AVERAGE CASE SELECTION

by

Walter Cunto

Research Report CS-84-02

Department of Computer Science
University of Waterloo
Waterloo, Ontario, Canada

December 1983

Average Case Selection

by

Walter Cunto

A thesis

presented to the University of Waterloo

in partial fulfillment of the

requirements for the degree of

Doctor of Philosophy

in

Computer Science

Waterloo, Ontario, 1983

Abstract

Average Case Selection

This thesis deals with the average case complexity of selecting elements of given ranks from a totally ordered set. The basic approach is to classify different kinds of comparisons in such a way that, by knowing their relationship, tight upper and lower bounds can be achieved. This approach is applied to prove that the Floyd-Rivest selection algorithm SELECT [Floyd-Rivest 75] is optimal up to lower order terms. Selection of the minimum and the maximum of a set is shown to be of almost the same difficulty in the worst and average cases.

By the use of sampling techniques similar to those of Floyd and Rivest, we develop simple and efficient algorithms for selecting the maximum (or minimum), and an element of any given rank. The problem restricted to selecting the maximum and any element no smaller than the median is optimally solved. These results are extended to produce upper and lower bounds on the number of comparisons needed to select the minimum, the maximum and an element of any given rank.

Another application studied is the problem of finding the closest neighbour(s) of a given random element in a set. By using an adversary argument, a simple algorithm is shown to be optimal in the worst case. For the average case version of the problem, we again make use of sampling techniques to design an optimal algorithm.

Acknowledgements

I am fortunate to have been under the supervision of Dr. Ian Munro. Not only he took an active interest in my work, but guided me along my doctoral studies.

Discussions with Dr. Dorom Rotem, Gregory Rawlins and Joseph Culberson produced many helpful suggestions. This thesis was typeset using software developed by Dr. Howard Johnson.

I would like to thank Dr. Derick Wood and Dr. David Kirkpatrick, both members of my committee, for their valuable comments on my thesis draft. The improvement on Theorem 2.2 was actually suggested by Dr. Kirkpatrick.

I am grateful to the Centro de Formacion y Adiestramiento Petrolero y Petroquimico (CEPET) for providing the much appreciated financial support during my studies at University of Waterloo.

Table of Contents

CHAPTER 1

INTRODUCTION AND PRELIMINARIES

1.1. The problem and related work.

When performing statistical analyses of large amounts of data, relevant information is often provided by looking at a subset of elements with specific ranks. Such a subset, called the summary of the data and commonly defined by the median, the maximum, the minimum, and quartile elements, provides descriptive statistics of the population. With the summary, frequency histograms are elaborated to give a pictorial distribution of the data. Clearly, for this type of application, it is important to have available efficient selection algorithms. In attempting to study the complexity of problems of this type, the number of comparisons has been the usual measure of work. This measure seems to elegantly capture the inherent difficulty of the problems. The computational complexity of comparison based problems is a well established area in the theory of computing (see [Knuth 73]). Motivated by practical considerations, such as those suggested above, and spurred on by a relatively long history of surprising and, on occasion, deep results, this subarea continues to be fruitful. Hence we have both practical motivation and mathematical interest for this study on the computational complexity of selection problems. Our main interest is then, in the average case difficulty of selecting several elements of given ranks.

Because selection problems are commonly associated with a single element of given rank, we will specifically call the selection of two or more elements of given ranks *multiple selection*, denoting its complexity by $M(k_1, k_2, \cdots, k_m; n)$, where we are selecting m elements of ranks $k_1, \cdots, k_m$ from a set of n elements. Actually, this concept generalises both selection and sorting problems since, if the list of ranks is unitary ($m = 1$), the multiple selection is just a selection, and, if the list of ranks includes all ($m = n$), the problem transforms into sorting.

The multiple selection problem can be succinctly stated as follows: given a set X of n elements with an underlying total order and a list of ranks, find the elements in X of such ranks, with as few comparisons as possible.

Our measure of difficulty will be the number of pairwise comparisons among the elements of the given input. Under our model, it is assumed these are the only operations performed on the data. However, the algorithms may retain any information learned through such comparisons, and make computations on the information free of charge.

Occasionally we will be interested in the worst case. The majority of the thesis, however, deals with average case behaviour. It may be assumed that all possible relative orderings of the input are equally likely. Equivalently, it may be assumed an adversary has initially ordered the data, but the algorithms may make use of a random number generator to map it into the former case.

Although the problem of (single element) selection has been widely investigated for the worst case complexity, the average case has received less attention. The first practical selection algorithm seems to be that of C.A.R. Hoare [Hoare 61]. His algorithm FIND is an application of the procedure PARTITION used in Quicksort. In its simplest version, the algorithm randomly chooses an element called the pivot, from the current set of elements. The set is split into two, one with elements smaller and the other with elements larger than the pivot, by simply comparing all the elements with the pivot. Recursion on the subset with the element of desired rank completes the process. The algorithm is certainly easy to understand, but its analysis is not trivial. Somewhat later, Donald E. Knuth [Knuth 71] proved that the average number of comparisons performed by FIND is

$$M(k\,;\,n) \leq 2\,[(n+1)H_n - (n+3-k)H_{n-k} + 1 - (k+2)H_k + n + 3]\ ,$$

where

$$H_m = \sum_{i=1}^{m} \frac{1}{i}\ .$$

If selecting the median, this yields,

$$M(\lceil n/2 \rceil\ ,\ n) \leq 3.39n + o(n)\ .$$

This is somewhat above the totally naive estimate of $2n$, but one's intuition is restored by noting that the desired element is, at each step, more likely to fall in the larger subset than the smaller.

In 1970, Robert Floyd and Ronald Rivest developed a new approach to selection by making use of sampling techniques [Floyd-Rivest 73] (later published

in [Floyd-Rivest 75]). We will discuss their approach in much greater detail in section 3.1. The basic idea is to find, on the basis of a small sample, elements just above and just below the desired element. Having done this, the selection problem is reduced to one of size $o(n)$, upon which even a sort would be satisfactory. Their algorithm SELECT provides an upper bound on the expected number of comparisons required as

$$M(k\,;\,n) \leq n + \min(k\,,\,n-k+1) + O(n^{1/2}) \ .$$

In the particularly interesting case of median selection, they have

$$M(\lceil n/2 \rceil\,;\,n) \leq 1.5n + O(n^{1/2}) \ .$$

Floyd and Rivest observed the interrelation between two types of comparisons: those which first join two disconnected substructures and those which constitute the (a posteriori) proof that the correct element has been found. Based on the solution of a linear programming problem, a lower bound for the selection problem is also proven in [Floyd-Rivest 75]. In the case of selecting the median the best of a sequence of lower bounds achieved was

$$M(\lceil n/2 \rceil\,;\,n) \geq 1.375n \ .$$

Although the method is weak, it supplied the first, and up to this point, only nontrivial lower bound for the problem.

The problem restricted to finding elements of small fixed rank, i.e. independent of n, was first devised by David W. Matula [Matula 73]. His elegant and surprising algorithm shows that for a fixed rank k,

$$M(k\,;\,n) \leq n + k\left\lceil \log_2(k) \right\rceil (11 + \ln\ln(n))$$

(see also [Knuth 73]). A lower bound confirming Matula's conjecture that the method was near optimal was formalised by Andrew C. Yao and F. Frances Yao [Yao-Yao 82]. They showed that for a fixed rank $k \geq 2$, there exists a number N_k, such that,

$$M(k\,;\,n) \geq n + \frac{k}{2}\,(\ln\ln(n) - \ln(k) - 9) \text{, for all } n \geq N_k \ .$$

Multiple selection problems have been investigated only in the worst case. While some specific instances of the problem have been studied for long time, an algorithm for the general problem first appeared in [Munro-Spira 76] and was

improved in [Dobkin-Munro 81]. This upper bound has been recently improved still further, approaching the lower bound supplied by information theory [Cunto-Munro 83]. The average case complexity for special instances of multiple selection are, to the best of our knowledge, first considered in this work.

1.2. Scope of the thesis and main results.

There were essentially two goals in the research leading to this thesis. The first was that of a specific theorem to close the 12 years old problem of proving the optimality of the Floyd-Rivest SELECT algorithm. The second was to introduce and develop some particular instances of the average case multiple selection problem.

Although the first goal is very specific, it plays an important role in deriving the results for the second. The second goal is that of opening a subarea that has received very little attention. The results on multiple selection obtained may be considered work still in progress. Certainly some interesting problems and conjectures remain to be investigated.

Chapters 2 through 5 constitute the body of the thesis. The main results are briefly outlined below.

Chapter 2 deals with the average case problem of selecting the minimum and the maximum of a set. Although this particular problem is not hard to solve, and the average case differs little from the worst case, the bounds obtained will be used explicitly in deriving the more important result of Chapter 4. Furthermore, this problem provides a relatively natural introduction to the type of arguments used later. Identical upper and lower bounds for selecting the minimum and the maximum are proven. That is,

$$M(1\,,\,n\,;\,n) = \frac{3n}{2} - 2 + \frac{1}{2n} \bmod(n\,,\,2)\,.$$

Compared with the worst case bounds for the same problem, an improvement is made when the set has odd cardinality. Certainly the most significant result of this chapter is the proof of the tight lower bound.

Chapter 3 is concerned with the selection of a single element such as the median. It begins with a detailed description of Floyd-Rivest selection algorithm SELECT. In addition to its place in providing the relevant upper bounds for Chapter 3, it also constitutes the basis of the algorithms to be developed in Chapter 4 and Chapter 5. A lower bound on selection due to Floyd and Rivest is

also described. It is shown that their approach cannot yield a substantially stronger lower bound. We reformulate their model, avoiding the restrictions implicitly imposed. The main part of the chapter is presented in the third section, in which a new model is used to prove that the algorithm SELECT is indeed optimal to within lower order terms. That is,

$$n + \min(k\,,\, n-k+1) - O(\log(n)) \leq M(k\,;\, n) \leq n + \min(k\,,\, n-k+1) + O(n^{1/2})\,.$$

In the case of selecting the median,

$$M(\lceil n/2 \rceil\,,\, n\,;\, n) = 1.5n \pm o(n)\ .$$

In Chapter 4 we turn to specific average case multiple selection problems. In particular we consider the problem of finding the maximum (or minimum) and any other rank, and also, of finding the minimum, the maximum and any other rank. Algorithms and lower bounds which depend on the rank of the 'other' element are developed. Typical of the results shown is that the complexity of selecting the median and the maximum (or minimum) is

$$M(\lceil n/2 \rceil\,,\, n\,;\, n) = 1.75n \pm o(n)\ .$$

Some of the algorithms developed in the chapter not only are conceptually simple, but also quite practical. This is the case for the algorithms finding the maximum and the median; the minimum and the median; the maximum, the minimum and the median.

Chapter 5 concludes with the problem of finding the closest neighbour(s) of a given random element x for which no rank is known a priori. The problem is analysed for both worst and average cases. In the worst case, we prove that lower and upper bounds are equal to

$$2n-3$$

comparisons. In the average case an algorithm is given with running time of

$$n + \min(k\,,\, n-k+1) + o(n)$$

comparisons if x is of rank k in the set. Averaging over all ranks, since x is a random element, this gives

$$\frac{5}{4}n + o(n)\ .$$

Our main purpose in achieving this result is to show that the basic techniques

used in solution of the problems presented in the previous chapters, can also be applied for a different kind of problem.

1.3. Types of comparisons in selection problems.

At each stage in any comparison based algorithm, the information which has been obtained can be represented as a collection of connected directed acyclic graphs, called fragments. These fragments can be depicted as a directed acyclic graph in which nodes denote elements and edges establish order relations. Such a figure is called a *Hasse diagram*. By convention, elements with bigger value are higher in the diagram, thus each edge is implicitly directed downward. We define a special kind of fragment, *maxtree*, to be a tree with edges oriented such that there is a path from a specific element, the maximum or *root*, to every other element. Because of its extensive use, we will refer to any maxtree simply as a *tree*. We will call a set of maxtrees a *maxforest*, or simply a *forest*. A tree of special interest is the *binomial tree*, being a tree of size a power of two, recursively defined as two binomial trees of equal size joined by the roots. A singleton is the minimal binomial tree.

In order to be able to identify the k-th smallest element of a set, any algorithm has to partition the set into a subset of $k-1$ bigger elements, the k-th, and a subset of $n-k$ smaller elements. Thus, starting with a collection of n unitary fragments, or singletons, representing the absence of known order among the

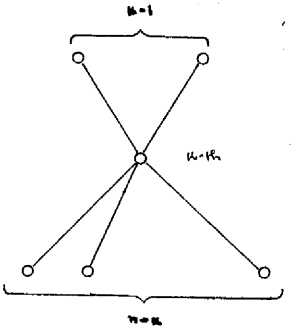


Figure 1.1
Subgraph present in any final configuration

elements in the set, any selection algorithm must produce a connected fragment whose transitive closure graph includes the graph given in Figure 1.1 as a subgraph. Based on these observations, Floyd and Rivest [Floyd-Rivest 75] defined two classes of comparisons. A *join comparison* is one which connects two disjoint fragments by comparing one element in each fragment. The *key comparison* of an element is the first one made between that element and either the k-th or some element lying between it and the k-th smallest. Observe that the set of $n-1$ key comparisons constitute the 'earliest' comparisons which can be used as a proof that the k-th smallest is indeed of that rank. Furthermore, a key comparison may be recognizable only a posteriori, after the k-th smallest is known. It is clear that any algorithm must perform exactly $n-1$ key comparisons and $n-1$ join comparisons as well. The approach of Floyd and Rivest was to show that a substantial number of join comparisons are usually not key comparisons.

For multiple selection problems, the concept of key comparison can be generalised to the *key-k_i comparison*, $1 \leq i \leq m$, that is, the key comparisons associated with the selection of the element with rank k_i.

We consider another class of comparisons called *straddle comparisons*, by which we mean those between elements that will fall in different partitions. Clearly, straddle comparisons are not key comparisons. Similar definitions of this type of comparisons can be found in the literature. See for example [Yao 73]. Schematically, the relation between the three classes of comparisons is presented in Figure 1.2.

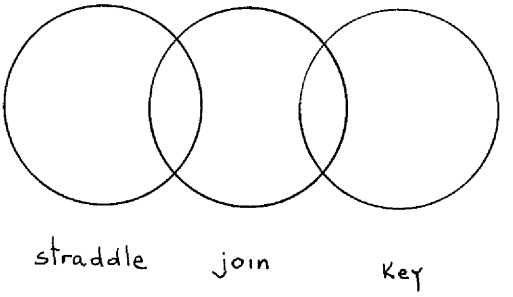


Figure 1.2
Relation between straddle, join and key comparisons

CHAPTER 2

MINMAX AVERAGE CASE

It is well known that in the worst case, $\lceil 3n/2 \rceil - 2$ comparisons are necessary and sufficient to find the minimum and maximum elements of n elements, usually called the *minmax* result. The bound was first proven in [Pohl 72], but a simpler proof of the lower bound can be found in [Knuth 73]. The same lower bound is again derived in [Fussenegger-Gabow 79], as a corollary of the more general problem of selecting the first t and last s elements of a set. In this chapter we study the average case version of the minmax problem. More precisely we will prove that

$$M(1\ ,\ n\ ;\ n) = \frac{3n}{2} - 2 + \frac{1}{2n}\ \mathrm{mod}(n\ ,\ 2)\ ,$$

where $\mathrm{mod}(n\ ,\ 2)$ is the residue of dividing n by 2.

Following the approach presented in [Pohl 72], we will distinguish four classes of elements: *virgins*, *winners*, *losers* and *others*. They will denote, respectively, the elements that have not yet been compared, elements that have won every (and at least one) comparison, elements that have lost every (and at least one) comparison, and elements that have won and lost at least one comparison.

2.1. The upper bound.

Pohl suggested the following optimal algorithm to find the maximum and the minimum of n elements, using at most $\lceil 3n/2 \rceil - 2$ comparisons [Pohl 72],

i) Make $\lfloor n/2 \rfloor$ pairs.

ii) Find the maximum of the pair maxima.

iii) Find the minimum of the pair minima.

iv) If n is odd, the remaining virgin is compared with the maximum computed in step (ii). If it loses, then it still has to be compared with the minimum found in step (iii).

Given our interest in the average case, we will see that a small improvement can be made when n is odd. Only in that case does the following algorithm

differ slightly from the one above. The algorithm will try to pair virgins as much as possible. In the case of odd cardinality, the remaining virgin will be compared in such a way that the virgin's probability of winning is as close as possible to its probability of losing.

More formally,

i) Pair $2\lfloor n/2 \rfloor$ virgin elements, then find the maximum of the pair maxima. Since the Hasse diagram of this part of the computation is a tree, call the maximum of the pair maxima the root.

ii) If n is odd then compare the remaining virgin with the element which lost the first comparison to the root.

If the virgin wins, perform an extra comparison with the tree's root, in order to decide the maximum of the set.

iii) Find the minimum of the set, by comparing the remaining losers located at the leaves of the resulting tree.

Except for the second step, all the steps are unconditional. Furthermore, during step (ii), the extra virgin is compared with an element that has lost only to the maximum of $n-1$ elements. This loser may almost be considered a virgin in the sense that we have virtually no information about its rank.

LEMMA 2.1. When n is odd, the probability of the extra virgin losing its first comparison during step (ii) of the algorithm, is $\frac{n-1}{2n}$.

Proof. Let X denote the set of n elements including x the remaining virgin after step i of the the previous algorithm. The element which lost to the root of the tree in the first round, y, has probability $\frac{1}{n-2}$ of being the j-th smallest, $1 \leq j \leq n-2$, of X–x. Before being compared, x has probability $\frac{1}{n}$ of having rank i, $1 \leq i \leq n$, in X.

If x is of rank i, the probability that $y > x$ is 0 if $i = n$ or $i = n-1$, and $\frac{n-1-i}{n-2}$ otherwise.

