Medians and Order Statistics

CS 624 — Analysis of Algorithms

February 20, 2024

Midterm Exam 1

- **Tentative dates:** The midterm exam will take place on either March 5 (Tuesday) or March 7 (Thursday), in class.
- Covered material: Induction, runtime analysis, sorting (mergesort, insertion sort, quicksort, heapsort, lower bounds), heaps, medians and order statistics, binary search trees. Not covered: dynamic programming.
- The previous class will be partly a review class.
- Prepare your own questions to ask me!

Midterm Exam 1

- Probably 4 questions. Assume every topic will be covered.
- No books, no computers, no cellphones/smartphones/tablets, strictly no friends.
- You may bring up to 20 pages of **handwritten notes**.
  (That is, 20 pieces of paper, up to letter size.)
  No printouts, no photocopies.

Medians and Order Statistics

Definition (Order Statistic)

The i^{th} **order statistic** is the i^{th} smallest element of a set of n elements.

In particular:

- **minimum** = 1^{st} order statistic
- **maximum** = n^{th} order statistic
- **median**: "half-way point" of the set
  - the **lower median** is at $\lfloor (n+1)/2 \rfloor$
  - the **upper median** is at $\lceil (n+1)/2 \rceil$
  - same when n is odd, different when n is even
  - for simplicity, "median" refers to the lower median

Selection Problem

Definition (Selection Problem)

The **selection problem** is defined as follows:

- Input: A set A of n **distinct** numbers and a number k, with $1 \leq k \leq n$.
- Output: the element $x \in A$ that is larger than exactly $k - 1$ other elements of A (that is, the k^{th} order statistic).

Can be solved in $O(n \log n)$ time. How?

Selection Problem

Definition (Selection Problem)

The **selection problem** is defined as follows:

- Input: A set A of n **distinct** numbers and a number k, with $1 \leq k \leq n$.
- Output: the element $x \in A$ that is larger than exactly $k - 1$ other elements of A (that is, the k^{th} order statistic).

Can be solved in $O(n \log n)$ time. How?

There are faster, linear-time algorithms.

- For the special cases when k = 1 and k = n.
- For the general problem.

Minimum and Maximum

The minimum or maximum can be found in $\Theta(n)$ time.

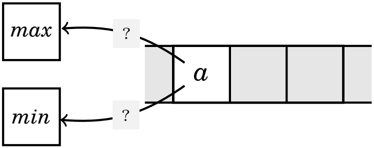
- Simply scan all the elements and find the smallest/largest.

Simultaneous Minimum and Maximum

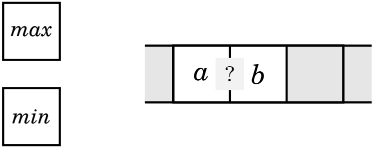
Some applications need to determine both the minimum and maximum of a set of elements.

- Example: Graphics program trying to fit a set of points onto a rectangular display.

Calculating the minimum and maximum *independently* requires $2n - 2$ comparisons. Can we reduce this number?



Simultaneous Minimum and Maximum

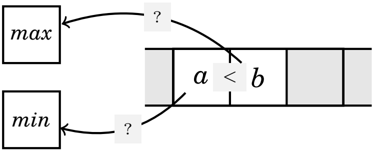


The algorithm sketch:

- maintain min and max elements seen so far
- process elements in *pairs*, compare to get smaller and larger
- compare smaller to min and larger to max, update

There are 3 comparisons per pair, and $\lfloor n/2 \rfloor$ pairs.

Simultaneous Minimum and Maximum



The algorithm sketch:

- maintain min and max elements seen so far
- process elements in *pairs*, compare to get smaller and larger
- compare smaller to min and larger to max, update

There are 3 comparisons per pair, and $\lfloor n/2 \rfloor$ pairs.

Simultaneous Minimum and Maximum

Analysis:

- For odd n: initialize min and max to $A[1]$. Pair the remaining elements. So, number of pairs $= \lfloor n/2 \rfloor$.
- For even n: initialize min to the smaller of the first pair and max to the larger. So, remaining number of pairs $= (n-2)/2 < \lfloor n/2 \rfloor$.
- Total number of comparisons, $C \leq 3\lfloor n/2 \rfloor$.
- For odd n: $C = 3\lfloor n/2 \rfloor$.
- For even n: $C = 3(n-2)/2 + 1 = 3n/2 - 2 < 3\lfloor n/2 \rfloor$.

