2 Lower Bounds for Selection (January 23 and 28)

If, however, it be thought that, under the proposed system, the very inferior Players would feel so hopeless of a prize that they would not enter a tournament, this can easily be remedied by a process of handicapping, as is usual in races, &c. This would give everyone a reasonable chance at a prize, and therefore a sufficient motive for entering.

— Lewis Carroll, "Lawn Tennis Tournaments: The True Method of Assigning Prizes with a Proof of the Fallacy of the Present Method" (1883)

2.1 Selection and Comparison Trees (a.k.a Tennis Tournaments)

Given a set of n distinct elements $x_1, x_2, \ldots, x_n$ from some totally ordered universe (real numbers, strings, etc.) and an integer k, the *selection* problem is to determine the kth smallest element in the set according to the total order, which we will denote $x_{(k)}$. For the moment, we will consider algorithms that use only comparisons to solve the selection problem—no "bit-fiddling" (like hashing or radix-sort) or more complex algebraic operations. The number of comparisons will be our measure of complexity for these algorithms.

Any comparison-based algorithm can be modeled as a family of *comparison trees*, one for each input size n. A comparison tree is a decision tree. Each internal node is labeled with two indices $1 \leq i, j \leq n$ and two edges labeled $<$ and $>$, corresponding to the two cases $x_i < x_j$ and $x_i > x_j$. As usual, the complexity of a comparison tree is its depth.

This is sometimes called the *tennis tournament model*. We have n players that can be ordered by their tennis-playing prowess, and we want to adaptively schedule matches to determine their ranking. (The Lewis Carroll paper quoted above was perhaps the first to use this model formally; the paper describes a tennis tournament where the three best players always win the top three prizes, regardless of the initial seeding.) To abuse the metaphor further, after a node compares x and y, if $x < y$, we call x the *loser* and y the *winner* of the comparison.

2.2 Background: Partial Orders

A partial order $\prec$ is an asymmetric, transitive, irreflexive binary relation over some set X; that is, for all elements $x, y, z \in X$,

- if $x \prec y$ then $y \not\prec x$,
- if $x \prec y$ and $y \prec z$ then $x \prec z$, and
- $x \not\prec x$.

We say that two elements x and y are *comparable* with respect to a partial order $\prec$ if $x = y$, $x \prec y$, or $y \prec x$. Otherwise, we say that x and y are *incomparable*. For any partial order $\prec$ there is an *opposite* partial order $\succ$ over the same set such that $x \succ y$ if and only if $y \prec x$.

Any partial order $\prec$ over a finite sett X can be presented by a *Hasse diagram*. A Hasse diagram is a directed graph with a node for every element in X, and a directed path from x to y if $x \prec y$. Typically, the nodes are drawn in horizontal levels such that all edges point upward; with this convention in place, we can pretend the graph is undirected. It is also convenient to remove any redundant edges, so that the Hasse diagram has an edge (x, y) if and only if $x \prec y$ and there is no z such that $x \prec z \prec y$. In other words, the standard Hasse diagram is the transitive reduction of any directed graph consistent with the partial order.

A *total* (or *linear*) order is a partial order where every pair of elements is comparable. The Hasse diagram of a total order consists of a single path. A *linear extension* of a partial order $\prec$ is any total order $<$ such that $x < y$ implies that $x \prec y$. Unless the partial order $\prec$ is actually total, it has more than one linear extension.

Any path from the root to another node in a comparison tree defines a partial order over the input elements. Each new comparison reduces the number of possible linear extensions.

Comparison trees are normally introduced to model comparison-based sorting algorithms. A comparison tree correctly sorts if each leaf is associated with a total order. Since there are $n!$ different total orders over any n-element set, a comparison tree that sorts n elements must have at least $n!$ leaves. As we saw in the first lecture, any binary tree with N leaves has depth at least $\lceil \lg N \rceil$, so the complexity of sorting, in the comparison tree model, is at least $\lceil \lg(n!) \rceil = \Omega(n \log n)$.

2.3 Counting Leaves

Let $V(n, k)$ denote the complexity of finding the kth largest element of an n-element set. Finding the kth largest element is obviously the "same" problem as finding the kth smallest element, so $V(n, k) = V(n, n - k + 1)$.

We easily observe that $V(n, 1) \leq n - 1$, since we can scan through the set keeping the largest element seen so far. Conversely, we have $V(n, 1) \geq n - 1$, since in any tournament, every element except the largest must lose at least one comparison. Thus, $V(n, 1) = n-1$. In fact, this observation implies that in any comparison tree to find the largest element, *every* leaf has depth at least $n - 1$, which implies that there must be at least 2^{n-1} leaves.

