

Chapter 5: Selection and Adversary Arguments

Selection is the problem of finding, given a set of some n keys and an integer k , $1 \leq k \leq n$, the k^{th} smallest number in the set. The number in question is called the k^{th} **order statistic**.

1. Finding the Maximum

To find the maximum of n keys using key comparison, $n - 1$ comparisons are necessary and sufficient. The reason is that a number is known to be the maximum only if everyone else is lost at least once. So, there have to be at least $n - 1$ comparisons.

A same argument holds for the minimum.

Finding both the Maximum and the Minimum

Theorem A To find both the maximum and the minimum of n keys using key comparison, $3n/2 - 2$ key comparisons are necessary and sufficient.

Sufficiency Consider the following strategy that deals with the input numbers in pairs:

- Take the first pair, (x, y) . Compare x and y ; Set A to the larger and B to the other.
- For each remaining pair, (x, y) , compare x and y ; set u to the larger and v to the smaller; set A to the larger of u and A ; set B to the larger of v and B .
- If one key remains, compare it with A and B .

The number of comparisons is $\lceil 3n/2 \rceil - 2$.

Necessity

We will show that for each algorithm that makes less than $3n/2 - 2$ key comparisons, there is an input for which the algorithm either fails to compute the maximum or fails to compute the minimum.

Adversary Argument

To prove this we will use what is called an **adversary argument**.

An **adversary** is an opponent that a key-comparison algorithm plays against. Its ultimate goal is to maximize the number of key comparisons that the algorithm makes while constructing an input to the problem. At the beginning there is no restriction on the input, but when the algorithm asks about a pair (a, b) of keys the adversary must return either $a < b$ or $a > b$, then only the inputs that are inconsistent with the answer will be removed from further consideration.

Assumptions

We assume the following:

- The keys are pairwise distinct.
- The algorithm cannot remember an key.
- At any point during computation, the algorithm can infer $>$ or $<$ based on the outcomes of the comparisons that have been made so far, and the adversary's answers should never contradict them.

Graph Representation

The outcomes of the comparisons that have been made so far can be viewed as a directed graph, where the nodes represent the keys and the arcs represent the outcomes.

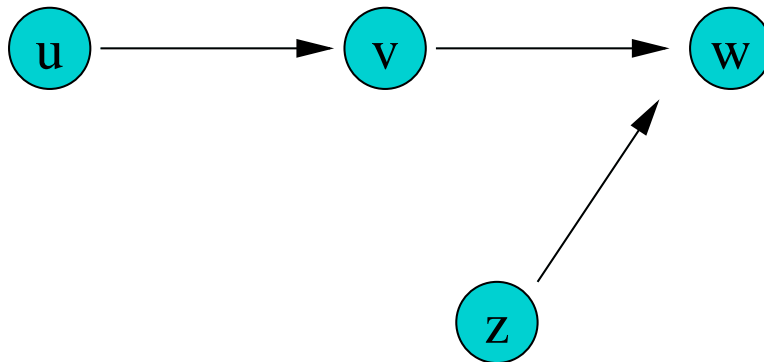
We draw an arc from a node u to a node v if the algorithm has asked about the pair (u, v) and the strategy has provided an answer $u < v$.

A relation $a < b$ is inferred from the previous outcomes if and only if there is a path from a to b .

Also, the graph has no cycles.

Example

The comparisons that have been made are $u < v$, $v < w$, and $z < w$. It can be inferred that $u < w$. So, when asked about the pair (u, w) , the strategy must answer $u < w$.