Finding k^{th} Smallest Element

Can we use a similar method for any order statistic in linear time?

- The cost of finding the k^{th} order statistic using either of these methods is $\Theta(kn)$. If k is fixed, this is $\Theta(n)$.
- If k is not fixed, this is not so good. For instance, suppose we want to find the median. Then k is $n/2$, and the cost is $\Theta(n^2)$, worse than sorting the array.

Is there an $O(n)$ time (independent of k) algorithm for selecting the k^{th} order statistic?

Finding k^{th} Smallest Element

Can we use a similar method for any order statistic in linear time?

- The cost of finding the k^{th} order statistic using either of these methods is $\Theta(kn)$. If k is fixed, this is $\Theta(n)$.
- If k is not fixed, this is not so good. For instance, suppose we want to find the median. Then k is $n/2$, and the cost is $\Theta(n^2)$, worse than sorting the array.

Is there an $O(n)$ time (independent of k) algorithm for selecting the k^{th} order statistic? Yes:

- a simple algorithm with expected $O(n)$ complexity
- a variant with worst-case $O(n)$ complexity

General Selection Problem

Given: array A of size n and k such that $1 \leq k \leq n$

- If the array A were sorted, we would simply find the k^{th} order statistic at $A[k]$. But we don't actually care if A is *completely sorted*, as long as $A[k]$ contains the right element.
- That is one of the properties that Quicksort establishes:

  > *Once Partition chooses a pivot and that call to Partition completes, that pivot never moves again.*

- We modify Quicksort to eliminate unnecessary work:
  We only recur on the side containing k.
- In the average case, the cost of the Partition steps should be

$$n + \frac{n}{2} + \frac{n}{4} + \frac{n}{8} + \cdots = 2n$$

That is, $O(n)$ *average-case* complexity.

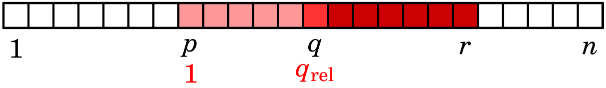
Randomized Select

Algorithm 1 $\text{RandomizedSelect}(A, p, r, k_{\text{rel}})$

Require: $1 \le k_{\text{rel}} \le r - p + 1$

```
if p = r then
    return A[p]
end if
q ← RandomizedPartition(A, p, r)
q_rel ← q − p + 1
if k_rel = q_rel then
    return A[q]
else if k_rel < q_rel then
    return RandomizedSelect(A, p, q − 1, k_rel)
else
    return RandomizedSelect(A, q + 1, r, k_rel − q_rel)
end if
```

Randomized Select



Notation used in the algorithm RandomizedSelect:

- p, q, and r are indices in the original array A.
- q_{rel} is the 1-based index of the pivot $A[q]$ in the subarray $A[p \ldots r]$ — that is, *relative* to the range $[p \mathrel{..} r]$.

Analysis

We see that Randomized-Select is divided into 3 cases:

1. $q_{\text{rel}} < k_{\text{rel}}$, so we search for the $(k_{\text{rel}} - q_{\text{rel}})^{th}$ element in $A[q+1 \,..\, r]$
2. $q_{\text{rel}} = k_{\text{rel}}$, so we found it, and we return $A[q]$
3. $q_{\text{rel}} > k_{\text{rel}}$, so we search for the k_{rel}^{th} element in $A[p \,..\, q-1]$

Analysis

Worst-case complexity:

- $\Theta(n^2)$ — (Like Quicksort) We could get unlucky and always recur on a subarray that is only one element smaller.

Average-case complexity:

- $\Theta(n)$ — Intuition: Because the pivot is chosen at random, we expect that we get rid of half of the list each time we choose a random pivot q.
- Why $\Theta(n)$ and not $\Theta(n \log n)$?

Average-Case Analysis

Let $C(n, i)$ denote the average running time of $\mathrm{RandomizedSelect}(A, 1, n, i)$.

Let $T(n)$ denote the worst average-case time of computing *any* i^{th} element of an array of size n using RandomizedSelect. That is:

$$T(n) = \max\{C(n, i) \mid 1 \leq i \leq n\}$$

We will prove that $T(n) = O(n)$.