Hence the probability that $y > x$ is

$$\sum_{i=1}^{n-2} \frac{n-1-i}{n-2} \frac{1}{n} = \frac{1}{n(n-2)} \sum_{i=1}^{n-2} i$$

$$= \frac{1}{n(n-2)} \frac{(n-1)(n-2)}{2}$$

$$= \frac{n-1}{2n} .$$

Hence, we can show,

THEOREM 2.2. The average number of comparisons performed by the minmax algorithm outlined above on a set of size n is

$$M(1, n; n) \leq \frac{3n}{2} - 2 + \frac{1}{2n} \bmod(n, 2) .$$

Proof. When n is even, the algorithm performs exactly $\frac{3n}{2} - 2$ comparisons. If $n = 2k + 1$, step (i) will perform $2k - 1$ comparisons in finding the maximum of $n - 1$ elements. At step (ii), the remaining virgin is compared for its first time, winning with probability $\frac{n+1}{2n}$ and therefore being compared again with the tree's root. Independent of the outcome, step (iii) involves k losers located at the leaves of final tree, and hence $k - 1$ more comparisons are needed for selecting the minimum.

In total $\frac{3n}{2} - 2 + \frac{1}{2n} \bmod(n, 2)$ comparisons are performed on average.

The comparisons performed can be classified as *key-min*, *key-max* and *join* comparisons. If n is even, then, during the first step of the algorithm, the $n/2$ pairings correspond to comparisons that are simultaneously key-min, key-max and join. The next $n/2 - 1$ comparisons in step (i) are key-max and join. Finally, in the third step, $n/2 - 1$ key-min comparisons are made in finding the minimum of the set. Observe that, exactly $n - 1$ comparisons of each type are performed in the algorithm.

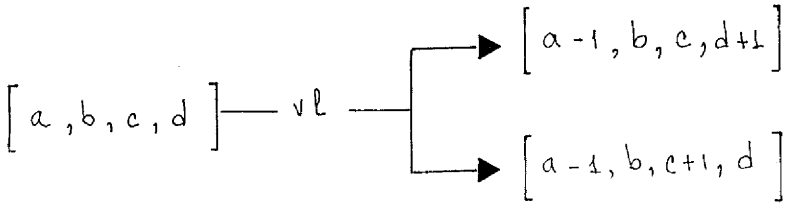
2.1.1. The lower bound.

The crucial information regarding the state of any computation which is determining the maximum and the minimum can be summarised by the

quadruple $[a, b, c, d]$ called an *instance* (see [Pohl 72]). It specifies the number of virgins, losers, winners and others, respectively. From an instance, one of ten possible comparisons can be performed (virgin and virgin, virgin and loser, etc), making a probabilistic transition to one of two possible instances, depending on the outcome of the comparison. Therefore, starting from the initial configuration of n virgins, any minmax algorithm will follow a sequence of instance-comparisons, ending with the final instance consisting of no virgins, one loser representing the minimum, one winner representing the maximum, and $n-2$ others. Figure 2.1 illustrates the transitional process from one instance to another. Although the outcome probabilities depend on the partial order of the instances, they can be easily bounded. As the second property is the only one needed in proving the lower bound for the minmax problem, the notation used in Figure 2.1 does not relate these probabilities to the partial order of the instances.

Elements of each type are assigned a weight referring to a lower bound on the asymptotic average number of comparisons the element contributes to the total work. Let x_1, x_2, x_3 and x_4 denote the weights assigned to virgins, losers, winners and others, respectively. The weight of an instance is the sum of each element weight. Initially, the weight is nx_1, and finally, $x_2 + x_3 + (n-2)x_4$.

The model has to ensure that, as a result of a comparison, the average difference between incoming and outgoing instance weights is not greater than one (the comparison performed during the transition). This guarantees that the difference between the initial and the final instance weights is a lower bound on the average number of comparisons every minmax algorithm must perform. For example, if a virgin and a loser are compared, the transition is,



No matter which partial order contains the loser, the probability, p_1, of the loser winning the comparison is less than half ($p_1 < 1/2$), since it has already lost at least one comparison. A similar argument applies to p_2, when virgin and winner

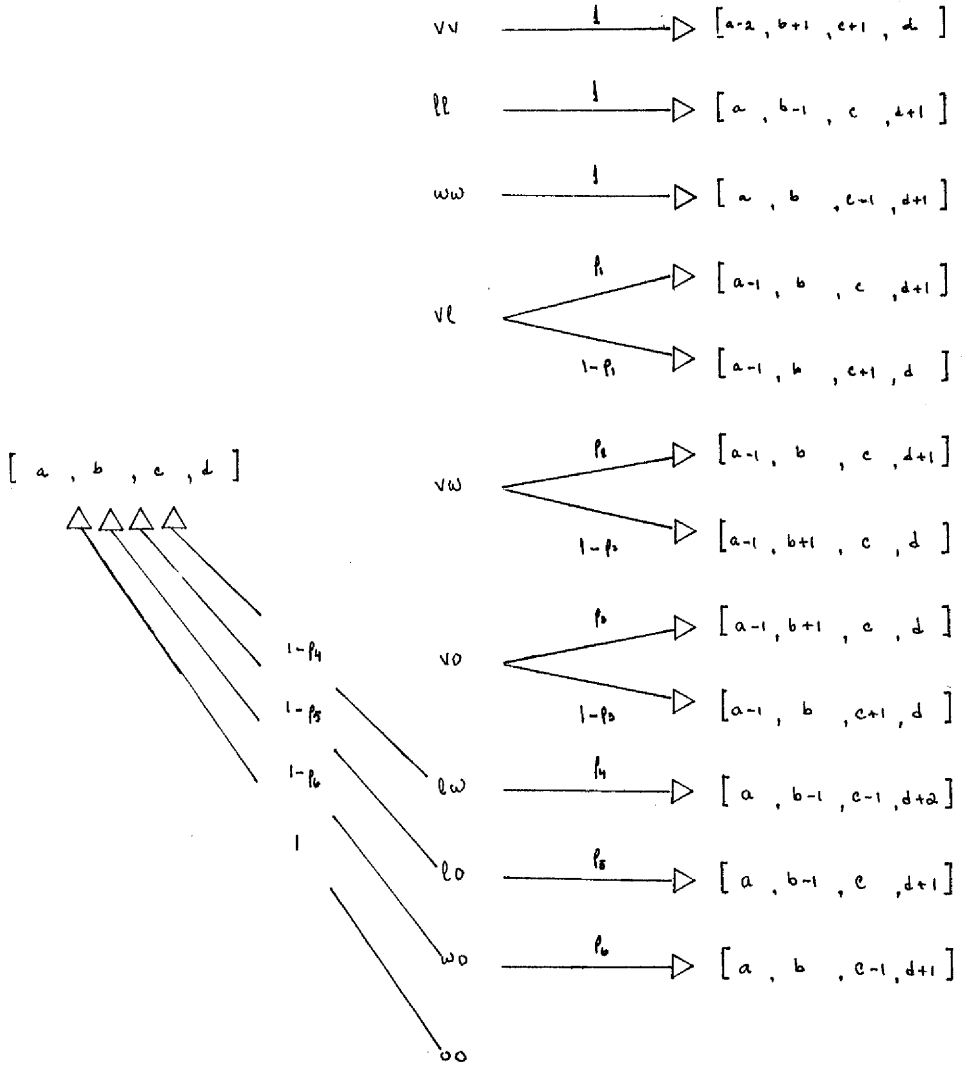


Figure 2.1
Transitions among instances

are compared. By taking the average difference of the instance weights, we get

$$\Big\{ ax_1 + bx_2 + cx_3 + dx_4 \Big\} - p_1 \Big\{ (a-1)x_1 + bx_2 + cx_3 + (d+1)x_4 \Big\} -$$

$$- (1-p) \Big\{ (a-1)x_1 + bx_2 + (c+1)x_3 + dx_4 \Big\} = x_1 - (1-p_1)x_3 - p_1 x_4 \ ,$$

therefore, the associated inequality is,

$$x_1 - (1-p_1)x_3 - p_1 x_4 \leq 1 \ .$$

The inequalities associated with the 'other' nine possible comparisons are derived similarly from the transitional diagram shown in Figure 2.1. Any feasible solution in the model constitutes a lower bound for the problem. Hence, the goal is to get the maximum lower bound possible. By using linear programming techniques, the following lower bound for the average case minmax problem will be shown.

Linear programming has already been used as a technique for proving lower bounds in related problems. It is explicitly used in [Floyd-Rivest 75] to derive a lower bound for selecting the median. Also, lower bounds appearing in other references as [Pohl 72], [Schonhage 73] and [Munro-Poblete 82] can be formulated as linear programming problems. We make use again of such a technique exploiting also the information provided by the dual of the linear programming problem derived for the minmax problem. As a useful reference for looking at the basic concepts of linear programming we suggest [Hillier-Lieberman 74].

THEOREM 2.3. The number of comparisons necessary on the average for the minmax problem is

$$M(1\ ,\ n\ ;\ n) \geq \frac{3n}{2} - 2 + \frac{1}{2n} \bmod(n\ ,\ 2) \ .$$

Proof. As mentioned above, for each possible transition in Figure 2.1, the lower bound is given by solving the linear programming problem,

$$\max \ z = nx_1 - (x_2 + x_3 + (n-2)x_4)$$

such that,

$$2x_1 - x_2 - x_3 \leq 1$$

$$x_2 - x_4 \leq 1$$

$$x_3 - x_4 \leq 1$$

$$x_1 - (1 - p_1)x_3 - p_1x_4 \leq 1$$

$$x_1 - (1 - p_2)x_2 - p_2x_4 \leq 1$$

$$x_1 - p_3x_2 - (1 - p_3)x_3 \leq 1$$

$$p_4(x_2 - x_3) - 2p_4x_4 \leq 1$$

$$p_5x_2 - p_5x_4 \leq 1$$

$$p_6x_3 - p_6x_4 \leq 1$$

$$x_1 , x_2 , x_3 , x_4 \geq 0 \ .$$

The inequality associated with the case when two 'others' are compared is not included, since trivially, $0 \leq 1$. Note that the last three inequalities are redundant since the second and the third are more restrictive. These last three inequalities and the one from the comparison of two 'other' elements correspond to the cases in which, with positive probability, one possible outcome from the transition is the same incoming instance, that is, the transition is stationary at the same instance. This is represented in Figure 2.1 by the transitions starting at the instance $[a , b , c , d]$ and with one outcome to the same instance $[a , b , c , d]$.

Discarding the redundant inequalities, the primal problem is then

$$\max \ z = nx_1 - (x_2 + x_3 + (n - 2)x_4)$$

such that,

$$2x_1 - x_2 - x_3 \leq 1$$

$$x_2 - x_4 \leq 1$$

$$x_3 - x_4 \leq 1$$

$$x_1 - (1 - p_1)x_3 - p_1x_4 \leq 1$$

$$x_1 - (1 - p_2)x_2 - p_2x_4 \leq 1$$

$$x_1 - p_3x_2 - (1 - p_3)x_3 \leq 1$$

$$x_1 , x_2 , x_3 , x_4 \geq 0 \ .$$

It is easy to see that both p_1 and p_2 are strictly less than one half, since losers and winners have already lost and won a comparison already, respectively. The solution of the linear programming problem is then,

$$z = \frac{3n}{2} - 2 , x_1 = \frac{3}{2} , x_2 = x_3 = 1 , x_4 = 0 \ .$$

and, $\frac{3n}{2} - 2$ is a lower bound for the problem. Figure 2.2 is the final tableau of the Simplex algorithm [Hillier-Lieberman 74] applied to our primal problem.

	z	x_1	x_2	x_3	x_4	y_1 x_5	y_2 x_6	y_3 x_7	y_4 x_8	y_5 x_9	y_6 x_{10}	sol
z	1					$\frac{n}{2}$	$\frac{n}{2}-1$	$\frac{n}{2}-1$				$\frac{3}{2}n-2$
x_1		1			-1	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{2}$				$\frac{3}{2}$
x_2			1		-1		1					1
x_3				1	-1			1				1
x_8						$-\frac{1}{2}$	$-\frac{1}{2}$	$\frac{1}{2}-p_1$	1			$\frac{1}{2}-p_1$
x_9						$-\frac{1}{2}$	$\frac{1}{2}-p_2$	$-\frac{1}{2}$		1		$\frac{1}{2}-p_2$
x_{10}						$-\frac{1}{2}$	$-\frac{1}{2}+p_3$	$\frac{1}{2}-p_3$			1	$\frac{1}{2}$

Figure 2.2
Final tableau for the primal problem

The lower bound derived above is tight when n is even, but not when it is odd. Deriving the dual problem from the primal problem,

$$\min g = \sum_{i=1}^{6} y_i$$

such that,

$$\begin{aligned} 2y_1 + y_4 + y_5 + y_6 &\geq n \\ -y_1 + y_2 - (1 - p_2)\, y_5 - p_3\, y_6 &\geq -1 \\ -y_1 + y_3 - (1 - p_1)\, y_4 - (1 - p_3)\, y_6 &\geq -1 \\ -y_2 - y_3 - p_1\, y_4 - p_2\, y_5 &\geq -(n-2) \end{aligned}$$

$$y_i \geq 0 \quad 1 \leq i \leq 6 \ ,$$

we observe that y_i is the number of comparisons of type i (see Figure 2.1) necessary to solve the minmax problem. The final tableau gives the optimal solution of the primal problem and also displays the values of the dual variables

$$y_1 = \frac{n}{2} \ , \ y_2 = y_3 = \frac{n}{2} - 1 \ , \ y_4 = y_5 = y_6 = 0 \ ,$$

indicating that only *vv*, *ll*, and *ww* comparisons are performed. When n is odd, at least one comparison of the type *vl*, *vw* or *vo* must be made. In Figure 2.2, the value of the slack variable x_{10} associated with the *vo* comparisons is greater than x_8 and x_9, which are the slack variables associated with the *vl* and *vw* comparisons respectively. This means that a local perturbation of the optimal solution due to the fourth or fifth equation is smaller than the one due to the sixth equation in the primal problem. Since the fourth and fifth equations in the primal problem are symmetric, we can arbitrarily choose to add

$$y_4 \geq 1 \quad \text{or} \quad y_5 \geq 1$$

to the dual problem. The new optimal solution is then

$$g = \frac{3n}{2} - 2 + (1 - p_1) \ .$$

The maximum value of p_1 is the one computed in Lemma 2.1, that is

$$p_1 \leq \frac{1}{2} - \frac{1}{2n} \ .$$

Consequently,

$$\mathrm{M}(1\ ,\ n\ ;\ n) \geq \frac{3n}{2} - 2 + \frac{1}{2n} \bmod(n\ ,\ 2) \ .$$

CHAPTER 3

COMPLEXITY OF SELECTION PROBLEMS

For years the median selection problem has proved intriguing to many people. Although published more than a decade ago, the worst case linear selection algorithm of [Blum et al 73] still constitutes a surprising achievement and a landmark in algorithm design. After a number of efforts, the best lower and upper bounds for the worst case median selection are $79/43\ n - O(1)$ [Munro-Poblete 82] and $3n + o(n)$ [Schonhage et al 76] comparisons respectively.

For practical purposes, selection algorithms with linear worst case behaviour have been unsatisfactory as they have had very complex implementations. However, 'practical' selection algorithms with linear average running time exist. For example, the idea of randomly partitioning a set, as is used in Quicksort, can be easily adapted for selection [Hoare 61], with running costs of $3.39n + o(n)$ comparisons on average [Knuth 71].

A very neat algorithm for selection called SELECT was presented in [Floyd-Rivest 75], and rapidly became the preferred selection algorithm because of its simplicity and average running time of $n + \min(k, n - k + 1) + o(n)$ comparisons for selecting the k-th smallest of n elements. The main feature of the algorithm is the use of sampling techniques for selecting the desired rank. Floyd and Rivest also demonstrated the first nontrivial lower bound for the average case median selection problem ($1.375n$ comparisons). They conjecture on the existence of a lower bound differing by o(n) comparisons from the upper bound supplied by the average running time analysis of SELECT [Floyd-Rivest 73]. We will discuss the limitations of their model and give an informal description of how it can be modified in order to get better lower bounds.

The main result of this chapter is the proof that the Floyd-Rivest algorithm is indeed optimal (up to a lower order term), i.e. $n + \min(k, n - k + 1) - o(n)$ comparisons are necessary, on the average, for selecting the k-th of a set of n elements. For the case of selecting the median, this gives a lower bound of $1.5n - o(n)$ comparisons. Therefore, problem 5.3.4.24 in [Knuth 73] is solved.

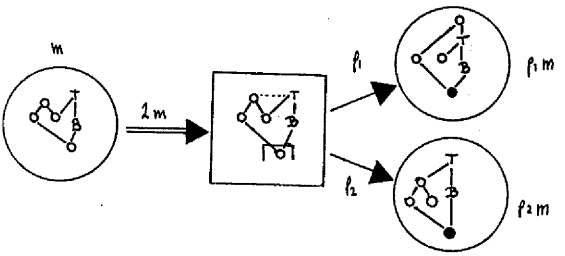
3.1. The Floyd-Rivest selection algorithm.

The main idea of the algorithm SELECT is to estimate the solution by first taking a sample that, in negligible time ($o(n)$ comparisons), will give quite a good approximation. More precisely, the algorithm will take a random sample of size $o(n)$ from the set of n elements, and in $o(n)$ time will find two elements, bottom cut and top cut, denoted by B and T respectively, so that, with very high probability, the k-th smallest element falls between them and the difference between their ranks in the full set is also $o(n)$. The algorithm then partitions the remaining singletons, elements that have not been in a comparison yet, by comparing them directly with the two cuts. If $k \leq \lceil n/2 \rceil$, the singletons are first compared with T and subsequently with B if necessary (the order reverses if $k > \lceil n/2 \rceil$). The algorithm can then be recursively applied on the subset containing the actual k-th. Because of the properties of the two cuts, the algorithm will recurse, with probability almost one, on the subset of $o(n)$ elements between the two cuts.