We can generalize this leaf-counting argument to prove a lower bound for $V(n, k)$ for arbitrary values of k. This generalization is due to Fussnegger and Gabow[1]

Lemma 1. *Suppose $\prec$ is a partial order over a set X such that some element $x \in X$ has the same rank in every linear extension of $\prec$. Then in the partial order $\prec$, every element of X is comparable with x^*.*

Proof: An easy homework exercise. □

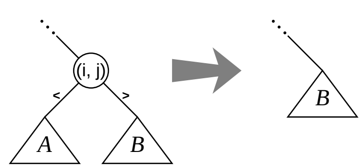
Theorem 1. $V(n, k) \geq n - k + \left\lceil \lg \binom{n}{k-1} \right\rceil$.

Proof: Let T be a comparison tree that identifies the kth largest element $x_{(k)} \in X$. Lemma 1 implies that at each leaf ℓ in T, we can not only identify $x_{(k)}$ but also the set $L(\ell)$ of $k-1$ elements larger than $x_{(k)}$.

Now suppose a little birdie tells us the set of $k - 1$ largest elements of X:

$$L = \{x_{(1)}, x_{(2)}, \ldots, x_{(k-1)}\}.$$

With this knowledge in hand, we call a comparison *wasted* if it compares an element of L with an element of $X \setminus L$. Since we already know the outcome of any wasted comparison, we can simply remove those comparisons from T. Call the resulting tree T_L.



Pruning a comparison tree when $x_i \in L$ and $x_j \notin L$.

Since the smaller comparison tree T_L identifies the largest element of $X \setminus L$, it must have at least 2^{n-k} leaves. But the leaves of T_L are exactly the leaves ℓ of T such that $L(\ell) = L$. Since there are $\binom{n}{k-1}$ choices for the set L, we conclude that T has at least $\binom{n}{k-1} \cdot 2^{n-k}$ leaves. □

[1]Frank Fussnegger and Harold Gabow. A counting approach to lower bounds for the selection problem. *Journal of the ACM* 26:165–172, 1965.

2.4 An Adversary Argument

We can also use an adversary argument to derive a lower bound for the selection problem. The resulting lower bound, first proved by Hyafil[2], is weaker than Fussnegger and Gabow's leaf-counting bound for all $k \geq 3$, but it illustrates some useful techniques.

Recall that every path in a comparison tree corresponds to a partial order. Consider a comparison tree that solves the k-selection problem. Lemma 1 implies that in the partial order associated with any leaf, the target element x^* must be comparable with every other element of X. In other words, for every element $y \neq x^*$, the algorithm must have made at least one comparison $y : z$ where either $y < z \leq x^*$ or $x^* \leq z < y$, establishing the order between y and x^*. We call such a comparison *crucial* for y. Since any comparison is crucial for at most one of the two elements, there must be at least $n-1$ crucial comparisons. In other words, $\boxed{V(n,k) \geq n-1}$ for all n and k.

Theorem 2. $V(n,k) \geq n - k + (k-1)\left\lceil \lg \dfrac{n}{k-1} \right\rceil$.

Proof: Let $L = \{x_{(1)}, x_{(2)}, \ldots, x_{(k-1)}\}$ denote the set of $k-1$ largest elements of X. The algorithm's goal is to identify both L and the largest element $x_{(k)}$ of $X \setminus L$. Each element of $X \setminus L$ is the loser of at least one crucial comparison, so the number of crucial comparisons is at least $n-k$. The adversary strategy described below forces each element of L to win at least $\lceil \lg \frac{n}{k-1} \rceil$ comparisons.

The adversary maintains a set of n *tokens* associated with the elements of X, an integer r, and a set $\tilde{L}$ containing the r largest elements of X. Let $t(x)$ denote the number of tokens associated with element x. Initially, $t(x) = 1$ for all x, $r = 0$, and L is the empty set. During the algorithm's execution, the adversary occasionally adds an element to $\tilde{L}$ and increments r. By definition, elements are always added to $\tilde{L}$ in order from largest to smallest.

Whenever the algorithm compares two elements x and y, the adversary answers using the following procedure:

```
COMPARE(x, y):
  if x ∈ L̃ or y ∈ L̃
      answer correctly
  else
      wlog spose t(x) ≥ t(y)
      if t(x) + t(y) < n/(k−1)
          t(x) ← t(x) + t(y)
          t(y) ← 0
      else
          r ← r + 1
          L̃ ← L̃ ∪ {x}    ⟨⟨x_(r) ← x⟩⟩
      return "x > y"
```

This adversary strategy maintains several invariants.