Average-Case Analysis

The cost of Partition is $O(n)$, so we can bound it by an for some a.

Therefore:

$$C(n,i) \le an + \frac{1}{n}\left(\sum_{q=1}^{i-1} C(n-q, i-q) + 0 + \sum_{q=i+1}^{n} C(q-1, i)\right)$$

- The call to RandomizedSelect has two parts:
  - the Partition call, whose cost is an, and
  - the recursive call, whose cost varies depending on the location of the pivot which we denote q (really should be q_{rel}).
- We assume that the pivot is equally likely to wind up in any of the n positions in the array, and we average over all those n possibilities.
- Inside the parentheses is the sum of the n possible pivots q:
  - the first term is if the pivot falls before i
  - the second term is if the pivot is exactly i (we just return)
  - the third term is if the pivot falls after i

Average-Case Analysis

$$
\begin{aligned}
C(n,i) &\le an + \frac{1}{n}\left(\sum_{q=1}^{i-1} C(n-q, i-q) + 0 + \sum_{q=i+1}^{n} C(q-1, i)\right) \\
&\le an + \frac{1}{n}\left(\sum_{q=1}^{i-1} T(n-q) + \sum_{q=i+1}^{n} T(q-1)\right) \\
&\le \max\left\{ an + \frac{1}{n}\left(\sum_{q=1}^{i-1} T(n-q) + \sum_{q=i+1}^{n} T(q-1)\right) \;\middle|\; 1 \le i \le n \right\} \\
&= an + \max\left\{ \frac{1}{n}\left(\sum_{q=1}^{i-1} T(n-q) + \sum_{q=i+1}^{n} T(q-1)\right) \;\middle|\; 1 \le i \le n \right\}
\end{aligned}
$$

Average-Case Analysis — Explanation

- Note: i is a separate variable, different from the i in the left-hand side $C(n,i)$. In fact, in the final inequality for $C(n,i)$, the right-hand side no longer depends on i.
- Substituting in the definition of $T(n)$, we get:

$$\begin{aligned} T(n) &= \max\{C(n,i) \mid 1 \le i \le n\} \\ &\le an + \max\left\{ \frac{1}{n}\left(\sum_{q=1}^{i-1} T(n-q) + \sum_{q=i+1}^{n} T(q-1) \right) \;\middle|\; 1 \le i \le n \right\} \end{aligned}$$

We'll guess that $T(n) = O(n)$ and prove by induction that $T(n) = Cn$ satisfies the inequality above.

Average-Case Analysis — Proof by Induction

Theorem

Suppose that

$$T(n) \leq an + \max\left\{ \frac{1}{n}\left(\sum_{q=1}^{i-1} T(n-q) + \sum_{q=i+1}^{n} T(q-1) \right) \,\middle|\, 1 \leq i \leq n \right\}$$

Then $T(n) \leq Cn$ *for some* $C > 0$.

Proof.

Base Case: We can arrange that this is true for $n = 2$ by making sure (when we finally figure out an appropriate value for C) that $C \geq a$.

Inductive Case: We must show that $T(n) \leq Cn$.

IH: Assume that $T(k) \leq Ck$ for all $1 \leq k < n$.

Average-Case Analysis — Proof by Induction

Proof.

We start with the recursive inequality:

$$
\begin{aligned}
T(n) &\leq an + \max\left\{ \frac{1}{n} \left(\sum_{q=1}^{i-1} T(n-q) + \sum_{q=i+1}^{n} T(q-1) \right) \middle| \, 1 \leq i \leq n \right\} \\
&\leq an + \max\left\{ \frac{C}{n} \left(\sum_{q=1}^{i-1} (n-q) + \sum_{q=i+1}^{n} (q-1) \right) \middle| \, 1 \leq i \leq n \right\} && \text{(by IH)} \\
&= an + \max\left\{ \frac{C}{n} \left((i-1)n - \frac{(i-1)i}{2} + \frac{(n-1)n}{2} - \frac{(i-1)i}{2} \right) \middle| \, 1 \leq i \leq n \right\} \\
&= an + \max\left\{ \frac{C}{n} \left((i-1)n - (i-1)i + \frac{(n-1)n}{2} \right) \middle| \, 1 \leq i \leq n \right\}
\end{aligned}
$$

Average-Case Analysis — Proof by Induction

Proof.