For a fixed probability that the k-th smallest element fails to lie between B and T, there is an inverse relationship between s, the size of the sample S, and the expected number of elements falling between the cuts. Therefore, balancing these values, i.e. $s = \Theta(n^{2/3}(\ln(n))^{1/3})$, minimises the expected cost subject to a fixed tolerance that the k-th is not between B and T. The rank in X of any element with known rank in S follows a negative hypergeometric distribution. Hence, the average and the variance of the distribution can be computed. SELECT is then recursively applied on S to select B and T in such a way that the k-th value lies between the expected ranks of both cuts in X, and the probability of the k-th being less than the rank of B or bigger than the rank of T in X is $o(1/n)$. The latter condition ensures that the expected work will be $o(1)$ even if SELECT 'misses' the target element which means it still has $O(n)$ work to do. The preceding conditions are satisfied by choosing the ranks of B and T in S to be $k\frac{s+1}{n+1} - [\ln(n)\, s]^{1/2}$ and $k\frac{s+1}{n+1} + [\ln(n)\, s]^{1/2}$, respectively. This approach to SELECT gives a running time of $n + \min(k, n-k+1) + O(n^{2/3}\ln(n)^{1/3})$ comparisons. With some additional complication the sublinear term can be improved to $O(n^{1/2})$ (see [Floyd-Rivest 75]).

A diagrammatic description of the algorithm is given in Figure 3.1. Circles represent configurations and squares comparisons on elements. Heavier arrows represent comparisons and lighter arrows outcomes.

Labels on heavier arrows denote the number of fragments, and labels on lighter arrows the outcome probabilities. For example, in the following diagram



the configuration has m fragments. Two comparisons per fragment are then performed, one involving finding the minimum of the minima. Black nodes indicate the elements that have already been compared in finding the minimum (or the maximum) of the set, and p_1 and p_2 denote the outcome probabilities.

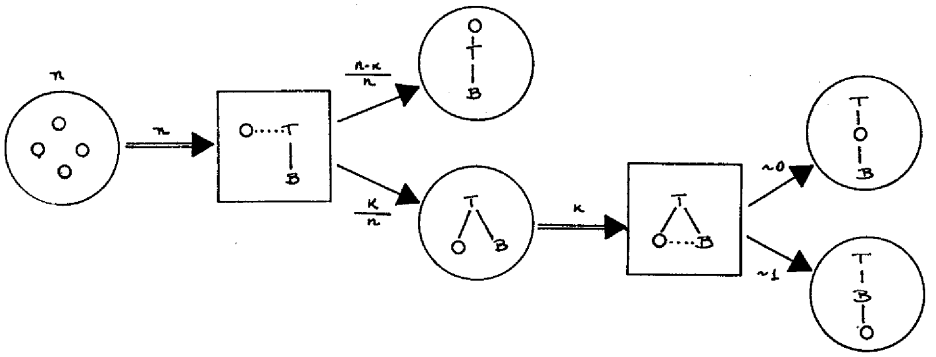


Figure 3.1
Algorithm SELECT for $k \leq \lceil n/2 \rceil$

Discarding lower order terms, the algorithm basically requires $\min(k, n-k+1)$ straddle comparisons, and the same number of key and non-join comparisons. In total, the algorithm will find the k-th smallest of a set of size n, in $n + \min(k, n-k+1) + o(n)$ comparisons on average.

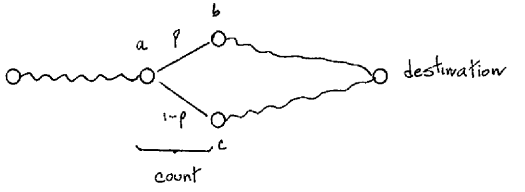
3.2. The Floyd-Rivest median selection lower bound, weaknesses and extensions.

This section will be entirely devoted to median selection, that is $k = \lceil n/2 \rceil$. Although it presents new extensions of a proof technique, the median lower bound developed here is subsumed by that of the next section, and the notation in this section is not necessary for the subsequent one.

It is not expected that all join comparisons will be key as well. Essentially, the lower bound of [Floyd-Rivest 75] is based on counting the expected number of join nonkey comparisons. Adding the $n-1$ key comparisons to this value produces their lower bound for the problem.

The counting is done by looking at a family of 'small' fragments that consist of all possible directed acyclic graphs containing at most some specified number of nodes. Larger fragments are declared 'big' and ignored since it is almost impossible to handle the large number of possible derivable fragments. Because join comparisons cause fragments to be joined into larger fragments, it seems natural to charge them the join nonkey comparisons. To accomplish this, the model associates a nonnegative weight with each fragment, representing the fragment's individual contribution to the lower bound. Fragments of the same type (i.e. isomorphic) will have the same weight, and weights will be independent of the specific configuration to which the corresponding fragment belongs. The weight of a configuration can be computed by adding the weights of the component fragments.

Configurations and their corresponding weights can be interpreted as points and lower bounds on their path length to a destination, respectively. The initial configuration consists of n singletons and the final configuration of only one big fragment.



Lower bound in a $\leq$ p Lower bound in b + (1-p) Lower bound in c + count

The count of the comparisons during a transition is a lower bound on the distance between the two consecutive points representing the outgoing and incoming configuration in the transition. Then, in order to compute a valid lower bound, the model must ensure that, during any transition, the difference between the weight of the outgoing configuration and the average weight of the incoming configurations is at most the count performed during the transition. As a consequence, the sum of weights of the initial n singletons is a valid lower bound for the problem. Joining two big fragments creates another big fragment. Since no count is performed during such a transition and because all weights must be nonnegative, their weight will be 0. In particular the weight of the final configuration is 0. The inequalities derived from the transitions will supply valid values for the fragment weights, and the maximum possible value for the singleton weight has to be found.

At this point, the model can be represented as a linear programming problem with format

$$\max \; z = e_1^T w$$

$$A w \leq b$$

$$w \geq 0 \; ,$$

where the first canonical vector, e_1, is the cost vector, w is the vector of nonnegative weights, with w_1 corresponding to the singleton weights, and w_i the weight of a fragment type i. A is the transition matrix, and b, the counting vector is also taken from the transitions.

The Floyd-Rivest model is based on the belief that, asymptotically, the average probability of a join comparison being a key comparison, is at most 1/2 . It assumes a negligible probability of a join comparison involving the true median. Furthermore, the kind of counting that is performed is quite 'local' as it deals primarily with join comparisons of specific types of fragments.

This model will handle two kinds of join comparisons,

i) between two small fragments,

ii) between small and big fragments.

Its weakness is due to the second type of join comparison. In those cases, the probability of a join comparison being nonkey is equivalent to the probability of that comparison being a straddle comparison. This fact will cause the weights to be lower bounds on the number of join and straddle comparisons, rather than on

the number of join nonkey comparisons alone. Actually, the lower bound obtained by formulating the model for the family of fragments with size up to three (see Figure 3.2) is not improved by any other formulation of the model with families of bigger size. This follows from the transitions associated with the join of a pair to a big fragment and the join of two singletons creating a pair. The model will associate with the first transition the inequality,

$$w_2 \leq 1/4 \ ,$$

and to the second,

$$2\ w_1 - w_2 \leq 1/2 \ .$$

Both inequalities are included in every formulation obtained by the model for any family of fragments up to size at least three, and both of them will satisfy equality in all linear programming solutions. Consequently,

$$w_1 \leq 3/8 \ ,$$

and, the best possible lower bound for selecting the median achievable with this model is $11/8\ n$ comparisons.

We observe that if join and straddle comparisons are the only comparisons counted, then it is possible to design an algorithm that joins the elements in such a way that the average number of join and straddle comparisons performed closely matches the lower bound on join and straddle comparisons given by the

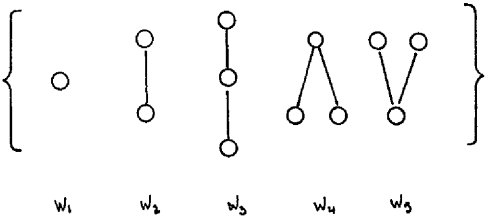


Figure 3.2
Family of fragments of size at most 3

Floyd-Rivest model for the median selection problem. The idea is simple, and deals with making join comparisons that are meaningless to the problem of selecting the median, namely by building a binomial tree, and therefore finding the maximum of the set.

LEMMA 3.1. The expected number of comparisons which are join and also straddle with respect to the median can be as low as

$$\frac{8}{21}\, n - \frac{2}{3n} + \frac{2}{7n^2} < 0.38..\, n \ .$$

This demonstrates that models based on the counting of join-straddle comparisons, such as the one given in [Floyd-Rivest 75], will not be suitable for proving that the algorithm SELECT is optimal up to lower order terms.

Proof. A binomial tree on $n = 2^m$ elements is formed by pairing winners on successive passes until one remains. We observe that as the singletons are paired, resulting in $n/2$ pairs, $n/4$ of those $n/2$ comparisons are expected to be straddles. On the second iteration, the maxima of pairs are joined, forming binomial trees of size 4. Since 1/4 of those pairs are expected to consist of two elements below the median, it follows that

$$2\,\frac{1}{4}\,\frac{3}{4} = \frac{3}{8}$$

of those $n/4$ comparisons, or $3n/32$ comparisons, are straddle. More precisely, at the beginning of the i-th pass, $n/2^{i-1}$ binomial trees of size 2^{i-1} are joined by their maxima in a total of $n/2^i$ comparisons, yielding $n/2^i$ binomial trees of size 2^i each. A straddle is obtained if and only if only one of the two binomial trees being joined is completely below the median and the other is not. This happens with probability

$$2\,\frac{1}{2^i}\left(1 - \frac{1}{2^i}\right) \ .$$

Then, during the i-th pass

$$2\,\frac{1}{2^i}\left(1 - \frac{1}{2^i}\right)\,\frac{n}{2^i}$$

of the $n/2^i$ join comparisons are expected to be straddle. This yields an expected total of

$$2n\left(\sum_{i=1}^{m} 2^{-2i} - \sum_{i=1}^{m} 2^{-3i}\right) = \frac{8}{21}n - \frac{2}{3n} + \frac{2}{7n^2}$$

comparisons which are both join and straddle.

Note, however, that for $n \geq 64$, this bound is greater than the asymptotic lower bound supplied in [Floyd-Rivest 75].

Next, we will try to overcome the restrictions imposed by the 'local counting' in the Floyd-Rivest model. Our main idea will be to count not only straddles, but also any join nonkey comparisons possible. For example, consider finding the median of 5 elements, M(3 , 5) = 88/15 (see 5.3.3 in [Knuth 73]). A possible intermediate outcome during the computations is illustrated in Figure 3.3. Note that the comparisons performed between elements a and b is, with probability 1/4, a join, nonkey, non-straddle comparison, if c is the actual median and b is larger than it. Instead of 'locally' counting join comparisons and their probabilities of being nonkey, we join fragments until they have to be declared 'big'. At this point a less 'local' count is made. Any count, therefore, is performed a bit more 'globally'. The situations to be considered are

i) joining two small fragments, giving another small fragment,
ii) joining two small fragments, getting a big one,
iii) hooking a small fragment into a big one.

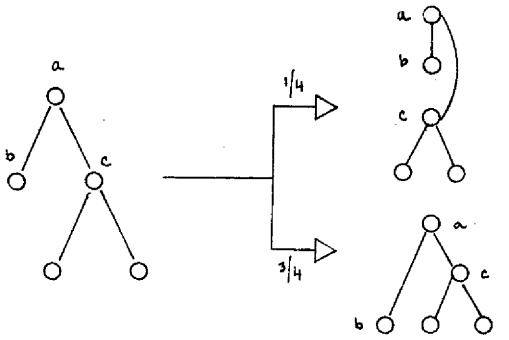


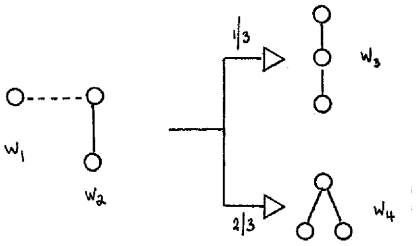
Figure 3.3
Possible joins in the optimal
algorithm for median of five elements

For the first type of join, because the result is still in the family and no count is performed, the right side of the corresponding inequality is less than or equal to 0. Figure 3.4 illustrates an example of such join comparison (a family of fragments with sizes less or equal to 3 is assumed).

The outcome of the second type of join has to be declared 'big', and the expected number of join nonkey comparisons computed. This value is based on the total orderings possible and the binomial distribution of how the elements in these total orders may be partitioned about the median. Figure 3.5 schematically explains the counting performed when the maxima of two pairs are joined. Because the joining of two fragments can be done by any pair of elements, one in each fragment, the model must consider all these possibilities. The joining of the fragments through different pairs of elements will give inequalities differing only by their right sides, since the counting is different but not the fragments. The most restrictive inequality (inequality with minimum right side) is then kept, and associated with the joining of these two fragments.

Asymptotically, the elements in a 'big' fragment may be thought of as having continuous rank αn, $\alpha \in [0, 1]$. Again, each possible element to join and each possible total order in the small fragment is considered. The model relies on the Floyd-Rivest lemma proving that the expected number of joins involving the median is small. When joining a 'small fragment' and a 'big' fragment, the model must consider

i) all possible permutations consistent with the 'small' fragment, and



$$W_1 + W_2 - \frac{1}{3} W_3 - \frac{2}{3} W_4 \leq 0$$

Figure 3.4
Small fragment from the join of two small fragments

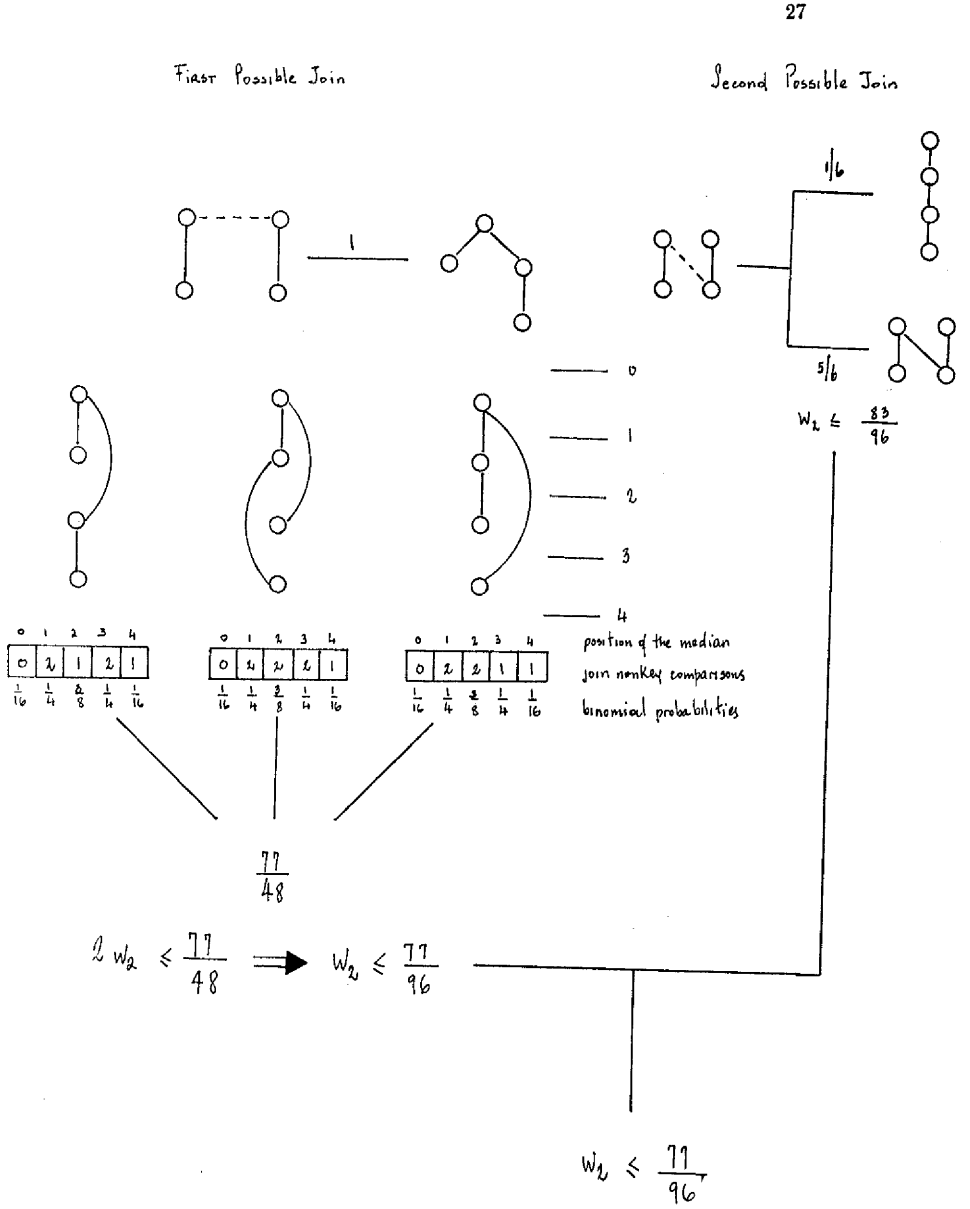


Figure 3.5
Joining two pairs giving a big fragment

their corresponding probabilities,

ii) joining by each element in the 'small' fragment,

iii) joining by an element in the big fragment that may be above or below the median.

Then, for every total order and each element in it, the count is performed for the case that the element in the 'big' fragment is above the median, and for the case it is below the median. In either case, the average of the counts (trinomially distributed) is a polynomial in α with a maximum value in its subinterval $[0\ ,\ 1/2)$ in the first case and in $(1/2\ ,\ 1]$ in the second case. For each element in the 'small' fragment there are two cases. In one case, the element in the 'big' fragment is above the median, and in the other, it is below the median. The most restrictive of the two cases is then kept. The final inequality associated with the 'small' fragment is given by the most restrictive inequality associated with any element in it. Figure 3.6 gives an example of this type of join.

