \sum_{n=0}^{\infty} \frac{(a)_n (b)_n}{n! (c)_n} z^n, \quad (a)_n = \begin{cases} 1, & \text{if } n = 0 \\ a(a+1) \dots (a+n-1), & \text{if } n = 1, 2, \dots \end{cases}$$

Considering all the permissible fixed *Insert-Delete* bracket sequences in an input of length n , we obtain a recurrence equation on the total number of permutations, s_n , as follows.

$$\begin{aligned}
s_n &= \sum_{B} k(B) \\
&= \sum_{i=0}^{n-1} \sum_{B_1} \sum_{B_2} k(iB_1 dB_2) \\
&= \sum_{i=0}^{n-1} \sum_{B_1} \sum_{B_2} (k(B_1) + 1) k(B_2), \\
&\quad \text{since } k(iBd) = k(B) + 1 \quad \text{and} \quad k(B_1 B_2) = k(B_1) k(B_2) \\
&\quad \hspace{15em} (\text{from previous lemma}) \\
&= \sum_{i=0}^{n-1} \left(\sum_{B_1} k(B_1) + \sum_{B_1} 1 \right) \sum_{B_2} k(B_2) \\
&= \sum_{i=0}^{n-1} (s_i + c_i) s_{n-i-1} \quad \text{where } c_i = \frac{\binom{2i}{i}}{(i+1)!}, \text{ is the } i\text{th Catalan number.} \\
s_n &= \sum_{i=0}^{n-1} (s_i + c_i) s_{n-i-1}, \quad s_0 = 1
\end{aligned}$$

$$\begin{aligned}
\text{So, } \sum_{n=1}^{\infty} s_n t^{n-1} &= \sum_{n=1}^{\infty} t^{n-1} \sum_{i=0}^{n-1} (s_i + c_i) s_{n-1-i} \\
&= \sum_{n=0}^{\infty} t^n \sum_{i=0}^n (s_i + c_i) s_{n-i} = (s(t) + c(t)) s(t) \quad (1)
\end{aligned}$$

where $s(t) = \sum_{n=0}^{\infty} s_n t^n$

$$\text{So, } t^{-1}(s(t)-1) = s^2(t) + s(t)c(t) \quad \text{or} \quad s^2(t) - (t^{-1} - c(t))s(t) + t^{-1} = 0$$

$$\text{Hence, } 2s(t) = t^{-1} - c(t) \pm \sqrt{(t^{-1} - c(t))^2 - 4t^{-1}} \quad \text{where } c(t) = \frac{1 - \sqrt{1-4t}}{2t}$$

$$\text{So, } t^{-1} - c(t) = \frac{1}{2t} (1 + \sqrt{1-4t})$$

$$\begin{aligned}
\text{Hence, } s(t) &= \frac{1 + \sqrt{1-4t} - \sqrt{(1 + \sqrt{1-4t})^2 - 16t}}{4t} \\
&= \frac{1 + \sqrt{1-4t}}{4t} \left\{ 1 - \left(1 - \frac{16t}{(1 + \sqrt{1-4t})^2} \right)^{\frac{1}{2}} \right\} \quad (2)
\end{aligned}$$

$$\text{However we know that, } (1-z)^{\frac{1}{2}} = \sum_{m=0}^{\infty} \frac{\left(-\frac{1}{2}\right)_m}{m!} z^m$$

$$\begin{aligned}
\text{So, } s(t) &= \frac{1 + \sqrt{1-4t}}{4t} \left\{ 1 - \sum_{m=0}^{\infty} \frac{\left(-\frac{1}{2}\right)_m}{m!} \frac{(16t)^m}{(1 + \sqrt{1-4t})^{2m}} \right\} \quad (3) \\
&= \frac{1}{4t} \left\{ 1 + \sqrt{1-4t} - \sum_{m=0}^{\infty} \frac{\left(-\frac{1}{2}\right)_m}{m!} (16t)^m (1 + \sqrt{1-4t})^{1-2m} \right\}
\end{aligned}$$

By formula (6) in [8] §2.8,

$$(1 + \sqrt{1-z})^{1-2a} = 2^{1-2a} F\left(a - \frac{1}{2}, a; 2a; z\right) \quad (4)$$

So we have,

$$(1 + \sqrt{1-4t})^{1-2m} = 2^{1-2m} F\left(m - \frac{1}{2}, m; 2m; 4t\right), \text{ if } |4t| < 1, \quad (5)$$

where,

$$F(a, b; c; z) = \sum_{n=0}^{\infty} \frac{(a)_n (b)_n}{n! (c)_n} z^n, \quad (a)_n = \begin{cases} 1, & \text{if } n = 0 \\ a(a+1) \dots (a+n-1), & \text{if } n = 1, 2, \dots \end{cases}$$

is the Gaussian hypergeometric function.

(Note: For $m = 0$ in (5), the r.h.s must be evaluated as $\lim_{m \rightarrow 0} [\dots]$.)

Indeed,

$$1 + \sqrt{1-4t} = 2 + \sum_{n=1}^{\infty} \frac{\left(-\frac{1}{2}\right)_n}{n!} (4t)^n \quad (6)$$

Now,

$$\begin{aligned} R(t) &= \sum_{m=0}^{\infty} \frac{\left(-\frac{1}{2}\right)_m}{m!} (16t)^m (1 + \sqrt{1-4t})^{1-2m} \\ &= 2 \sum_{m=0}^{\infty} \frac{\left(-\frac{1}{2}\right)_m}{m!} (4t)^m \sum_{n=0}^{\infty} \frac{\left(m - \frac{1}{2}\right)_n (m)_n}{n! (2m)_n} (4t)^n \\ &= 2 \sum_{m=0}^{\infty} \sum_{n=0}^{\infty} \frac{\left(-\frac{1}{2}\right)_{m+n} (m)_n}{m! n! (2m)_n} (4t)^{m+n} \\ &= 2 \sum_{m=0}^{\infty} \sum_{n=0}^{\infty} \frac{\left(-\frac{1}{2}\right)_{m+n} \Gamma(m+n) \Gamma(2m)}{m! n! \Gamma(m) \Gamma(2m+n)} (4t)^{m+n} \end{aligned} \quad (7)$$

Now set $m+n = k$, or $m = k-n$, to get the series on the r.h.s of (7) as

$$R(t) = 2 \sum_{k=0}^{\infty} \frac{\left(-\frac{1}{2}\right)_k \Gamma(k)}{k! \Gamma(2k)} (4t)^k \sum_{n=0}^k \frac{(-k)_n \Gamma(2k-2n) \Gamma(2k)}{n! \Gamma(k-n) \Gamma(2k-n)} (-1)^n \quad (8)$$

$$\text{since, } \frac{1}{(k-1)!} = \frac{1}{k!} (-k)_n (-1)^n.$$

By the duplication formula (see [8], page 5),

$$\Gamma(2z) = \frac{\Gamma(z) \Gamma\left(z + \frac{1}{2}\right)}{2\sqrt{\pi}} 2^{2z} \quad (9)$$

$$\text{Hence we find that, } \frac{\Gamma(2k-2n)}{\Gamma(k-n)} = \frac{\Gamma\left(k-n + \frac{1}{2}\right)}{2\sqrt{\pi}} 2^{2k-2n}$$

$$\frac{\Gamma(2k)}{\Gamma(k)} = \frac{\Gamma(k + \frac{1}{2})}{2\sqrt{\pi}} 2^{2k}$$

Thus,

$$\begin{aligned} R(t) &= 2 \sum_{k=0}^{\infty} \frac{(-\frac{1}{2})_k}{k! \Gamma(k + 1/2)} (4t)^k \sum_{n=0}^k \frac{(-k)_n \Gamma(k + \frac{1}{2} - n) \Gamma(2k)}{n! \Gamma(2k - n)} \left(-\frac{1}{4}\right)^n \quad (10) \\ &= 2 \sum_{k=0}^{\infty} \frac{(-\frac{1}{2})_k}{k!} (4t)^k \sum_{n=0}^k \frac{(-k)_n (1 - 2k)_n}{k! (1/2 - k)_n} \left(-\frac{1}{4}\right)^n \\ &= 2 \sum_{k=0}^{\infty} \frac{(-\frac{1}{2})_k}{k!} (4t)^k F\left(-k, 1 - 2k; \frac{1}{2} - k; -\frac{1}{4}\right). \end{aligned}$$

Thus from (6) and (10),

$$\begin{aligned} s(t) &= \frac{1}{4t} \left\{ 2 + \sum_{n=1}^{\infty} \frac{(-\frac{1}{2})_n}{n!} (4t)^n - 2 - 2 \sum_{n=1}^{\infty} \frac{(-\frac{1}{2})_n}{n!} (4t)^n F\left(-n, 1 - 2n; \frac{1}{2} - n; -\frac{1}{4}\right) \right\} \\ &= \frac{1}{4t} \sum_{n=1}^{\infty} \frac{(-\frac{1}{2})_n}{n!} (4t)^n \left\{ 1 - 2F\left(-n, 1 - 2n; \frac{1}{2} - n; -\frac{1}{4}\right) \right\} \\ &= -\frac{1}{2} \sum_{n=0}^{\infty} \frac{(\frac{1}{2})_n}{(n+1)!} (4t)^n \left\{ 1 - 2F\left(-n - 1, -2n - 1; -n - \frac{1}{2}; -\frac{1}{4}\right) \right\} \\ &= \sum_{n=0}^{\infty} \frac{(\frac{1}{2})_n}{(n+1)!} (4t)^n \left\{ F\left(-n - 1, -2n - 1; -n - \frac{1}{2}; -\frac{1}{4}\right) - \frac{1}{2} \right\} \end{aligned}$$

Hence,

$$\begin{aligned} S(x) &= \sum_{n=0}^{\infty} \frac{(\frac{1}{2})_n}{(n+1)!} \left\{ F\left(-n - 1, -2n - 1; -n - \frac{1}{2}; -\frac{1}{4}\right) - \frac{1}{2} \right\} (4x)^n \\ &= 2x + 7x^2 + 29x^3 + 131x^4 + 625x^5 + \dots \end{aligned}$$

And,

$$s_n = \frac{(\frac{1}{2})_n}{(n+1)!} \left\{ F\left(-n - 1, -2n - 1; -n - \frac{1}{2}; -\frac{1}{4}\right) - \frac{1}{2} \right\} 4^n$$

Hence the average number of output permutations (averaged over all permissible *Insert-Delete* sequences) is,

$$\frac{\frac{(\frac{1}{2})_n}{(n+1)!} \left\{ F\left(-n - 1, -2n - 1; -n - \frac{1}{2}; -\frac{1}{4}\right) - \frac{1}{2} \right\} 4^n}{c_n}$$

Example 3 Compute the total number of output sequences generated by a binary priority queue using fixed Insert-Delete sequences for inputs of size $n = 5$.

The total number of output sequences can be computed using the previous theorem as follows:

$$s_n = \frac{\left(\frac{1}{2}\right)_n}{(n+1)!} \left\{ F\left(-n-1, -2n-1; -n-\frac{1}{2}; -\frac{1}{4}\right) - \frac{1}{2} \right\} 4^n$$

$$s_3 = \frac{\left(\frac{1}{2}\right)_3}{24} \left\{ F\left(-4, -7; \frac{-7}{2}; \frac{-1}{4}\right) - \frac{1}{2} \right\} 64$$

$$\begin{aligned} \left(\frac{1}{2}\right)_3 &= \frac{1}{2} \cdot \frac{3}{2} \cdot \frac{5}{2} = \frac{15}{8} \\ F\left(-4, -7; \frac{-7}{2}; \frac{-1}{4}\right) &= \sum_{n=0}^{\infty} \frac{(-4)_n (-7)_n}{n! \left(\frac{-7}{2}\right)_n} \left(\frac{-1}{4}\right)^n \\ &= 1 + 2 + \frac{9}{5} + 1 + \frac{1}{2} \\ s_3 &= \frac{15}{8 \cdot 24} \cdot \frac{29}{5} \cdot 64 \\ &= 29 \end{aligned}$$

matching the numerical results given in table 12

The following table lists the total number of output permutations obtained using fixed Insert-Delete sequences in a binary priority queue.

Table 12: Number of τ for Fixed Insert-Delete Sequences

Input Length	Total Number of τ
1	2
2	7
3	29
4	131
5	625
6	3099

Chapter 3

Double Ended Priority Queues

A simple modification to the ordinary priority queue, where the *DeleteMin* operation is substituted with either a *DeleteMin* or *DeleteMax* operation, results in another useful data type called double ended priority queue. The input to a double ended priority queue can be any numerical sequence, but as before, for ease of manipulation only the set of distinct positive integers from 1 to n is considered. One or more numbers from the input string can be input to the queue at each time. Every time an output is generated, the algorithm removes only either the smallest number or the largest number currently held in the queue. Thus it is clear that the double ended priority queue is a generalized form of ordinary priority queue. It is also easy to observe that due to the availability of the additional operation, *DeleteMax*, a double ended priority queue produces a greater number of output permutations for a given input length than an ordinary priority queue.

It was discussed in Chapter 2 that there are c_n legal *Insert* and *Delete* (*DeleteMin*) operations for a given input string of length n . But, unlike the case of ordinary priority queues, here we have two different ways of carrying out each delete operation, namely *DeleteMin* and *DeleteMax*. A double ended priority queue computation is determined by an *Insert-Delete* sequence and a *DeleteMin-DeleteMax* sequence. An *Insert-Delete* sequence is similar to the *Insert-DeleteMin* sequence of the ordinary priority queue case where the number of *Deletes* counted from left to right do not

exceed the number of preceding *Inserts*. Each *Delete* operation in the *Insert-Delete* sequence can be a *DeleteMin* or a *DeleteMax* operation specified by the *DeleteMin-DeleteMax* sequence. Since for each of the c_n legal ways of arranging n *Insert* and n *Delete* operations there are 2^n ways of choosing the n *Delete* operations we have exactly $c_n 2^n$ different double ended priority queue computations on inputs of length n . We recall that with the ordinary priority queue when the input was the sequence $1, 2, \dots, n$, there was only one output, which is $1, 2, \dots, n$ itself. Also when the input was the sequence $n, n-1, \dots, 2, 1$, every legal sequence of *Inserts* and *DeleteMins* gave a different output. But in the case of a double ended priority queue, we obviously have a more complicated situation. Let us look at the simplest case where the input is $1, 2, \dots, n-1, n$. It is apparent that we cannot have output of the form $\dots, i+3, i+1, i, i+2, \dots$ or $\dots, i+3, i+1, i+2, i, \dots$. This is because, when we output $i+3$, we should already have $i, i+1$, and $i+2$ in the queue. Now having output $i+3$, we can either output i by applying *DeleteMin* or output $i+2$ by applying *DeleteMax*. Hence with an input string of the form $1, 2, \dots, n-1, n$ we cannot, at any time, output $i+1$ after $i+3$. This observation leads us to the conclusion that an input sequence of the form $1, 2, \dots, n-1, n$ will give rise to strictly less than $n!$ different output permutations.

3.1 Constraints on the Input

Let σ be some input sequence which when passed through the double ended priority queue, gives rise to some output sequence τ . We consider the problem of determining which input permutations σ can produce a given output permutation τ using a double ended priority queue using some given sequence of *DeleteMin*, *DeleteMax* operations. From now on we shall use the terms *d-operation* and *D-operation* interchangeably with *DeleteMin* and *DeleteMax* respectively. Let i, j , and k be three symbols in the output sequence τ . We will now look at four possible scenarios by which we could output i, j and k from the queue in some specified order.

We observe that if $i > j$ and i precedes j in some τ , and i was output from the queue using a d -operation, then i should precede j in σ . This is because, if j were to precede i in σ , then in order to output i first, j would still be in the queue when i was inserted. But since i is associated with a d -operation and since j is smaller than i , j would have to be output before i , a contradiction to our original assumption.

Let us look at the second scenario where $i > j$ and k , i , and j appear in the order $\dots k \dots i \dots j \dots$ in τ . Using the previous argument, in order for i to precede j in τ , if a d -operation is used to remove i from the queue then i precedes j in σ . Hence it would appear that i, j, k can come in any of the following three orders in σ .

1. $\dots k \dots i \dots j \dots$
2. $\dots i \dots k \dots j \dots$
3. $\dots i \dots j \dots k \dots$

However the third of these is impossible. For, in order to output k first, i, j must still be in the queue when k is inserted; then assuming k is smaller than i and j , k is output using a d -operation (or if k is larger than i and j , output using a D -operation). Having output k , we now have i and j remaining in the queue. Since the next delete operation is a d -operation, j , being smaller than i , will be output before i . This results in an output sequence of the form, $\dots k \dots j \dots i \dots$, a contradiction to our assumption. Thus it follows only the input sequences 1 and 2 can produce the desired output sequence, or in other words, k should precede j in σ in order for us to produce an output sequence of the form $\dots k \dots i \dots j \dots$, where $i > j$ and i is associated with a d -operation.

Exactly similar arguments can be used when $i < j$ and a D -operation is used to output i . Summarizing, we have found four conditions on how σ is constrained:

- If i precedes j in all τ , $i > j$, and if a d -operation is used to output i , then i precedes j in σ .
- If k precedes i precedes j in τ , $i > j$, and if a d -operation is used to output i , then k precedes j in σ .

- If i precedes j in all τ , $i < j$, and if a D -operation is used to output i , then i precedes j in σ .
- If k precedes i and i precedes j in all τ , if $i < j$, and if a D -operation is used to output i , then k precedes j in σ .

Using the above results and denoting the sequence of d - and D -operations by $\mathcal{D}$, we shall now define a partially ordered set $P(\tau, \mathcal{D})$, depending only on τ and $\mathcal{D}$, whose set of linear extensions is precisely the set of permutations from which τ can be generated using the sequence $\mathcal{D}$.

The definition of $P(\tau, \mathcal{D})$ is as follows:

1. If i precedes j in τ , $i > j$, and i is a d -symbol then $i \prec j$
2. If k precedes i precedes j in τ , $i > j$, and i is a d -symbol then $k \prec j$
3. If i precedes j in τ , $i < j$, and i is a D -symbol then $i \prec j$
4. If k precedes i precedes j in τ , $i < j$, and i is a D -symbol then $k \prec j$.

Lemma 9 *If σ is a permutation which, if input to a double ended priority queue, can produce the output τ with the sequence $\mathcal{D}$ of DeleteMin, DeleteMax operations then σ is a linear extension of $P(\tau, \mathcal{D})$.*

Proof: Proof follows directly from the discussion above and the definition of $P(\tau, \mathcal{D})$.

Lemma 10 *If σ is a linear extensions of $P(\tau, \mathcal{D})$ then there is an algorithm which transforms σ into τ by means of the sequence $\mathcal{D}$ of DeleteMin, DeleteMax operations.*

Proof: We use induction on n . Let $\tau = \alpha v$ where $v = \tau_n$ is the last symbol of τ . The constraints of $P(\tau, \mathcal{D})$ consist of the constraints of $P(\alpha, \mathcal{D})$ together with constraints $\tau_i \prec v$ for all $i \leq t$ where

$$t = \max_{j \leq n} \{j | \tau_j \text{ is a } d\text{-symbol and } \tau_j > \tau_n \text{ or } \tau_j \text{ is a } D\text{-symbol and } \tau_j < \tau_n \}$$

The linear extension σ of $P(\tau, \mathcal{D})$ has the form $\mu = \lambda v \rho$ ($\lambda \rho$ with v inserted where $\lambda \rho$ is a linear extension of $P(\alpha, \mathcal{D})$). By induction on n there is an algorithm $\mathcal{A}$ which transforms $\lambda \rho$ into α . For all $r \in \rho$, $r \prec v$ is *not* a constraint. Hence for all $r \in \rho$ we have one of

1. $r < v$ and r is a d -symbol, or
2. $r > v$ and r is a D -symbol

If we apply $\mathcal{A}$ to σ until $v = \tau_n$ is encountered we can place v in the priority queue and resume $\mathcal{A}$. Because of the two properties above v will not be output before any $r \in \rho$ and when $\mathcal{A}$ terminates the priority queue will contain only v , which may be output by either of a d or D -operation.

3.2 Generating Output Sequences Using Double Ended Priority Queues

The functional difference between ordinary priority queues and double ended priority queues is the way the delete operation is carried out. In ordinary priority queue always the smallest element currently held in the queue is the one to be deleted next. In double ended priority queue, either the smallest or the largest of the elements currently held in the queue is deleted next. This functional difference of double ended priority queue from ordinary priority queue produces a set of *Insert-Delete* sequences which is 2^n times larger than that of an ordinary priority queue of the same length. In chapter 2 we looked at two different methods of generating output sequences with ordinary priority queues. Now that there are two types of *Delete* operations we can regard the output of a double ended priority queue as being dependent on,

- (i) the input permutation,
- (ii) the sequence of *Inserts* and *Deletes*,
- (iii) the sequence of *DeleteMins* and *DeleteMaxs*.