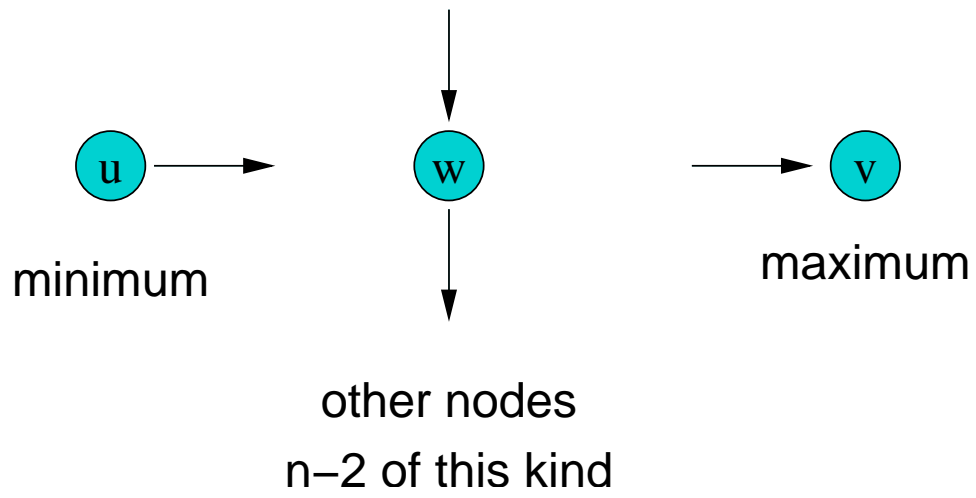
Stopping Condition

The algorithm stops examination when the graph acquires a certain property. For computing the maximum and the minimum, the stopping condition is when

- there is a node from which all the other nodes are reachable (thus, it's the minimum), and
- there is a node to which all the other nodes are reachable (thus, it's the maximum).

Since there is no cycle, when the stopping condition holds,

- each of the other two nodes has at least one incoming edge and at least one outgoing edge.



Terminology

We say that a key x **wins** at y (respectively, **loses** to y) if x and y have been compared and the adversary has assigned a value to each of the two so that $x < y$ (respectively, so that $x > y$).

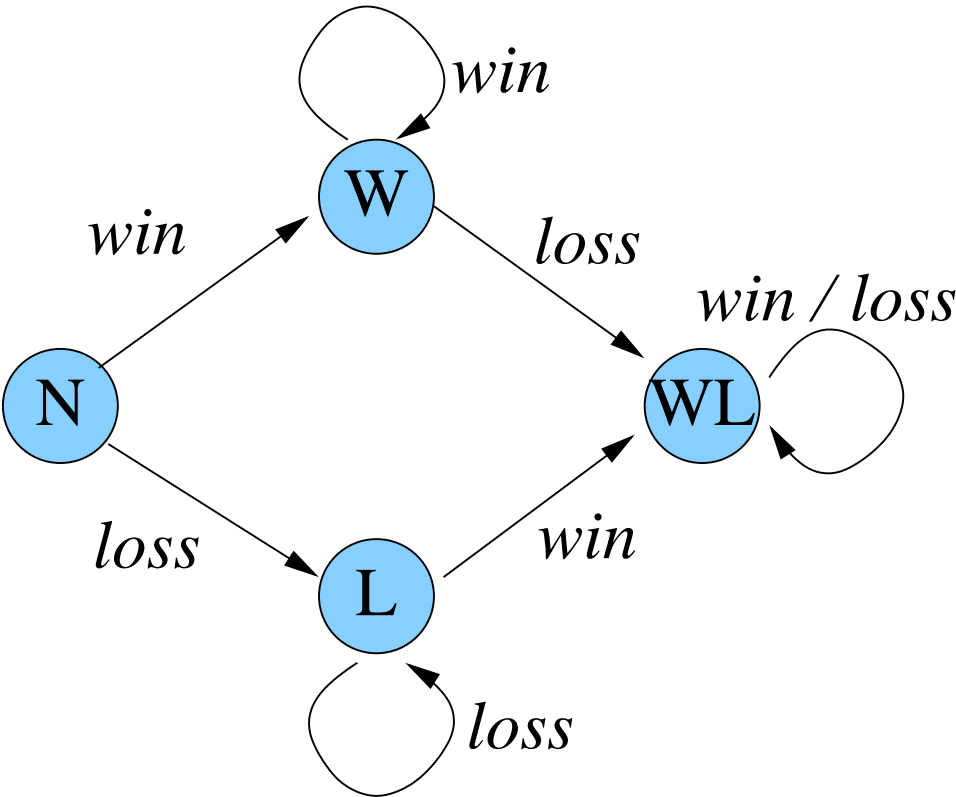
Adversary Strategy For each key we consider the following situations depending on its number of wins and its number of losses:

state	# wins	# losses
W	≥ 1	0
L	0	≥ 1
WL	≥ 1	≥ 1
N	0	0

That is, W, L, WL, and N stand for “has won but never lost,” “has lost but never won,” “has won and has lost,” and “has never participated in a comparison.” At the beginning every key is in the N state. The stopping condition is that there are:

- only one key in the W state,
- only one key in the L state,
- exactly $n - 2$ keys in the WL state.

State Transition



The Basic Rule of the Adversary

x, y	Outcome	New Status	# moves
N, N	$x > y$	W, L	2
W, W	$x > y$	W, WL	1
WL, N	$x < y$	WL, W	1
L, L	$x > y$	WL, L	1
W, N	$x > y$	W, L	1
L, N	$x < y$	L, W	1
W, L	$x > y$	W, L	0
WL, W	$x < y$	WL, W	0
WL, L	$x > y$	WL, L	0
WL, WL	Consistent with assign.	WL, WL	0

Argument About the Number of Moves

To arrive at the stopping condition, along the arrows of the diagram one move has to be made for each of max and min and two moves for each of the rest, so the total number of moves that have to be made is $2(n - 2) + 2 \cdot 1 = 2n - 2$. The number of key comparisons is minimized when two N's are compared $\lfloor n/2 \rfloor$ times and then the other moves are made one at a time. In this happens, the number of comparisons is

$$\lfloor n/2 \rfloor + (2n - 2 - 2\lfloor n/2 \rfloor).$$

This is equal to

$$\lceil 3n/2 \rceil - 2.$$




2. Finding the Second Largest Key

There is an algorithm that uses $n - 2 + \lceil \lg 2 \rceil$ key comparisons:

- Form a tournament so that the height is the smallest and use that to compute the maximum.
- Collect all the keys that directly lost to the maximum and run the tournament on them to compute their maximum. That is, the second largest key. (The other keys are smaller than at least two keys.)

Must Identify the Largest Key

Claim B The second largest key cannot be found without finding the largest key.

The stopping condition has to include: there is a set S of $n - 2$ keys such that for each u of S there are at least two keys from which u is reachable. Let p and q be the remaining two. Then one of them must be the largest and the other must be the second largest. If both lack incoming edges, there is no way to tell which is larger, so it is impossible to find the second largest. So, one of them has an incoming edge. Now there are $n - 1$ keys with an incoming edge, the remaining key must be the maximum. 

Counting the Losses

To prove the lower bound, we pay attention to the number of keys that loses at least once and the number of keys that loses at least twice. Let a and b respective be the two numbers. The total number of losses is at least $a + b$. Since the maximum has to be identified, it must be the case that $a = n - 1$. So, we have only to show that $b \geq \lceil \lg n \rceil - 1$.

Counting the Number of Keys Compared Against the Largest Key

Suppose that the largest key participates in p comparisons and wins in all. Out of the p keys that are compared directly against the largest key, only one can lose exactly once. Otherwise, there is more than one candidate for the second largest key.

So, $b \geq p - 1$. Thus, it suffices to show:

Lemma C $p \geq \lceil \lg n \rceil$.

The Adversary

We'll use the adversary we used for max-plus-min with a slight modification.

According to that adversary, when a key loses **for the first time**, it does so by losing to a **key that has never lost**.

Call a comparison of two keys that have never lost a **critical comparison**. We call a win in a critical comparison a **critical win**.

The rule we add is the following:

- (*) The winner of a critical comparison is the one having more critical wins than the other, where a tie can be broken arbitrarily.

The Strategy

x, y	Outcome	New Status
N/W, L/WL	$x > y$	W, L/WL
WL, L	$x > y$	WL, L
N, N	$x < y$	WL, W
W, N	$x > y$	W, L
L/WL, L/WL	Consist. with assign.	WL, L/WL
W, W	*	*

* The winner is the one having more critical wins than the other.