We were able to implement the whole model as a package with the aid of a simplex routine developed by R. Bartels [Bartels 80] and facilities from the symbolic manipulator Maple [Geddes et al. 82]. We obtained the lower bounds of Floyd and Rivest by considering the family of singletons and also the family of singletons and pairs. The first improvement with the new approach appears when the family of fragments of sizes at most 3 is considered. This provides a lower bound for selecting the median of $269/192\ n = 1.40..n$ comparisons. Figure 3.7 illustrates the linear programming problem associated with this first improvement.

The advantages of our approach over the one of Floyd-Rivest are:

i) it seems to improve the lower bound whenever a bigger family of fragments is considered,

ii) the join between small and big fragments does not restrict the model as in the case of Floyd-Rivest's approach, and

iii) it actually bounds the set of join nonkey comparisons rather than the set of join and straddle comparisons.

With this new approach, we are able to get $1.44..\ n$ as a lower bound. The family of fragments with sizes less than or equal to 5 was considered in the computation.

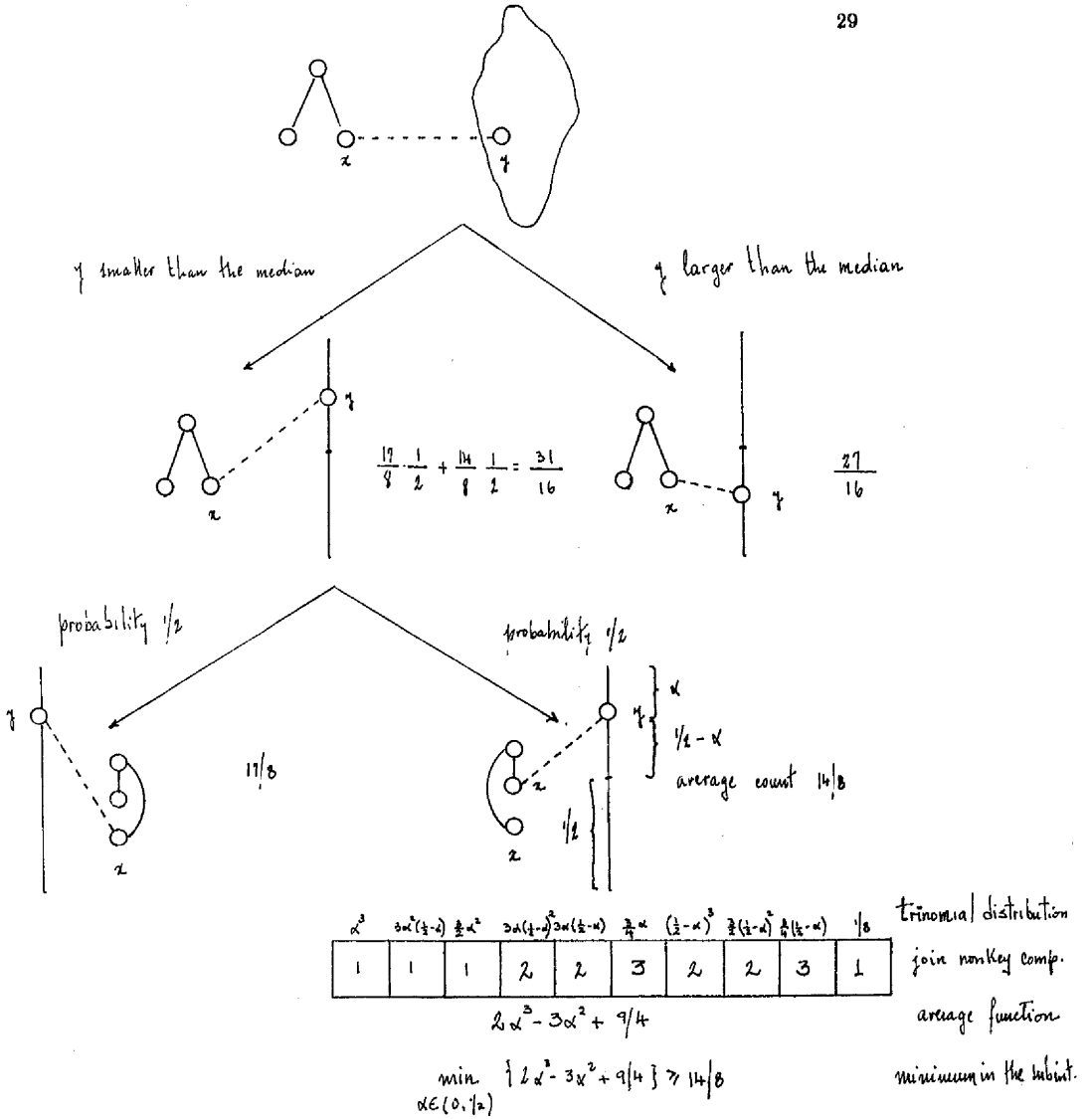


Figure 3.6
Hooking a small fragment into a big fragment

$$\max \; z = e_1^T w$$

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 2 & -1 & 0 & 0 \\ 1 & 1 & -1/3 & -2/3 \\ 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} w_1 \\ w_2 \\ w_3 \\ w_4 \end{pmatrix} \leq \begin{pmatrix} 1/2 \\ 0 \\ 0 \\ 21/16 \\ 57/32 \\ 77/96 \\ 153/80 \\ 9/4 \\ 173/160 \\ 407/160 \\ 1831/1280 \end{pmatrix}$$

$$w_i \geq 0 \qquad i = 1, 2, 3, 4 \; .$$

$$z = \frac{269}{192} \; , w_1 = \frac{77}{192} \; , w_2 = \frac{154}{192} \; , w_3 = \frac{175}{192} \; , w_4 = \frac{265}{192} \; .$$

Figure 3.7
Model for median lower bound

3.3. SELECT is within o(n) of optimal.

The proof of our main theorem begins with simple observations extracted from the Floyd-Rivest SELECT algorithm. In this algorithm almost all elements are initially compared with one of the cuts, B or T, that are expected to be close in rank to the k-th smallest. With high probability, about k of these comparisons are straddles. Furthermore, virtually all the nonkey comparisons correspond to those straddles. Also, because both cuts are close in rank to the k-th smallest element, if a singleton is compared with one of the two cuts and the comparison is not a straddle, then with probability almost one, its comparison with the other cut is a straddle.

As no model based on the counting of join and straddles comparisons can provide a satisfactory lower bound, our main approach is to relate the disjoint sets of straddle and key comparisons. The crucial notion is what we call a *close comparison*. The *close comparison* for the j-th smallest element, $1 \leq j \leq n$ and $j \neq k$, is *the first comparison* made between that element and another of rank i such that, $|k-i| \leq |k-j|$. In other words, the *close comparison* of an element

is the first comparison between it and some other element which is at least as close to it as the k-th smallest. Observe that this kind of comparison is either a straddle or a key comparison, but not both. Furthermore, it is not necessarily a join comparison.

For purposes of notational convenience, we will think of X, the set of elements, as being the set of integers 1 through n. This assumption will allow us to refer to the i-th smallest in X as the value i. Π will denote the set of $n!$ permutations of X. It is helpful to think of an input permutation of X being stored as a vector. Algorithms are allowed to access the vector cells but not to interchange or modify their contents. Information regarding the relative order among elements is obtained by pairwise comparisons. The model represents any algorithm as a decision tree with internal nodes indicating the pairwise comparisons between two vector cells. The left branch of an internal node is taken if the first argument of its comparison is smaller than the second argument. The right branch is taken otherwise. Final configurations displaying the solution are at the fringe of the decision tree.

Let us focus our attention on the elements $k+l$ and $k-l$ with $1 \leq l \leq \min(k-1, n-k)$. By fixing l, $k+l$ has a close comparison when compared for the first time with an element with rank in the range $[k-l, k+l-1]$, and $k-l$, when it is compared for the first time with an element of rank in the range $[k-l+1, k+l]$. Then, relative to $k-l$ and $k+l$, our approach is to consider a pair of permutations in Π such that the algorithm follows the same sequence of comparisons up to the point at which the close comparison is made for $k+l$ in one permutation and $k-l$ in the other. Moreover, the close comparison of $k+l$ will be a straddle if and only if the close comparison for $k-l$ is a key. By presenting a bijection from Π to itself, we will be able to prove that, on average, the number of close comparisons of $k-l$ and $k+l$ which are straddles is at least $1-o(1)$. By adding this average for all l in the range, a lower bound on the number of straddle comparisons can be computed. Finally if the $n-1$ key comparisons are added to the lower bound on the number of straddle comparisons, a lower bound for the selection problem is achieved. Having proven this, it follows immediately that SELECT is within a lower order term of optimal.

The definition of the class of close comparisons may seem very 'algorithm dependent'. Actually, this is not the first time 'algorithm dependent' operations have been defined in order to prove the optimality of a certain algorithm. For example, in order to prove the optimality of the Horner's polynomial evaluation

rule with respect to multiplications, a restricted type of multiplication (active multiplication) was defined [Pan 66]. The new operation allowed that proof to concentrate on the relevant ways of performing multiplications. In our case, a similar direction is followed.

LEMMA 3.2. For any algorithm selecting the k-th smallest and any l in $[1, \min(k-1, n-k)]$, the expected number of close comparisons (over all input permutations) of $k+l$ which are straddles plus the expected number for $k-l$ is 1.

Proof. Let π_j denote the j-th element of the permutation π. We will define a bijection, f, mapping $\Pi \to \Pi$ and adopt the notation that $f(j)$ denotes the image of j in $f(\pi)$. Hence $f(\pi_j)$ denotes the j-th element in $f(\pi)$. The bijection $f(\pi)$ and its inverse $f^{-1}(\pi)$ are cyclic shifts of the values $k-l, \cdots, k+l$ in π. More formally:

$$f(k+l) = k-l$$

$$f(j) = j+1 \quad j \in [k-l, k+l-1]$$

and

$$f(j) = j \quad \text{otherwise} .$$

Clearly f^{-1} is given by

$$f^{-1}(k-l) = k+l$$

$$f^{-1}(j) = j-1 \quad j \in [k-l+1, k+l]$$

and

$$f^{-1}(j) = j \quad \text{otherwise} .$$

Consider the algorithm working with the permutation π as input. Suppose $\pi_j = k+l$ and $v = \pi_i : \pi_j$ is the close comparison of π_j. This comparison is key if $\pi_i \in [k, k+l-1]$ and straddle if $\pi_i \in [k-l, k-1]$.

Now consider the action of the algorithm on $f(\pi)$. Up to the operation in question, the outcomes of all comparisons $f(\pi_p) : f(\pi_q)$ are identical to those of $\pi_p : \pi_q$ because the relative order of two permutations differ only with respect to $k+l$ in π and $k-l$ in $f(\pi)$, and these elements have not yet had their close comparison. Hence, the algorithm behaves in the same manner on $f(\pi)$ as it does on π, up to the close comparison which we will call v. In other words, the algorithm is unable to distinguish between π and $f(\pi)$ up to v.

The comparison on $f(\pi)$ corresponding to v will be the close comparison of $f(\pi_j) = k - 1$, and so be a straddle if $f(\pi_i) \in [k + 1, k + l]$ (i.e. $\pi_i \in [k, k + l - 1]$) and key if $f(\pi_i) \in [k - l + 1, k]$ (i.e. $\pi_i \in [k - l, k - 1]$). In other words the close comparison of $k + l$ in π is straddle if and only if the close comparison of $k - l$ is key.

The lemma now follows from the fact that f is a bijection.

A superficial analysis might suggest we have the desired result, since we can immediately conclude that the sum over all $l \in [1, \min(k - 1, n - k)]$ of the expected number of straddles involving $k + l$ and those involving $k - l$ is at least $\min(k - 1, n - k)$. We must ensure, however, that comparisons directly comparing $k + l$ and $k - l$ do not cause serious difficulties as the above analysis counts such straddles twice. This problem is alleviated by noting that such 'double close' comparisons occur rarely.

LEMMA 3.3. The probability that the close comparison of $k + l$ is also the close comparison of $k - l$ is at most

$$\frac{1}{2l} .$$

Proof. Consider the algorithm operating on some permutation, π, such that $k + l$ and $k - l$ happen to be directly compared and that comparison is close for both elements. Call such a permutation a *double close permutation* and such a comparison a *double close comparison.*

We will show that in any algorithm, for every double close permutation having a double close comparison at some internal node v in the decision tree, there exist (at least) $2l - 1$ permutations which are not double close and perform the same sequence of comparisons from the root to v in the decision tree. In particular we will partition a subset of Π into equivalence classes of size $2l$. Each equivalence class contains a double close permutation and the $2l - 1$ permutations which are not double close. Having proved this, it is immediate that the ratio

$$\frac{1}{2l}$$

is an upper bound on the probability of a double close comparison for $k + l$ and $k - l$ happening in any algorithm.

For each double close permutation π we define $2l-1$ functions denoted by g_i, $1 \leq i \leq 2l-1$, such that $g_i(\pi)$ differs from π in that the position containing $k-l$ in π is replaced by $k-l+i$ and all the positions in π containing elements in the subrange $[k-l+1, k-l+i]$ are decremented.

Note that excluding the two elements being compared at v, π and all the $g_i(\pi)$ are the same permutations in the sense that the remaining elements have the same relative order. Any comparison involving $k+l$ or $g_i(k-l)$ prior to v do not involve another element in the range $[k-l, k+l]$ and so have the same outcome as the counterparts in π. Hence, an algorithm acting on $g_i(\pi)$, $1 \leq i \leq 2l-1$, will follow the same sequence of comparisons from the root to v. Note also that the comparison at v for each $g_i(\pi)$ is not a double close comparison.

Consider how we could determine inverse images under g_j, $1 \leq j \leq 2l-1$ of $\pi' = g_i(\pi)$ where π is a double close permutation. An inverse must map the element to be compared with $k+l$ in its close comparison to $k-l$. This value is, then, $k-l+j$, fixing $j=i$. Hence the inverse is unique and the lemma follows as for each double close permutation we can exhibit at least $2l$ distinct permutations.

The previous lemma could obviously be strengthened since we did not consider all possible non double close permutations associated to a double close permutations. For example, by fixing the position of $k-l$, transformations similar to those defined in Lemma 3.3 (by fixing the position of $k+l$) can also be defined, and therefore larger groups of permutations can be obtained.

THEOREM 3.4. The expected number of straddle comparisons necessary to determine the k-th smallest of n elements is at least

$$\min(k-1, n-k) - \frac{1}{2} H_{\min(k-1, n-k)} .$$

Proof As noted above, it follows from Lemma 3.2 that the sum over all l in $[1, \min(k-1, n-k)]$ of the expected number of close comparisons for $k+l$ which are straddle plus that number for $k-l$ is $\min(k-1, n-k)$.

From Lemma 3.3 we see that the expected number of comparisons which are close for both elements is at most

$$\sum_{i=1}^{\min(k-1,\,n-k)} \frac{1}{2i} = \frac{1}{2} H_{\min(k-1,\,n-k)} \ .$$

From this, the theorem follows.

From Theorem 3.4, the observation that $n-1$ key comparisons are required for selecting any rank, and the Floyd-Rivest SELECT algorithm, our main result follows immediately.

THEOREM 3.5.

$$n + \min(k\,,\, n-k+1) - \frac{1}{2} H_{\min(k-1\,,\,n-k)} - 2 \leq \mathrm{M}(k\,;\,n) \leq n + \min(k\,,\, n-k+1) + \mathrm{O}(n^{1/2}) \ .$$

Finally, we have the important corollary for the case in which the median is selected.

COROLLARY 3.6.

$$\mathrm{M}(\lceil n/2 \rceil\,;\,n) = 1.5n \pm \mathrm{o}(n) \ .$$

CHAPTER 4

MULTIPLE SELECTION PROBLEMS

4.1. Selecting the maximum and any other rank.

If an element of a set of size n has rank k, its *relative rank* is defined to be the quotient k/n. Thus the relative rank of the maximum is 1 and of the minimum, $1/n$. Asymptotically, the relative rank of the elements can be seen to approach the continuous interval $(0\ ,\ 1]$.

In this section we will study the average case selection of the maximum and any other element with a given relative rank $\alpha \in (0\ ,\ 1]$. This will be called the max–α problem. Clearly, both upper and lower bounds apply to the symmetric problem of selecting the minimum and another rank.

We will present algorithms and lower bounds for different subintervals of $(0\ ,\ 1]$. For $\alpha \in [1/2\ ,\ 1]$ we present an algorithm and prove its optimality to within lower order terms. For $\alpha \in (0\ ,\ 1/2)$, we develop a family of methods, conjecturing that the corresponding lower bound, which we developed, is tight. The algorithms are based on sampling techniques and ways of partitioning small binomial trees with the cuts B and T. The lower bounds are derived from the average case lower bounds proven for the median selection and minmax problems.

Our most quotable result is that $1.75n \pm o(n)$ comparisons on average are necessary and sufficient for the max-median problem, i.e. finding the maximum and the median. For any α, the upper bounds supplied for the problem are strictly less than $2n$ comparisons.

4.1.1. The lower bounds.

Selecting the maximum, and an element x with relative rank α , can be viewed as two subproblems. The first is to partition the set into those elements with relative ranks in $(0\ ,\ \alpha)$, and those in $[\alpha\ ,\ 1]$. The second is to find the minimum and the maximum of the elements in $[\alpha\ ,\ 1]$. With this observation, the lower bound follows from the results derived in the previous two chapters.