In chapter 2 we looked at the number of outputs when either (i) or (ii) was held constant. In this chapter we consider the corresponding problems when zero, one or two of (i), (ii), (iii) are held constant. Specifically we consider the following six cases,

- (1) Determining the total number of allowable pairs (σ, τ) . Recalling our definition in Chapter 2, the set of allowable pair consists of all pairs (σ, τ) such that σ, τ are sequences of length n and a double ended priority queue computation exists which transforms the input sequence σ into the output sequence τ .
- (2) Given any fixed *Insert-Delete* sequence $\mathcal{B}$ determining the number of allowable pairs (σ, τ) when an input σ is transformed into an output τ by using the *Insert-Delete* sequence $\mathcal{B}$, and any *DeleteMin-DeleteMax* sequences.
- (3) Given any fixed *DeleteMin-DeleteMax* sequence $\mathcal{D}$ determining the number of allowable pairs (σ, τ) when an input σ is transformed into an output τ by using the *DeleteMin-DeleteMax* sequence $\mathcal{D}$, and any *Insert-Delete* sequences.
- (4) For a fixed input σ and fixed *Insert-Delete* sequence $\mathcal{B}$ determining the number of outputs τ obtained as the *DeleteMin-DeleteMax* sequence varies.
- (5) For a fixed input σ and fixed *DeleteMin-DeleteMax* sequence $\mathcal{D}$ determining the number of outputs τ obtained as the *Insert-Delete* sequence varies.
- (6) For a fixed *Insert-Delete* sequence $\mathcal{B}$ and fixed *DeleteMin-DeleteMax* sequence $\mathcal{D}$ determining the number of outputs τ obtained as the input sequence σ varies.

We shall look at each of these cases in detail in the following sections.

3.2.1 The Total Number of Allowable Pairs

We proved in Chapter 2 that the number of allowable pairs for the ordinary priority queue is equal to $(n + 1)^{n-1}$. An expression for the same quantity has not yet been found for the case of double ended priority queue. Even though we lack such an expression, determining if a (σ, τ) pair is an allowable one is rather easy. The following algorithm, based upon the one given in [2] for ordinary priority queues, precisely serves this purpose.

```

function allowable-pair( $\sigma, \tau$ : string): boolean;
begin
   $Q :=$  empty priority queue;
   $j := 1$ ;
  while  $j \leq n$  do
    while  $\tau_j \notin Q$  do
      endwhile
    if ( $\min(Q) \neq \tau_j$ ) or ( $\max(Q) \neq \tau_j$ ) then
      return(false) else
         $j := j + 1$ ;
      endif;
    endwhile;
  return(true);
end; { allowable-pair }

```

At the beginning the queue is empty, so the inner while loop is entered. The inner while loop is now executed until we input the element equivalent to τ_j . Now we exit from the while loop and check to see if τ_j is either the smallest or the largest element currently in the queue. If not we immediately know that we couldn't have output τ_j with any valid B and D sequences and return the pair (σ, τ) as not allowable, otherwise we continue until we exhaust all the symbols.

Analysis of the above algorithm shows that the time complexity of finding if a given pair (σ, τ) is an allowable one is $O(n)$.

Figure 27: Generation of output sequences for $\sigma = 2\ 3\ 1$

In the absence of an expression for the number of allowable pairs some numerical results were obtained by explicit enumeration. Our method to enumerate the allowable pairs (σ, τ) considered each possibility for σ in turn and computed the number of possible τ . Before giving the details of the method (*DoubleEndedPriorityQueue0*) we consider the example of $\sigma = 2\ 3\ 1$. There are $2^3 c_3 = 8 \times 5 = 40$ *Insert-Delete* sequences that can be used to generate the output permutations. Each of the 8 *DeleteMin-DeleteMax* sequences (*ddd, ddD, dDd, Ddd, dDD, DdD, DDd, DDD*) is applied onto each of the 5 *Insert-Delete* bracket sequences (*iiixxx, iixxix, iixixx, ixixxx, ixixix*

where every x is either a d or D). Figure 27 shows a portion of the output generations with the fixed input permutation $\sigma = 2\ 3\ 1$, where we can see each of the 8 *DeleteMin-DeleteMax* sequences is applied onto the *Insert-Delete* bracket sequence *nrirrr*.

```

procedure DoubleEndedPriorityQueue0;
begin
  NoOf- $\sigma := n!$ ;
  NoOf- $\mathcal{B} := \frac{\binom{2n}{n}}{n+1}$ ;
  NoOf- $\mathcal{D} := 2^n$ ;
  EmptyPriorityQueue( $Q$ );
  for  $i := 1$  to NoOf- $\sigma$  do
    input :=  $\sigma[i]$ ;
     $m := 1$ ;
    for  $j := 1$  to NoOf- $\mathcal{B}$  do
      for  $k := 1$  to NoOf- $\mathcal{D}$  do
        for  $l := 1$  to  $n$  do { insert each symbol in  $\sigma[i]$  }
          insert( $\sigma[i]$ );
           $\eta := deleteMinMax_m(Q)$ ;
        endfor;
      endfor;
      inc( $m$ );
    endfor;
    output( $\tau$ ); { if not already generated }
  endfor;
end { DoubleEndedPriorityQueue0 }

```

Using this method, we process each of the $n!$ input permutations through each of the $2^n c_n$ valid *Insert-Delete* sequences using the algorithm *DoubleEndedPriorityQueue0* listed below. For the implementation of *DoubleEndedPriorityQueue0* we assume that the input sequences are pre-stored in the array σ . Let *Dd*-sequence be the sequence of *DeleteMin-DeleteMax* operations. We assume that a routine called *deleteMinMax*, responsible for determining the next *Dd*-sequence is available to our algorithm. The routine *deleteMinMax* computes all legal $2^n c_n$ *Insert-Delete* sequences

Table 13: Number of Allowable Pairs (σ, τ)

Length of Input	number of τ
1	1
2	4
3	32
4	392
5	6488
6	135360

(2^n *Dd*-sequences for each of c_n *Insert-Delete* bracket sequences) and applies each of them one at a time on a consecutive set of n input symbols. The *deleteMinMax* routine may perform one of the following actions after every input symbol is inserted into the queue: (a) remove the smallest element from the queue by applying a *d*-operation, (b) remove the largest element from the queue by applying a *D*-operation, (c) perform no action and choose to wait until one or more input symbols are inserted into the queue. Note that *deleteMinMax_m* denotes the m th deletion algorithm and is not invoked on an empty queue.

Time Complexity: Assuming that the m th deletion algorithm is computed in linear time, the running time of the algorithm is obviously, $t = O\left\{n!2^n \frac{\binom{2n}{n}}{n+1}n\right\} = O\left\{2^n \frac{(2n)!}{n!}\right\}$.

Processing all $n!$ input permutations in this way and removing the duplicate output permutations for each input permutation will give us the total number of allowable pairs (σ, τ) that can be obtained with input permutations of length n .

The method above draws attention to the problem of computing the number of outputs that can be generated by a given input (as the *Insert-Delete* sequence $\mathcal{B}$ and *DeleteMin DeleteMax* sequence $\mathcal{D}$ vary). The following lemmas 11 and 12 are some preliminary work on the problem.

Lemma 11 *For an input sequence $\alpha m \beta$, where m is an arbitrary symbol, m cannot become the first symbol for any legal output, if m is neither the smallest nor the largest of the symbols in αm .*

Proof: The double ended priority queue outputs either the smallest or the largest symbol currently held in the queue. When m is inserted into the queue there would be at least two elements already in the queue or any of them would have been already processed and output from the queue. If either one of them had been already output, then obviously m is not the first symbol of τ , otherwise m becomes the third or subsequent symbol (depending upon the size of α) in the queue, where it is neither the smallest nor the largest and thus cannot be obviously output first.

The following definitions are used in lemma 12 for computing the total number of output permutations.

Trailing symbol: A symbol m in a given input sequence $\sigma = \alpha m \beta$, is called a trailing symbol if it cannot become the first symbol for any legal output sequence τ of σ .

Partial sum $s(\sigma, m)$: Consider an input sequence σ of length n . Partial sum of σ for m , $s(\sigma, m)$, is defined as the number of output permutations generated by inserting all the symbols preceding m in σ simultaneously in the queue, followed by inserting the symbols succeeding m in σ individually or as sub-strings.

Lemma 12 *If $\sigma = m_1 m_2 \cdots m_{n-1} m_n$, then the number of output sequences τ generated by σ in a double ended priority queue is,*

$$s = \sum_{\substack{i=1 \\ m_i \notin T_\sigma}}^{i=n} s(\sigma, m_i)$$

where T_σ is the set containing all the trailing symbols in σ , and $s(\sigma, m_i)$ is the partial sum of σ for the symbol m_i .

Proof: Since trailing symbols cannot become the first symbols in the output permutations, only m_i , $m_i \notin T_o$ can become the first symbols in τ . Assume that σ is of the form $\sigma = \alpha m \beta$ where $|\alpha| \geq 0$, $|\beta| \geq 0$ and $m \notin T_o$. Let us consider the output permutations τ which have m as their first symbol. Just after m is output from the queue, we will have $\alpha\gamma$ left in the queue where γ can be either an empty string or a string consisting of some consecutive symbols from β . Now the total number of output sequences with m as their first symbol are formed by processing all the symbols in $\alpha\gamma$ in the queue, along with the remaining symbols in β . From the definition of partial sum $s(\sigma, m)$, it is apparent that the above input symbols $\alpha\gamma$ now in the queue, along with the remaining symbols of β waiting to be input to the queue generate the partial sum of σ for the symbol m .

Since all the symbols in σ , except the trailing symbols, can become the first symbol of output permutations, in order to obtain the total number of output sequences for the given input σ , we compute the summation of all the partial sums of σ for the above symbols.

Example 4 Given that $s(1234) = 22$ and $s(3124) = 16$, compute the total number of output sequences generated in a double ended priority queue for the input sequence $\sigma = 41235$, i.e: $s(41235)$.

The trailing symbols for $\sigma = 41235$ are 2 and 3. Hence the total number of output sequences is,

$$s(\sigma) = s(\sigma, 1) + s(\sigma, 4) + s(\sigma, 5)$$

Now $s(\sigma, 1)$ is the number of output sequences with 1 as the first symbol. Obviously this is equal to the output permutations of the remaining input string, $4235 = s(3124) = 16$. Similarly $s(\sigma, 4) = s(1235) = s(1234) = 22$. But $s(\sigma, 5) \neq s(4123)$, as $(\sigma, 5)$ is generated by inserting all the symbols in σ into the queue at once and then outputting 5 followed by the other symbols. Obviously this value is equal to $2^{4-1} = 8$. This gives us,

$$\begin{aligned} s(41235) &= 16 + 22 + 8 \\ &= 46 \end{aligned}$$

3.2.2 The Number of Allowable Pairs with Fixed Insert-Delete Sequence

There is a simple formula for the number of allowable pairs associated with a fixed *Insert-Delete* sequence which is stated by the following theorem.

Theorem 4 *The number of allowable pairs associated with a fixed Insert-Delete sequence $\mathcal{B}$ is $n!2^{n-t}$ where t is the number of well-formed segments in the Insert-Delete sequence.*

Proof: By theorem 7 in §3.2.4 the number of outputs generated from a particular input σ using $\mathcal{B}$ is 2^{n-t} . The result follows since there are $n!$ inputs.

The above theorem solves the problem in the title of this subsection but suggests the following two additional questions:

- (a) How many outputs $k(\mathcal{B})$ can a given *Insert-Delete* sequence $\mathcal{B}$ generate (as the input sequence and *DeleteMin-DeleteMax* sequence both vary)?
- (b) What is the average number of allowable pairs (averaged over all *Insert-Delete* sequence)?

The following theorem answers question (a).

Theorem 5 *In a double ended priority queue, when considering the Insert-Delete sequences,*

1. *If $\mathcal{B}_1, \mathcal{B}_2$ are well-formed Insert-Delete sequences of length $2a, 2b$ then*

$$k(\mathcal{B}_1\mathcal{B}_2) = k(\mathcal{B}_1)k(\mathcal{B}_2) \binom{a+b}{a}$$

2. *If $\mathcal{B}$ is a well-formed Insert-Delete sequence of length u then*

$$k(i\mathcal{B}\delta) = (a+1)2^t k(\mathcal{B})$$

where t is the number of well-formed Insert-Delete segments in $\mathcal{B}$, δ is either a *DeleteMin* or a *DeleteMax*.

Proof: 1. Since $\mathcal{B}_1$ is a well-formed *Insert-Delete* sequence, the priority queue in the computation $\mathcal{B}_1\mathcal{B}_2$ will be empty just before the $(a + 1)$ th symbol of the input is read into the queue. So it is clear that the first a symbols of the output permutation are a permutation of the first a symbols of the input. Since there is a total of $(a + b)$ input symbols, the first a symbols can be chosen in $\binom{a+b}{a}$ ways and, once chosen, can be permuted by $\mathcal{B}_1$ in $k(\mathcal{B}_1)$ ways. After the first a symbols are output, the remaining b symbols can be permuted by $\mathcal{B}_2$ in $k(\mathcal{B}_2)$ ways and the first part of the lemma thus follows.

2. Considering each of the well-formed segments $\mathcal{B}_1, \mathcal{B}_2, \dots, \mathcal{B}_t$ of $\mathcal{B}$ individually, we know that the input symbol corresponding to the last operation of each *Insert-Delete* segment is not affected by whether it is removed from the queue by a *DeleteMin* or a *DeleteMax* operation. In other words, both *DeleteMin* and *DeleteMax* applied on to the above input symbol of each segment will result in the same output permutation. Assume that the input permutation σ , when processed through the priority queue computation $\mathcal{B}$ results in the output permutation τ . For the priority queue computation $i\mathcal{B}\delta$, let us consider an input permutation of the form $x\sigma$, where x does not necessarily have to be the maximal number in the sequence. Due to the well-formed nature of the *Insert-Delete* segments, it is obvious that when $x\sigma$ is passed through the computation $i\mathcal{B}_1\mathcal{B}_2 \dots \mathcal{B}_t\delta$, at the end of exhausting the *Insert-Delete* operations in each segments $\mathcal{B}_1, \mathcal{B}_2, \dots, \mathcal{B}_t$, there will always be one symbol left in the queue. This means that there will be two symbols left in the queue just before the last delete operation of each *Insert-Delete* segment. Since we now have two symbols in the queue, we could either apply *DeleteMin* and output the smaller one or apply *DeleteMax* and output the larger one, resulting in two different permutations. In other words, the last delete operation in each well-formed segment in $\mathcal{B}$ will now give rise to two different permutations. Since there are t such well-formed segments, we obtain 2^t different ways of outputting the symbols corresponding to the last delete operation in each segment $\mathcal{B}_1, \mathcal{B}_2, \dots, \mathcal{B}_t$ using the computation $i\mathcal{B}\delta$, as opposed to

Figure 28: Generation of output sequences for $\mathcal{B} = iixirx$

just one way of outputting the same in the computation $\mathcal{B}$. Since x can be chosen from one of $1, 2, \dots, (a+1)$, it is easy to observe that we will now have $(a+1)2^t k(\mathcal{B})$ output sequences for the priority queue computation $i\mathcal{B}\delta$.

To answer question (b) we define the total number of allowable pairs for all the *Insert-Delete* sequences to be,

$$s_n = \sum_{\mathcal{B}} n! 2^{n-t}$$

$$\begin{aligned}
&= \sum_t \sum_{B \text{ with } t \text{ segments}} n!2^{n-t} \\
&= \sum_t \alpha(n, t) n!2^{n-t}
\end{aligned}$$

where $\alpha(n, t)$ is the total number of legal *Insert-Delete* sequences of size n (n inserts and n deletes) with t well formed segments.

Table 14 below lists $\alpha(n, t)$ for $n = 1 \dots 6$.

Table 14: $\alpha(n, t)$ for $n = 1 \dots 6$

t	$n = 1$	$n = 2$	$n = 3$	$n = 4$	$n = 5$	$n = 6$
1	1	1	2	5	14	42
2		1	2	5	14	42
3			1	3	9	28
4				1	4	14
5					1	5
6						1

The following lemma provides a recurrent expression for $\alpha(n, t)$.

Lemma 13 *The total number of legal Insert-Delete sequences of size n (n inserts and n deletes) with t well formed segments is,*

$$\alpha(n, t) = \sum_{i=t-1}^{n-1} \alpha(n-1, i) \quad \text{for } t > 1$$

Proof: The above expression is equivalent to saying *the number of n -pair bracket sequences with t segments is equal to the number of $(n-1)$ -pair bracket sequences with at least $t-1$ segments*. Hence showing a one-to-one correspondence between,

- (a) the n -pair bracket sequences with t segments, and
 - (b) the $(n-1)$ -pair bracket sequences with at least $t-1$ segments
- will prove the above recurrent expression.

Table 15: Number of Allowable Pairs and Bracket Sequences for $n = 4$

β -Sequence	Number of τ	β -Sequence	Number of τ
10101010	24	11010010	24
10101100	24	11010100	24
10110010	24	11011000	16
10110100	24	11100010	16
10111000	16	11100100	20
11001010	24	11101000	16
11001100	24	11110000	8

One can easily observe that,

if $(\sigma_1)(\sigma_2) \dots (\sigma_{t-1})(\sigma_t)$ is a sequence of type (a)
 then, $(\sigma_1)(\sigma_2) \dots (\sigma_{t-1})\sigma_t$ is a sequence of type (b).

This correspondence is obviously reversible and so is one-to-one.

Average Number of Outputs for a Random Priority Queue Computation

Theorem 6 *The average number of output permutations (allowable pairs) for a random double ended priority queue computation is,*

$$\frac{n! + \sum_{t=1}^{n-1} \left\{ \binom{2n-t-1}{n-1} - \binom{2n-t-1}{n-t-1} \right\} n! 2^{n-t}}{c_n}$$

where n is the length of input permutations.

Proof: From lemma 13,

$$\alpha(n, t) = \alpha(n-1, t-1) + \alpha(n-1, t) + \cdots + \alpha(n-1, n-1)$$

$$\text{But } \alpha(n-1, t) + \cdots + \alpha(n-1, n-1) = \alpha(n, t+1)$$

Hence we obtain the following recurrence,

$$\alpha(n, t) = \alpha(n-1, t-1) + \alpha(n, t+1)$$

This recurrence shows that the numbers $\alpha(n, t)$ (some of which are shown in table 14) are the same as those considered by Atkinson and Chang [3] (with appropriate changes to subscripts). The work of Atkinson and Chang therefore gives the formula,

$$\alpha(n, t) = \binom{2n-t-1}{n-1} - \binom{2n-t-1}{n-t-1} \quad \text{for } n > 0$$

Since the number of output permutations for a given input bracket sequence with t well-formed segments is $n!2^{n-t}$, the total number of output permutations for $t < n$ is,

$$\sum_{t=1}^{n-1} \left\{ \binom{2n-t-1}{n-1} - \binom{2n-t-1}{n-t-1} \right\} n!2^{n-t}.$$

When $t = n$, $\alpha(n, t) = 1$ and the number of output permutations in this case is $n!$. Hence the total number of output permutations is,

$$s_n = n! + \sum_{t=1}^{n-1} \left\{ \binom{2n-t-1}{n-1} - \binom{2n-t-1}{n-t-1} \right\} n!2^{n-t}.$$

Hence the average number of allowable pairs (averaged over all Insert-Delete sequences) is,

$$\frac{s_n}{c_n} = \frac{n! + \sum_{t=1}^{n-1} \left\{ \binom{2n-t-1}{n-1} - \binom{2n-t-1}{n-t-1} \right\} n!2^{n-t}}{c_n}.$$

Table 16: Number of allowable pairs for all *Insert-Delete* Sequences

Length of Input	Number of τ
1	1
2	6
3	78
4	1608
5	45720

Example 5 Compute the total number of allowable pairs, s_n , considering all legal $\mathcal{B}$ (*Insert-Delete*) sequences for a double ended priority queue of size $n = 3$.

$$\begin{aligned}
 s_n &= n! + \sum_{t=1}^{n-1} \left\{ \binom{2n-t-1}{n-1} - \binom{2n-t-1}{n-t-1} \right\} n! 2^{n-t} \\
 s_3 &= 3! + \sum_{t=1}^2 \left\{ \binom{5-t}{2} - \binom{5-t}{2-t} \right\} 6 \cdot 2^{3-t} \\
 &= 6 + \left\{ \binom{4}{2} - \binom{4}{1} \right\} 6 \cdot 4 + \left\{ \binom{3}{2} - \binom{3}{0} \right\} 6 \cdot 2 \\
 &= 6 + 48 + 24 \\
 &= 78
 \end{aligned}$$

matching with the numerical result given in Table 16

3.2.3 The Number of Allowable Pairs with Fixed *DeleteMin-DeleteMax* Sequences

In this section we study $A(\mathcal{D})$, the number of allowable pairs associated with a *DeleteMin-DeleteMax* sequence $\mathcal{D}$.