- First, and most obviously, $t(x) < \frac{n}{k-1}$ for all x. Thus, there are enough tokens for the adversary to put $k-1$ elements into $\tilde{L}$.
- Next, if $t(x) > 0$, then x has lost only to elements of $\tilde{L}$. This invariant implies that elements are added to $\tilde{L}$ in decreasing order, as promised.

[2] Laurent Hyafil. Bounds for selection. *SIAM Journal of Computing* 5:150–165, 1976

- Whenever x wins a comparison, $t(x)$ at most doubles. Thus, every element x has won at least $\lceil \lg t(x) \rceil$ comparisons.
- Finally, for any element $x \in L$, we have $t(x) \geq \frac{n}{2(k-1)}$. Thus, every element of L has won at least $\lceil \lg \frac{n}{t-1} \rceil$ comparisons: $\lceil \lg \frac{n}{2(t-1)} \rceil = \lceil \lg \frac{n}{t-1} \rceil - 1$ to pay for the tokens, plus one more to actually put x into L.

When the algorithm terminates, the elements of $\tilde{L} = L$ have won a total of $(k-1)\lceil \lg \frac{n}{t-1} \rceil$ comparisons. None of these comparisons are crucial for elements of $X \setminus L$. Adding the $n - k$ comparisons that are crucial for $X \setminus L$ completes the proof. □

Observe that whenever the adversary adds an element to $\tilde{L}$, he might as well tell the algorithm its rank in the fictional input set. If the algorithm listens to this information, it never needs to compare an element of $\tilde{L}$ with anything else. By the "Little Birdie" Principle, this extra information only helps the algorithm. Indeed, our argument never counts comparisons involving elements of $\tilde{L}$.

2.5 Better Bounds and Open Problems

Bent and John[3] combined the leaf-counting and adversary arguments to obtain what is currently the best known lower bound for the selection problem for all values of k:

Theorem 3. *$V(n,k) \geq n + R(n,k) - 2\sqrt{R(t,k)}$, where $R(n,k) = \lg \binom{n}{k} - \lg(n-k+1) + 3 = \lg \binom{n+1}{k} - \lg(n+1) + 3$.*

For the special case of finding the median element, we have $R(n, \frac{n+1}{2}) = n - O(\lg n)$, which implies that $V(n, \frac{n+1}{2}) \geq 2n - 2\sqrt{n} - O(\lg n)$. Since Bent and John's result, the only improvement has been the following amazing result by Dor and Zwick:[4]

$$\boxed{V(n, \tfrac{n+1}{2}) > (2 + 2^{-50})n}$$

This bound is not known to be tight, except asymptotically. The famous deterministic algorithm of Blum, Floyd Pratt, Rivest, and Tarjan establishes the upper bound $V(n, \frac{n+1}{2}) < 6n + o(n)$, and a beautiful algorithm of Schönhage, Paterson, and Pippenger[5] shows that $V(n, \frac{n+1}{2}) < 3n + o(n)$. This was the fastest algorithm known for over twenty years, until the following improvement by Dor and Zwick (again!):[6]

$$\boxed{V(n, \tfrac{n+1}{2}) < 2.95n}$$

These are the best bounds known.

On the other hand, there is a very simple randomized algorithm to find the median of an n-element set in only $3n/2 + o(n)$ expected comparisons. Essentially the only lower bound known for randomized selection is the trivial $n - 1$.

It is not hard to show that Theorems 1 and 2 actually gives *exact* bounds for the special cases $k = 1$ and $k = 2$ (and the symmetric special cases $k = n$ and $k = n-1$), but those are the only cases whose exact complexity is known. Even the exact value of $V(n, 3)$ is an open problem, although the gap between the best upper and lower bounds is only an additive constant term.

---

[3] Samuel W. Brent and John W. John. Finding the median requires $2n$ comparisons. *Proc. STOC* 213–216, 1985.

[4] Dorit Dor and Uri Zwick. Median selection requires $(2+\varepsilon)n$ comparisons. *SIAM Journal on Discrete Mathematics* 14:312–325, 2001.

[5] Arnold Schönhage, Michael Paterson, and Mick Pippenger. Finding the median. *Journal of Computer and System Sciences* 13:184–199, 1976.

[6] Dorit Dor and Uri Zwick. Selecting the median. *SIAM Journal on Computing* 28:1722-1758, 1999.