THEOREM 4.1. Selecting the maximum and the element x with relative rank α from a set of n elements, requires on average, at least

$$\left(\frac{3}{2} + \frac{1}{2}\alpha \right) n - o(n) \text{ comparisons if } \alpha \in (0 , 1/2) ,$$

and

$$\left(\frac{5}{2} - \frac{3}{2}\alpha \right) n - o(n) \text{ comparisons if } \alpha \in [1/2 , 1] .$$

Proof. As noted before, the max-α problem must

i) find the elements with relative rank in $[\alpha , 1]$,

ii) determine the minmax of these elements.

Theorem 3.4 states that any algorithm to find an element of rank αn requires at least $\min(\alpha , 1 - \alpha)n - o(n)$ straddle comparisons. We emphasise that these straddle comparisons involve elements of relative rank in $(0 , \alpha)$. Second, $\alpha n - 1$ key-α comparisons involving elements with relative rank $(0 ,\alpha]$ are necessary. This gives a total of $\alpha n + \min(\alpha , 1 - \alpha)\, n - o(n)$ comparisons involving elements of relative rank in $(0 , \alpha)$.

Turning our attention to part (ii), we will claim that $1.5\,(1 - \alpha)n - 2$ comparisons are necessary between elements of relative rank in $[\alpha , 1]$. We argue by contradiction as follows: Assume an algorithm A performs fewer comparisons. Then a new algorithm B, based on algorithm A could take $(1 - \alpha)n$ elements of the set and add αn 'dummy elements' of 'value' $-\infty$. B also conditions the execution of

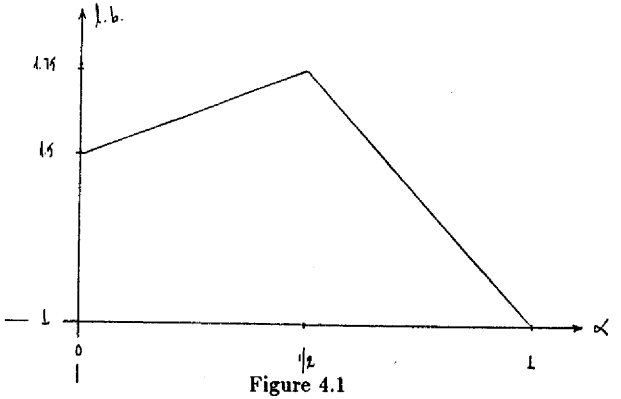


Figure 4.1

Average case max-α lower bound

every pairwise comparison in A by first comparing the two elements involved in the comparison with $-\infty$, allowing the pairwise comparison only if both elements are different from $-\infty$. Observe that by conditioning in B the pairwise comparisons performed in A, only pairwise comparisons on the $(1-\alpha)n$ elements chosen initially will be performed in B. As the number of pairwise comparisons is our complexity measure, algorithm B performs fewer comparisons than stated in Theorem 2.2. Thus A requires at least $1.5(1-\alpha)n - 2$ comparisons as claimed.

Observing that the types of comparisons, which we count, are disjoint, their sum leads to a total of

$$\left\{ \frac{3}{2} + \frac{1}{2}\alpha \right\} n - o(n)$$

comparisons if $\alpha \in (0\ ,\ 1/2)$, and,

$$\left\{ \frac{5}{2} - \frac{3}{2}\alpha \right\} n - o(n)$$

comparisons if $\alpha \in [1/2\ ,\ 1]$.

This lower bound has its maximum value in the case of selecting the median and the maximum of the set.

COROLLARY 4.2. Selecting the median and the maximum of a set of n elements requires at least

$$1.75n - o(n)$$

comparisons in average.

Figure 4.1 displays the linear term coefficient of the lower bound expression, as a function of α . In the case of selecting the minimum instead of the maximum, the lower bound function is symmetric to the previous one at $\alpha = 1/2$.

4.1.2. The upper bounds.

The first algorithm we present is a modification of the Floyd-Rivest algorithm SELECT which finds elements with ranks larger than the median. Consider the $(1-\alpha)n + o(n)$ elements which are found to be greater than the bottom cut B. These elements are paired such that the pair minima will be bigger than

the top cut T. At this stage, the loser of each pair is not a candidate for the maximum, and the winner is very unlikely to be of relative rank αn . Therefore, this comparison is with the same previous probability, a key-max comparison for the pair minimum and and a key-α comparison for the pair maximum.

In any algorithm we develop, it should be understood that the step of sampling, and therefore guessing the element with relative rank α, consists of determining the cut satisfying all the properties discussed in Section 1.3 for the algorithm SELECT. The algorithm can be stated as,

i) Sample to guess x with relative rank α.
ii) The remaining elements are first compared with the bottom cut B.
iii) Pair the set of straddling elements (elements larger than B).
iv) For each such pair compare the pair loser with the top cut T. If it loses again, then compare the pair winner with T.
v) Find the maximum from the set of maxima larger than T.
vi) Select x from its subset.

Again, with probability $1 - o(1)$, step vi will be applied on the subset of elements falling in between the two cuts. The algorithm is schematically presented in Figure 4.3. Configurations with almost zero probability are not further developed in the figure, although they must be considered in any implementation of the algorithm.

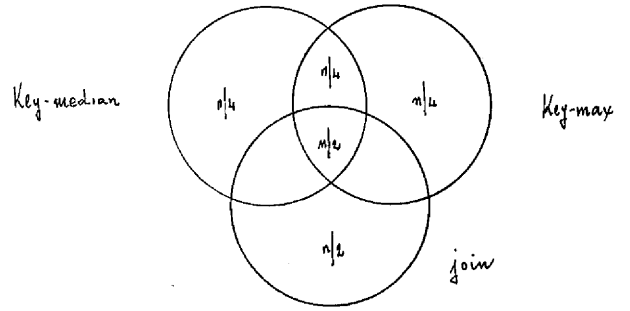


Figure 4.2
Classifying comparisons in max-median selection

LEMMA 4.3. The previous algorithm performs on average,

$$\left(\frac{5}{2} - \frac{3}{2}\alpha \right) n + o(n)$$

comparisons, when selecting an element of relative rank α .

Proof. A very brief analysis of the previous algorithm shows that step (i) takes $o(n)$ comparisons (the same as in SELECT). Step (ii) performs one comparison per singleton, that is in total $n - o(n)$ comparisons. Also, from the analysis of SELECT, we know that after the second step $(1 - \alpha)n + o(n)$ elements will be larger than B, so step (iii) makes $\frac{1-\alpha}{2} n + o(n)$ comparisons. For each pair obtained at step (iii), step iv will make a comparison with T, and an extra $\frac{1-\alpha}{2} n + o(n)$ comparisons are performed during this step. Since, with high probability, the pair minimum will be larger than T during the previous step, it is also expected that $\frac{1-\alpha}{2} n + o(n)$ pair maxima are involved in finding the maximum of the set at step (v). By adding all the comparisons made at each step, the lemma follows.

Of course, if the sampling at step (i) fails to satisfy the properties assumed above, the maximum has already been found in $O(n)$ comparisons, and x can be computed by simply sorting the subset in which it belongs, then performing at most $O(n\log(n))$ comparisons. Since this events happens with probability $o(1/n)$, its contribution in the total average performance is $o(n)$ comparisons only.

Comparing Lemma 4.3 and Theorem 4.1, we can deduce that for $\alpha \in [1/2 , 1]$ the algorithm is optimal up to lower order terms. Surprisingly, for $\alpha = 1/2$, this gives $1.75n + o(n)$ comparisons. Thus, when selecting the median, an extra $0.25n + o(n)$ comparisons are sufficient to determine the maximum of the set as well.

It is helpful to classify the comparisons performed by the algorithm. Figure 4.2 illustrates the kinds of comparisons and their numbers (lower order terms are not considered), when $\alpha = 1/2$. Better upper bounds are obtained when larger numbers of comparisons which belong to the intersections of the comparison sets, are performed.

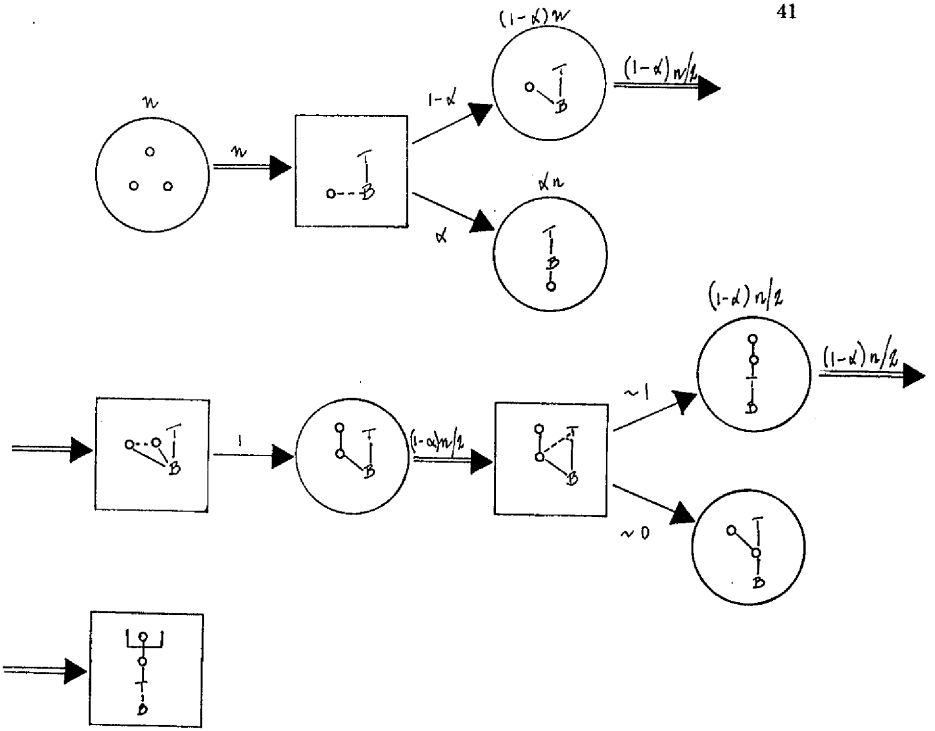


Figure 4.3
The optimal max-α algorithm for $\alpha \in [1/2, 1]$

The previous algorithm is not suitable for relative ranks $\alpha \in (0, 1/2)$. For example, it solves a problem close to minmax by performing $2.5n + o(n)$ comparisons. This is a poor result, since we already know that the algorithm presented in Section 2.1 makes $1.5n - 2$ comparisons for the same problem. One reason for such behaviour is due to the bad choice of cut for the singletons first comparison. An easy modification is, to compare the singleton's with the top cut T first. More precisely,

i) Sample to guess x with relative rank α.
ii) Compare the remaining singletons against T.
iii) Compare the straddling singletons (smaller than T), with B.
iv) The maximum is computed from the set of maxima larger than T.
v) Select x from its subset.

It is not difficult to see that this algorithm will perform $2n + o(n)$ comparisons regardless of α. After step (ii), $(1 - \alpha) - o(n)$ elements will be candidates for the maximum, and $\alpha n + o(n)$ elements will straddle. The two methods have the same running time for $\alpha = 1/3$. The behaviour of the second algorithm is still not satisfactory, since when α approaches 0, we would like the upper bound to converge to $1.5n + o(n)$. A look at the kind of comparisons performed by the second algorithm when selecting the minimum as well, shows that no comparison in the intersection of key-min, key-max and join comparisons is performed (see Figure 4.4). At this point we conclude that 'minmax like' comparisons (pairing of singletons) should be made. In a further modified algorithm, singletons are paired before comparing them against the cuts.

Because each input permutation is equally likely, each pair would be partitioned by x (if it were known), in such a way that, with binomial probability of,

i) $(1 - \alpha)^2$, the pair is bigger than x,

ii) $2\alpha(1 - \alpha)$, x is between the two elements,

iii) α^2, x is bigger than the pair

(see Figure 4.5).

The algorithm can be stated as,

i) Sample to guess x with relative rank α.

ii) Pair the remaining singletons, calling a the winner and b the loser for each pair.

iii) For each pair compare b and T.

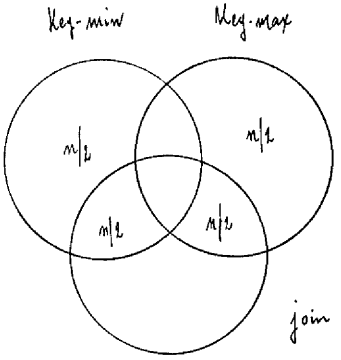


Figure 4.4

Types of comparisons in the second algorithm

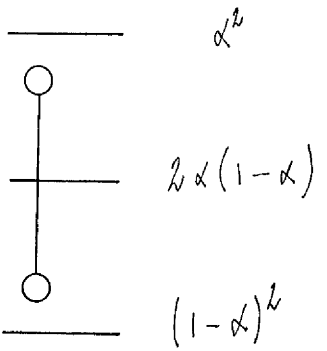


Figure 4.5
Partitioning a pair

iv) For pairs straddling at step (iii) (b smaller than T), compare a with T.

1) If a is larger than T then compare b with B.

2) Otherwise, compare a and B. If a is larger than B, compare b with B also.

v) Select the maximum from the maximal elements.

vi) Select x from its subset.

Figure 4.6 is a schematic presentation of the algorithm stated above (again, configurations happening with negligible probability are not shown).

Due to the properties of B and T, the probabilities, when partitioning the pairs by using the cuts instead of the actual x, will differ by a negligible o(1) term.

LEMMA 4.4. The algorithm stated above performs on average,

$$\left\{ \frac{3}{2} + 2\alpha - \frac{3}{2}\alpha^2 \right\} n + o(n)$$

comparisons, when selecting an element of relative rank α .

Proof. From Figure 4.6 (lower order terms are not shown), observe that (neglecting lower order terms) $n/2$ comparisons are performed during step (ii), and another $n/2$ comparisons are performed at step (iii). During step (iv), $\alpha(2-\alpha)$ comparisons are made, not including the $\frac{\alpha(1-\alpha)}{2}\, n$ comparisons at substep (1)

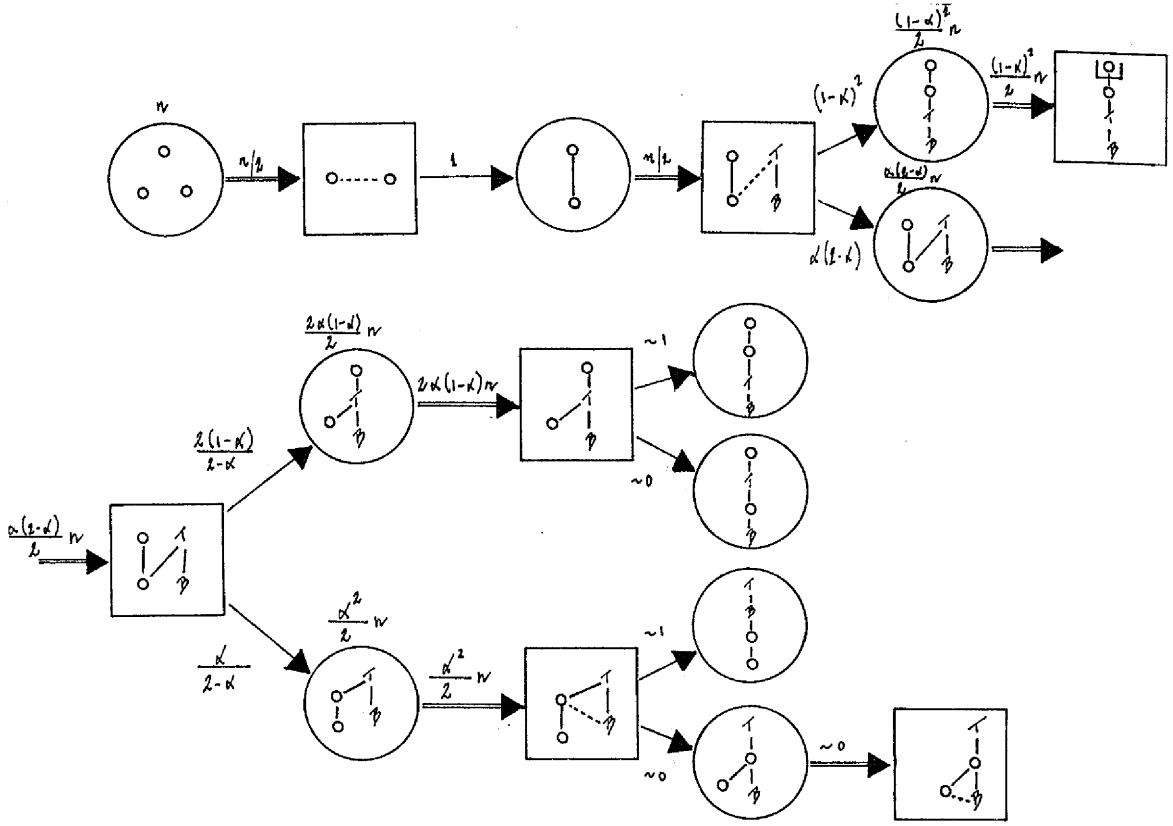


Figure 4.6
Algorithm for partitioning pairs

and, the $\frac{\alpha^2}{2}\,n$ comparisons during substep (2). At step v, two subsets of maximal elements one with $\frac{(1-\alpha)^2}{2}\,n$ and the other with $\alpha(1-\alpha)n$ elements are involved in the computation of the maximum of the set.

Again, since the probability of failure at step (i) is $o(1/n)$, sorting is suitable in the case of such a failure for selecting x, but on average this only adds $o(n)$ comparisons to the overall performance.

An important observation is that, when α approaches 0, the upper bound supplied by this algorithm converges to $1.5n + o(n)$. This algorithm and the first one proved in Lemma 4.3, meet each other at $\alpha = 1/3$ with value $2n + o(n)$.

Another improvement can be made in the region near $\alpha = 1/3$. The goal is to ensure that for any α, the max-α problem can be solved in less than $2n$ comparisons. The main idea is to build a more complex structure, a binomial tree of size 4, before partitioning it with the cuts.