Lemma 14 *When considering the sequences of DeleteMin-DeleteMax in a double ended priority queue, if $A(\mathcal{D}) = |\{(\sigma, \tau) : \sigma \text{ is transformed into } \tau \text{ by some priority queue computation whose DeleteMin-DeleteMax sequence is } \mathcal{D}\}|$ then,*

1. *The number of allowable pairs $A(\mathcal{D}) = A(\bar{\mathcal{D}})$ where $\bar{\mathcal{D}}$ is obtained from $\mathcal{D}$ by replacing each DeleteMin by DeleteMax and vice versa.*
2. *$A(\mathcal{D}\delta) = A(\bar{\mathcal{D}}\bar{\delta})$ where δ is either a DeleteMin or DeleteMax and $\bar{\delta}$ is the corresponding complementary delete operation.*

Proof: 1. We prove the first part of the lemma by establishing a one-to-one correspondence between the allowable pairs (σ, τ) produced by $\mathcal{D} = \delta_1\delta_2\cdots\delta_{n-1}\delta_n$ and the allowable pairs (σ', τ') produced by $\bar{\mathcal{D}} = \bar{\delta}_1\bar{\delta}_2\cdots\bar{\delta}_{n-1}\bar{\delta}_n$. Let us assume with no loss of generality that the input sequences are made up of ordered set of integers from 1 to n . We know for every input sequence $\sigma = a_1a_2\cdots a_i\cdots a_{n-1}a_n$, there is a unique complementary input sequence $\sigma' = a'_1a'_2\cdots a'_i\cdots a'_{n-1}a'_n$ whose elements are obtained by the transformation $a'_i = (n+1) - a_i$. We show that for every allowable pair (σ, τ) obtained by using $\mathcal{D}$, there is a corresponding allowable pair (σ', τ') obtained by using $\bar{\mathcal{D}}$.

Assume that $\tau = b_1b_2\cdots b_{n-1}b_n$ is the output sequence obtained by applying $\mathcal{D}$ onto $\sigma = a_1a_2\cdots a_i\cdots a_{n-1}a_n$. Assume that the element a_i is now transpositioned onto the r th position in τ (i.e: $b_r = a_i$). Then using the same *Insert-Delete* sequence and applying $\bar{\mathcal{D}}$ onto the input sequence $\sigma' = a'_1a'_2\cdots a'_i\cdots a'_{n-1}a'_n$, we will obtain the output sequence $\tau' = b'_1b'_2\cdots b'_{n-1}b'_n$ whose r th element $b'_r = a'_i$. In other words $b'_r = (n+1) - b_r = a'_i$. Hence if $\sigma = a_1a_2\cdots a_{n-1}a_n$ produces $\tau = b_1b_2\cdots b_{n-1}b_n$ using $\mathcal{D}$ then $\sigma' = a'_1a'_2\cdots a'_{n-1}a'_n$ will produce $\tau' = b'_1b'_2\cdots b'_{n-1}b'_n$ using the same *Insert-Delete* sequence and $\bar{\mathcal{D}}$, where $a'_i = (n+1) - a_i$ and $b'_i = (n+1) - b_i$ for $i = 1\cdots n$.

Table 17: Number of τ and Dd Sequences for $n = 4$

<i>Dd</i> Sequence	Number of τ	<i>Dd</i> Sequence	Number of τ
dddd	125	dddD	125
Dddd	111	DddD	111
dDdd	107	dDdD	107
DDdd	113	DDdD	113
ddDd	113	ddDD	113
DdDd	107	DdDD	107
dDDd	111	dDDD	111
DDDd	125	DDDD	125

Hence for every pair of (σ, τ) we have one corresponding pair of (σ', τ') . Arguing on similar lines we can show for every pair of (σ', τ') we have one corresponding pair of (σ, τ) and a one-to-one correspondence between the allowable pairs produced by $\mathcal{D}$ and the allowable pairs produced by $\bar{\mathcal{D}}$ is thus established.

2. When applying the last delete operation of a *DeleteMin-DeleteMax* sequence $\mathcal{D}$ on to an input sequence σ , there will be only one input element remaining in the queue. Hence the resulting output sequence τ will be the same regardless of whether the last delete operation is a *DeleteMin* or a *DeleteMax*. Hence $A(\mathcal{D}\delta) = A(\mathcal{D}\bar{\delta})$.

Theorem 1 of Chapter 2 and the previous lemma show that $A(\mathcal{D}) = (n+1)^{n-1}$ for $\mathcal{D}$ any of $d^n, D^n, d^{n-1}D, D^{n-1}d$. No other exact results are known. However the numerical evidence for $n \leq 6$ (see Appendix B.2.4) is that $A(\mathcal{D}) \leq (n+1)^{n-1}$ for all $\mathcal{D}$. The table 17 lists the values of $A(\mathcal{D})$ for $n = 4$.

3.2.4 The Number of Outputs for Fixed Inputs and Fixed Insert-Delete Sequences

There is a simple formula for the number of outputs obtainable from a fixed input and fixed *Insert-Delete* sequence. This is stated by the following theorem.

Theorem 7 *The number of output permutations τ for any fixed input and Insert Delete sequence in a double ended priority queue is 2^{n-t} , where t is the number of well formed segments in the Insert-Delete sequence.*

Proof: Consider an arbitrary input sequence of length n . Assume that the following well formed *Insert-Delete* sequence of delete operations are performed on the input sequence. Here a ')' denotes either a *DeleteMin* or a *DeleteMax* and a '(' denotes an *Insert* operation.

$$(\overbrace{((()))}^{\alpha_t}) \dots (\overbrace{(())}^{\alpha_i}) \dots ((\dots (\overbrace{(())}^{\alpha_1}) \dots))$$

Consider a segment with the length of $2\alpha_1$ (i.e: there are α_1 additions and α_1 deletions in this segment). Now, since there are two ways of carrying out each deletion (either *DeleteMin* or *DeleteMax*) except for the last element of the segment for which both *DeleteMin* and *DeleteMax* produce the same output, we obtain 2^{α_1-1} distinct output

Table 18: Number of τ and Bracket Sequences for $n = 4$

B-Sequence	Number of τ	B-Sequence	Number of τ
10101010	24	11010010	96
10101100	48	11010100	192
10110010	48	11011000	192
10110100	96	11100010	96
10111000	96	11100100	192
11001010	48	11101000	192
11001100	96	11110000	192

2 of/de 2

MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS
STANDARD REFERENCE MATERIAL 1010a
(ANSI and ISO TEST CHART No. 2)

permutations for the above bracket segment. Similarly considering all the other segments, we obtain $2^{\alpha_2-1}, \dots, 2^{\alpha_i-1}, \dots, 2^{\alpha_t-1}$ output permutations. This results in a total of $2^{\alpha_1+\alpha_2+\dots+\alpha_t-t} = 2^{n-t}$ distinct output permutations.

Table 18 lists the total number of output permutations τ for all legal *Insert-Delete* bracket sequences in $\mathcal{B}$ for input sequence of length $n = 4$.

Example 6 Compute the number of τ for an input sequence of length $n = 4$, when the bracket sequence is 11010010.

The sequence 11010010 consists of two well formed segments, namely, (110100) and (10). Hence $t = 2$. Hence the number of τ produced by the bracket sequence is $n!2^{n-t} = 4!2^{4-2} = 96$, matching with the numerical result given in Table 18.

3.2.5 Number of Outputs for Fixed Inputs and Fixed *DeleteMin-DeleteMax* Sequences

In this section we discuss how many outputs can be generated from a fixed input σ using a fixed sequence $\mathcal{D}$ of *DeleteMin-DeleteMax* as the *Insert-Delete* sequence vary over all possibilities.

Since there are c_n different *Insert-Delete* sequences there can be no more than c_n different outputs. This upper bound is attained in the case that $\sigma = n, n-1, \dots, 2, 1$ and $\mathcal{D} = d^n$ as mentioned in section 2.2. Arguing as in Lemma 14 this upper bound is also attained for,

$$\begin{aligned} \sigma &= n, n-1, \dots, 2, 1, & \mathcal{D} &= d^{n-1}D \\ \sigma &= 1, 2, \dots, n-1, n, & \mathcal{D} &= D^n, D^{n-1}d \end{aligned}$$

We know no other cases where this upper bound is attained. Exhaustive enumeration for $n \leq 5$ shows that the upper bound is attained only in the above cases. (See Appendix B.2.5).

3.2.6 The Number of Outputs for Fixed *Insert-Delete* and Fixed *DeleteMin-DeleteMax* Sequences

In this section we consider a fixed double ended priority queue computation (i.e: both the *Insert-Delete* sequence $\mathcal{B}$ and the *DeleteMin-DeleteMax* sequence $\mathcal{D}$ are fixed).

Table 19: Number of τ for Fixed $\mathcal{D}$ and Fixed $\mathcal{B}$ -sequences

Length of Input	Number of τ
1	2
2	12
3	120
4	1672
5	29768

Let $k(\mathcal{B}, \mathcal{D})$ be the number of outputs as the input varies over all possibilities. In section 2.3, lemma 7 the case $\mathcal{D} = d^n$ was studied and recurrences to compute $k(\mathcal{B}, \mathcal{D})$ were found. Similar recurrences are not known in the general case but numerical evidence for $n \leq 5$ suggest (see Appendix B.2.6) the conjecture

$$k(\mathcal{B}, \mathcal{D}) \leq k(\mathcal{B}, d^n).$$

Chapter 4

Conclusion

4.1 Summary of Results

We considered the two abstract data types namely, ordinary priority queues and double ended priority queues and studied their operations on numeric input sequences in this research work. We also looked at binary priority queue as a specialized form of ordinary priority queue in which the input sequence is binary. We reviewed some of the existing implementation schemes for ordinary priority queues and double ended priority queues and their complexity analysis. Comparison of time complexities of most of the existing implementation schemes in the literature was given. Applications for ordinary and double ended priority queues were discussed and example applications were provided.

In chapter 2 we looked at two different methods of generating output sequences using ordinary priority queues. In the first method, each of the $n!$ input permutations was processed through all c_n legal *Insert-DeleteMin* priority queue computations and the unique output permutations were returned. In the second method, we fixed the *Insert-DeleteMin* bracket sequences and processed all $n!$ distinct input sequences through the priority queue. This method enabled us to compute the total number of output permutations for each of the c_n *Insert-DeleteMin* bracket sequences. We proved using induction on n that the size of the set of all σ such that (σ, τ) is allowable

is $s(\tau) = \prod_{i=1}^n \{i - b(i)\}$ where $b(i) = \max_j \{j < i, \sigma_j > \sigma_i\}$, $1 \leq i \leq n$. We also obtained an important result on the number of allowable pairs for ordinary priority queues. This result, given in the form of a theorem, states that the number of allowable pairs (σ, τ) on a set of size n is $(n + 1)^{n-1}$. Moreover considering the *Insert-DeleteMin* bracket sequences, we found that

- If $\mathcal{B}$ is a well formed bracket sequence then $k(i\mathcal{B}d) = k(\mathcal{B})$ and
- If $\mathcal{B}_1, \mathcal{B}_2$ are well formed bracket sequences of length $2a, 2b$ then $k(\mathcal{B}_1\mathcal{B}_2) = k(\mathcal{B}_1)k(\mathcal{B}_2)\binom{a+b}{a}$

where $k(\mathcal{B})$ is the number of output permutations for the bracket sequence $\mathcal{B}$. Using these results we computed the total number of output permutations taken over all c_n permissible bracket sequences using the average number of output permutations for a random priority queue computation. We also found that for binary priority queues

- If $\mathcal{B}$ is a well formed bracket sequence then $k(i\mathcal{B}d) = 1 + k(\mathcal{B})$ and
- If $\mathcal{B}_1, \mathcal{B}_2$ are well formed bracket sequences then $k(\mathcal{B}_1\mathcal{B}_2) = k(\mathcal{B}_1)k(\mathcal{B}_2)$

where $k(\mathcal{B})$ is the number of output permutations for the bracket sequence $\mathcal{B}$. Finally we also derived the average number of output permutations for a random binary priority queue computation.

Unlike ordinary priority queues, due to the presence of the additional operation *DeleteMax*, double ended priority queues are more difficult to analyze. We stated six different methods for generating output sequences using double ended priority queues and discussed each of them in detail. Numerical results obtained in each of these six methods were tabulated and attempts were made to describe the output patterns. Unlike the case of ordinary priority queues where the total number of allowable pairs is equal to $(n + 1)^{n-1}$, no general formula is known for the total number of allowable pairs in the case of double ended priority queues. We found that when considering the *Insert-Delete* bracket sequences, an analogous results for the ordinary priority queues can be obtained for the doubled ended priority queues. Given in chapter 3 as a lemma, this result states,

- If $\mathcal{B}$ is a well formed bracket sequence of length a then $k(i\mathcal{B}\delta) = (a+1)2^t k(\mathcal{B})$ and
- If $\mathcal{B}_1, \mathcal{B}_2$ are well formed bracket sequences of length $2a, 2b$ then
$$k(\mathcal{B}_1\mathcal{B}_2) = k(\mathcal{B}_1)k(\mathcal{B}_2)\binom{a+b}{a}$$

where t is the number of well formed bracket segments in $\mathcal{B}$, δ is either a *DeleteMin* or a *DeleteMax* and $k(\mathcal{B})$ is the number of output permutations for the bracket sequence $\mathcal{B}$. We also found that the number of output permutations for a given input *Insert-Delete* bracket sequence is $n!2^{n-t}$, where t is the number of well formed segments in the bracket sequence. Using an idea illustrated in [3] we also showed that the average number of output permutations (taken over all legal bracket sequences) is, $(n! + \sum_{t=1}^{n-1} \{ \binom{2n-t-1}{n-1} - \binom{2n-t-1}{n-t-1} \} n!2^{n-t}) / c_n$. Finally we conjectured that the number of output permutations for a given *DeleteMin-DeleteMax* sequence is $\leq (n+1)^{n-1}$ and the number of output permutations for a fixed *Insert-Delete* sequence and a fixed *DeleteMin-DeleteMax* sequence $k(\mathcal{B}, \mathcal{D})$ is less than the quantity obtained for ordinary priority queues, $k(\mathcal{B}, d^n)$.

4.2 Future Work

Despite the rich collection of numerical data obtained for double ended priority queues, we accomplished only a moderate amount of progress as far as deriving mathematical relationships between the variables involved. Most notably we lack a formula that can describe the number of allowable pairs (σ, τ) for a double ended priority queue. Any further work in this area may involve unravelling the mathematical relationships between the input, output variables in the number of tables listed in chapter 3. Two conjectures stated in sections 3.2.5 and 3.2.6 based upon the numerical results also need to be proved.

Bibliography

- [1] M.D. Atkinson. Transforming binary sequences using priority queues. *Technical Report, Dept. of Math. and Comp. Sciences, St. Andrews University*, Sep 1992.
- [2] M.D. Atkinson and Robert Beals. Priority queues and permutations. *Technical Report, Dept. of Math and Comp. Sciences, St. Andrews, Scotland*, Nov 1992.
- [3] M.D. Atkinson and H.W. Chang. Computing the number of mergings with constraints. *Information Processing Letters*, 24(5), Mar 1987.
- [4] M.D. Atkinson, J.R. Sack, N. Santoro and T. Strothotte. An efficient implicit double ended priority queue. *Technical Report SCS-TR-55, School of Comp. Science, Carleton University*, Jul 1984.
- [5] M.D. Atkinson, J.R. Sack, N. Santoro and T. Strothotte. Min-max heaps and generalized priority queues. *Communications of the ACM*, 29(10), Oct 1986.
- [6] M.D. Atkinson and Murali Thiagarajah. Permutational power of priority queues. *BIT*, 33(1):2-6, Jul 1993.
- [7] Svante Carlsson. The deap - a double ended heap to implement double ended priority queues. *Information Processing Letters*, Sep 1987.
- [8] A. Erdélyi (ed). *Higher Transcendental Functions*. McGraw-Hill, New York, 1953.
- [9] R.W. Floyd. Algorithm 245: treesort3. *Communications of the ACM*, 7, Dec 1964.

- [10] G.H. Gonnet and R. Baeza-Yates. *Handbook of Algorithms and Data Structures*. Addison-Wesley, Reading, Massachusetts, 1991.
- [11] D. A. Huffman. A method for the construction of minimum-redundancy codes. *Proceedings of the Institute of Radio Engineers*, 40:1098-1101, 1952.
- [12] Douglas W. Jones. An empirical comparison of priority-queue and event-set implementations. *Communications of the ACM*, 29(4), Apr 1986.
- [13] D.E. Knuth. *Sorting and Searching, The Art of Computer Programming Vol 3*. Addison-Wesley, Reading, Massachusetts, 1973.
- [14] M.C. Monard. Design and analysis of external quicksort algorithms. *PhD Dissertation, PUC University of Rio de Janeiro*, Feb 1980.
- [15] J. Riordan. *An Introduction to Combinatorial Analysis*. Wiley, New York, 1958.
- [16] Porter. T and Simon. I. Random insertion into a priority queue structure. *IEEE Transactions on Software Engineering*, 1:292-298, 1975.
- [17] J.S. Viter. Design and analysis of dynamic huffman codes. *Journal of the ACM*, 34:825-845, 1987.
- [18] J. Vuillemin. A data structure for manipulating priority queues. *Communications of the ACM*, 21:309-315, 1978.
- [19] Mark Allen Weiss. *Data Structures and Algorithm Analysis*. Benjamin / Cummings, California, 1992.
- [20] J.W.J. Williams. Algorithm 232: heapsort. *Communications of the ACM*, 7(6):347-348, 1964.
- [21] Christopher J. Van Wyk. *Data Structures and C Programs*. Addison-Wesley, Reading, Massachusetts, 1988.

Appendix A

Turbo Pascal Listings of Codes: Ordinary and Double Ended Priority Queues

All the code listings which appear in this appendix have been written using Borland Turbo Pascal Version 7.0.