- We have to find the maximum of $(i-1)n-(i-1)i$ $= -i^2+(n+1)i-n$ between $i=1$ and $i=n$. This is a concave function of i; in fact, it's an "upside-down parabola", and so its maximum occurs where the derivative is 0.
- The derivative is $-2i+(n+1)$ and this is 0 when $i=\frac{n+1}{2}$.
- So the maximum value of the expression $(i-1)n-(i-1)i$, which is also $(i-1)(n-i)$, is

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

Average-Case Analysis — Proof by Induction

Proof.

So we have

$$\begin{aligned}
T(n) &\le an + \frac{C}{n}\left(\frac{(n-1)^2}{4} + \frac{(n-1)n}{2}\right) \\
&= an + \frac{C}{n}\left(\frac{n^2-2n+1}{4} + \frac{n^2-n}{2}\right) \\
&= an + \frac{C}{n}\left(\frac{3n^2}{4} - n + \frac{1}{4}\right) \\
&= an + C\left(\frac{3n}{4} - 1 + \frac{1}{4n}\right) \\
&\le an + C\frac{3n}{4} \qquad\qquad \text{for } n \ge 1 \\
&= \left(a + \frac{3}{4}C\right)n
\end{aligned}$$

Average-Case Analysis — Proof by Induction

Proof.

So we can fix C finally so that

- $C \geq a$, and
- $a + (3/4)C \leq C$

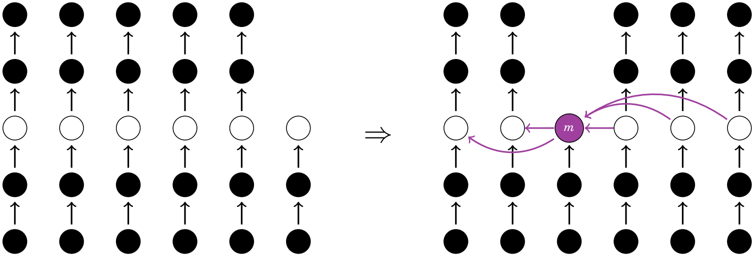
For instance, $C = 4a$ would work.
Then we get $T(n) \leq Cn$ and we are done. $\square$

Selection in Worst-Case Linear-Time

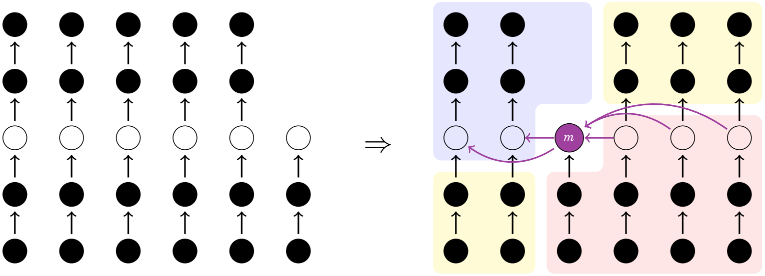
The previous algorithm has *expected* $O(n)$ running time, but the worst case is $O(n^2)$, like Quicksort.

There is a variant that runs in $O(n)$ time in the worst case:

- instead of picking a random pivot, use the **median of medians**
- divide the input range $A[p \mathrel{..} r]$ into $\lfloor n/5 \rfloor$ groups of 5
- sort each group to find its median
- recursively find the median of the group medians
- that is a good enough pivot to guarantee linear complexity



$\Rightarrow$
m



Let m be the median of medians. In each *full* column to the left or right of m, 3 of the 5 elements are $<$ or $>$ to m, respectively.

Analysis of Select

Let $T(n)$ be the running time of Select using median of medians:

- sort each group of 5 to find its median — $O(5^2)\lfloor n/5 \rfloor = \Theta(n)$
- recursively find the median of the group medians — $T(\lfloor n/5 \rfloor)$
- partition using the median of medians as pivot — $\Theta(n)$
- recur on one of the partitions — $\leq T(7n/10)$ (?!)

To summarize:

$$T(n) \leq T(n/5) + T(7n/10) + \Theta(n)$$

We can use guess and prove to show that this is $O(n)$.
We also know $T(n) = \Omega(n)$, so we get $T(n) = \Theta(n)$.