Again, if the actual x is considered, the probabilities of the different ways of x splitting the structures depend upon the frequencies of the total orders consistent with the partitions, and the binomial distribution of elements falling below and above x. Figure 4.7 shows what these probabilities are. They have also been grouped in terms of the situations presented during different steps of the algorithm. Note for example, the probability of partitioning the structure with one element below x and three above x is $4(1-\alpha)^3\alpha$. There are two ways of making this partition (follow lines with label 1 in Figure 4.7), the first allowing only one total order $a > c > d > b$, and the second allowing two, $a > b > c > d$ and $a > c > b > d$. As all total orders are equally likely, the probability associated with the first case is $\frac{4}{3}(1-\alpha)^3\alpha$ and for the second case $\frac{8}{3}(1-\alpha)^3\alpha$. The remaining probabilities are derived in a similar way.

The decision of what element and cut to choose when making a comparison during the partitioning process is

i) Choose the element having the smallest difference in probabilities of being above and below x.

ii) Then, select the cut corresponding to the larger probability in (i). If the probability of being above x is larger, select T, otherwise select

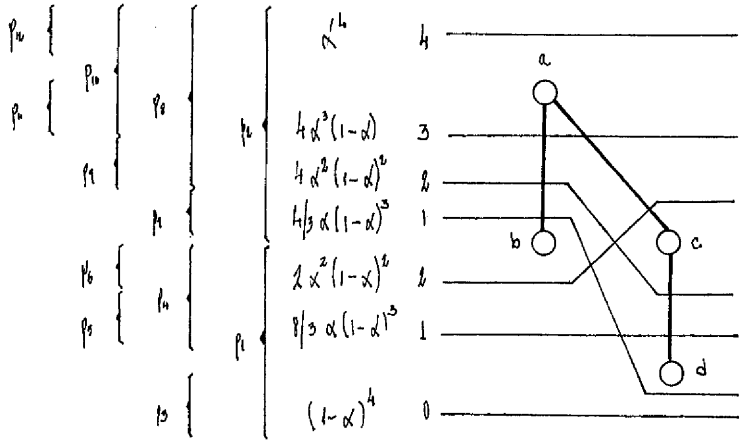


Figure 4.7
Partitioning a binomial tree of 4 elements

B.

The algorithm can be stated informally as:

i) Sample to guess x with relative rank α.

ii) Build with the singletons binomial trees of size four.

iii) Partition the small structure with the cuts.

iv) Find the maximum by comparing the maximal elements obtained after the partition.

v) Select x from its subset.

Clearly, the most important step is the third one partitioning binomial trees of size four. Briefly,

For each tree, compare the element labeled b (see Figure 4.7) with T.

If b wins, compare d with B.

If d wins, compare it with T, in which case virtually the whole tree is above T.

Otherwise, d having lost to B, compare c with T, and finally with B if c loses its previous comparison.

Else if b loses against T, d is compared with B.

If d wins, with high probability a and c are larger than T and b smaller than B; consequently, c is compared with T, and b is compared with B.

Else, if d loses to B, c is compared with T.

If c wins, b is the only one left to be compared with B.

Otherwise in case of c losing the previous comparison, a is compared with T.

If a wins the comparison, b and c have still to be compared with B.

Else, it is sufficient to compare a with B, in which case the whole tree happens to be below B.

Figure 4.8 presents a description of the algorithm.

LEMMA 4.5. In total, the previous algorithm makes

$$\left(\frac{7}{4} + \frac{1}{3}\alpha + \frac{1}{2}\alpha^2 + \alpha^3 - \frac{19}{12}\alpha^4 \right) n + o(n)$$

comparisons on average, when selecting an element of relative rank α .

Proof. A brief analysis (discarding lower order terms) can be derived from Figure

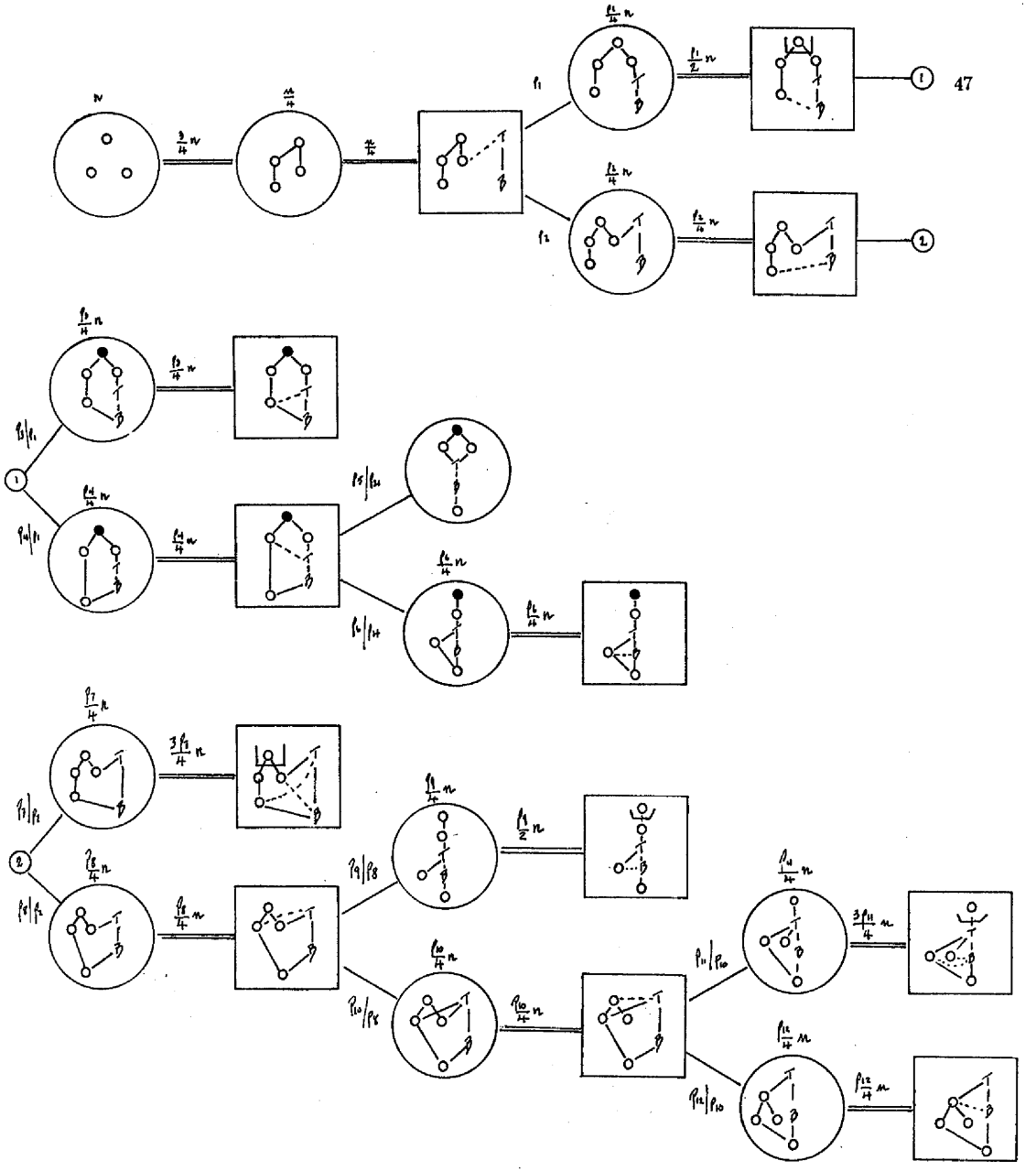


Figure 4.8
The algorithm for α close to 1/3

4.8, by adding the comparisons shown on the heavier arrows. The outcome probabilities for each type of comparison can be obtained from Figure 4.7.

Discarding lower order terms, the second step performs $\frac{3}{4}n$ comparisons. During the partitioning of the binomial trees of size four at step (iii),

$$\left(\frac{3}{4} + \frac{1}{3}\alpha + \frac{1}{2}\alpha^2 + \alpha^3 - \frac{4}{3}\alpha^4 \right) n + o(n)$$

comparisons are performed. There are $\frac{1-\alpha}{4}\ n$ maximal candidates for the maximum remaining after the third step. The number of comparisons performed by steps (i) and step (v), and the case of step (i) missing the target element x can all be neglected.

The function obtained above has to be compared with the previous two upper bounds obtained for the subintervals $(0\ ,\ 1/2)$ and $[1/2\ ,\ 1]$. Let p_1 and p_2 denote the intersection points with the other two curves derived in Lemma 4.4 and Lemma 4.3, respectively. Calculations show that $p_1 \in [\ 0.203\ ,\ .204\ ]$ and $p_2 \in [\ 0.362\ ,\ 0.363]$. As the function is monotonically increasing in the subinterval $[p_1\ ,\ p_2]$, the maximum in the subinterval is given at p_2 , with value around $1.957..\ n + o(n) < 2n$.

The next Theorem summarises the best upper bounds obtained for different values of α from Lemma 4.3 through Lemma 4.5.

THEOREM 4.6. On average, the max–α problem can be solved in the following number of comparisons,

i) $\left(\frac{3}{2} + 2\alpha - \frac{3}{2}\alpha^2 \right) n + o(n)$, if $\alpha \in (0\ ,\ p_1]$,

ii) $\left(\frac{7}{4} + \frac{1}{3}\alpha + \frac{1}{2}\alpha^2 + \alpha^3 - \frac{19}{12}\alpha^4 \right) n + o(n)$, if $\alpha \in (p_1\ ,\ p_2]$,

iii) $\left(\frac{5}{2} - \frac{3}{2}\alpha \right) n + o(n)$, if $\alpha \in (p_2\ ,\ 1]$.

for constants p_1 and p_2 , where, $p_1 \in [\ 0.203\ ,\ .204\ ]$ and $p_2 \in [\ 0.362\ ,\ 0.363]$.

These bounds are strictly less than $2n$. For $\alpha \in [1/2\ ,\ 1]$ it is within a lower order term of optimal, and for $\alpha = 1/2$ its value is $1.75n + o(n)$.

COROLLARY 4.7. For all α, the max-α problem can be solved in an average of $1.958n + o(n)$ comparisons.

Figure 4.9 displays the linear term coefficient of the upper bound expressions as a function of α. Through the development of these upper bounds, it becomes clear that the algorithms for the max-α problem in the subinterval $(0\ ,\ 1/2)$ have a radically different behaviour to those for α in the subinterval $[1/2\ ,\ 1]$. In the second subinterval, the optimal algorithm develops virtually no substructure at all. Singletons are compared directly with the cuts. However, when α is in the first half of the interval, very complex procedures and structures have to be designed in order to match the lower bound. We conjecture that indeed, the lower bound for the first half of the interval can be approached by buiding a binomial tree with all the elements and partitioning it in the right way. We have not been able to reach this solution yet, leaving it as an open question.

CONJECTURE 4.8. The upper bound on the average number of comparisons for the max-α problem, α restricted to the subinterval $(0\ ,\ 1/2)$, is

$$\left(\frac{3}{2} + \frac{1}{2}\alpha\right) n + o(n)\ .$$

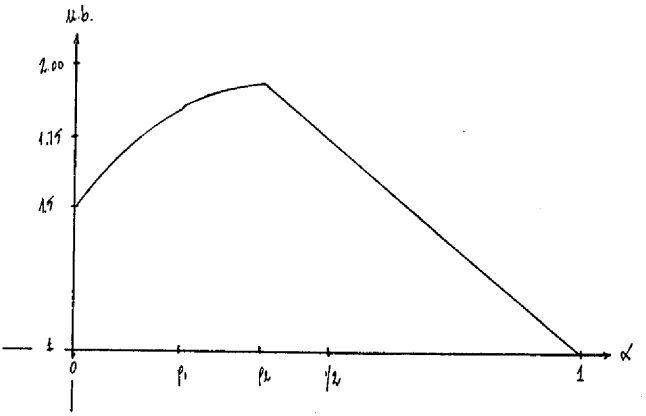


Figure 4.9
Max-α upper bounds

4.2. Selecting the maximum, the minimum and another element.

Selecting the median and the maximum can be carried out in $1.75n + o(n)$ comparisons and this is optimal. Hence, it is quite reasonable to assume that selecting the minimum, the maximum and the median should be possible in at most $2n + o(n)$ comparisons. Although we are able to prove a lower bound of $2n + o(n)$ comparisons for this problem, we have not been successful in obtaining the corresponding upper bound. The best upper bound we obtain is $2.25n + o(n)$ comparisons. That means that when finding the median you only have to pay as little as $0.25n + o(n)$ comparisons for one of the extreme elements, but for both, the cost is three times as much, viz $0.75n + o(n)$ comparisons. However, even if the algorithm we present is not optimal, it does have the virtue of simplicity.

In this section, we study upper and lower bounds for the problem of selecting the minimum, the maximum an element of relative rank α from a set of n elements. This particular problem is called the minmax-α problem. Lower and upper bounds for the problem follow in a similar manner to the corresponding bounds derived for the max-α problem.

THEOREM 4.9. On average,

$$\left\lceil \frac{3}{2} + \min(\alpha , 1 - \alpha) \right\rceil n - o(n)$$

comparisons are necessary to solve the minmax-α problem with n elements.

Proof. The lower bound is another corollary of the lower bounds for the minmax and selection problems. This time the problem consists of three subproblems,

i) selecting x with relative rank α,

ii) minmax on the smallest αn elements and,

iii) minmax on the remaining $(1 - \alpha)n$ elements.

From Theorem 3.4 we already know that the first subproblem requires $\min(\alpha , 1 - \alpha) - o(n)$ straddle comparisons. We emphasise that straddle comparisons must involve one element strictly less than and one greater than that of rank αn. The second subproblem requires $\frac{3}{2}\alpha n - 2$ comparisons among the smallest αn elements, otherwise a similar argument to the one presented in the proof of Theorem 4.1 can be given. For the third subproblem, $\frac{3}{2}(1 - \alpha)n - \frac{1}{2}$ comparisons among the $(1 - \alpha)n$ largest elements are required. As we can observe, the classes of counted comparisons are disjoint, so in total, $\left\lceil \frac{3}{2} + \min(\alpha , 1 - \alpha) \right\rceil n - o(n)$

comparisons on average are needed.

As the problem is symmetric about $\alpha = 1/2$, it need only be considered in the subinterval $[1/2, 1]$. The upper bounds we derive are similar to the corresponding ones for the max-α problem. The basic strategy is to pair in the subinterval with α relatively close to 1 (or 0), build binomial trees of 4 elements in the subinterval around the relative rank 2/3 (or 1/3), and perform comparisons of singletons with cuts for the subinterval around the median.

The first algorithm applies to values of α close to 1/2. It is assumed for simplicity that $\alpha \geq 1/2$.

i) Sample to guess x with relative rank α.
ii) The remaining elements are first compared with the bottom cut B.
iii) Elements smaller than B are compared again to find the minimum.
iv) Pair the set of straddling elements (elements larger than B). Since with very high probability (almost one), the pair minimum is larger than T, compare the pair losers with T.
v) Find the maximum from the maxima larger than T.
vi) Select x from its subset.

Figure 4.11 is a pictorial description of the algorithm.

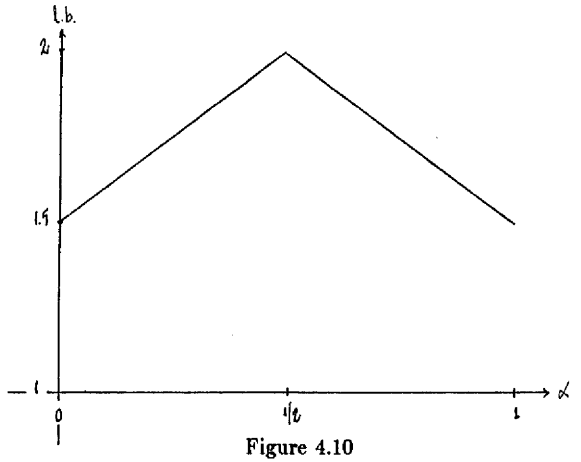


Figure 4.10
Average case minmax-α lower bound

LEMMA 4.10. The previous algorithm solving the minmax-α problem performs

$$\left[\frac{5}{2} - \frac{1}{2}\alpha\right] n + o(n)$$

comparisons on average.

Proof. The algorithm differs from that of Lemma 4.3 only with the inclusion of step (ii). Therefore, by Lemma 4.3, we know that

$$\left[\frac{5}{2} - \frac{3}{2}\alpha\right] n + o(n)$$

comparisons on average are made excluding step (ii)

The lemma follows by adding the αn comparisons of step (ii).

For $\alpha = 1/2$, the upper bound becomes $2.25n + o(n)$ comparisons on average. Consider the kinds of comparisons performed when selecting the median by using the above algorithm (lower order terms are discarded). After the second step is

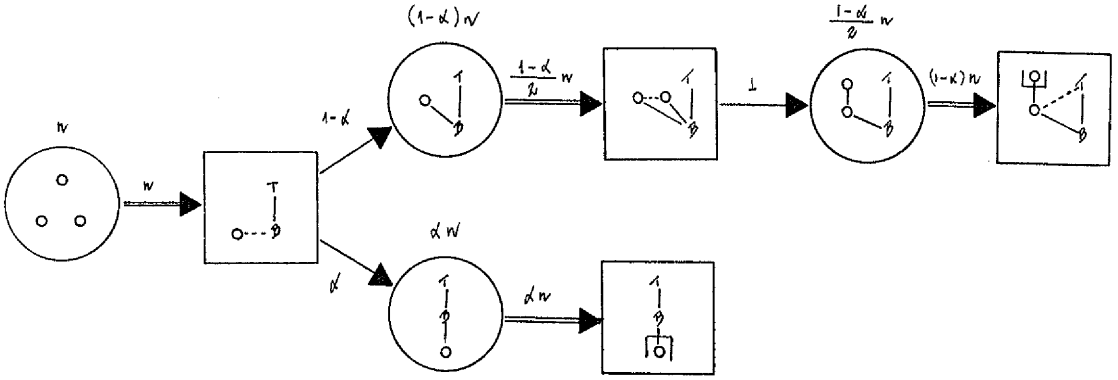


Figure 4.11
Minmax-α algorithm for $(1/2, p_3]$

completed, on average, $\frac{n}{2}$ comparisons are simultaneously key-median and key-max, while $\frac{n}{2}$ are key-min and straddle comparisons. After step (iv), the re-pairing will produce $\frac{n}{4}$ comparisons, each of which is key-max for the pair's loser and almost certainly key-med for the pair's winner. Note that the intersection of the subsets of key-min, key-median and key-max comparisons is empty.