A.1 Listing of Priority Queue Engine

The following Turbo Pascal unit implements the functions and procedures necessary for the operations (*Insert*, *DeleteMin*, *DeleteMax* etc) of ordinary and double ended priority queues. Actual programs implementing the various output generation methods discussed in the chapters are listed in the sections following this unit.

```
unit pqengine;

interface

const
    len = 4; { length of input sequence }

type
    BracketString = string[2*len];
    DdString = string[len];
    NumberArray = array[1..len] of integer;
    StringArray = array[1..500] of BracketString;
```

```

DdStringArray = array[1..500] of DdString;

procedure ConvertBinary(i : integer; var b: BracketString);
function first(len : integer) : longint;
function last(len : integer) : longint;
procedure DeleteMin(var sink, queue : NumberArray);
procedure DeleteMax(var sink, queue : NumberArray);
procedure Insert(var source, queue : NumberArray);
procedure Initialize(var sink, queue: NumberArray);
procedure MakeSource(var source : NumberArray; x : longint);
function Good(n : longint) : boolean;
procedure GenerateBracketSequences(var Bracket : StringArray);
function BracketOperation(B : StringArray; i, j : integer) : integer;
procedure WriteArray(a : NumberArray);
procedure RemoveDuplicates(var a : StringArray; var max : integer);
function NoOfBracketSequences(len : integer) : longint;
function NoOfDOperations(len : integer) : integer;
procedure GenerateDdSequences(var D : DdStringArray);
procedure DeleteOperation(D : DdStringArray; l, j : integer; var sink, queue:
NumberArray);

implementation

procedure ConvertBinary(i : integer; var b: BracketString);
var
    n, j : integer;
    a : array[1..2*len] of integer;
begin
    j := i;
    n := 0;
    while j <> 0 do
        begin
            Inc(n);
            a[n] := j mod 2;
            j := j div 2;
        end;
    for j := n downto 1 do
        b[n+1-j] := char(a[j]+48);
    end;
end;

```

```
function first(len : integer) : longint;
{* returns the first permutation in longint - e.g: for len = 3, first
   permutation is 123 *}
var
    i, j : integer;
    n, power : longint;
begin
    n := 0;
    for i := 1 to len do
        begin
            power := 1;
            for j := 1 to (len-i) do power := power*10;
            n := n + i*power;
        end;
    first := n;
end;
```

```
function last(len : integer) : longint;
{* returns the last permutation in longint - e.g: for len = 3, last
   permutation is 321 *}
var
    i, j : integer;
    n, power : longint;
begin
    n := 0;
    for i := 1 to len do
        begin
            power := 1;
            for j := 1 to (len-i) do power := power*10;
            n := n + (len-i+1)*power;
        end;
    last := n;
end;
```

```
procedure DeleteMin(var sink, queue : NumberArray);
{* deletes the smallest element in the queue and appends it to the sink *}
var
    i, index, smallest : longint;
```

```
    success : boolean;
begin
    index := 0;
    smallest := maxint;
    success := false;
    for i := 1 to len do
        if (queue[i] <> 0) and (queue[i] < smallest)
            then begin
                smallest := queue[i];
                index := i;
                success := true;
            end;
    if success then
        begin
            i := 1;
            while (sink[i] <> 0) do inc(i);
            sink[i] := smallest;
            queue[index] := 0;
        end;
end;

procedure DeleteMax(var sink, queue : NumberArray);
(* deletes the largest element in the queue and appends it to the sink *)
var
    i, index, largest : integer;
    success : boolean;
begin
    index := 0;
    largest := 0;
    success := false;
    for i := 1 to len do
        if (queue[i] <> 0) and (queue[i] > largest)
            then begin
                largest := queue[i];
                index := i;
                success := true;
            end;
    if success then
        begin
```

```
        i := 1;
        while (sink[i] <> 0) do inc(i);
        sink[i] := largest;
        queue[index] := 0;
    end;

end;

procedure Insert(var source, queue : NumberArray);
(* removes the next element from the source and appends it to the queue *)
var
    i, j : integer;
    done : boolean;
begin
    i := 1;
    done := false;
    while (i <= len) and (not done) do if source[i] <> 0 then done := true
        else inc(i);
        if done then begin
            j := 1;
            while queue[j] <> 0 do Inc(j);
            queue[j] := source[i];
            source[i] := 0;
        end;
    end;

end;

procedure Initialize(var sink, queue: NumberArray);
(* initializes sink and queue *)
var
    i : integer;
begin
    for i := 1 to len do
        begin
            queue[i] := 0;
            sink[i] := 0;
        end;
    end;

end;
```

```
procedure MakeSource(var source : NumberArray; x : longint);
  (* initializes source with the given input sequence *)
var
  i : integer;
begin
  for i := 1 to len do
    begin
      source[len-i+1] := (x mod 10);
      x := (x div 10);
    end;
  end;

function Good(n : longint) : boolean;
  (* determine if a given number has any duplicate digits.  good number should
     not have any duplicate digits *)
var
  temp : NumberArray;
  i, j : integer;
  found : boolean;
begin
  for i := 1 to len do temp[i] := 0;
  for i := 1 to len do
    begin
      temp[i] := (n mod 10);
      n := n div 10;
    end;
    found := false;
  i := 1;
  while (i <= len) and (not found) do
    begin
      j := 1;
      while (j <= len) and (not found) do
        if (((i <> j) and (temp[i] = temp[j]))
          or (temp[j] > len) or (temp[j] <= 0)
          or (temp[i] > len) or (temp[i] <= 0)) then found := true else
          inc(j);
        inc(i);
      end;
      Good := not found;
    end;
  end;
```



```
function HasEqualOnesAndZeros(number : longint) : boolean;
(* determine if a given bracket sequence has equal number of inserts and
   deletes *)
var
    one, zero: integer;
begin
    if (number mod 2 = 0) then
        begin
            zero := 0;
            one := 0;
            while number <> 0 do
                begin
                    if (number mod 2) = 0 then inc(zero) else inc(one);
                    number := number div 2;
                end;
            if zero = one then HasEqualOnesAndZeros := true else
                HasEqualOnesAndZeros := false;
        end else HasEqualOnesAndZeros := false;
    end;
```

```
function PerfectBracket(i : integer) : boolean;
(* determine if a bracket sequence is a well-formed one *)
var
    j, p, InsertCount, DeleteCount : integer;
    a : array[1..2*len] of integer;
    flag : boolean;
begin
    flag := true;
    if HasEqualOnesAndZeros(i) then
        begin
            j := i;
            p := 0;
            while j <> 0 do
                begin
                    Inc(p);
                    a[p] := j mod 2;
                    j := j div 2;
                end;
            end;
```

```
        InsertCount := 0;
        DeleteCount := 0;
        j := p;
        while flag and (j > 0) do
        begin
            if a[j] = 1 then Inc(InsertCount) else Inc(DeleteCount);
            Dec(j);
            If InsertCount < DeleteCount then flag := false;
        end;
        end else flag := false;
    PerfectBracket := flag;
end;

function fact(n : integer) : longint;
begin
    if n = 0 then fact := 1 else
        fact := n*fact(n-1);
end;

function NoOfBracketSequences(len : integer) : longint;
{ * computes the no of valid bracket sequences, clen
  DO NOT compute NoOfBracketSequences for len > 6 (overflow errors) *}
begin
    NoOfBracketSequences := round(fact(2*len)/(fact(len)*fact(len)*(len+1)));
end;

procedure InitializeBracket(var Bracket : StringArray);
var
    i : integer;
begin
    for i := 1 to 2*len do
        Bracket[i] := '';
    end;
end;

procedure GenerateBracketSequences(var Bracket : StringArray);
{ * generates all the Insert-Delete bracket sequences and stores them
  in an array *}
end;
```

```
var
    i, j, k, m, n : longint;
begin
    InitializeBracket(Bracket);
    m := round(exp((2*len-1)*ln(2)));
    n := round(exp(2*len*ln(2))-1);
    j := 1;
    for i := m to n do
        if PerfectBracket(i) then
            begin
                ConvertBinary(i, Bracket[j]);
                inc(j);
            end;
    end;

function BracketOperation(B : StringArray; i, j : integer) : integer;
{* determines the jth bracket operation in the ith bracket sequence *}
{* bracket operation 1 denotes insert and 0 denotes delete *}
begin
    if B[i][j] = '1' then BracketOperation := 1 else
        BracketOperation := 0;
end;

procedure WriteArray(a : NumberArray);
var
    i : integer;
    outfile : text;
begin
    assign(outfile, 'temp.out');
    append(outfile);
    for i := 1 to len do write(outfile, a[i]);
    writeln(outfile);
    close(outfile);
end;

procedure RemoveDuplicates(var a : StringArray; var max : integer);
{* this procedure reads in the output sequences from the file 'temp.out'
    and removes all the duplicates *}
end;
```

```
var
    s : string[len];
    infile : text;
    found : boolean;
    i : integer;
begin
    Assign(infile, 'temp.out');
    reset(infile);
    max := 0;
    while not eof(infile) do
        begin
            readln(infile, s);
            found := false;
            i := 1;
            while (i <= Max) and (not found) do
                if s = a[i] then found := true else Inc(i);
            if not found then begin
                Inc(max);
                a[max] := s;
            end;
        end;
        close(infile);
    end;

function NoOfddOperations(len : integer) : integer;
var
    i, j : integer;
begin
    j := 1;
    for i := 1 to len do j := j*2;
    NoOfddOperations := j;
end;

procedure GenerateDdSequences(var D : DdStringArray);
{* generates all possible dd sequences and stores them in array D *}
var
    i, m, j : integer;
    x : DdString;
```

```
procedure ConvertNumberToDd(i: integer; var b : DdString);
var
n, j : integer;
a : array[1..len] of integer;
begin
    for j := 1 to len do a[j] := 0;
    j := i;
    n := 0;
    while j <> 0 do
        begin
            Inc(n);
            a[n] := j mod 2;
            j := j div 2;
        end;
    for j := n downto 1 do
        b[len+1-j] := char(a[j]+48);
    end;

begin
    m := 1;
    for i := 1 to len do m := m*2;
    m := m - 1; {maximum value of Dd sequence in decimal eg:for len=3, m=7}
    for i := 0 to m do
        begin
            for j := 1 to len do x[j] := '0';
            ConvertNumberToDd(i,x);
            D[i+1] := x;
        end;
    end;

procedure DeleteOperation(D : DdStringArray; l, j : integer; var sink,queue:
NumberArray);
begin
    if D[l][j] = '0' then DeleteMin(sink,queue) else
    if D[l][j] = '1' then DeleteMax(sink,queue);
end;

end.
```

A.2 Generating Output Sequences Using Ordinary Priority Queues

```
program OrdinaryPQ(input,output);
{* returns the number of tau for each sigma *}

uses crt,pqengine;

var
    source, sink, queue : NumberArray;
    B, TauArray : StringArray;
    k, n, n1, n2, i, j, m : longint;
    max, count : integer;
    outfile : text;

begin
    ClrScr;
    Initialize(sink,queue);
    assign(outfile,'final.out');
    rewrite(outfile); close(outfile);
    count := 0;
    n1 := first(len);
    n2 := last(len);
    m := NoOfBracketSequences(len);
    GenerateBracketSequences(B);
    for n := n1 to n2 do
        if good(n) then
            begin
                assign(outfile,'temp.out');
                rewrite(outfile); close(outfile);
                for i := 1 to m do
                    begin
                        Initialize(sink,queue);
                        MakeSource(source,n);
                        for j := 1 to 2*len do
                            if BracketOperation(B,i,j) = 1 then Insert(source,queue) else
                            if BracketOperation(B,i,j) = 0 then DeleteMin(sink,queue);
                        WriteArray(sink);
                        write(n, ' ');
                        for j := 1 to 2*len do write(B[i][j]); write(' ');
```

```
        for j := 1 to len do write(sink[j]); writeln;
    end;
    RemoveDuplicates(TauArray,max);
    assign(outfile,'final.out');
    append(outfile);
    writeln(outfile,'input = ',n);
    writeln(outfile,'=====');
    for j := 1 to max do writeln(outfile,' ',TauArray[j]);
    close(outfile);
    count := count + max;
end;
assign(outfile,'final.out');
append(outfile);
writeln(outfile,'total = ',count);
close(outfile);
readln;
end.
```

A.3 Generating Output Sequences Using Double Ended Priority Queues

A.3.1 The Total Number of Allowable Pairs

```
program DPQ(input,output);
(* output permutations tau for all input permutations sigma *)

uses crt,pqengine;

var
    source, sink, queue : NumberArray;
    B, TauArray : StringArray;
    k, n, n1, n2, i, j, m : longint;
    max, count, d, l, DeletionNo : integer;
    outfile : text;
    Dd : DdStringArray;

begin
    ClrScr;
    Initialize(sink,queue);
```

```

assign(outfile,'final.out');
rewrite(outfile); close(outfile);
count := 0;
n1 := first(len);
n2 := last(len);
m := NoOfBracketSequences(len);
d := NoOfDdOperations(len);
GenerateBracketSequences(B);
GenerateDdSequences(Dd);
for n := n1 to n2 do
  if good(n) then
    begin
      assign(outfile,'temp.out');
      rewrite(outfile); close(outfile);
      for i := 1 to m do { for all bracket sequences do }
        begin
          writeln(i,' of ',m);
          for l := 1 to d do { for all Dd sequences do }
            begin
              Initialize(sink,queue);
              MakeSource(source,n);
              DeletionNo := 1; { index to the next Dd operation }
              for j := 1 to 2*len do
                if BracketOperation(B,i,j) = 1 then Insert(source,queue)
                else if BracketOperation(B,i,j) = 0 then
                  begin
                    DeleteOperation(Dd,l,DeletionNo,sink,queue);
                    Inc(DeletionNo);
                  end;
              WriteArray(sink);
            end;
          end;
        RemoveDuplicates(TauArray,max);
        assign(outfile,'final.out');
        append(outfile);
        writeln(outfile,'input = ',n);
        writeln(outfile,'=====');
        for j := 1 to max do writeln(outfile,' ',TauArray[j]);
        close(outfile);
        count := count + max;
    end;
  end;
end;

```



```
end;
assign(outfile,'final.out');
append(outfile);
writeln(outfile,'total = ',count);
close(outfile);
readln;
end.
```

A.3.2 The Number of Allowable Pairs with Fixed *Insert-Delete* Sequences

```
program DPQ(input,output);
{* output permutations tau for each bracket sequence *}

uses crt,pqengine;

var
    source, sink, queue : NumberArray;
    B, TauArray : StringArray;
    k, n, n1, n2, i, j, m : longint;
    max, count, d, l, DeletionNo : integer;
    outfile : text;
    Dd : DdStringArray;

begin
    ClrScr;
    Initialize(sink,queue);
    assign(outfile,'final.out');
    rewrite(outfile); close(outfile);
    count := 0;
    n1 := first(len);
    n2 := last(len);
    m := NoOfBracketSequences(len);
    d := NoOfDdOperations(len);
    GenerateBracketSequences(B);
    GenerateDdSequences(Dd);
    for i := 1 to m do { for all bracket sequences do }
    begin
        writeln('working on ',i,' of ',m);
```

```

assign(outfile,'temp.out');
rewrite(outfile); close(outfile);
for n := n1 to n2 do
  if good(n) then
    for l := 1 to d do { for all Dd sequences do }
    begin
      Initialize(sink,queue);
      MakeSource(source,n);
      DeletionNo := 1; { index to the next Dd operation }
      for j := 1 to 2*len do
        if BracketOperation(B,i,j) = 1 then Insert(source,queue) else
        if BracketOperation(B,i,j) = 0 then
          begin
            DeleteOperation(Dd,l,DeletionNo,sink,queue);
            Inc(DeletionNo);
          end;
        WriteArray(sink);
      end;
      RemoveDuplicates(TauArray,max);
      assign(outfile,'final.out');
      append(outfile);
      write(outfile,'input = ');
      for j := 1 to 2*len do write(outfile,B[i][j]); { ith bracket sequence }
      writeln(outfile);
      writeln(outfile,'=====');
      for j := 1 to max do
        begin
          if j = max then writeln(outfile,' ',TauArray[j], ' count = ',max) else
            writeln(outfile,' ',TauArray[j]);
        end;
      close(outfile);
      count := count + max;
    end;
  assign(outfile,'final.out');
  append(outfile);
  writeln(outfile,'total = ',count);
  close(outfile);
  readln;
end.

```

A.3.3 The Number of Outputs with Fixed Inputs and Fixed Insert-Delete Sequences

```
program DPQ(input,output);
{* output permutations tau for each bracket sequence and each sigma *}

uses crt,pqengine;

var
    source, sink, queue : NumberArray;
    B, TauArray : StringArray;
    k, n, n1, n2, i, j, m : longint;
    max, count, d, l, DeletionNo : integer;
    outfile : text;
    Dd : DdStringArray;

begin
    ClrScr;
    Initialize(sink,queue);
    assign(outfile,'final.out');
    rewrite(outfile); close(outfile);
    count := 0;
    n1 := first(len);
    n2 := last(len);
    m := NoOfBracketSequences(1er);
    d := NoOfDdOperations(len);
    GenerateBracketSequences(B);
    GenerateDdSequences(Dd);
    for i := 1 to m do { for all bracket sequences do }
    for n := n1 to n2 do { for each sigma do }
    if good(n) then
    begin
        assign(outfile,'temp.out');
        rewrite(outfile); close(outfile);
        for l := 1 to d do { for all Dd sequences do }
        begin
            Initialize(sink,queue);
            MakeSource(source,n);
            DeletionNo := 1; { index to the next Dd operation }
            for j := 1 to 2*len do
```

```

    if BracketOperation(B,i,j) = 1 then Insert(source,queue) else
    if BracketOperation(B,i,j) = 0 then
    begin
        DeleteOperation(Dd,l,DeletionNo,sink,queue);
        Inc(DeletionNo);
    end;
    WriteArray(sink);
end;
RemoveDuplicates(TauArray,max);
assign(outfile,'final.out');
append(outfile);
for j := 1 to 2*len do write(outfile,B[i][j]);
writeln(outfile,' sigma = ',n);
for j := 1 to max do
begin
    if j = max then writeln(outfile,' ',TauArray[j],' count = ',max) else
    writeln(outfile,' ',TauArray[j]);
end;
close(outfile);
count := count + max;
end;
assign(outfile,'final.out');
append(outfile);
writeln(outfile,' TOTAL = ',count);
close(outfile);
writeln('done');
readln;
end.

```

A.3.4 Output Generation with Fixed *DeleteMin-DeleteMax* Sequences

```

program DPQ(input,output);
(* output permutations tau for each Dd sequence and each sigma *)

uses crt,pqengine;

var
    source, sink, queue : NumberArray;

```

```
B, TauArray : StringArray;
k, n, n1, n2, i, j, m : longint;
max, count, d, l, DeletionNo : integer;
outfile : text;
Dd : DdStringArray;

begin
  ClrScr;
  Initialize(sink,queue);
  assign(outfile,'final.out');
  rewrite(outfile); close(outfile);
  count := 0;
  n1 := first(len);
  n2 := last(len);
  m := NoOfBracketSequences(len);
  d := NoOfDdOperations(len);
  GenerateBracketSequences(B);
  GenerateDdSequences(Dd);
  begin
    for l := 1 to d do { for all Dd sequences do }
      for n := n1 to n2 do
        if good(n) then
          begin
            assign(outfile,'temp.out');
            rewrite(outfile); close(outfile);
            for i := 1 to m do { for all bracket sequences do }
              begin
                Initialize(sink,queue);
                MakeSource(source,n);
                DeletionNo := 1; { index to the next Dd operation }
                for j := 1 to 2*len do
                  if BracketOperation(B,i,j) = 1 then Insert(source,queue) else
                  if BracketOperation(B,i,j) = 0 then
                    begin
                      DeleteOperation(Dd,l,DeletionNo,sink,queue);
                      Inc(DeletionNo);
                    end;
                WriteArray(sink);
              end;
            end;
            RemoveDuplicates(TauArray,max);
```

```

        assign(outfile,'final.out');
        append(outfile);
        for j := 1 to len do write(outfile,Dd[l][j]);
        writeln(outfile,' sigma = ',n);
        for j := 1 to max do
            if j = max then writeln(outfile,' ',TauArray[j],' count = ',max)
            else writeln(outfile,' ',TauArray[j]);
        close(outfile);
        count := count + max;
    end;
end;
assign(outfile,'final.out');
append(outfile);
writeln(outfile,'total = ',count);
close(outfile);
writeln('done');
readln;
end.

```

A.3.5 The Number of Allowable Pairs with Fixed Inputs and Fixed *DeleteMin-DeleteMax* Sequences

```

program DPQ(input,output);
{* output permutations tau for each Dd sequence *}

uses crt,pqengine;

var
    source, sink, queue : NumberArray;
    B, TauArray : StringArray;
    k, n, n1, n2, i, j, m : longint;
    max, count, d, l, DeletionNo : integer;
    outfile : text;
    Dd : DdStringArray;

begin
    ClrScr;
    Initialize(sink,queue);
    assign(outfile,'final.out');

```

```
rewrite(outfile); close(outfile);
count := 0;
n1 := first(len);
n2 := last(len);
m := NoOfBracketSequences(len);
d := NoOfDdOperations(len);
GenerateBracketSequences(B);
GenerateDdSequences(Dd);
begin
  for l := 1 to d do { for all Dd sequences do }
  begin
    writeln(l, ' of ', d);
    assign(outfile, 'temp.out');
    rewrite(outfile); close(outfile);
    for n := n1 to n2 do
      if good(n) then
        for i := 1 to m do { for all bracket sequences do }
        begin
          Initialize(sink, queue);
          MakeSource(source, n);
          DeletionNo := 1; { index to the next Dd operation }
          for j := 1 to 2*len do
            if BracketOperation(B, i, j) = 1 then Insert(source, queue) else
            if BracketOperation(B, i, j) = 0 then
              begin
                DeleteOperation(Dd, l, DeletionNo, sink, queue);
                Inc(DeletionNo);
              end;
            WriteArray(sink);
          end;
          RemoveDuplicates(TauArray, max);
          assign(outfile, 'final.out');
          append(outfile);
          for j := 1 to len do write(outfile, Dd[l][j]); { ith B sequence }
          writeln(outfile, '-----');
          for j := 1 to max do
            if j = max then writeln(outfile, ' ', TauArray[j], ' count = ', max)
            else writeln(outfile, ' ', TauArray[j]);
          close(outfile);
          count := count + max;
        end;
      end;
  end;
end;
```

```
        end;
    end;
    assign(outfile,'final.out');
    append(outfile);
    writeln(outfile);
    writeln(outfile,' TOTAL = ',count);
    close(outfile);
    writeln('done');
    readln;
end.
```

A.3.6 Output Generation with Fixed *Insert-Delete* Bracket Sequences and *DeleteMin-DeleteMax* Sequences

```
program DPQ(input,output);
{* output permutations tau for each Dd sequence and each bracket sequence *}

uses crt,pqengine;

var
    source, sink, queue : NumberArray;
    B, TauArray : StringArray;
    k, n, n1, n2, i, j, m : longint;
    max, count, d, l, DeletionNo : integer;
    outfile : text;
    Dd : DdStringArray;

begin
    ClrScr;
    Initialize(sink,queue);
    assign(outfile,'final.out');
    rewrite(outfile); close(outfile);
    count := 0;
    n1 := first(len);
    n2 := last(len);
    m := NoOfBracketSequences(len);
    d := NoOfDdOperations(len);
    GenerateBracketSequences(B);
    GenerateDdSequences(Dd);
```



```
begin
  for l := 1 to d do { for all Dd sequences do }
  for i := 1 to m do { for all bracket sequences do }
  begin
    assign(outfile,'temp.out');
    rewrite(outfile); close(outfile);
    for n := n1 to n2 do { for all sigma do }
    if good(n) then
    begin
      Initialize(sink,queue);
      MakeSource(source,n);
      DeletionNo := 1; { index to the next Dd operation }
      for j := 1 to 2*len do
        if BracketOperation(B,i,j) = 1 then Insert(source,queue) else
        if BracketOperation(B,i,j) = 0 then
        begin
          DeleteOperation(Dd,l,DeletionNo,sink,queue);
          Inc(DeletionNo);
        end;
        WriteArray(sink);
      end;
      RemoveDuplicates(TauArray,max);
      assign(outfile,'final.out');
      append(outfile);
      for j := 1 to len do write(outfile,Dd[l][j]); write(outfile,' ');
      for j := 1 to 2*len do write(outfile,B[i][j]); writeln(outfile);
      for j := 1 to max do
        if j = max then writeln(outfile,' ',TauArray[j],' count = ',max)
        else writeln(outfile,' ',TauArray[j]);
      close(outfile);
      count := count + max;
    end;
  end;
  assign(outfile,'final.out');
  append(outfile);
  writeln(outfile,'total = ',count);
  close(outfile);
  readln;
end.
```

Appendix B

Output Listings for Ordinary and Double Ended Priority Queues

All output listings given in the appendices are for input permutations of size $n = 4$.