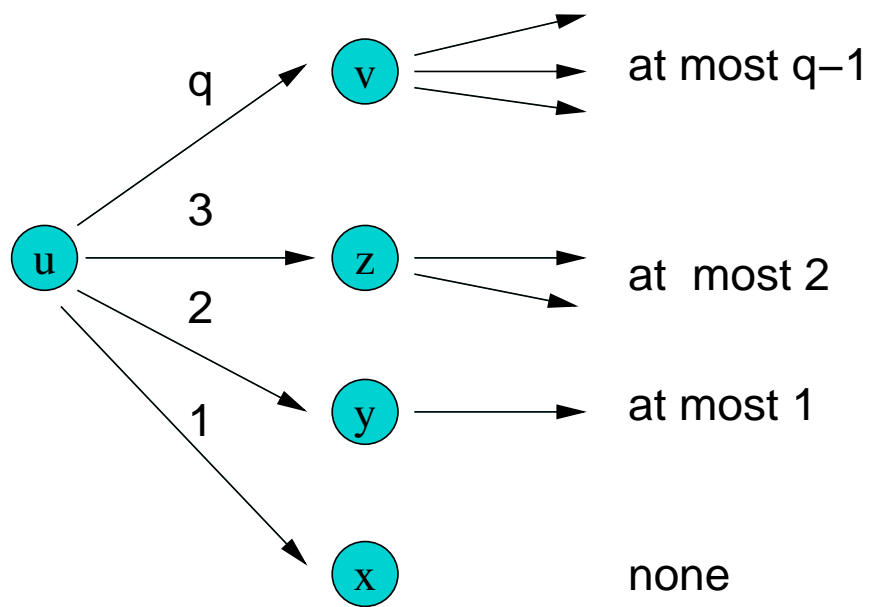
Graph of Critical Comparisons

Consider the graph consisting only of arcs representing critical comparisons. For each node, label its going edges according to the order in which they are added.

Each key other than the maximum key loses at least once. According to the strategy, its first loss occurs in a critical comparison. Since a node will never participate in a critical comparison once it has lost in a critical comparison, its out-degree is equal to the number of its critical wins.

When a node u acquires its q^{th} outgoing edge, say (u, v) , v has at most $q - 1$ wins and u has just made its q^{th} critical wins. So, we have:

(★) **For all u, v , and q , if there is an edge from u and v and if the label of (u, v) is q , then the out-degree of v is at most $q - 1$.**



Bounding the Number of Reachable Nodes

Claim D If a node u has out-degree q then the number of nodes reachable from u is at most 2^q .

We prove the claim by induction on q .

Base Case

The base case is when $q = 0$. u doesn't have a critical win, u is the only node reachable from u . Since $2^0 = 1$, the claim holds.

Induction Step

Let $q = k > 0$ and suppose that the claim holds for all q , $0 \leq q \leq k - 1$.

Let u be a node whose out-degree is q . For each i , $1 \leq i \leq q$, let v_i be the node to which there is an edge from u having label i .

For all i , $1 \leq i \leq q$, the out-degree of v_i is at most $i - 1$, so by our induction hypothesis, the number of nodes reachable from v_i is at most 2^{i-1} .

The nodes reachable from u consists of u and the nodes reachable from one of u_1, \dots, u_q . So, there are at most $1 + \sum_{i=1}^q 2^{i-1} = 2^q$ of them. Thus, the claim holds. ■

Every non-maximum key must lose at least once, so it has to lose in a critical comparison, and once it's lost in a critical comparison it can never participate in a critical comparison. So, every non-maximum key has exactly one incoming edge. So, the graph is a forest. Furthermore, since there are $n - 1$ edges in the graph, the graph is actually a tree. Thus, the maximum is the root and every node is reachable from the root.

This implies that the degree d of the root has to be at least $\lceil \lg n \rceil$. This proves the lemma.