As mentioned before, for α close to 1, minmax-like comparisons are suitable. Observe the repairing of elements performed at substep 1. These kinds of comparisons belong to the intersection of key-α and key-max comparisons.

i) Sample to guess x with relative rank α.

ii) Pair the remaining singletons, and for each pair call a the winner and b the loser.

iii) For each pair compare a with B.

iv) For pairs which straddle at step (iii) (a bigger than B), compare b with B.

1) If b is smaller than B, pair the set of a's. Then compare the losers of these new pairs with T, since the pairs are virtually larger than T.

2) If b wins the comparison, compare b with T. Note that b is with probability almost 1 larger than T.

v) Select the maximum from the remaining maxima.

vi) Select the minimum from the remaining minima.

vii) Select x from the subset in which it is contained.

Figure 4.12 describes the algorithm.

LEMMA 4.11. The above algorithm performs

$$\left[\frac{5}{2} + \frac{1}{2}\alpha - \frac{3}{2}\alpha^2 \right] n + o(n)$$

comparisons on average.

Proof. Steps (ii) and (iii) perform $n/2$ comparisons each. At step (iv), $\frac{(1-\alpha)(1+\alpha)}{2} n$ pairs straddle (lower order terms are not considered). Of these, $\alpha(1-\alpha)n$ elements are re-paired during substep (1), and $\frac{1-\alpha}{2} n$ are compared with T at substep (2). After the partitioning process, $\alpha(1-\frac{1}{2}\alpha)n$ minima are

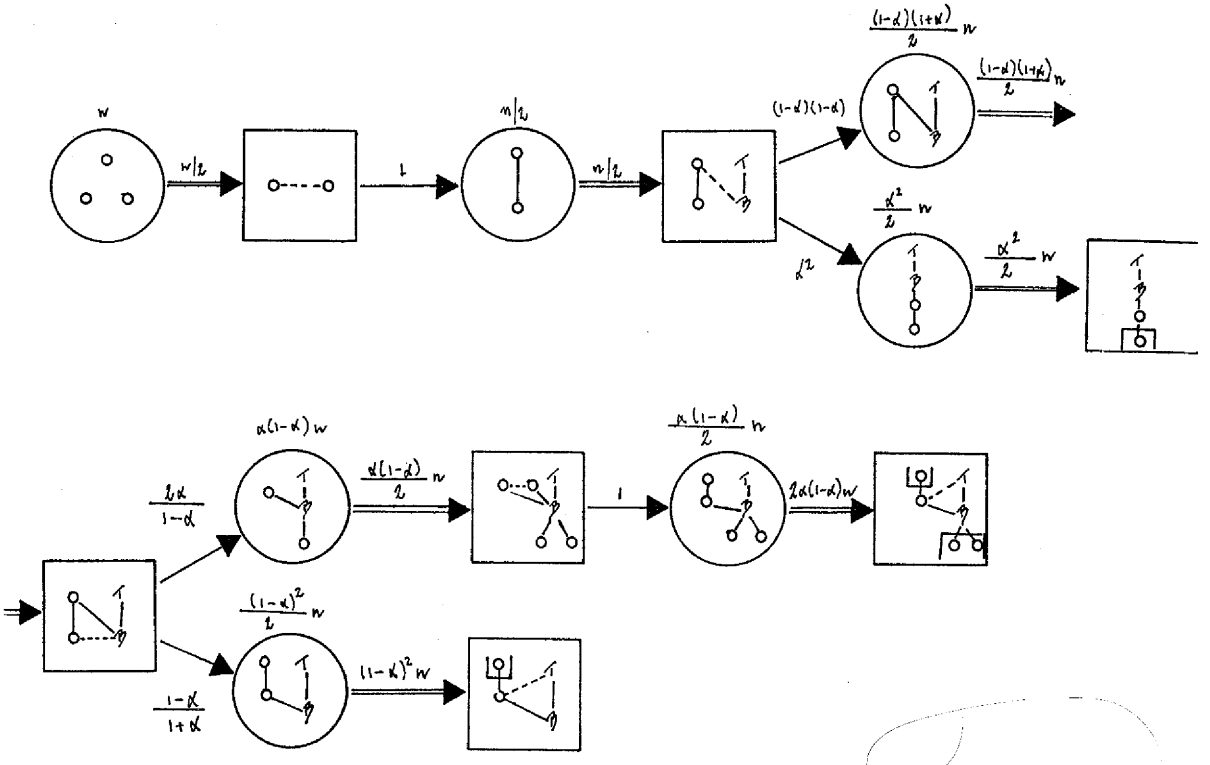


Figure 4.12
Minmax-α algorithm for $(p_4, 1]$

candidate for the minimum, and $\frac{1-\alpha}{2}\,n$ maxima remain candidates for the maximum.

Adding all the comparisons performed at each step and substep, the lemma then follows.

The last case to be considered is the subinterval around 1/3. Again, the main idea is to partition binomial trees of size four in such a way that as many comparisons as possible belong to more than one class. Briefly, the algorithm is

i) Sample to guess x with relative rank α.
ii) Build binomial trees of size four with the singletons.
iii) Partition the small structures with the cuts.
iv) Find the maximum from the remaining maximals.

v) Find the minimum from the remaining minimals.

vi) Select x from its subset.

The process of partitioning the binomial trees is similar to that appearing in the proof of Lemma 4.5. The only difference is to take advantage, when possible, of the re-pairing of elements in order to produce comparisons that belong to the intersection of two types of key comparisons. Figure 4.13 supplies the details of the algorithm.

LEMMA 4.12. On average ,

$$\left(\frac{7}{4} + \alpha + \frac{1}{2}\alpha^2 + \frac{1}{2}\alpha^3 - \frac{5}{4}\alpha^4 \right) n + o(n)$$

comparisons are performed by the above algorithm.

Proof. Steps (i) and (vi) are already known to be $o(n)$. Step (ii) requires $\frac{3}{4} n$ comparisons to build the binomial trees. Partitioning the binomial trees during step (iii) takes

$$\left(1 + \frac{2}{3}\alpha + \frac{1}{4}\alpha^4 + 2\alpha^3 - \frac{13}{6}\alpha^4 \right) n$$

comparisons. After the partition, $\frac{\alpha(1-\alpha)^2(1+2\alpha)}{3} n$ maxima are candidates for the maximum, and, $\frac{\alpha(2-2\alpha+\alpha^2)}{4} n$ minima are candidates for the minimum.

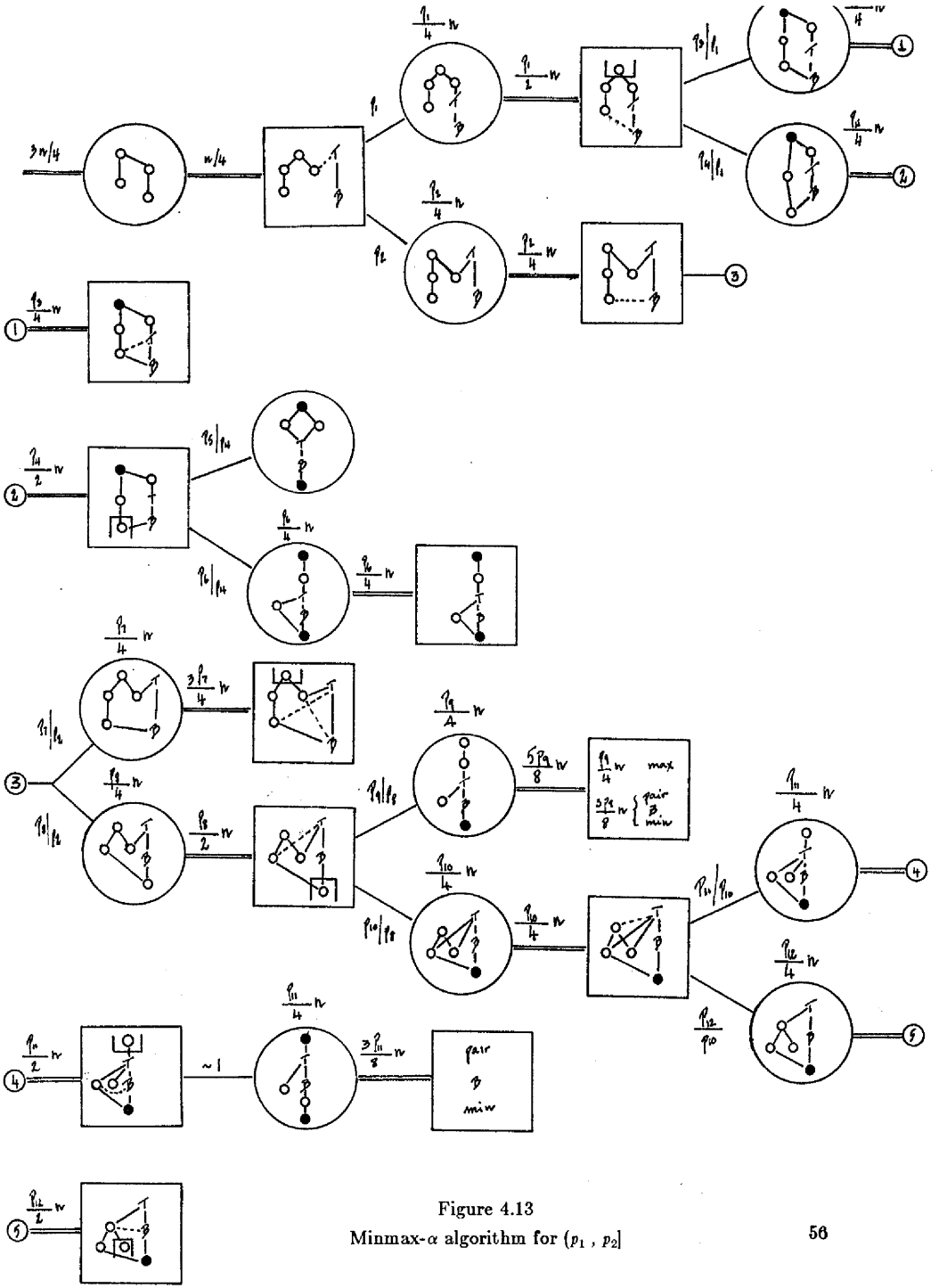


Figure 4.13
Minmax-α algorithm for $(p_1 , p_2]$

Our average case upper bounds for the minmax-α problem can be summarised

THEOREM 4.13.

$$\text{i)} \left(\frac{3}{2} + \frac{5}{2}\alpha - \frac{3}{2}\alpha^2 \right) n + o(n), \quad \text{if } \alpha \in (0, p_1),$$

$$\text{ii)} \left(\frac{7}{4} + \alpha + \frac{1}{2}\alpha^2 + \frac{1}{2}\alpha^3 - \frac{5}{4}\alpha^4 \right) n + o(n), \quad \text{if } \alpha \in (p_1, p_2],$$

$$\text{iii)} \left(2 + \frac{1}{2}\alpha \right) n + o(n), \quad \text{if } \alpha \in (p_2, 1/2],$$

$$\text{iv)} \left(\frac{5}{2} - \frac{1}{2}\alpha \right) n + o(n), \quad \text{if } \alpha \in (1/2, p_3],$$

$$\text{v)} \left(\frac{5}{2} + \frac{3}{2}\alpha - \frac{11}{2}\alpha^2 + \frac{9}{2}\alpha^3 - \frac{5}{4}\alpha^4 \right) n + o(n), \quad \text{if } \alpha \in (p_3, p_4],$$

$$\text{vi)} \left(\frac{5}{2} + \frac{1}{2}\alpha - \frac{3}{2}\alpha^2 \right) n + o(n), \quad \text{if } \alpha \in (p_4, 1],$$

for constants p_1, p_2, p_3, and p_4, where,

$$p_1 \in [0.256, 0.257], \; p_2 \in [0.363, 0.364], \; p_3 \in [0.636, 0.637], \; p_4 \in [0.743, 0.744].$$

Figure 4.14 illustrates the linear term coefficient as a function of α. Finally, we strongly believe that our algorithm solving the minmax-median problem is optimal up to lower order terms, and therefore, the lower bound is the one to be improved. Moreover for the minmax-α problem we conjecture

CONJECTURE 4.14. The complexity of solving the minmax-α problem is

$$\text{i)} \left(\frac{3}{2} + \frac{3}{2}\alpha \right) n \pm o(n), \quad \text{if } \alpha \in (0, \frac{1}{2}],$$

$$\text{ii)} \left(3 - \frac{3}{2}\alpha \right) n \pm o(n), \quad \text{if } \alpha \in (\frac{1}{2}, 1],$$

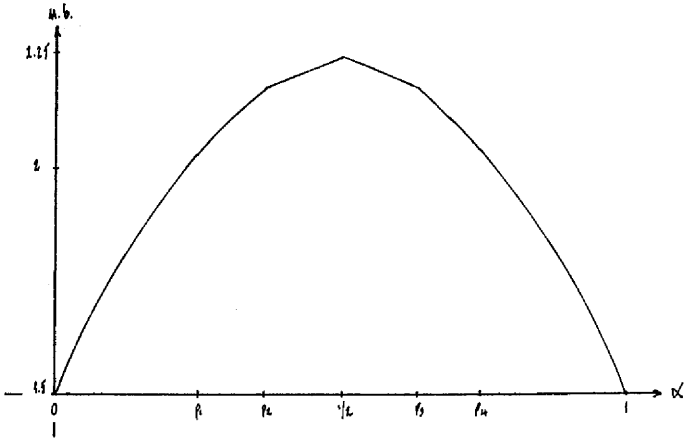


Figure 4.14
Minmax-α upper bounds

CHAPTER 5

THE CLOSEST NEIGHBOUR(S) PROBLEM

Consider the following problem: given a set X of n elements including a designated $x \epsilon X$, find the closest neighbour(s) of x in the set X. That is, denoting θ the rank operator, $y \epsilon X$ is a neighbour of x if and only if $| y \theta X - x \theta X | = 1$. The element x may have one or two neighbours depending on whether or not x is extreme in X. This condition is not known in advance.

We will study the worst case complexity of this problem as well as its average case. For the worst case, a simple algorithm making at most $2n - 3$ comparisons will be shown to be optimal. For the average case, we design an algorithm having similar running time to the selection algorithm SELECT. That is, if x is the k-th smallest in X, $n + \min(k, n - k + 1) + o(n)$ comparisons on average are performed by the algorithm. Averaging over all possible ranks of x, an expected run time of $\frac{5}{4}n + o(n)$ comparisons is obtained. If the rank of x is known, the problem reduces to selection. Therefore, the lower bound on selection applies to it.

5.1. Worst case optimality for closest neighbour(s).

An immediate algorithm for the problem is,

i) Partition X into $X_<$ and $X_>$, subsets of elements less and greater than x, respectively.

ii) Find the maximum in $X_<$ if the subset is not empty.

iii) Find the minimum in $X_>$ if the subset in not empty.

If $X_<$ and $X_>$ are both nonempty, the number of comparisons performed is $2n - 4$. Otherwise, if one of them is empty, only one neighbour is defined, and $2n - 3$ comparisons are performed. Note there are $n - 1$ elements in addition to x.

From the previous analysis we see:

LEMMA 5.1 Finding the closest neighbour(s) of $x \epsilon X$, $| X | = n$, requires at most $2n - 3$ comparisons.

This particular problem presents an interesting anomaly. If x turns out to have two neighbours, the algorithm performs faster (one comparison less) than the case when only one neighbour exists since x is an extreme in X. To take advantage of this anomaly, let us consider the simplified version of the problem, to find the closest left (smaller) neighbour of x, if this exists. Certainly the algorithm given above solves this variant, and any lower bound derived for the simplified version is a lower bound for the more general problem. By designing a simple adversary that resembles the one given in [Blum et al 73] for worst case selection, we will show that our algorithm is indeed optimal.

THEOREM 5.2 In the worst case, $2n-3$ comparisons are needed to determine whether a left neighbour of x exists, and if so, to find it. Hence, the above algorithm is optimal.

Proof. The adversary will partition $X-x$, at any time, into five different subsets.

i) X_v, the set of virgins having no comparisons associated with them,

ii) X_p, the set of pairs created from pairing virgins.

iii) $X_{2,>}$, the set of elements known to be greater than x, each element charged with two comparisons,

iv) $X_{2,<}$, the subset of elements less than x, and charged with two comparisons per element.

v) $X_{1,<}$, the subset of elements less than x, but larger than any element in $X_{2,<}$. Elements in this category are candidates for closest left neighbour, and each one of them has been charged with one comparison.

The only restriction on the adversary is that any given answer must be consistent. The relative order the adversary will always keep at any stage, is given in Figure 5.1. As an element is declared in $X_{2,<}$, it is declared to be smaller than any other element still in one of the other subsets. Similarly entry into $X_{2,>}$ declares an element larger than all remaining ones. $X_{2,<}$ and $X_{2,>}$ can be viewed as chains, which means that even though the adversary provides the total order on those elements, no useful information is gained toward solving the problem at hand.

If the algorithm chooses to compare two elements a and b, the adversary will answer according to the following rules,

Case 1 : $a \in X_v$.

i) If a is compared with x, a is transferred to $X_{1,<}$.

