Medians and Order Statistics

September 30, 2019
The midterm exam will take place on Thursday, October 17, in class.

Covered material: Induction, runtime analysis, sorting (mergesort, insertion sort, quicksort, heapsort, lower bounds), heaps, medians and order statistics (maybe, depends on whether you get your HW in time).

The class on October 15 will be partly a review class.

Prepare your own questions to ask me!!!
• Probably 4-5 questions. Assume every topic will be covered.
• Bring class notes and homework assignments.
• No books, no computers, no cellphones/smartphones/tablets, strictly no friends.
• It will count towards 20% of your final grade.
- $i^{th}$ order statistic: $i^{th}$ smallest element of a set of $n$ elements.
- Minimum: first order statistic.
- Maximum: $n^{th}$ order statistic.
- Median: “half-way point” of the set.
- Unique, when $n$ is odd – occurs at $i = (n+1)/2$.
- Two medians when $n$ is even.
- Lower median, at $i = n/2$.
- Upper median, at $i = n/2+1$.
- For consistency, ”median” will refer to the lower median.
Selection Problem

- Selection problem:
  - 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 $A$ (the $k^{th}$ order statistics).

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

- We will study faster linear-time algorithms.
  - For the special cases when $k = 1$ and $k = n$.
  - For the general problem.
Minimum (Maximum) can be found in $\Theta(n)$ time.
Simply scan all the elements and find the smallest (largest).
Some applications need to determine both the maximum and minimum of a set of elements.
Example: Graphics program trying to fit a set of points onto a rectangular display.
Independent determination of maximum and minimum requires $2n-2$ comparisons.
Can we reduce this number?
Simultaneous Minimum and Maximum

- Maintain minimum and maximum elements seen so far
- Process elements in pairs
- Compare the smaller to the current minimum and the larger to the current maximum
- Update current minimum and maximum based on the outcomes
- No. of comparisons per pair = 3. How?
- No. of pairs \( \leq \lfloor n/2 \rfloor \).
Simultaneous Minimum and Maximum

- For odd \( n \): initialize min and max to \( A[1] \). Pair the remaining elements. So, no. 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 no. of pairs 
  \( = (n - 2)/2 < \lfloor n/2 \rfloor \).
- Total no. of comparisons, \( C \leq 3\lfloor n/2 \rfloor \).
- For odd \( n \): \( C = 3\lfloor n/2 \rfloor \).
- For even \( n \): \( C = 3(n - 2)/2 + 1 \) (For the initial comparison).
  \( = 3n/2 - 2 < 3\lfloor n/2 \rfloor \)
Why can’t we use a similar method for any order statistics in a 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 about $n/2$, and the cost would be quadratic.
- Worse than sorting the array...
- Even using heaps won’t do better than sorting the array for finding the median.
Find the $i^{th}$ order statistics.

Seems more difficult than Minimum or Maximum.

Yet, has solutions with same asymptotic complexity as Minimum and Maximum.

We will study an algorithm for the general problem with expected linear-time complexity (independent of $k$).

A second algorithm, whose worst-case complexity is linear, can be found in the text.
General Selection Problem

- Modeled after randomized quicksort.
- Exploits the abilities of Randomized-Partition (RP).
- It uses Partition repeatedly, except that at each step, we only have to use recursion on one side of the partitioned set.
- We hope that in the “average case”, we recurse on a subarray that is about half the size of the previous subarray.
- Then the total cost will be the cost of the partitions, which will be roughly some constant times
  \[ n + \frac{n}{2} + \frac{n}{4} + \frac{n}{8} + \cdots = 2n \]
- So the total cost on average should be \( O(n) \).
Algorithm 1 RandomizedSelect(A,p,r,i)

1: if $p = r$ then
2:     return $A[p]$
3: end if
4: $q \leftarrow RandomizedPartition(A, p, r)$
5: $\pi \leftarrow q - p + 1$
6: if $i = \pi$ then
7:     return $A[q]$
8: else
9:     if $i < \pi$ then
10:        return RandomizedSelect($A, p, q - 1, i$)
11:    end if
12: else
13:    return RandomizedSelect($A, q + 1, r, i - \pi$)
14: end if
Notation used in the algorithm RandomizedSelect:

- $p$, $q$, and $r$ are indices in the original array $A$.
- $\pi$ is the 1-based index of the pivot $A[q]$ in the subarray $A[p \ldots r]$. *(Be careful – $\pi$ here is used to denote just an ordinary variable.)*
We see that Randomized-Select is divided into 3 cases:

1. \( \pi < i \), so we search for the \((i - \pi)^{th}\) element in \(A[q+1..r]\)
2. The pivot \( \pi = i \). We finish and exit.
3. \( \pi > i \), so we search for the \(i^{th}\) element in \(A[p..q-1]\).
Worst-case Complexity:

- $\Theta(n^2)$ – As we could get unlucky and always recurse on a subarray that is only one element smaller than the previous subarray.