B.1 Ordinary Priority Queues

B.1.1 Output Generation: Method I

The following table was obtained by processing each of the $4! = 24$ input permutations through all $c_4 = 14$ permissible *Insert-DeleteMin* priority queue computations and eliminating duplicate output permutations. Note that the total number of outputs for $n = 4$ is $(n + 1)^{n-1} = 125$.

Table 22: Output Generation with Ordinary Priority Queues: Method (i)

Inputs	Outputs	Total
1234	1234	1
1243	1243 1234	2
1324	1324 1234	2
1342	1342 1324 1234	3
1423	1423 1243 1234	3
1432	1432 1423 1342 1324 1234	5
2134	2134 1234	2
2143	2143 2134 1243 1234	4
2314	2314 2134 1234	3
2341	2341 2314 2134 1234	4
2413	2413 2143 2134 1243 1234	5
2431	2431 2413 2341 2314 2134 1234	6
3124	3124 1324 1234	3
3142	3142 3124 1342 1324 1234	5
3214	3214 3124 2314 2134 1234	5
3241	3241 3214 3124 2341 2314 2134 1234	7
3412	3412 3142 3124 1342 1324 1234	6
3421	3421 3412 3241 3214 3124 2341 2314 2134 1234	9
4123	4123 1423 1243 1234	4
4132	4132 4123 1432 1423 1342 1324 1234	7
4213	4213 4123 2413 2143 2134 1243 1234	7
4231	4231 4213 4123 2431 2413 2341 2314 2134 1234	9
4312	4312 4132 4123 3412 3142 3124 1342 1324 1234	9
4321	4321 4312 4231 4213 4123 3421 3412 3241 3214 3124 2341 2314 2134 1234	14
Total		125

B.1.2 Output Generation: Method (ii)

The following table was obtained by fixing the *Insert-DeleteMin* bracket sequences and processing all $4! = 24$ input permutations through the priority queue.

Table 23: Output Generation with Ordinary Priority Queues: Method (ii)

B Sequence	Input	Outputs	Total
10101010	1234	1234	
	1243	1243	
	1324	1324	
	1342	1342	
	1423	1423	
	1432	1432	
	2134	2134	
	2143	2143	
	2314	2314	
	2341	2341	
	2413	2413	
	2431	2431	
	3124	3124	
	3142	3142	
	3214	3214	
	3241	3241	
	3412	3412	
	3421	3421	
	4123	4123	
	4132	4132	
	4213	4213	
	4231	4231	
	4312	4312	
	4321	4321	
			24

Continued on the next page

<i>B</i> Sequence	Input	Outputs	Total
10110010	3142	3124	12
	3214	3214	
	3241	3214	
	3412	3412	
	3421	3412	
	4123	4123	
	4132	4123	
	4213	4213	
	4231	4213	
	4312	4312	
	4321	4312	
	1234	1234	
	1243	1243	
	1324	1234	
	1342	1342	
	1423	1243	
	1432	1342	
	2134	2134	
	2143	2143	
	2314	2134	
	2341	2341	
	2413	2143	
	2431	2341	
10110100	3124	3124	12
	3142	3142	
	3214	3124	
	3241	3241	
	3412	3142	
	3421	3241	
	4123	4123	
	4132	4132	
	4213	4123	
	4231	4231	
	4312	4132	
	4321	4231	
	1234	1234	
	1243	1234	

Continued on the next page

B Sequence	Input	Outputs	Total
10101100	1234	1234	
	1243	1234	
	1324	1324	
	1342	1324	
	1423	1423	
	1432	1423	
	2134	2134	
	2143	2134	
	2314	2314	
	2341	2314	
	2413	2413	
	2431	2413	
	3124	3124	
	1324	1234	
	1342	1324	
	1423	1234	
	1432	1324	
	2134	2134	
	2143	2134	
	2314	2134	
	2341	2314	
	2413	2134	
	2431	2314	
	3124	3124	
	3142	3124	
	3214	3124	
	3241	3214	
	3412	3124	
	3421	3214	
	4123	4123	
	4132	4123	
	4213	4123	
	4231	4213	
	4312	4123	
	4321	4213	
10111000	1234	1234	8

Continued on next page

<i>B</i> Sequence	Input	Outputs	Total
11001010	1243	1234	4
	1324	1234	
	1342	1234	
	1423	1234	
	1432	1234	
	2134	2134	
	2143	2134	
	2314	2134	
	2341	2134	
	2413	2134	
	2431	2134	
	3124	3124	
	3142	3124	
	3214	3124	
	3241	3124	
	3412	3124	
	3421	3124	
	4123	4123	
	4132	4123	
	4213	4123	
	4231	4123	
	4312	4123	
	4321	4123	
	1234	1234	
	1243	1243	
	1324	1324	
	1342	1342	
	1423	1423	
	1432	1432	
	2134	1234	
	2143	1243	
	2314	2314	
	2341	2341	
	2413	2413	

Continued on next page

B Sequence	Input	Outputs	Total
11001100	2431	2431	12
	3124	1324	
	3142	1342	
	3214	2314	
	3241	2341	
	3412	3412	
	3421	3421	
	4123	1423	
	4132	1432	
	4213	2413	
	4231	2431	
	4312	3412	
	4321	3421	
	1234	1234	
	1243	1234	
	1324	1324	
	1342	1324	
	1423	1423	
	1432	1423	
	2134	1234	
	2143	1234	
	2314	2314	
	2341	2314	
	2413	2413	
	2431	2413	
	3124	1324	
	3142	1324	
	3214	2314	
	3241	2314	
	3412	3412	
	3421	3412	
	4123	1423	
	4132	1423	
	4213	2413	
	4231	2413	
	4312	3412	

Continued on next page

β Sequence	Input	Outputs	Total
11010010	4321	3412	6
	1234	1234	
	1243	1243	
	1324	1234	
	1342	1342	
	1423	1243	
	1432	1342	8
	2134	1234	
	2143	1243	
	2314	2134	
	2341	2341	
	2413	2143	
	2431	2341	
	3124	1234	
	3142	1342	
	3214	2134	
	3241	2341	
	3412	3142	
	3421	3241	
	4123	1243	
	4132	1342	
	4213	2143	
	4231	2341	
	4312	3142	
11010100	4321	3241	
	1234	1234	
	1243	1234	
	1324	1234	
	1342	1324	
	1423	1234	
	1432	1324	
	2134	1234	
	2143	1234	
	2314	2134	
	2341	2314	
	2413	2134	

Continued on next page

B Sequence	Input	Outputs	Total
11011000	2431	2314	6
	3124	1234	
	3142	1324	
	3214	2134	
	3241	2314	
	3412	3124	
	3421	3214	
	4123	1234	
	4132	1324	
	4213	2134	
	4231	2314	
	4312	3124	
	4321	3214	
	1234	1234	
	1243	1234	
	1324	1234	
	1342	1234	
	1423	1234	
	1432	1234	
	2134	1234	
	2143	1234	
	2314	2134	
	2341	2134	
	2413	2134	
	2431	2134	
	3124	1234	
	3142	1234	
	3214	2134	
	3241	2134	
	3412	3124	
	3421	3124	
	4123	1234	
	4132	1234	
	4213	2134	
	4231	2134	
	4312	3124	

Continued on next page

B Sequence	Input	Outputs	Total
11100010	4321	3124	3
	1234	1234	
	1243	1243	
	1324	1234	
	1342	1342	
	1423	1243	
	1432	1342	
	2134	1234	
	2143	1243	
	2314	1234	
	2341	2341	
	2413	1243	
	2431	2341	
	3124	1234	
	3142	1342	
	3214	1234	
	3241	2341	
	3412	1342	
	3421	2341	
	4123	1243	
	4132	1342	
	4213	1243	
	4231	2341	
11100100	4312	1342	4
	4321	2341	
	1234	1234	
	1243	1234	
	1324	1234	
	1342	1324	
	1423	1234	
	1432	1324	
	2134	1234	
	2143	1234	
	2314	1234	
	2341	2314	
	2413	1234	

Continued on next page

B Sequence	Input	Outputs	Total
11101000	2431	2314	3
	3124	1234	
	3142	1324	
	3214	1234	
	3241	2314	
	3412	1324	
	3421	2314	
	4123	1234	
	4132	1324	
	4213	1234	
	4231	2314	
	4312	1324	
	4321	2314	
	1234	1234	
	1243	1234	
	1324	1234	
	1342	1234	
	1423	1234	
	1432	1234	
	2134	1234	
	2143	1234	
	2314	1234	
	2341	2134	
	2413	1234	
	2431	2134	
	3124	1234	
	3142	1234	
	3214	1234	
	3241	2134	
	3412	1234	
	3421	2134	
	4123	1234	
	4132	1234	
	4213	1234	
	4231	2134	
	4312	1234	

Continued on next page

B Sequence	Input	Outputs	Total
11110000	4321	2134	2
	1234	1234	
	1243	1234	
	1324	1234	
	1342	1234	
	1423	1234	
	1432	1234	
	2134	1234	
	2143	1234	
	2314	1234	
	2341	1234	
	2413	1234	
	2431	1234	
	3124	1234	
	3142	1234	
	3214	1234	
	3241	1234	
	3412	1234	
	3421	1234	
	4123	1234	
	4132	1234	
	4213	1234	
	4231	1234	
	4312	1234	
	4321	1234	
			1
Total			105

B.2 Double Ended Priority Queues

B.2.1 The Total Number of Allowable Pairs

Table 24: The Total Number of Allowable Pairs

Inputs	Outputs	Total
1234	1234 1243 1324 1342 1423 1432 2134 2143 2314 2341 2413 2431 3124 3214 3142 3241 3412 3421 4123 4132 4312 4321	22
1243	1243 1234 1423 1432 2143 2134 2413 2431 4123 4213 4132 4231 4312 4321	14
1324	1324 1342 1234 1243 1423 1432 3124 3142 3214 3241 3412 3421 4123 4132 4312 4321	16
1342	1342 1324 1432 1423 1234 1243 3142 3124 3412 3421 4132 4312 4123 4321	14
1423	1423 1432 1243 1234 4123 4132 4213 4231 4312 4321	10
1432	1432 1423 1342 1324 1234 1243 4132 4123 4312 4321	10
2134	2134 2143 2314 2341 2413 2431 1234 1243 1324 1342 1423 1432 3124 3214 3142 3241 3412 3421 4123 4132 4312 4321	22
2143	2143 2134 2413 2431 1243 1234 1423 1432 4123 4213 4132 4231 4312 4321	14
2314	2314 2341 2134 2143 2413 2431 3214 3241 3124 3142 3412 3421 1234 1324 1243 1342 1423 1432 4123 4132 4312 4321	22
2341	2341 2314 2431 2413 2134 2143 3241 3214 3421 3412 3124 3142 4231 4321 4213 4312 4123 4132 1234 1243 1423 1432	22

Continued on next page

Inputs	Outputs	Total
2413	2413 2431 2143 2134 4213 4231 4123 4132 4312 4321 1243 1423 1234 1432	14
2431	2431 2413 2341 2314 2134 2143 4231 4213 4321 4312 4123 4132 1234 1243 1423 1432	16
3124	3124 3142 3214 3241 3412 3421 1324 1342 1234 1243 1423 1432 4123 4132 4312 4321	16
3142	3142 3124 3412 3421 1342 1324 1432 1423 1234 1243 4132 4312 4123 4321	14
3214	3214 3241 3124 3142 3412 3421 2314 2341 2134 2143 2413 2431 1234 1324 1243 1342 1423 1432 4123 4132 4312 4321	22
3241	3241 3214 3421 3412 3124 3142 2341 2314 2431 2413 2134 2143 4231 4321 4213 4312 4123 4132 1234 1243 1423 1432	22
3412	3412 3421 3142 3124 4312 4321 4132 4123 1342 1432 1324 1423 1234 1243	14
3421	3421 3412 3241 3214 3124 3142 4321 4312 4231 4213 4123 4132 2341 2431 2314 2413 2134 2143 1234 1243 1423 1432	22
4123	4123 4132 4213 4231 4312 4321 1423 1432 1243 1234	10
4132	4132 4123 4312 4321 1432 1423 1342 1324 1234 1243	10
4213	4213 4231 4123 4132 4312 4321 2413 2431 2143 2134 1243 1423 1234 1432	14
4231	4231 4213 4321 4312 4123 4132 2431 2413 2341 2314 2134 2143 1234 1243 1423 1432	16
4312	4312 4321 4132 4123 3412 3421 3142 3124 1342 1432 1324 1423 1234 1243	14
4321	4321 4312 4231 4213 4123 4132 3421 3412 3241 3214 3124 3142 2341 2431 2314 2413 2134 2143 1234 1243 1423 1432	22
Total		392

B.2.2 The Number of Allowable Pairs with Fixed *Insert-Delete* Sequences

Table 25: The Number of Allowable Pairs with Fixed *Insert-Delete* Sequences

Input	Outputs	Total
10101010	1234 1243 1324 1342 1423 1432 2134 2143 2314 2341 2413 2431 3124 3142 3214 3241 3412 3421 4123 4132 4213 4231 4312 4321	24
10101100	1234 1243 1324 1342 1423 1432 2134 2143 2314 2341 2413 2431 3124 3142 3214 3241 3412 3421 4123 4132 4213 4231 4312 4321	24
10110010	1234 1324 1243 1423 1342 1432 2134 2314 2143 2413 2341 2431 3124 3214 3142 3412 3241 3421 4123 4213 4132 4312 4231 4321	24
10110100	1234 1243 1324 1342 1423 1432 2134 2143 2314 2341 2413 2431 3124 3142 3214 3241 3412 3421 4123 4132 4213 4231 4312 4321	24
10111000	1234 1243 1423 1432 2134 2143 2413 2431 3124 3142 3412 3421 4123 4132 4312 4321	16
11001010	1234 2134 1243 2143 1324 3124 1342 3142 1423 4123 1432 4132 2314 3214 2341 3241 2413 4213 2431 4231 3412 4312 3421 4321	24
11001100	1234 1243 2134 2143 1324 1342 3124 3142 1423 1432 4123 4132 2314 2341 3214 3241 2413 2431 4213 4231 3412 3421 4312 4321	24
11010010	1234 1324 2134 2314 1243 1423 2143 2413 3124 3214 1342 1432 3142 3412 4123 4213 4132 4312 2341 2431 3241 3421 4231 4321	24

Continued on next page

Input	Outputs	Total
11010100	1234 1243 1324 1342 2134 2143 2314 2341 1423 1432 2413 2431 3124 3142 3214 3241 3412 3421 4123 4132 4213 4231 4312 4321	24
11011000	1234 1243 1423 1432 2134 2143 2413 2431 3124 3142 3412 3421 4123 4132 4312 4321	16
11100010	1234 1324 3124 3214 1243 1423 4123 4213 1342 1432 4132 4312 2341 2431 4231 4321	16
11100100	1234 1243 1324 1342 3124 3142 3214 3241 1423 1432 4123 4132 4213 4231 4312 4321 2314 2341 2413 2431	20
11101000	1234 1243 1423 1432 3124 3142 3412 3421 4123 4132 4312 4321 2134 2143 2413 2431	16
11110000	1234 1243 1423 1432 4123 4132 4312 4321	8
Total		284

B.2.3 The Number of Outputs with Fixed Inputs and Fixed Insert-Delete Sequences

Table 26: Number of Outputs with Fixed Inputs and Fixed Bracket Sequences

B Sequence	Input	Outputs	Total
10101010	1234	1234	1
10101010	1243	1243	1
10101010	1324	1324	1
10101010	1342	1342	1
10101010	1423	1423	1
10101010	1432	1432	1
10101010	2134	2134	1
10101010	2143	2143	1
10101010	2314	2314	1
10101010	2341	2341	1
10101010	2413	2413	1
10101010	2431	2431	1
10101010	3124	3124	1
10101010	3142	3142	1
10101010	3214	3214	1
10101010	3241	3241	1
10101010	3412	3412	1
10101010	3421	3421	1
10101010	4123	4123	1
10101010	4132	4132	1
10101010	4213	4213	1
10101010	4231	4231	1
10101010	4312	4312	1
10101010	4321	4321	1
10101100	1234	1234 1243	2

Continued on next page

<i>B</i> Sequence	Input	Outputs	Total
10101100	1243	1234 1243	2
10101100	1324	1324 1342	2
10101100	1342	1324 1342	2
10101100	1423	1423 1432	2
10101100	1432	1423 1432	2
10101100	2134	2134 2143	2
10101100	2143	2134 2143	2
10101100	2314	2314 2341	2
10101100	2341	2314 2341	2
10101100	2413	2413 2431	2
10101100	2431	2413 2431	2
10101100	3124	3124 3142	2
10101100	3142	3124 3142	2
10101100	3214	3214 3241	2
10101100	3241	3214 3241	2
10101100	3412	3412 3421	2
10101100	3421	3412 3421	2
10101100	4123	4123 4132	2
10101100	4132	4123 4132	2
10101100	4213	4213 4231	2
10101100	4231	4213 4231	2
10101100	4312	4312 4321	2
10101100	4321	4312 4321	2
10110010	1234	1234 1324	2
10110010	1243	1243 1423	2
10110010	1324	1234 1324	2
10110010	1342	1342 1432	2
10110010	1423	1243 1423	2
10110010	1432	1342 1432	2
10110010	2134	2134 2314	2
10110010	2143	2143 2413	2
10110010	2314	2134 2314	2
10110010	2341	2341 2431	2
10110010	2413	2143 2413	2
10110010	2431	2341 2431	2
10110010	3124	3124 3214	2
10110010	3142	3142 3412	2

Continued on next page

B Sequence	Input	Outputs	Total
10110010	3214	3124 3214	2
10110010	3241	3241 3421	2
10110010	3412	3142 3412	2
10110010	3421	3241 3421	2
10110010	4123	4123 4213	2
10110010	4132	4132 4312	2
10110010	4213	4123 4213	2
10110010	4231	4231 4321	2
10110010	4312	4132 4312	2
10110010	4321	4231 4321	2
10110100	1234	1234 1243 1324 1342	4
10110100	1243	1234 1243 1423 1432	4
10110100	1324	1234 1243 1324 1342	4
10110100	1342	1324 1342 1423 1432	4
10110100	1423	1234 1243 1423 1432	4
10110100	1432	1324 1342 1423 1432	4
10110100	2134	2134 2143 2314 2341	4
10110100	2143	2134 2143 2413 2431	4
10110100	2314	2134 2143 2314 2341	4
10110100	2341	2314 2341 2413 2431	4
10110100	2413	2134 2143 2413 2431	4
10110100	2431	2314 2341 2413 2431	4
10110100	3124	3124 3142 3214 3241	4
10110100	3142	3124 3142 3412 3421	4
10110100	3214	3124 3142 3214 3241	4
10110100	3241	3214 3241 3412 3421	4
10110100	3412	3124 3142 3412 3421	4
10110100	3421	3214 3241 3412 3421	4
10110100	4123	4123 4132 4213 4231	4
10110100	4132	4123 4132 4312 4321	4
10110100	4213	4123 4132 4213 4231	4
10110100	4231	4213 4231 4312 4321	4
10110100	4312	4123 4132 4312 4321	4
10110100	4321	4213 4231 4312 4321	4
10111000	1234	1234 1243 1423 1432	4
10111000	1243	1234 1243 1423 1432	4