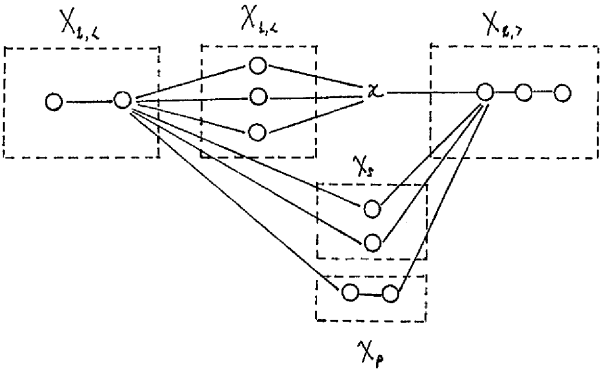


Figure 5.1
Partial order kept by the adversary

ii) If b is another virgin, they are paired together and inserted into X_p.

iii) If b is the minimum of a pair in X_p with pair maximum denoted by c, b is declared to be in $X_{2,<}$ (although for consistency b is larger than any element previously put in $X_{2,<}$) and c becomes a 'reconstituted' virgin.

iv) If b is the maximum of a pair in X_p with pair minimum denoted by d, b is declared to be the minimum in $X_{2,>}$ and d becomes a virgin again.

v) If $b \in X_{1,<}$, then b becomes the maximum of those currently in $X_{2,<}$.

Case 2 : a is the minimum of a pair in X_p, with corresponding maximum denoted by c.

i) If compared with x, a is declared to be the maximum in $X_{2,<}$ and c becomes a 'reconstituted' virgin.

ii) If $b \in X_{1,<}$ or either element of a pair then, a becomes the maximum in $X_{2,<}$, and c a 'reconstitute' virgin.

Case 3 : a is the maximum of a pair in X_p with corresponding minimum denoted by c.

i) If compared with x then a becomes the minimum in $X_{2,>}$ and c a 'reconstitute' virgin.

ii) If $b \in X_{1,<}$ or either element of a pair, b becomes the maximum in $X_{2,<}$.

Case 4 : $a, b \in X_{1,<}$, b becomes the maximum in $X_{2,<}$.

Any other case leads to a redundant comparison. In particular $X_{2,<}$ elements always lose and $X_{2,>}$ always win.

The computation begins with all elements in X_v and ends with all in $X_{2,<}$ or $X_{2,>}$ with the exception of one which may be in $X_{1,<}$. The latter condition occurs if x does have a left neighbour. The total charge to these $n-1$ elements is, then, $2n-3$.

5.2. Closest neighbour(s) average case.

Now consider the average number of comparisons necessary and sufficient to solve the same problem. The element x is randomly picked from the set X, therefore the probability of x having any rank in the set is $1/n$.

The algorithm we present is another application of the sampling technique also used in other probabilistic problems. We will consider the suitable sample size that will reduce, as much as possible, the lower order term. The algorithm can be stated as

i) Take a random sample S of size $s-1$ from $X-x$.

ii) Find a and b , the closest neighbours of x in S , by using the optimal worst case algorithm. This step and the previous one give ranks of a, x and b in S and so estimates of their ranks in X .

iii) If $x\,\theta\,(S \bigcup x) \neq 1$ and $x\,\theta\,(S \bigcup x) \neq s$, then partition the set $X-S$ into sets A , B , C , of elements less than a , between a and b , and greater than b , respectively, by comparing the singletons in $X-S$ against the two cuts a and b in such a way that if $x\,\theta\,S \leq s/2$ then the singletons are compared first with b and after with a if necessary, otherwise the order of comparisons is interchanged. Otherwise, the problem is solved by using the worst case optimal algorithm.

iv) Find the closest neighbours in B by using the worst case optimal algorithm.

Before analysing the algorithm, we will study the relation among ranks in the sample S and the set X. Let ρ denote the inverse function of θ . In general, if V is a random sample of size s , taken from a set T of size n , we denote

$$p_{(i,k),(j,r),s,n} = \Pr[(i\,\rho\,V)\,\theta\,T = k, (j\,\rho\,V)\,\theta\,T = r] ,$$

the probability of the i-th and the j-th smallest in V having ranks k and r in T, respectively. Then,

$$p_{(i,k),(j,r),s,n} = \frac{\left[\begin{matrix} k-1 \\ i-1 \end{matrix}\right]\left[\begin{matrix} r-k-1 \\ j-i-1 \end{matrix}\right]\left[\begin{matrix} n-r \\ s-j \end{matrix}\right]}{\left[\begin{matrix} n \\ s \end{matrix}\right]} ,$$

$$i \leq k \leq r-(j-i) ,$$

$$k+(j-i) \leq r \leq n-(s-j) .$$

By using the well known identity,

$$\sum_{j=i}^{n-(s-i)} \left[\begin{matrix} j-1 \\ i-1 \end{matrix}\right]\left[\begin{matrix} n-j \\ s-i \end{matrix}\right] = \left[\begin{matrix} n \\ s \end{matrix}\right] ,$$

we can observe that,

$$p_{(j,r),s,n} = \sum_{k=i}^{(r-1)-((j-1)-i)} p_{(i,k),(j,r)s,n}$$

$$= \frac{\left[\begin{matrix} r-1 \\ j-1 \end{matrix}\right]\left[\begin{matrix} n-r \\ s-j \end{matrix}\right]}{\left[\begin{matrix} n \\ s \end{matrix}\right]} ,$$

and,

$$p_{(i,k),s,n} = \sum_{r=k+j-i}^{n-(s-j)} p_{(i,j),(j,r)s,n}$$

$$= \frac{\left[\begin{matrix} k-1 \\ i-1 \end{matrix}\right]\left[\begin{matrix} n-k \\ s-i \end{matrix}\right]}{\left[\begin{matrix} n \\ s \end{matrix}\right]} ,$$

are both, negative hypergeometric distributions with known average, variance and ascending factorial moments [Feller 50] (see Figure 5.2).

Returning to the problem, we want to prove that

$$p_{(i,j),s,n} = \frac{\left[\begin{matrix} k-1 \\ s-1 \end{matrix}\right]\left[\begin{matrix} n-k \\ s-i \end{matrix}\right]}{\left[\begin{matrix} n \\ s \end{matrix}\right]} \qquad i \leq k \leq n-s+i$$

$$E\left[((i \rho V) \theta T)^{\overline{x}}\right] = i^{\overline{x}} \frac{(n+1)^{\overline{x}}}{(s+1)^{\overline{x}}}$$

$$V\left[(i \rho V) \theta T\right] = i \frac{(n+1)}{(s+1)^2} \frac{(s-i+1)(n-s)}{(s+2)}$$

Figure 5.2
Moments for a negative hypergeometric distribution

$$\Pr[\, x \theta X \mid x \theta (S \bigcup x) = i \,] = p_{(i,k),s,n} \ .$$

First,

$$\Pr[\, x \theta (S \bigcup x) = i \mid x \theta X = k \,] = \Pr[\, ((i-1) \rho S) \theta (X/x) \leq k-1 \,,$$

$$(i \rho S) \theta (X/x) \geq k \,]$$

$$= \sum_{r=k}^{(n-1)-((s-1)-1)} \sum_{j=i-1}^{k-1} \frac{\left[\begin{matrix} j-1 \\ (i-1)-1 \end{matrix}\right]\left[\begin{matrix} (n-1)-r \\ (s-1)-i \end{matrix}\right]}{\left[\begin{matrix} n-1 \\ s-1 \end{matrix}\right]}$$

$$= \frac{n}{s} \frac{\left[\begin{matrix} k-1 \\ i-1 \end{matrix}\right]\left[\begin{matrix} n-k \\ s-1 \end{matrix}\right]}{\left[\begin{matrix} n \\ s \end{matrix}\right]}$$

$$= \frac{n}{s} \; p_{(i,k),s,n} \quad .$$

Note that,

$$\Pr[\, x \; \theta \; (S \bigcup x) = i \,] = \sum_{k=i}^{n-(s-i)} \frac{n}{s} \; p_{(i,k),s,n} \frac{1}{n}$$

$$= \frac{1}{s} \quad ,$$

as it was expected.
Finally,

$$\Pr[x \; \theta \; X = k \mid x \; \theta \; (S \bigcup x) = i \,] = \frac{\Pr[x \; \theta \; (S \bigcup x) = i \mid x \; \theta \; X = k \,] \Pr[x \; \theta \; X = k \,]}{\Pr[x \; \theta \; (S \bigcup x) = i \,]}$$

$$= \frac{n}{s} \; p_{(i,k),s,n} \frac{1}{n} \; s \quad ,$$

$$= p_{(i,k),s,n} \quad .$$

implying that, picking x from X first, then choosing a sample of size $s-1$, and estimating the rank of x in X, is the same as picking a sample of size s first, then choosing x from the sample, and estimating its rank in X.

We are now ready for the analysis of the previous algorithm.

THEOREM 5.3. $\frac{5}{4} n + O(n^{1/2})$ comparisons are sufficient for computing the closest neighbours of x randomly picked from a set X of cardinality n.

Proof. We have to analyse three cases, when x is an extreme element in $S \bigcup x$, when the rank of x is between 2 and $s/2$, and when its rank is between $s/2 + 1$ and $s - 1$.
In the first case, the worst case optimal algorithm is applied, therefore, $2n - 3$ comparisons are performed by the algorithm.

When $x \; \theta \; (S \bigcup x) = i$, $2 \leq i \leq s/2$, with probability $p_{(i-1,k),(i+1,r),s,n}$, the two closest neighbours of x in $S \bigcup x$ have ranks k and r in X, respectively. Then

$2s-4$ comparisons are performed for computing a and b in $S \bigcup x$, $n-s$ comparisons of singletons in $X/(S \bigcup x)$ against b, $r-i-1$ comparisons of elements less than b against a, and $2r-2k-5$ comparisons for computing the neighbours in B. In total, $n+s+3r-2k-i-10$ comparisons. Averaging over r and k,

$$n+s+\frac{n+1}{s+1}\,i+5\,\frac{n+1}{s+1}-10$$

comparisons are performed, if $x\,\theta\,(S \bigcup x)$ is conditioned to be i.

The last case is symmetric to the previous one. Taking the average over all possible ranks of x in $S \bigcup x$, we get $\frac{5}{4}\,n+O(s+n/s)$. Optimising the lower order term, $s=O(n^{1/2})$, and the theorem follows.

The lower order term can be further improved from $O(n^{1/2})$ to $O(\log(n))$ by keeping the closest neighbours at any time during stage iii of the algorithm. The algorithm is also very simple. Let us call a and b the closest known neighbours of x at any stage of the computation. They will act in the same way as the two cuts in the selection algorithm. Also, A will denote the elements smaller than a, and B the ones larger than b. Then, the algorithm is,

i) Initially, the first singleton is compared with x, becoming one of the two closest neighbours.

ii) If only one neighbour is known, the next singleton is compared with it, and with x if necessary.

iii) If both neighbours are known, then compare the next singleton with a or b first, depending on which of the subsets A or B is bigger.

If the singleton straddles its first comparison, then compare it with the other cut.

If the element lies between a and b, then compare it with x and appropriately update one of the closest neighbours.

The algorithm can be modelled as a Markovian process, in which the states and transitional probabilities depend upon the sizes of both subsets. Let $C_{k,n}$ be the average number of comparisons performed after n singletons have been introduced in the process, $k-1$ of them being in A. It is not difficult to express $C_{k,n}$ as a recurrence. In fact, if $k \leq n/2$,

$$C_{0,n} = C_{0,n-1} + 1 + \frac{1}{n} \ ,$$

$$C_{1,n} = \frac{1}{2}C_{0,n-1} + \frac{1}{2}C_{1,n-1} + \frac{3}{2} + \frac{1}{n-1} \ ,$$

$$C_{k,n} = \frac{1}{2}C_{k-1,n-1} + \frac{1}{2}C_{k,n-1} + \frac{3}{2} + \frac{1}{2k} + \frac{1}{n-1} \quad 2 \leq k \leq n/2 \ ,$$

$$C_{0,0} = 0 \ .$$

An easy induction shows that,

$$C_{k,n} \leq n + k + H_k + 2H_{n-k} \ .$$

Averaging on k, the final running time is,

$$\frac{5}{4}\, n + O(\log n) \ .$$

Note that the selection lower bound applies to the problem if the rank of x, the random element, is known in advance, since in this case, the problem is converted into the selection of two elements with known rank.

CHAPTER 6

DIRECTIONS FOR FUTURE RESEARCH

Although, the two conjectures presented in Chapter 4 are the next natural steps to study in the direction defined by this work, there are other interesting problems that also should be considered.

We observe that the technique of sampling can also be used for external sorting. In this case, tradeoffs between the memory available and the number of I/O operations constitutes an important issue to consider when designing efficient algorithms. The idea is simple and deals with distributing the external file into a sequence of subfiles in such a way that any element in a subfile is less than any element in the next subfile in the sequence. Informally,

i) Read a random sample from the external file into memory.

ii) Sort the sample in memory.

iii) Distribute the file into subfiles according to the sorted sample.

iv) For each subfile, apply step (i) to (iii) recursively or any other standard procedure.

A superficial analysis shows that if $O(n^{1/2})$ memory space is available, and a sample of similar size is read into memory, then $O(n \log\log(n))$ I/O operations are sufficient.

Looking at other instances of the multiple selection problem, the best lower and upper bounds obtained so far for the expected number of comparisons when selecting the quartile elements (relative rank 1/4 and 3/4 respectively) are $2n - o(n)$ and $2.5n + o(n)$. The same results apply for some other instances such as selecting the 1/3 and 2/3 elements. As mentioned in the introduction (Chapter 1) finding the minimum, the maximum, the median and the two quartile elements of a set is a commonly required in performing statistical analyses. At this point, a simple algorithm can be designed. The ideas are,

i) Select the median and therefore partition the set into two halves.

ii) Apply the optimal min-median algorithm to the elements below the median thus obtaining the minimum and the lower quartile of the entire set.

iii) Apply the optimal max-median algorithm to the upper half to obtain the upper quartile and the maximum of the entire set.

This algorithm selecting all five ranks shows a reasonable average running time of $3.25n + o(n)$ comparisons. The question that still remains is of course, to find a faster algorithm or to prove a tight lower bound.

We believe that in order to analyse more complex instances of the multiple selection problem, stronger models than the one developed in Chapter 4 have to be devised.

Bibliography

[Bartels 80] Bartels R. H.;
A Penalty Linear Programming Method Using Reduced-Gradient Basis-Exchange Techniques;
Linear Algebra and Its Applications, 2:17-32 (1980).

[Blum et al 73] Blum M., Floyd R. W., Pratt V. R., Rivest R. L., Tarjan R. E.;
Time Bounds for Selection;
JCSS 7:448-461 (1973).

[Cunto-Munro 83] Cunto W., Munro J. I.;
Multiple Selection Worst Case;
Unpublished Notes.

[Dobkin-Munro 81] Dobkin D., Munro J. I.;
Optimal Time Minimal Space Selection Algorithms;
JACM 28:454-461 (1981).

[Feller 50] Feller W.;
An Introduction to Probability;
Addison Wesley (1950).

[Floyd-Rivest 73] Floyd R. W., Rivest R. L.;
Bounds on the Expected Time for Median Computations;
Courant Computer Science Symposium 9, Randall Rustin, Algorithmic Press, New York, pp 69-76 (1973).

[Floyd-Rivest 75] Floyd R. W., Rivest R. L.;
Expected Time Bounds for Selection;

CACM 18:165-172 (1975);
and,
Algorithm 489, The Algorithm SELECT for Finding the i-th Smallest of n elements;
CACM 18:173 (1975).

[Fussenegger-Gabow 79] Fussenegger F., Gabow H.;
A Counting Approach to Lower Bounds for Selection Problems;
JACM 26:227-238 (1979).

[Geddes et al 82] Geddes K. O., Gonnet G. H., Char B. W.;
MAPLE User's Manual;
University of Waterloo, Research Report CS-82-40 (1982).

[Hillier-Lieberman 74] Hillier F. S., Lieberman G. J.;
Introduction to Operation Research;
Second Edition, Holden-Day Inc. (1974).

[Hoare 61] Hoare C.A.R.;
Algorithm 63 PARTITION
and,
Algorithm 65 FIND;
CACM 4:321 (1961).

[Knuth 71] Knuth D. E.;
Mathematical Analysis of Algorithms;
Stanford University, Report STAN-CS-71-206 (1971).

[Knuth 73] Knuth D. E.;
The Art of Computer Programming: Sorting and Searching (Vol 3);
Addison Wesley (1973).

[Matula 73] Matula D.;
Selecting the k-th Best in Average n + O(lglgn) Comparisons;
Washington University (St. Louis), TR 73-9 (1973).

[Munro-Poblete 82] Munro J. I., Poblete P. V.;
A Lower Bound for Determining the Median;
University of Waterloo, Research Report CS-82-21 (1982).

[Munro-Spira 76] Munro J. I., Spira P.;
Sorting and Searching in Multisets;
SIAM J. COMPT. 5(1):1-9 (1976).

[Pan 66] Pan V. Y.;
Methods of Computing Values of Polynomials;
Russian Mathematical Surveys, 21:105-136 (1966).

[Pohl 72] Pohl I.;
A Sorting Problem and Its Complexity;
CACM 15:462-464 (1972).

[Schonhage 73] Schonhage A.;
Unpublished Manuscript (1973).

[Schonhage et al 76] Schonhage A., Paterson M., Pippinger N.;
Finding the Median;
JCSS 13:184-199 (1976).

[Yao 73] Yao F. F.;
On Lower Bounds for Selection Problems;
MAC TR-121 (1973).

[Yao-Yao 82] Yao A., Yao F. F.;
On the Average-case Complexity of Selecting the k-th

Best;
SIAM J. COMPT. 11:428-447 (1982).