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 $\Theta(n \log n)$?
Denote the average runtime of RandomizedSelect(A,1,n,i) by $C(n, i)$.

We will find an upper bound for $C(n, i)$:

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

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

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

First of all – the cost of partition is $O(n)$, so we can bound it by $a*n$ for some $a$. 
Therefore:

\[ C(n, i) \leq an + \frac{1}{n} \left( \sum_{\pi=1}^{i-1} C(n - \pi, i - \pi) + \sum_{\pi=i+1}^{n} C(\pi - 1, i) \right) \]

\[ \leq an + \frac{1}{n} \left( \sum_{\pi=1}^{i-1} T(n - \pi) + \sum_{\pi=i+1}^{n} T(\pi - 1) \right) \]

\[ \leq \max\{an + \frac{1}{n} \left( \sum_{\pi=1}^{i-1} T(n - \pi) + \sum_{\pi=i+1}^{n} T(\pi - 1) \right) : 1 \leq i \leq n\} \]

\[ = an + \max\{\frac{1}{n} \left( \sum_{\pi=1}^{i-1} T(n - \pi) + \sum_{\pi=i+1}^{n} T(\pi - 1) \right) : 1 \leq i \leq n\} \]
The call to RandomizedSelect has two parts: The partition which is the \( an \) part, and the recursive call whose cost varies depending on the location of the pivot which we denote \( \pi \).

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.

That accounts for the factor \( \frac{1}{n} \) just outside the big parenthesized term on the right-hand side.

Inside the parentheses is the sum of all the possibilities that can happen.

The first term is if the pivot falls before \( i \), the second term is if the pivot falls after \( i \).

In the third case (where the pivot is exactly \( i \)) we just end the run...
Once we have taken the maximum over \( i \) in the last line, we note that the final right-hand side is actually independent of \( i \).

Therefore since for each \( i \), \( C(n, i) \) on the left-hand side is \( \leq \) this expression, the maximum of them all is as well.

That is, we can take the maximum over \( i \) of the left-hand side, (using the previous equation) and get a term which we will use for proof by induction.

\[
T(n) \leq an + \max\left\{ \frac{1}{n} \left( \sum_{\pi=1}^{i-1} T(n - \pi) + \sum_{\pi=i+1}^{n} T(\pi - 1) \right) : 1 \leq i \leq n \right\}
\]
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 \( C \geq a \).

Prove that the inductive hypothesis remains true for \( k = n \). We have two things we can use:

- the inductive hypothesis which we can assume is true for \( 1 \leq k < n \)
- the recursive inequality above.
We start with the recursive inequality:

\[ T(n) \leq an + \max \left\{ \frac{1}{n} \left( \sum_{\pi=1}^{i-1} T(n - \pi) + \sum_{\pi=i+1}^{n} T(\pi - 1) \right) : 1 \leq i \leq n \right\} \]

\[ \leq an + \max \left\{ \frac{C}{n} \left( \sum_{\pi=1}^{i-1} (n - \pi) + \sum_{\pi=i+1}^{n} (\pi - 1) \right) : 1 \leq i \leq n \right\} \]

\[ = an + \max \left\{ \frac{C}{n} \left( (i - 1)n - \frac{(i - 1)i}{2} + \frac{(n - 1)n}{2} - \frac{(i - 1)i}{2} \right) : 1 \leq i \leq n \right\} \]

\[ = an + \max \left\{ \frac{C}{n} \left( (i - 1)n - (i - 1)i + \frac{(n - 1)n}{2} \right) : 1 \leq i \leq n \right\} \]
Average Runtime Analysis – Proof by Induction

- 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 the kind of thing we’ve seen before: 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 simply \(-2i + (n+1)\) 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

\[
\left(\frac{n + 1}{2} - 1\right)\left(n - \frac{n + 1}{2}\right) = \frac{n - 1}{2} \cdot \frac{n - 1}{2} = \frac{(n - 1)^2}{4}
\]
So we have

\[ T(n) \leq an + C \left( \frac{(n-1)^2}{4} + \frac{(n-1)n}{2} \right) \]

\[ = an + \frac{C}{n} \left( \frac{n^2 - 2n + 1}{4} + \frac{n^2 - n}{2} \right) \]

\[ = an + \frac{C}{n} \left( \frac{3n^2}{4} - n + \frac{1}{4} \right) \]

\[ = an + C\left( \frac{3n}{4} - 1 + \frac{1}{4n} \right) \]

\[ \leq an + C\frac{3n}{4} \quad \text{for } n \geq 1 \]

\[ = \left( a + \frac{3}{4}C \right)n \]
So we can fix $C$ once and for all 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.