Continued on next page

B Sequence	Input	Outputs	Total
10111000	1324	1234 1243 1423 1432	4
10111000	1342	1234 1243 1423 1432	4
10111000	1423	1234 1243 1423 1432	4
10111000	1432	1234 1243 1423 1432	4
10111000	2134	2134 2143 2413 2431	4
10111000	2143	2134 2143 2413 2431	4
10111000	2314	2134 2143 2413 2431	4
10111000	2341	2134 2143 2413 2431	4
10111000	2413	2134 2143 2413 2431	4
10111000	2431	2134 2143 2413 2431	4
10111000	3124	3124 3142 3412 3421	4
10111000	3142	3124 3142 3412 3421	4
10111000	3214	3124 3142 3412 3421	4
10111000	3241	3124 3142 3412 3421	4
10111000	3412	3124 3142 3412 3421	4
10111000	3421	3124 3142 3412 3421	4
10111000	4123	4123 4132 4312 4321	4
10111000	4132	4123 4132 4312 4321	4
10111000	4213	4123 4132 4312 4321	4
10111000	4231	4123 4132 4312 4321	4
10111000	4312	4123 4132 4312 4321	4
10111000	4321	4123 4132 4312 4321	4
11001010	1234	1234 2134	2
11001010	1243	1243 2143	2
11001010	1324	1324 3124	2
11001010	1342	1342 3142	2
11001010	1423	1423 4123	2
11001010	1432	1432 4132	2
11001010	2134	1234 2134	2
11001010	2143	1243 2143	2
11001010	2314	2314 3214	2
11001010	2341	2341 3241	2
11001010	2413	2413 4213	2
11001010	2431	2431 4231	2
11001010	3124	1324 3124	2
11001010	3142	1342 3142	2

Continued on next page

B Sequence	Input	Outputs	Total
11001010	3214	2314 3214	2
11001010	3241	2341 3241	2
11001010	3412	3412 4312	2
11001010	3421	3421 4321	2
11001010	4123	1423 4123	2
11001010	4132	1432 4132	2
11001010	4213	2413 4213	2
11001010	4231	2431 4231	2
11001010	4312	3412 4312	2
11001010	4321	3421 4321	2
11001100	1234	1234 1243 2134 2143	4
11001100	1243	1234 1243 2134 2143	4
11001100	1324	1324 1342 3124 3142	4
11001100	1342	1324 1342 3124 3142	4
11001100	1423	1423 1432 4123 4132	4
11001100	1432	1423 1432 4123 4132	4
11001100	2134	1234 1243 2134 2143	4
11001100	2143	1234 1243 2134 2143	4
11001100	2314	2314 2341 3214 3241	4
11001100	2341	2314 2341 3214 3241	4
11001100	2413	2413 2431 4213 4231	4
11001100	2431	2413 2431 4213 4231	4
11001100	3124	1324 1342 3124 3142	4
11001100	3142	1324 1342 3124 3142	4
11001100	3214	2314 2341 3214 3241	4
11001100	3241	2314 2341 3214 3241	4
11001100	3412	3412 3421 4312 4321	4
11001100	3421	3412 3421 4312 4321	4
11001100	4123	1423 1432 4123 4132	4
11001100	4132	1423 1432 4123 4132	4
11001100	4213	2413 2431 4213 4231	4
11001100	4231	2413 2431 4213 4231	4
11001100	4312	3412 3421 4312 4321	4
11001100	4321	3412 3421 4312 4321	4
11010010	1234	1234 1324 2134 2314	4
11010010	1243	1243 1423 2143 2413	4

Continued on next page

β Sequence	Input	Outputs	Total
11010010	1324	1234 1324 3124 3214	4
11010010	1342	1342 1432 3142 3412	4
11010010	1423	1243 1423 4123 4213	4
11010010	1432	1342 1432 4132 4312	4
11010010	2134	1234 1324 2134 2314	4
11010010	2143	1243 1423 2143 2413	4
11010010	2314	2134 2314 3124 3214	4
11010010	2341	2341 2431 3241 3421	4
11010010	2413	2143 2413 4123 4213	4
11010010	2431	2341 2431 4231 4321	4
11010010	3124	1234 1324 3124 3214	4
11010010	3142	1342 1432 3142 3412	4
11010010	3214	2134 2314 3124 3214	4
11010010	3241	2341 2431 3241 3421	4
11010010	3412	3142 3412 4132 4312	4
11010010	3421	3241 3421 4231 4321	4
11010010	4123	1243 1423 4123 4213	4
11010010	4132	1342 1432 4132 4312	4
11010010	4213	2143 2413 4123 4213	4
11010010	4231	2341 2431 4231 4321	4
11010010	4312	3142 3412 4132 4312	4
11010010	4321	3241 3421 4231 4321	4
11010100	1234	1234 1243 1324 1342 2134 2143 2314 2341	8
11010100	1243	1234 1243 1423 1432 2134 2143 2413 2431	8
11010100	1324	1234 1243 1324 1342 3124 3142 3214 3241	8
11010100	1342	1324 1342 1423 1432 3124 3142 3412 3421	8
11010100	1423	1234 1243 1423 1432 4123 4132 4213 4231	8
11010100	1432	1324 1342 1423 1432 4123 4132 4312 4321	8
11010100	2134	1234 1243 1324 1342 2134 2143 2314 2341	8
11010100	2143	1234 1243 1423 1432 2134 2143 2413 2431	8
11010100	2314	2134 2143 2314 2341 3124 3142 3214 3241	8
11010100	2341	2314 2341 2413 2431 3214 3241 3412 3421	8
11010100	2413	2134 2143 2413 2431 4123 4132 4213 4231	8
11010100	2431	2314 2341 2413 2431 4213 4231 4312 4321	8
11010100	3124	1234 1243 1324 1342 3124 3142 3214 3241	8
11010100	3142	1324 1342 1423 1432 3124 3142 3412 3421	8
11010100	3214	2134 2143 2314 2341 3124 3142 3214 3241	8

Continued on next page

B Sequence	Input	Outputs	Total
11010100	3241	2314 2341 2413 2431 3214 3241 3412 3421	8
11010100	3412	3124 3142 3412 3421 4123 4132 4312 4321	8
11010100	3421	3214 3241 3412 3421 4213 4231 4312 4321	8
11010100	4123	1234 1243 1423 1432 4123 4132 4213 4231	8
11010100	4132	1324 1342 1423 1432 4123 4132 4312 4321	8
11010100	4213	2134 2143 2413 2431 4123 4132 4213 4231	8
11010100	4231	2314 2341 2413 2431 4213 4231 4312 4321	8
11010100	4312	3124 3142 3412 3421 4123 4132 4312 4321	8
11010100	4321	3214 3241 3412 3421 4213 4231 4312 4321	8
11011000	1234	1234 1243 1423 1432 2134 2143 2413 2431	8
11011000	1243	1234 1243 1423 1432 2134 2143 2413 2431	8
11011000	1324	1234 1243 1423 1432 3124 3142 3412 3421	8
11011000	1342	1234 1243 1423 1432 3124 3142 3412 3421	8
11011000	1423	1234 1243 1423 1432 4123 4132 4312 4321	8
11011000	1432	1234 1243 1423 1432 4123 4132 4312 4321	8
11011000	2134	1234 1243 1423 1432 2134 2143 2413 2431	8
11011000	2143	1234 1243 1423 1432 2134 2143 2413 2431	8
11011000	2314	2134 2143 2413 2431 3124 3142 3412 3421	8
11011000	2341	2134 2143 2413 2431 3124 3142 3412 3421	8
11011000	2413	2134 2143 2413 2431 4123 4132 4312 4321	8
11011000	2431	2134 2143 2413 2431 4123 4132 4312 4321	8
11011000	3124	1234 1243 1423 1432 3124 3142 3412 3421	8
11011000	3142	1234 1243 1423 1432 3124 3142 3412 3421	8
11011000	3214	2134 2143 2413 2431 3124 3142 3412 3421	8
11011000	3241	2134 2143 2413 2431 3124 3142 3412 3421	8
11011000	3412	3124 3142 3412 3421 4123 4132 4312 4321	8
11011000	3421	3124 3142 3412 3421 4123 4132 4312 4321	8
11011000	4123	1234 1243 1423 1432 4123 4132 4312 4321	8
11011000	4132	1234 1243 1423 1432 4123 4132 4312 4321	8
11011000	4213	2134 2143 2413 2431 4123 4132 4312 4321	8
11011000	4231	2134 2143 2413 2431 4123 4132 4312 4321	8
11011000	4312	3124 3142 3412 3421 4123 4132 4312 4321	8
11011000	4321	3124 3142 3412 3421 4123 4132 4312 4321	8
11100010	1234	1234 1324 3124 3214	4
11100010	1243	1243 1423 4123 4213	4
11100010	1324	1234 1324 3124 3214	4
11100010	1342	1342 1432 4132 4312	4

Continued on next page

<i>B</i> Sequence	Input	Outputs	Total
11100010	1423	1243 1423 4123 4213	4
11100010	1432	1342 1432 4132 4312	4
11100010	2134	1234 1324 3124 3214	4
11100010	2143	1243 1423 4123 4213	4
11100010	2314	1234 1324 3124 3214	4
11100010	2341	2341 2431 4231 4321	4
11100010	2413	1243 1423 4123 4213	4
11100010	2431	2341 2431 4231 4321	4
11100010	3124	1234 1324 3124 3214	4
11100010	3142	1342 1432 4132 4312	4
11100010	3214	1234 1324 3124 3214	4
11100010	3241	2341 2431 4231 4321	4
11100010	3412	1342 1432 4132 4312	4
11100010	3421	2341 2431 4231 4321	4
11100010	4123	1243 1423 4123 4213	4
11100010	4132	1342 1432 4132 4312	4
11100010	4213	1243 1423 4123 4213	4
11100010	4231	2341 2431 4231 4321	4
11100010	4312	1342 1432 4132 4312	4
11100010	4321	2341 2431 4231 4321	4
11100100	1234	1234 1243 1324 1342 3124 3142 3214 3241	8
11100100	1243	1234 1243 1423 1432 4123 4132 4213 4231	8
11100100	1324	1234 1243 1324 1342 3124 3142 3214 3241	8
11100100	1342	1324 1342 1423 1432 4123 4132 4312 4321	8
11100100	1423	1234 1243 1423 1432 4123 4132 4213 4231	8
11100100	1432	1324 1342 1423 1432 4123 4132 4312 4321	8
11100100	2134	1234 1243 1324 1342 3124 3142 3214 3241	8
11100100	2143	1234 1243 1423 1432 4123 4132 4213 4231	8
11100100	2314	1234 1243 1324 1342 3124 3142 3214 3241	8
11100100	2341	2314 2341 2413 2431 4213 4231 4312 4321	8
11100100	2413	1234 1243 1423 1432 4123 4132 4213 4231	8
11100100	2431	2314 2341 2413 2431 4213 4231 4312 4321	8
11100100	3124	1234 1243 1324 1342 3124 3142 3214 3241	8
11100100	3142	1324 1342 1423 1432 4123 4132 4312 4321	8
11100100	3214	1234 1243 1324 1342 3124 3142 3214 3241	8
11100100	3241	2314 2341 2413 2431 4213 4231 4312 4321	8

Continued on next page

B Sequence	Input	Outputs	Total
11100100	3412	1324 1342 1423 1432 4123 4132 4312 4321	8
11100100	3421	2314 2341 2413 2431 4213 4231 4312 4321	8
11100100	4123	1234 1243 1423 1432 4123 4132 4213 4231	8
11100100	4132	1324 1342 1423 1432 4123 4132 4312 4321	8
11100100	4213	1234 1243 1423 1432 4123 4132 4213 4231	8
11100100	4231	2314 2341 2413 2431 4213 4231 4312 4321	8
11100100	4312	1324 1342 1423 1432 4123 4132 4312 4321	8
11100100	4321	2314 2341 2413 2431 4213 4231 4312 4321	8
11101000	1234	1234 1243 1423 1432 3124 3142 3412 3421	8
11101000	1243	1234 1243 1423 1432 4123 4132 4312 4321	8
11101000	1324	1234 1243 1423 1432 3124 3142 3412 3421	8
11101000	1342	1234 1243 1423 1432 4123 4132 4312 4321	8
11101000	1423	1234 1243 1423 1432 4123 4132 4312 4321	8
11101000	1432	1234 1243 1423 1432 4123 4132 4312 4321	8
11101000	2134	1234 1243 1423 1432 3124 3142 3412 3421	8
11101000	2143	1234 1243 1423 1432 4123 4132 4312 4321	8
11101000	2314	1234 1243 1423 1432 3124 3142 3412 3421	8
11101000	2341	2134 2143 2413 2431 4123 4132 4312 4321	8
11101000	2413	1234 1243 1423 1432 4123 4132 4312 4321	8
11101000	2431	2134 2143 2413 2431 4123 4132 4312 4321	8
11101000	3124	1234 1243 1423 1432 3124 3142 3412 3421	8
11101000	3142	1234 1243 1423 1432 4123 4132 4312 4321	8
11101000	3214	1234 1243 1423 1432 3124 3142 3412 3421	8
11101000	3241	2134 2143 2413 2431 4123 4132 4312 4321	8
11101000	3412	1234 1243 1423 1432 4123 4132 4312 4321	8
11101000	3421	2134 2143 2413 2431 4123 4132 4312 4321	8
11101000	4123	1234 1243 1423 1432 4123 4132 4312 4321	8
11101000	4132	1234 1243 1423 1432 4123 4132 4312 4321	8
11101000	4213	1234 1243 1423 1432 4123 4132 4312 4321	8
11101000	4231	2134 2143 2413 2431 4123 4132 4312 4321	8
11101000	4312	1234 1243 1423 1432 4123 4132 4312 4321	8
11101000	4321	2134 2143 2413 2431 4123 4132 4312 4321	8
11110000	1234	1234 1243 1423 1432 4123 4132 4312 4321	8
11110000	1243	1234 1243 1423 1432 4123 4132 4312 4321	8
11110000	1324	1234 1243 1423 1432 4123 4132 4312 4321	8
11110000	1342	1234 1243 1423 1432 4123 4132 4312 4321	8
11110000	1423	1234 1243 1423 1432 4123 4132 4312 4321	8

Continued on next page

<i>B</i> Sequence	Input	Outputs	Total
11110000	1432	1234 1243 1423 1432 4123 4132 4312 4321	8
11110000	2134	1234 1243 1423 1432 4123 4132 4312 4321	8
11110000	2143	1234 1243 1423 1432 4123 4132 4312 4321	8
11110000	2314	1234 1243 1423 1432 4123 4132 4312 4321	8
11110000	2341	1234 1243 1423 1432 4123 4132 4312 4321	8
11110000	2413	1234 1243 1423 1432 4123 4132 4312 4321	8
11110000	2431	1234 1243 1423 1432 4123 4132 4312 4321	8
11110000	3124	1234 1243 1423 1432 4123 4132 4312 4321	8
11110000	3142	1234 1243 1423 1432 4123 4132 4312 4321	8
11110000	3214	1234 1243 1423 1432 4123 4132 4312 4321	8
11110000	3241	1234 1243 1423 1432 4123 4132 4312 4321	8
11110000	3412	1234 1243 1423 1432 4123 4132 4312 4321	8
11110000	3421	1234 1243 1423 1432 4123 4132 4312 4321	8
11110000	4123	1234 1243 1423 1432 4123 4132 4312 4321	8
11110000	4132	1234 1243 1423 1432 4123 4132 4312 4321	8
11110000	4213	1234 1243 1423 1432 4123 4132 4312 4321	8
11110000	4231	1234 1243 1423 1432 4123 4132 4312 4321	8
11110000	4312	1234 1243 1423 1432 4123 4132 4312 4321	8
11110000	4321	1234 1243 1423 1432 4123 4132 4312 4321	8
Total			1608

B.2.4 Output Generation with Fixed *DeleteMin-DeleteMax* Sequences

Table 27: Output Generation with Fixed *DeleteMin-DeleteMax* Sequences

<i>D</i> Sequence	Input	Outputs	Total
dddd	1234	1234	1
dddd	1243	1243 1234	2
dddd	1324	1324 1234	2
dddd	1342	1342 1324 1234	3
dddd	1423	1423 1243 1234	3
dddd	1432	1432 1423 1342 1324 1234	5
dddd	2134	2134 1234	2
dddd	2143	2143 2134 1243 1234	4
dddd	2314	2314 2134 1234	3
dddd	2341	2341 2314 2134 1234	4
dddd	2413	2413 2143 2134 1243 1234	5
dddd	2431	2431 2413 2341 2314 2134 1234	6
dddd	3124	3124 1324 1234	3
dddd	3142	3142 3124 1342 1324 1234	5
dddd	3214	3214 3124 2314 2134 1234	5
dddd	3241	3241 3214 3124 2341 2314 2134 1234	7
dddd	3412	3412 3142 3124 1342 1324 1234	6
dddd	3421	3421 3412 3241 3214 3124 2341 2314 2134 1234	9
dddd	4123	4123 1423 1243 1234	4
dddd	4132	4132 4123 1432 1423 1342 1324 1234	7
dddd	4213	4213 4123 2413 2143 2134 1243 1234	7
dddd	4231	4231 4213 4123 2431 2413 2341 2314 2134 1234	9
dddd	4312	4312 4132 4123 3412 3142 3124 1342 1324 1234	9
dddd	21	4321 4312 4231 4213 4123 3421 3412 3241 3214 3124 2341 2314 2134 1234	14

Continued on next page

<i>D</i> Sequence	Input	Outputs	Total
dddD	1234	1234	1
dddD	1243	1243 1234	2
dddD	1324	1324 1234	2
dddD	1342	1342 1324 1234	3
dddD	1423	1423 1243 1234	3
dddD	1432	1432 1423 1342 1324 1234	5
dddD	2134	2134 1234	2
dddD	2143	2143 2134 1243 1234	4
dddD	2314	2314 2134 1234	3
dddD	2341	2341 2314 2134 1234	4
dddD	2413	2413 2143 2134 1243 1234	5
dddD	2431	2431 2413 2341 2314 2134 1234	6
dddD	3124	3124 1324 1234	3
dddD	3142	3142 3124 1342 1324 1234	5
dddD	3214	3214 3124 2314 2134 1234	5
dddD	3241	3241 3214 3124 2341 2314 2134 1234	7
dddD	3412	3412 3142 3124 1342 1324 1234	6
dddD	3421	3421 3412 3241 3214 3124 2341 2314 2134 1234	9
dddD	4123	4123 1423 1243 1234	4
dddD	4132	4132 4123 1432 1423 1342 1324 1234	7
dddD	4213	4213 4123 2413 2143 2134 1243 1234	7
dddD	4231	4231 4213 4123 2431 2413 2341 2314 2134 1234	9
dddD	4312	4312 4132 4123 3412 3142 3124 1342 1324 1234	9
dddD	4321	4321 4312 4231 4213 4123 3421 3412 3241 3214 3124 2341 2314 2134 1234	14
ddDd	1234	1234 1243	2
ddDd	1243	1243	1
ddDd	1324	1324 1342 1234 1243	4
ddDd	1342	1342 1243	2
ddDd	1423	1423 1432 1243	3
ddDd	1432	1432 1342 1243	3
ddDd	2134	2134 2143 1234 1243	4
ddDd	2143	2143 1243	2
ddDd	2314	2314 2341 2134 2143 1234 1243	6
ddDd	2341	2341 2143 1243	3
ddDd	2413	2413 2431 2143 1243	4