A Linear Time Selection Algorithm

Problem: Given a set of n keys and an integer k , find the k^{th} smallest key in the set.

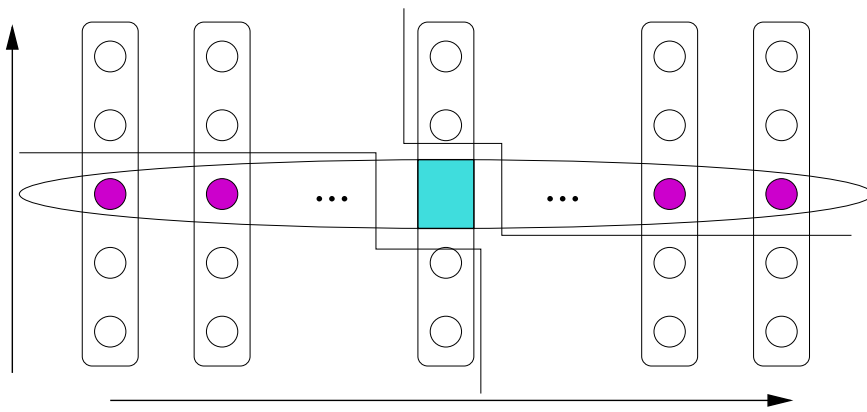
Idea: Divide the input array into blocks of five keys. Compute the median of the medians of the blocks and use it as a pivot to partition the array.

If necessary, continue searching in either the right or the left region.

Use recursion to find the “super-median.”
Also use the following stopping condition:

(*) If the size of the array is less than or equal to B then use brute-force search to find the desired order statistic.

1. Set $m = \lceil n/5 \rceil$. Add extra ∞ to make the size $5m$.
2. Divide the input array into blocks of five keys, and in each group find the median.
3. Using a recursive call compute the “super-median,” x , of the m medians.
4. Partition the input array using x as the pivot: L = the left region and R = the right region.
5. Set $p = \|L\| + 1$. If $p = k$, return x .
6. If $p < k$, return the $(k - p)^{th}$ order statistic in R (remove the extra ∞).
7. If $p > k$, return the k^{th} order statistic in L .



Assumption on the Algorithm

The algorithm can be modified so that identification of the order statistic in question is done as partition.

We will also assume that the keys are pairwise distinct.

Running Time Analysis

Let $T(n)$ be the worst-case running time of the above algorithm, in terms of the number of key comparisons.

Our goal is to show that there exists some α such that for all n , $T(n) \leq \alpha n$.

Note that for each block whose median is smaller than or equal to x , at least three keys are smaller than or equal to x . So,
 $p \geq 3\lceil m/2 \rceil$.

For much the same reason,
 $p \leq 5m - 3\lceil m/2 \rceil + 1$.

So, the size of the array when line 6 or line 7 is executed is at most

$$5m - 3\lceil m/2 \rceil \leq 7m/2.$$

The Number of Key Comparisons for Partitioning

Partitioning with x as pivot can be done with $2(m - 1)$ key comparisons.

- The group from which x is coming does not need extra key comparison.
- For each group whose median is larger than x , only the two keys smaller than the group's median have to be compared against x .
- For each group whose median is smaller than x , only the two keys larger than the group's median have to be compared against x .

It is possible to find the median of five keys using six comparisons. The proof is omitted. See Exercise 5.14.

Then, the total number of key comparisons, excluding those required during the recursive calls, is

$$6m + 2(m - 1) = 8m - 2.$$

So, the running time is

$$T(n) \leq 8m - 2 + T(m) + T(7m/2).$$

Apply the inequality $T(s) \leq \alpha s$ to each T on the right-hand side. Then,

$$T(n) \leq 8m - 2 + \alpha(9m/2).$$

Since $m \leq (n + 4)/5$, the right-hand side is at most

$$\frac{16 + 9\alpha}{10}n + \frac{44 + 36\alpha}{10}.$$

Then, we have only to choose α so that the above quantity is at most αn . That is,