Continued on next page

<i>D</i> Sequence	Input	Outputs	Total
ddDd	2431	2431 2341 2143 1243	4
ddDd	3124	3124 3142 1324 1342 1234 1243	6
ddDd	3142	3142 1342 1243	3
ddDd	3214	3214 3241 3124 3142 2314 2341 2134 2143 1234 1243	10
ddDd	3241	3241 3142 2341 2143 1243	5
ddDd	3412	3412 3421 3142 1342 1243	5
ddDd	3421	3421 3241 3142 2341 2143 1243	6
ddDd	4123	4123 4132 1423 1432 1243	5
ddDd	4132	4132 1432 1342 1243	4
ddDd	4213	4213 4231 4123 4132 2413 2431 2143 1243	8
ddDd	4231	4231 4132 2431 2341 2143 1243	6
ddDd	4312	4312 4321 4132 3412 3421 3142 1342 1243	8
ddDd	4321	4321 4231 4132 3421 3241 3142 2341 2143 1243	9
ddDD	1234	1234 1243	2
ddDD	1243	1243	1
ddDD	1324	1324 1342 1234 1243	4
ddDD	1342	1342 1243	2
ddDD	1423	1423 1432 1243	3
ddDD	1432	1432 1342 1243	3
ddDD	2134	2134 2143 1234 1243	4
ddDD	2143	2143 1243	2
ddDD	2314	2314 2341 2134 2143 1234 1243	6
ddDD	2341	2341 2143 1243	3
ddDD	2413	2413 2431 2143 1243	4
ddDD	2431	2431 2341 2143 1243	4
ddDD	3124	3124 3142 1324 1342 1234 1243	6
ddDD	3142	3142 1342 1243	3
ddDD	3214	3214 3241 3124 3142 2314 2341 2134 2143 1234 1243	10
ddDD	3241	3241 3142 2341 2143 1243	5
ddDD	3412	3412 3421 3142 1342 1243	5
ddDD	3421	3421 3241 3142 2341 2143 1243	6
ddDD	4123	4123 4132 1423 1432 1243	5
ddDD	4132	4132 1432 1342 1243	4
ddDD	4213	4213 4231 4123 4132 2413 2431 2143 1243	8

Continued on next page

<i>D</i> Sequence	Input	Outputs	Total
ddDD	4231	4231 4132 2431 2341 2143 1243	6
ddDD	4312	4312 4321 4132 3412 3421 3142 1342 1243	8
ddDD	4321	4321 4231 4132 3421 3241 3142 2341 2143 1243	9
dDdd	1234	1234 1324 1423	3
dDdd	1243	1243 1234 1423	3
dDdd	1324	1324 1423	2
dDdd	1342	1342 1324 1432 1423	4
dDdd	1423	1423	1
dDdd	1432	1432 1423	2
dDdd	2134	2134 2314 2413 1234 1324 1423	6
dDdd	2143	2143 2134 2413 1243 1234 1423	6
dDdd	2314	2314 2413 1324 1423	4
dDdd	2341	2341 2314 2431 2413 1423	5
dDdd	2413	2413 1423	2
dDdd	2431	2431 2413 1423	3
dDdd	3124	3124 3214 3412 1324 1423	5
dDdd	3142	3142 3124 3412 1342 1324 1432 1423	7
dDdd	3214	3214 3412 2314 2413 1324 1423	6
dDdd	3241	3241 3214 3421 3412 2341 2314 2431 2413 1423	9
dDdd	3412	3412 1432 1423	3
dDdd	3421	3421 3412 2431 2413 1423	5
dDdd	4123	4123 4213 4312 1423	4
dDdd	4132	4132 4123 4312 1432 1423	5
dDdd	4213	4213 4312 2413 1423	4
dDdd	4231	4231 4213 4321 4312 2431 2413 1423	7
dDdd	4312	4312 3412 1432 1423	4
dDdd	4321	4321 4312 3421 3412 2431 2413 1423	7
dDdD	1234	1234 1324 1423	3
dDdD	1243	1243 1234 1423	3
dDdD	1324	1324 1423	2
dDdD	1342	1342 1324 1432 1423	4
dDdD	1423	1423	1
dDdD	1432	1432 1423	2
dDdD	2134	2134 2314 2413 1234 1324 1423	6
dDdD	2143	2143 2134 2413 1243 1234 1423	6
dDdD	2314	2314 2413 1324 1423	4

Continued on next page

<i>D</i> Sequence	Input	Outputs	Total
dDdD	2341	2341 2314 2431 2413 1423	5
dDdD	2413	2413 1423	2
dDdD	2431	2431 2413 1423	3
dDdD	3124	3124 3214 3412 1324 1423	5
dDdD	3142	3142 3124 3412 1342 1324 1432 1423	7
dDdD	3214	3214 3412 2314 2413 1324 1423	6
dDdD	3241	3241 3214 3421 3412 2341 2314 2431 2413 1423	9
dDdD	3412	3412 1432 1423	3
dDdD	3421	3421 3412 2431 2413 1423	5
dDdD	4123	4123 4213 4312 1423	4
dDdD	4132	4132 4123 4312 1432 1423	5
dDdD	4213	4213 4312 2413 1423	4
dDdD	4231	4231 4213 4321 4312 2431 2413 1423	7
dDdD	4312	4312 3412 1432 1423	4
dDdD	4321	4321 4312 3421 3412 2431 2413 1423	7
dDDd	1234	1234 1243 1324 1342 1432	5
dDDd	1243	1243 1423 1432	3
dDDd	1324	1324 1342 1432	3
dDDd	1342	1342 1432	2
dDDd	1423	1423 1432	2
dDDd	1432	1432	1
dDDd	2134	2134 2143 2314 2341 2431 1234 1243 1324 1342 1432	10
dDDd	2143	2143 2413 2431 1243 1423 1432	6
dDDd	2314	2314 2341 2431 1324 1342 1432	6
dDDd	2341	2341 2431 1432	3
dDDd	2413	2413 2431 1423 1432	4
dDDd	2431	2431 1432	2
dDDd	3124	3124 3142 3214 3241 3421 1324 1342 1432	8
dDDd	3142	3142 3412 3421 1342 1432	5
dDDd	3214	3214 3241 3421 2314 2341 2431 1324 1342 1432	9
dDDd	3241	3241 3421 2341 2431 1432	5
dDDd	3412	3412 3421 1432	3
dDDd	3421	3421 2431 1432	3
dDDd	4123	4123 4132 4213 4231 4321 1423 1432	7
dDDd	4132	4132 4312 4321 1432	4

Continued on next page

D Sequence	Input	Outputs	Total
dDDd	4213	4213 4231 4321 2413 2431 1423 1432	7
dDDd	4231	4231 4321 2431 1432	4
dDDd	4312	4312 4321 3412 3421 1432	5
dDDd	4321	4321 3421 2431 1432	4
dDDD	1234	1234 1243 1324 1342 1432	5
dDDD	1243	1243 1423 1432	3
dDDD	1324	1324 1342 1432	3
dDDD	1342	1342 1432	2
dDDD	1423	1423 1432	2
dDDD	1432	1432	1
dDDD	2134	2134 2143 2314 2341 2431 1234 1243 1324 1342 1432	10
dDDD	2143	2143 2413 2431 1243 1423 1432	6
dDDD	2314	2314 2341 2431 1324 1342 1432	6
dDDD	2341	2341 2431 1432	3
dDDD	2413	2413 2431 1423 1432	4
dDDD	2431	2431 1432	2
dDDD	3124	3124 3142 3214 3241 3421 1324 1342 1432	8
dDDD	3142	3142 3412 3421 1342 1432	5
dDDD	3214	3214 3241 3421 2314 2341 2431 1324 1342 1432	9
dDDD	3241	3241 3421 2341 2431 1432	5
dDDD	3412	3412 3421 1432	3
dDDD	3421	3421 2431 1432	3
dDDD	4123	4123 4132 4213 4231 4321 1423 1432	7
dDDD	4132	4132 4312 4321 1432	4
dDDD	4213	4213 4231 4321 2413 2431 1423 1432	7
dDDD	4231	4231 4321 2431 1432	4
dDDD	4312	4312 4321 3412 3421 1432	5
dDDD	4321	4321 3421 2431 1432	4
Dddd	1234	1234 2134 3124 4123	4
Dddd	1243	1243 1234 2143 2134 4123	5
Dddd	1324	1324 1234 3124 4123	4
Dddd	1342	1342 1324 1234 3142 3124 4132 4123	7
Dddd	1423	1423 1243 1234 4123	4
Dddd	1432	1432 1423 1342 1324 1234 4132 4123	7
Dddd	2134	2134 3124 4123	3

Continued on next page

D Sequence	Input	Outputs	Total
Dddd	2143	2143 2134 4123	3
Dddd	2314	2314 2134 3214 3124 4123	5
Dddd	2341	2341 2314 2134 3241 3214 3124 4231 4213 4123	9
Dddd	2413	2413 2143 2134 4213 4123	5
Dddd	2431	2431 2413 2341 2314 2134 4231 4213 4123	8
Dddd	3124	3124 4123	2
Dddd	3142	3142 3124 4132 4123	4
Dddd	3214	3214 3124 4123	3
Dddd	3241	3241 3214 3124 4231 4213 4123	6
Dddd	3412	3412 3142 3124 4312 4132 4123	6
Dddd	3421	3421 3412 3241 3214 3124 4321 4312 4231 4213 4123	10
Dddd	4123	4123	1
Dddd	4132	4132 4123	2
Dddd	4213	4213 4123	2
Dddd	4231	4231 4213 4123	3
Dddd	4312	4312 4132 4123	3
Dddd	4321	4321 4312 4231 4213 4123	5
DddD	1234	1234 2134 3124 4123	4
DddD	1243	1243 1234 2143 2134 4123	5
DddD	1324	1324 1234 3124 4123	4
DddD	1342	1342 1324 1234 3142 3124 4132 4123	7
DddD	1423	1423 1243 1234 4123	4
DddD	1432	1432 1423 1342 1324 1234 4132 4123	7
DddD	2134	2134 3124 4123	3
DddD	2143	2143 2134 4123	3
DddD	2314	2314 2134 3214 3124 4123	5
DddD	2341	2341 2314 2134 3241 3214 3124 4231 4213 4123	9
DddD	2413	2413 2143 2134 4213 4123	5
DddD	2431	2431 2413 2341 2314 2134 4231 4213 4123	8
DddD	3124	3124 4123	2
DddD	3142	3142 3124 4132 4123	4
DddD	3214	3214 3124 4123	3
DddD	3241	3241 3214 3124 4231 4213 4123	6
DddD	3412	3412 3142 3124 4312 4132 4123	6
DddD	3421	3421 3412 3241 3214 3124 4321 4312 4231 4213 4123	10

Continued on next page

<i>D</i> Sequence	Input	Outputs	Total
DddD	4123	4123	1
DddD	4132	4132 4123	2
DddD	4213	4213 4123	2
DddD	4231	4231 4213 4123	3
DddD	4312	4312 4132 4123	3
DddD	4321	4321 4312 4231 4213 4123	5
DdDd	1234	1234 1243 2134 2143 3124 3142 4132	7
DdDd	1243	1243 2143 4123 4132	4
DdDd	1324	1324 1342 1234 1243 3124 3142 4132	7
DdDd	1342	1342 1243 3142 4132	4
DdDd	1423	1423 1432 1243 4123 4132	5
DdDd	1432	1432 1342 1243 4132	4
DdDd	2134	2134 2143 3124 3142 4132	5
DdDd	2143	2143 4123 4132	3
DdDd	2314	2314 2341 2134 2143 3214 3241 3124 3142 4132	9
DdDd	2341	2341 2143 3241 3142 4231 4132	6
DdDd	2413	2413 2431 2143 4213 4231 4123 4132	7
DdDd	2431	2431 2341 2143 4231 4132	5
DdDd	3124	3124 3142 4132	3
DdDd	3142	3142 4132	2
DdDd	3214	3214 3241 3124 3142 4132	5
DdDd	3241	3241 3142 4231 4132	4
DdDd	3412	3412 3421 3142 4312 4321 4132	6
DdDd	3421	3421 3241 3142 4321 4231 4132	6
DdDd	4123	4123 4132	2
DdDd	4132	4132	1
DdDd	4213	4213 4231 4123 4132	4
DdDd	4231	4231 4132	2
DdDd	4312	4312 4321 4132	3
DdDd	4321	4321 4231 4132	3
DdDD	1234	1234 1243 2134 2143 3124 3142 4132	7
DdDD	1243	1243 2143 4123 4132	4
DdDD	1324	1324 1342 1234 1243 3124 3142 4132	7
DdDD	1342	1342 1243 3142 4132	4
DdDD	1423	1423 1432 1243 4123 4132	5
DdDD	1432	1432 1342 1243 4132	4

Continued on next page

<i>D</i> Sequence	Input	Outputs	Total
DdDD	2134	2134 2143 3124 3142 4132	5
DdDD	2143	2143 4123 4132	3
DdDD	2314	2314 2341 2134 2143 3214 3241 3124 3142 4132	9
DdDD	2341	2341 2143 3241 3142 4231 4132	6
DdDD	2413	2413 2431 2143 4213 4231 4123 4132	7
DdDD	2431	2431 2341 2143 4231 4132	5
DdDD	3124	3124 3142 4132	3
DdDD	3142	3142 4132	2
DdDD	3214	3214 3241 3124 3142 4132	5
DdDD	3241	3241 3142 4231 4132	4
DdDD	3412	3412 3421 3142 4312 4321 4132	6
DdDD	3421	3421 3241 3142 4321 4231 4132	6
DdDD	4123	4123 4132	2
DdDD	4132	4132	1
DdDD	4213	4213 4231 4123 4132	4
DdDD	4231	4231 4132	2
DdDD	4312	4312 4321 4132	3
DdDD	4321	4321 4231 4132	3
DDdd	1234	1234 1324 1423 2134 2314 2413 3214 3412 4312	9
DDdd	1243	1243 1234 1423 2143 2134 2413 4213 4312	8
DDdd	1324	1324 1423 3124 3214 3412 4312	6
DDdd	1342	1342 1324 1432 1423 3142 3124 3412 4312	8
DDdd	1423	1423 4123 4213 4312	4
DDdd	1432	1432 1423 4132 4123 4312	5
DDdd	2134	2134 2314 2413 3214 3412 4312	6
DDdd	2143	2143 2134 2413 4213 4312	5
DDdd	2314	2314 2413 3214 3412 4312	5
DDdd	2341	2341 2314 2431 2413 3241 3214 3421 3412 4321 4312	10
DDdd	2413	2413 4213 4312	3
DDdd	2431	2431 2413 4231 4213 4321 4312	6
DDdd	3124	3124 3214 3412 4312	4
DDdd	3142	3142 3124 3412 4312	4
DDdd	3214	3214 3412 4312	3
DDdd	3241	3241 3214 3421 3412 4321 4312	6
DDdd	3412	3412 4312	2

Continued on next page

$\mathcal{D}$ Sequence	Input	Outputs	Total
DDdd	3421	3421 3412 4321 4312	4
DDdd	4123	4123 4213 4312	3
DDdd	4132	4132 4123 4312	3
DDdd	4213	4213 4312	2
DDdd	4231	4231 4213 4321 4312	4
DDdd	4312	4312	1
DDdd	4321	4321 4312	2
DDdD	1234	1234 1324 1423 2134 2314 2413 3214 3412 4312	9
DDdD	1243	1243 1234 1423 2143 2134 2413 4213 4312	8
DDdD	1324	1324 1423 3124 3214 3412 4312	6
DDdD	1342	1342 1324 1432 1423 3142 3124 3412 4312	8
DDdD	1423	1423 4123 4213 4312	4
DDdD	1432	1432 1423 4132 4123 4312	5
DDdD	2134	2134 2314 2413 3214 3412 4312	6
DDdD	2143	2143 2134 2413 4213 4312	5
DDdD	2314	2314 2413 3214 3412 4312	5
DDdD	2341	2341 2314 2431 2413 3241 3214 3421 3412 4321 4312	10
DDdD	2413	2413 4213 4312	3
DDdD	2431	2431 2413 4231 4213 4321 4312	6
DDdD	3124	3124 3214 3412 4312	4
DDdD	3142	3142 3124 3412 4312	4
DDdD	3214	3214 3412 4312	3
DDdD	3241	3241 3214 3421 3412 4321 4312	6
DDdD	3412	3412 4312	2
DDdD	3421	3421 3412 4321 4312	4
DDdD	4123	4123 4213 4312	3
DDdD	4132	4132 4123 4312	3
DDdD	4213	4213 4312	2
DDdD	4231	4231 4213 4321 4312	4
DDdD	4312	4312	1
DDdD	4321	4321 4312	2
DDDd	1234	1234 1243 1324 1342 1432 2134 2143 2314 2341 2431 3214 3241 3421 4321	14
DDDd	1243	1243 1423 1432 2143 2413 2431 4213 4231 4321	9
DDDd	1324	1324 1342 1432 3124 3142 3214 3241 3421 4321	9

Continued on next page

D Sequence	Input	Outputs	Total
DDDd	1342	1342 1432 3142 3412 3421 4312 4321	7
DDDd	1423	1423 1432 4123 4132 4213 4231 4321	7
DDDd	1432	1432 4132 4312 4321	4
DDDd	2134	2134 2143 2314 2341 2431 3214 3241 3421 4321	9
DDDd	2143	2143 2413 2431 4213 4231 4321	6
DDDd	2314	2314 2341 2431 3214 3241 3421 4321	7
DDDd	2341	2341 2431 3241 3421 4321	5
DDDd	2413	2413 2431 4213 4231 4321	5
DDDd	2431	2431 4231 4321	3
DDDd	3124	3124 3142 3214 3241 3421 4321	6
DDDd	3142	3142 3412 3421 4312 4321	5
DDDd	3214	3214 3241 3421 4321	4
DDDd	3241	3241 3421 4321	3
DDDd	3412	3412 3421 4312 4321	4
DDDd	3421	3421 4321	2
DDDd	4123	4123 4132 4213 4231 4321	5
DDDd	4132	4132 4312 4321	3
DDDd	4213	4213 4231 4321	3
DDDd	4231	4231 4321	2
DDDd	4312	4312 4321	2
DDDd	4321	4321	1
DDDD	1234	1234 1243 1324 1342 1432 2134 2143 2314 2341 2431 3214 3241 3421 4321	14
DDDD	1243	1243 1423 1432 2143 2413 2431 4213 4231 4321	9
DDDD	1324	1324 1342 1432 3124 3142 3214 3241 3421 4321	9
DDDD	1342	1342 1432 3142 3412 3421 4312 4321	7
DDDD	1423	1423 1432 4123 4132 4213 4231 4321	7
DDDD	1432	1432 4132 4312 4321	4
DDDD	2134	2134 2143 2314 2341 2431 3214 3241 3421 4321	9
DDDD	2143	2143 2413 2431 4213 4231 4321	6
DDDD	2314	2314 2341 2431 3214 3241 3421 4321	7
DDDD	2341	2341 2431 3241 3421 4321	5
DDDD	2413	2413 2431 4213 4231 4321	5
DDDD	2431	2431 4231 4321	3
DDDD	3124	3124 3142 3214 3241 3421 4321	6
DDDD	3142	3142 3412 3421 4312 4321	5

Continued on next page

<i>D</i> Sequence	Input	Outputs	Total
DDDD	3214	3214 3241 3421 4321	4
DDDD	3241	3241 3421 4321	3
DDDD	3412	3412 3421 4312 4321	4
DDDD	3421	3421 4321	2
DDDD	4123	4123 4132 4213 4231 4321	5
DDDD	4132	4132 4312 4321	3
DDDD	4213	4213 4231 4321	3
DDDD	4231	4231 4321	2
DDDD	4312	4312 4321	2
DDDD	4321	4321	1
Total			1824

B.2.5 The Number of Allowable Pairs with Fixed Inputs and Fixed *DeleteMin-DeleteMax* Sequences

Table 28: The Number of Allowable Pairs with Fixed Inputs and Fixed *DeleteMin-DeleteMax* Sequences

<i>D</i> Sequence	Outputs	Total
dddd	1234 1243 1324 1342 1423 1432 2134 2143 2314 2341 2413 2431 3124 3142 3214 3241 3412 3421 4123 4132 4213 4231 4312 4321	24
dddD	1234 1243 1324 1342 1423 1432 2134 2143 2314 2341 2413 2431 3124 3142 3214 3241 3412 3421 4123 4132 4213 4231 4312 4321	24
ddDd	1234 1243 1324 1342 1423 1432 2134 2143 2314 2341 2413 2431 3124 3142 3214 3241 3412 3421 4123 4132 4213 4231 4312 4321	24
ddDD	1234 1243 1324 1342 1423 1432 2134 2143 2314 2341 2413 2431 3124 3142 3214 3241 3412 3421 4123 4132 4213 4231 4312 4321	24
dDdd	1234 1324 1423 1243 1342 1432 2134 2314 2413 2143 2341 2431 3124 3214 3412 3142 3241 3421 4123 4213 4312 4132 4231 4321	24
dDdD	1234 1324 1423 1243 1342 1432 2134 2314 2413 2143 2341 2431 3124 3214 3412 3142 3241 3421 4123 4213 4312 4132 4231 4321	24
dDDd	1234 1243 1324 1342 1432 1423 2134 2143 2314 2341 2431 2413 3124 3142 3214 3241 3421 3412 4123 4132 4213 4231 4321 4312	24
dDDD	1234 1243 1324 1342 1432 1423 2134 2143 2314 2341 2431 2413 3124 3142 3214 3241 3421 3412 4123 4132 4213 4231 4321 4312	24

Continued on next page

D Sequence	Outputs	Total
Dddd	1234 2134 3124 4123 1243 2143 1324 1342 3142 4132 1423 1432 2314 3214 2341 3241 4231 4213 2413 2431 3412 4312 3421 4321	24
DddD	1234 2134 3124 4123 1243 2143 1324 1342 3142 4132 1423 1432 2314 3214 2341 3241 4231 4213 2413 2431 3412 4312 3421 4321	24
DdDd	1234 1243 2134 2143 3124 3142 4132 4123 1324 1342 1423 1432 2314 2341 3214 3241 4231 2413 2431 4213 3412 3421 4312 4321	24
DdDD	1234 1243 2134 2143 3124 3142 4132 4123 1324 1342 1423 1432 2314 2341 3214 3241 4231 2413 2431 4213 3412 3421 4312 4321	24
DDdd	1234 1324 1423 2134 2314 2413 3214 3412 4312 1243 2143 4213 3124 1342 1432 3142 4123 4132 2341 2431 3241 3421 4321 4231	24
DDdD	1234 1324 1423 2134 2314 2413 3214 3412 4312 1243 2143 4213 3124 1342 1432 3142 4123 4132 2341 2431 3241 3421 4321 4231	24
DDDd	1234 1243 1324 1342 1432 2134 2143 2314 2341 2431 3214 3241 3421 4321 1423 2413 4213 4231 3124 3142 3412 4312 4123 4132	24
DDDD	1234 1243 1324 1342 1432 2134 2143 2314 2341 2431 3214 3241 3421 4321 1423 2413 4213 4231 3124 3142 3412 4312 4123 4132	24
Total		384

B.2.6 Output Generation with Fixed *Insert-Delete* Bracket Sequences and *DeleteMin-DeleteMax* Dd Sequences

Table 29: Output Generation with Fixed *Insert-Delete* Bracket Sequences and *DeleteMin-DeleteMax* Dd Sequences

<i>D</i> Sequence	<i>B</i> Sequence	Outputs	Total
dddd	10101010	1234 1243 1324 1342 1423 1432 2134 2143 2314 2341 2413 2431 3124 3142 3214 3241 3412 3421 4123 4132 4213 4231 4312 4321	24
dddd	10101100	1234 1324 1423 2134 2314 2413 3124 3214 3412 4123 4213 4312	12
ddcd	10110010	1234 1243 1342 2134 2143 2341 3124 3142 3241 4123 4132 4231	12
dddd	10110100	1234 1324 2134 2314 3124 3214 4123 4213	8
dddd	10111000	1234 2134 3124 4123	4
dddd	11001010	1234 1243 1324 1342 1423 1432 2314 2341 2413 2431 3412 3421	12
dddd	11001100	1234 1324 1423 2314 2413 3412	6
dddd	11010010	1234 1243 1342 2134 2341 2143 3142 3241	8
dddd	11010100	1234 1324 2134 2314 3124 3214	6
dddd	11011000	1234 2134 3124	3
dddd	11100010	1234 1243 1342 2341	4
dddd	11100100	1234 1324 2314	3
dddd	11101000	1234 2134	2
dddd	11110000	1234	1
dddD	10101010	1234 1243 1324 1342 1423 1432 2134 2143 2314 2341 2413 2431 3124 3142 3214 3241 3412 3421 4123 4132 4213 4231 4312 4321	24
dddD	10101100	1234 1324 1423 2134 2314 2413 3124 3214 3412 4123 4213 4312	12

Continued on next page

<i>D</i> Sequence	<i>B</i> Sequence	Outputs	Total
dddD	10110010	1234 1243 1342 2134 2143 2341 3124 3142 3241 4123 4132 4231	12
dddD	10110100	1234 1324 2134 2314 3124 3214 4123 4213	8
dddD	10111000	1234 2134 3124 4123	4
dddD	11001010	1234 1243 1324 1342 1423 1432 2314 2341 2413 2431 3412 3421	12
dddD	11001100	1234 1324 1423 2314 2413 3412	6
dddD	11010010	1234 1243 1342 2134 2341 2143 3142 3241	8
dddD	11010100	1234 1324 2134 2314 3124 3214	6
dddD	11011000	1234 2134 3124	3
dddD	11100010	1234 1243 1342 2341	4
dddD	11100100	1234 1324 2314	3
dddD	11101000	1234 2134	2
dddD	11110000	1234	1
ddDd	10101010	1234 1243 1324 1342 1423 1432 2134 2143 2314 2341 2413 2431 3124 3142 3214 3241 3412 3421 4123 4132 4213 4231 4312 4321	24
ddDd	10101100	1243 1342 1432 2143 2341 2431 3142 3241 3421 4132 4231 4321	12
ddDd	10110010	1234 1243 1342 2134 2143 2341 3124 3142 3241 4123 4132 4231	12
ddDd	10110100	1243 1342 2143 2341 3142 3241 4132 4231	8
ddDd	10111000	1243 2143 3142 4132	4
ddDd	11001010	1234 1243 1324 1342 1423 1432 2314 2341 2413 2431 3412 3421	12
ddDd	11001100	1243 1342 1432 2341 2431 3421	6
ddDd	11010010	1234 1243 1342 2134 2341 2143 3142 3241	8
ddDd	11010100	1243 1342 2143 2341 3142 3241	6
ddDd	11011000	1243 2143 3142	3
ddDd	11100010	1234 1243 1342 2341	4
ddDd	11100100	1243 1342 2341	3
ddDd	11101000	1243 2143	2
ddDd	11110000	1243	1
ddDD	10101010	1234 1243 1324 1342 1423 1432 2134 2143 2314 2341 2413 2431 3124 3142 3214 3241 3412 3421 4123 4132 4213 4231 4312 4321	24

Continued on next page

D Sequence	B Sequence	Outputs	Total
ddDD	10101100	1243 1342 1432 2143 2341 2431 3142 3241 3421 4132 4231 4321	12
ddDD	10110010	1234 1243 1342 2134 2143 2341 3124 3142 3241 4123 4132 4231	12
ddDD	10110100	1243 1342 2143 2341 3142 3241 4132 4231	8
ddDD	10111000	1243 2143 3142 4132	4
ddDD	11001010	1234 1243 1324 1342 1423 1432 2314 2341 2413 2431 3412 3421	12
ddDD	11001100	1243 1342 1432 2341 2431 3421	6
ddDD	11010010	1234 1243 1342 2134 2341 2143 3142 3241	8
ddDD	11010100	1243 1342 2143 2341 3142 3241	6
ddDD	11011000	1243 2143 3142	3
ddDD	11100010	1234 1243 1342 2341	4
ddDD	11100100	1243 1342 2341	3
ddDD	11101000	1243 2143	2
ddDD	11110000	1243	1
dDdd	10101010	1234 1243 1324 1342 1423 1432 2134 2143 2314 2341 2413 2431 3124 3142 3214 3241 3412 3421 4123 4132 4213 4231 4312 4321	24
dDdd	10101100	1234 1324 1423 2134 2314 2413 3124 3214 3412 4123 4213 4312	12
dDdd	10110010	1324 1423 1432 2314 2413 2431 3214 3412 3421 4213 4312 4321	12
dDdd	10110100	1324 1423 2314 2413 3214 3412 4213 4312	8
dDdd	10111000	1423 2413 3412 4312	4
dDdd	11001010	1234 1243 1324 1342 1423 1432 2314 2341 2413 2431 3412 3421	12
dDdd	11001100	1234 1324 1423 2314 2413 3412	6
dDdd	11010010	1324 1423 1432 2314 2431 2413 3412 3421	8
dDdd	11010100	1324 1423 2314 2413 3412	5
dDdd	11011000	1423 2413 3412	3
dDdd	11100010	1324 1423 1432 2431	4
dDdd	11100100	1324 1423 2413	3
dDdd	11101000	1423 2413	2
dDdd	11110000	1423	1

Continued on next page

<i>D</i> Sequence	<i>B</i> Sequence	Outputs	Total
dDdD	10101010	1234 1243 1324 1342 1423 1432 2134 2143 2314 2341 2413 2431 3124 3142 3214 3241 3412 3421 4123 4132 4213 4231 4312 4321	24
dDdD	10101100	1234 1324 1423 2134 2314 2413 3124 3214 3412 4123 4213 4312	12
dDdD	10110010	1324 1423 1432 2314 2413 2431 3214 3412 3421 4213 4312 4321	12
dDdD	10110100	1324 1423 2314 2413 3214 3412 4213 4312	8
dDdD	10111000	1423 2413 3412 4312	4
dDdD	11001010	1234 1243 1324 1342 1423 1432 2314 2341 2413 2431 3412 3421	12
dDdD	11001100	1234 1324 1423 2314 2413 3412	6
dDdD	11010010	1324 1423 1432 2314 2431 2413 3412 3421	8
dDdD	11010100	1324 1423 2314 2413 3412	5
dDdD	11011000	1423 2413 3412	3
dDdD	11100010	1324 1423 1432 2431	4
dDdD	11100100	1324 1423 2413	3
dDdD	11101000	1423 2413	2
dDdD	11110000	1423	1
dDDd	10101010	1234 1243 1324 1342 1423 1432 2134 2143 2314 2341 2413 2431 3124 3142 3214 3241 3412 3421 4123 4132 4213 4231 4312 4321	24
dDDd	10101100	1243 1342 1432 2143 2341 2431 3142 3241 3421 4132 4231 4321	12
dDDd	10110010	1324 1423 1432 2314 2413 2431 3214 3412 3421 4213 4312 4321	12
dDDd	10110100	1342 1432 2341 2431 3241 3421 4231 4321	8
dDDd	10111000	1432 2431 3421 4321	4
dDDd	11001010	1234 1243 1324 1342 1423 1432 2314 2341 2413 2431 3412 3421	12
dDDd	11001100	1243 1342 1432 2341 2431 3421	6
dDDd	11010010	1324 1423 1432 2314 2431 2413 3412 3421	8
dDDd	11010100	1342 1432 2341 2431 3421	5
dDDd	11011000	1432 2431 3421	3
dDDd	11100010	1324 1423 1432 2431	4
dDDd	11100100	1342 1432 2431	3

Continued on next page

$\mathcal{D}$ Sequence	$\mathcal{B}$ Sequence	Outputs	Total
dDDd	11101000	1432 2431	2
dDDd	11110000	1432	1
dDDD	10101010	1234 1243 1324 1342 1423 1432 2134 2143 2314 2341 2413 2431 3124 3142 3214 3241 3412 3421 4123 4132 4213 4231 4312 4321	24
dDDD	10101100	1243 1342 1432 2143 2341 2431 3142 3241 3421 4132 4231 4321	12
dDDD	10110010	1324 1423 1432 2314 2413 2431 3214 3412 3421 4213 4312 4321	12
dDDD	10110100	1342 1432 2341 2431 3241 3421 4231 4321	8
dDDD	10111000	1432 2431 3421 4321	4
dDDD	11001010	1234 1243 1324 1342 1423 1432 2314 2341 2413 2431 3412 3421	12
dDDD	11001100	1243 1342 1432 2341 2431 3421	6
dDDD	11010010	1324 1423 1432 2314 2431 2413 3412 3421	8
dDDD	11010100	1342 1432 2341 2431 3421	5
dDDD	11011000	1432 2431 3421	3
dDDD	11100010	1324 1423 1432 2431	4
dDDD	11100100	1342 1432 2431	3
dDDD	11101000	1432 2431	2
dDDD	11110000	1432	1
Dddd	10101010	1234 1243 1324 1342 1423 1432 2134 2143 2314 2341 2413 2431 3124 3142 3214 3241 3412 3421 4123 4132 4213 4231 4312 4321	24
Dddd	10101100	1234 1324 1423 2134 2314 2413 3124 3214 3412 4123 4213 4312	12
Dddd	10110010	1234 1243 1342 2134 2143 2341 3124 3142 3241 4123 4132 4231	12
Dddd	10110100	1234 1324 2134 2314 3124 3214 4123 4213	8
Dddd	10111000	1234 2134 3124 4123	4
Dddd	11001010	2134 2143 3124 3142 4123 4132 3214 3241 4213 4231 4312 4321	12
Dddd	11001100	2134 3124 4123 3214 4213 4312	6
Dddd	11010010	2134 2143 3124 3142 4123 4132 3241 4231	8
Dddd	11010100	2134 3124 4123 3214 4213	5
Dddd	11011000	2134 3124 4123	3

Continued on next page

<i>D</i> Sequence	<i>B</i> Sequence	Outputs	Total
Dddd	11100010	3124 4123 4132 4231	4
Dddd	11100100	3124 4123 4213	3
Dddd	11101000	3124 4123	2
Dddd	11110000	4123	1
DddD	10101010	1234 1243 1324 1342 1423 1432 2134 2143 2314 2341 2413 2431 3124 3142 3214 3241 3412 3421 4123 4132 4213 4231 4312 4321	24
DddD	10101100	1234 1324 1423 2134 2314 2413 3124 3214 3412 4123 4213 4312	12
DddD	10110010	1234 1243 1342 2134 2143 2341 3124 3142 3241 4123 4132 4231	12
DddD	10110100	1234 1324 2134 2314 3124 3214 4123 4213	8
DddD	10111000	1234 2134 3124 4123	4
DddD	11001010	2134 2143 3124 3142 4123 4132 3214 3241 4213 4231 4312 4321	12
DddD	11001100	2134 3124 4123 3214 4213 4312	6
DddD	11010010	2134 2143 3124 3142 4123 4132 3241 4231	8
DddD	11010100	2134 3124 4123 3214 4213	5
DddD	11011000	2134 3124 4123	3
DddD	11100010	3124 4123 4132 4231	4
DddD	11100100	3124 4123 4213	3
DddD	11101000	3124 4123	2
DddD	11110000	4123	1
DdDd	10101010	1234 1243 1324 1342 1423 1432 2134 2143 2314 2341 2413 2431 3124 3142 3214 3241 3412 3421 4123 4132 4213 4231 4312 4321	24
DdDd	10101100	1243 1342 1432 2143 2341 2431 3142 3241 3421 4132 4231 4321	12
DdDd	10110010	1234 1243 1342 2134 2143 2341 3124 3142 3241 4123 4132 4231	12
DdDd	10110100	1243 1342 2143 2341 3142 3241 4132 4231	8
DdDd	10111000	1243 2143 3142 4132	4
DdDd	11001010	2134 2143 3124 3142 4123 4132 3214 3241 4213 4231 4312 4321	12
DdDd	11001100	2143 3142 4132 3241 4231 4321	6
DdDd	11010010	2134 2143 3124 3142 4123 4132 3241 4231	8

Continued on next page

<i>D</i> Sequence	<i>B</i> Sequence	Outputs	Total
DdDd	11010100	2143 3142 4132 3241 4231	5
DdDd	11011000	2143 3142 4132	3
DdDd	11100010	3124 4123 4132 4231	4
DdDd	11100100	3142 4132 4231	3
DdDd	11101000	3142 4132	2
DdDd	11110000	4132	1
DdDD	10101010	1234 1243 1324 1342 1423 1432 2134 2143 2314 2341 2413 2431 3124 3142 3214 3241 3412 3421 4123 4132 4213 4231 4312 4321	24
DdDD	10101100	1243 1342 1432 2143 2341 2431 3142 3241 3421 4132 4231 4321	12
DdDD	10110010	1234 1243 1342 2134 2143 2341 3124 3142 3241 4123 4132 4231	12
DdDD	10110100	1243 1342 2143 2341 3142 3241 4132 4231	8
DdDD	10111000	1243 2143 3142 4132	4
DdDD	11001010	2134 2143 3124 3142 4123 4132 3214 3241 4213 4231 4312 4321	12
DdDD	11001100	2143 3142 4132 3241 4231 4321	6
DdDD	11010010	2134 2143 3124 3142 4123 4132 3241 4231	8
DdDD	11010100	2143 3142 4132 3241 4231	5
DdDD	11011000	2143 3142 4132	3
DdDD	11100010	3124 4123 4132 4231	4
DdDD	11100100	3142 4132 4231	3
DdDD	11101000	3142 4132	2
DdDD	11110000	4132	1
DDdd	10101010	1234 1243 1324 1342 1423 1432 2134 2143 2314 2341 2413 2431 3124 3142 3214 3241 3412 3421 4123 4132 4213 4231 4312 4321	24
DDdd	10101100	1234 1324 1423 2134 2314 2413 3124 3214 3412 4123 4213 4312	12
DDdd	10110010	1324 1423 1432 2314 2413 2431 3214 3412 3421 4213 4312 4321	12
DDdd	10110100	1324 1423 2314 2413 3214 3412 4213 4312	8
DDdd	10111000	1423 2413 3412 4312	4
DDdd	11001010	2134 2143 3124 3142 4123 4132 3214 3241 4213 4231 4312 4321	12

Continued on next page

<i>D</i> Sequence	<i>B</i> Sequence	Outputs	Total
DDdd	11001100	2134 3124 4123 3214 4213 4312	6
DDdd	11010010	2314 2413 3214 3412 4213 4312 3421 4321	8
DDdd	11010100	2314 2413 3214 3412 4213 4312	6
DDdd	11011000	2413 3412 4312	3
DDdd	11100010	3214 4213 4312 4321	4
DDdd	11100100	3214 4213 4312	3
DDdd	11101000	3412 4312	2
DDdd	11110000	4312	1
DDdD	10101010	1234 1243 1324 1342 1423 1432 2134 2143 2314 2341 2413 2431 3124 3142 3214 3241 3412 3421 4123 4132 4213 4231 4312 4321	24
DDdD	10101100	1234 1324 1423 2134 2314 2413 3124 3214 3412 4123 4213 4312	12
DDdD	10110010	1324 1423 1432 2314 2413 2431 3214 3412 3421 4213 4312 4321	12
DDdD	10110100	1324 1423 2314 2413 3214 3412 4213 4312	8
DDdD	10111000	1423 2413 3412 4312	4
DDdD	11001010	2134 2143 3124 3142 4123 4132 3214 3241 4213 4231 4312 4321	12
DDdD	11001100	2134 3124 4123 3214 4213 4312	6
DDdD	11010010	2314 2413 3214 3412 4213 4312 3421 4321	8
DDdD	11010100	2314 2413 3214 3412 4213 4312	6
DDdD	11011000	2413 3412 4312	3
DDdD	11100010	3214 4213 4312 4321	4
DDdD	11100100	3214 4213 4312	3
DDdD	11101000	3412 4312	2
DDdD	11110000	4312	1
DDDd	10101010	1234 1243 1324 1342 1423 1432 2134 2143 2314 2341 2413 2431 3124 3142 3214 3241 3412 3421 4123 4132 4213 4231 4312 4321	24
DDDd	10101100	1243 1342 1432 2143 2341 2431 3142 3241 3421 4132 4231 4321	12
DDDd	10110010	1324 1423 1432 2314 2413 2431 3214 3412 3421 4213 4312 4321	12
DDDd	10110100	1342 1432 2341 2431 3241 3421 4231 4321	8
DDDd	10111000	1432 2431 3421 4321	4

Continued on next page

D Sequence	B Sequence	Output	Total
DDd	11001010	2134 2143 3124 3142 4123 4132 3214 3241 4213 4231 4312 4321	12
DDd	11001100	2143 3142 4132 3241 4231 4321	6
DDd	11010010	2314 2413 3214 3412 4213 4312 3421 4321	8
DDd	11010100	2341 2431 3241 3421 4231 4321	6
DDd	11011000	2431 3421 4321	3
DDd	11100010	3214 4213 4312 4321	4
DDd	11100100	3241 4231 4321	3
DDd	11101000	3421 4321	2
DDd	11110000	4321	1
DDDD	10101010	1234 1243 1324 1342 1423 1432 2134 2143 2314 2341 2413 2431 3124 3142 3214 3241 3412 3421 4123 4132 4213 4231 4312 4321	24
DDDD	10101100	1243 1342 1432 2143 2341 2431 3142 3241 3421 4132 4231 4321	12
DDDD	10110010	1324 1423 1432 2314 2413 2431 3214 3412 3421 4213 4312 4321	12
DDDD	10110100	1342 1432 2341 2431 3241 3421 4231 4321	8
DDDD	10111000	1432 2431 3421 4321	4
DDDD	11001010	2134 2143 3124 3142 4123 4132 3214 3241 4213 4231 4312 4321	12
DDDD	11001100	2143 3142 4132 3241 4231 4321	6
DDDD	11010010	2314 2413 3214 3412 4213 4312 3421 4321	8
DDDD	11010100	2341 2431 3241 3421 4231 4321	6
DDDD	11011000	2431 3421 4321	3
DDDD	11100010	3214 4213 4312 4321	4
DDDD	11100100	3241 4231 4321	3
DDDD	11101000	3421 4321	2
DDDD	11110000	4321	1
Total			1672

END

2 0 1 0 7 - 9 4

FIN