$$\frac{\alpha - 16}{10}n \geq \frac{44 + 36\alpha}{10}.$$

Let $\alpha = 16 + \delta$ for some positive δ . Then the above inequality is equivalent to

$$\delta n \geq 620 + 36\delta.$$

Thus,

$$n \geq 36 + \frac{620}{\delta}.$$

This means that our analysis is valid only for n greater than equal to $36 + \frac{620}{\delta}$. For n less than this, we cannot use recursion and have to show that finding the median can be done in αn steps. So, we set B to

$$\left\lceil 36 + \frac{620}{\delta} \right\rceil.$$

The choice that satisfies these conditions is not unique.

Suppose we set δ to 1. Then, $\alpha = 17$ and $B = 656$. Mergesort requires $\lceil n \lg n - n + 1 \rceil$ key comparisons for sorting. For all n , $2 \leq n \leq 656$, $\lceil n \lg n - n + 1 \rceil < 9.36n < \alpha n$. So, this works.

If we set δ to 0.001, then $\alpha = 16.001$ and $B \geq 62,036$. For all n , $2 \leq n \leq 62,036$, $\lceil n \lg n - n + 1 \rceil \leq 14.93n < \alpha n$. so, this works, too.

A Lower Bound for Finding the Median

Theorem E Let $n \geq 3$ be an odd number. The number of key comparisons required for finding the median of n keys is at least $\frac{3(n-1)}{2}$.

To prove the theorem we will use an adversary argument.

Adversary Argument

Our adversary plays against a given algorithm and dynamically designates a key to be the median, $(n - 1)/2$ keys to be larger than the median, and $(n - 1)/2$ keys to be smaller than the median.

The adversary uses four states for each key

L: the key has been designated to be the larger than the median,

S: the key has been designated to be the smaller than the median,

M: the key has been designated to be the median, and

N: the key has not been given any role.

Every key starts in *N*. The state of each node can be changed at most once. There can be at most one key in *M*, at most $(n - 1)/2$ keys in *L*, and at most $(n - 1)/2$ keys in *S*.

Graph Representation

The rest is about how to determine the state or how to assign/adjust the value of a key. The information gained by the algorithm can be represented using a graph, where each directed edge representing the outcome of the comparison between the keys.

Then the median finding algorithm finishes its work when there has emerged a node from which $(n - 1)/2$ other nodes are reachable and to which $(n - 1)/2$ other nodes are reachable.

Let λ , σ , and ν be the number of nodes in L , S , and N , respectively.

Rule 0 As soon as $\lambda + \sigma = n - 1$, the state of the remaining key is set to M . Before this, no key will enter M .

The following are the rules before rule 0 is applicable. Thus, $\lambda + \sigma + \nu = n - 1$. Let $\delta = \lambda - \sigma$. During the execution of the strategy we will maintain that $\delta \leq 1$.

Suppose that x and y will be compared. We'll omit symmetric cases.

1. $x, y \in N, \mu \geq 3$
Put x in L , y in S , return $x > y$.
2. $x, y \in N, \mu = 2, \delta = 1$
Put x in S , return $x < y$, apply Rule 0.
3. $x, y \in N, \mu = 2, \delta = -1$
Put x in L , return $x > y$, apply Rule 0.
4. $x \notin N, y \in N, \delta = 1$
Put y in S . Apply one of (8) and (9).
5. $x \notin N, y \in N, \delta = -1$
Put y in L . Apply one of (8) and (9).
6. $x \in M, y \in L$
Return $x < y$.
7. $x \in M, y \in S$
Return $x > y$.
8. $x \in L, y \in S$
Return $x > y$.
9. Either $x, y \in L$ or $x, y \in S$
Return consistent relation.

Counting the Number of Edges

For the algorithm to determine the median, $n - 1$ have to be put into either L or S and $n - 1$ have to be added to establish reachability to and from the median. To accomplish the former $(n - 1)/2$ comparisons are necessary. To accomplish the latter $n - 1$ comparisons are necessary. Thus, we have $3(n - 1)/2$, as desired. 