CS 624
Lecture 6: Medians and Order Statistics

1 Definitions

Definition *The i^{th}* order statistic *of a set of n elements is the i^{th} smallest element. (So the minimum element is the first order statistic, and the maximum element is the n^{th} order statistic.)*

Definition *The* median *of a set of n elements is the "one in the middle"—that is,*

- *if n is odd, it is the $\big((n+1)/2)\big)^{th}$ order statistic.*
- *if n is even, it is either the $\big\lfloor(n+1)/2\big\rfloor^{th}$ or $\big\lceil(n+1)/2\big\rceil^{th}$ order statistic. Or, to be more precise, there are two medians in this case—either one of those two elements can be referred to as the median.*

2 Computing the k^{th} order statistic

How fast can we compute the k^{th} order statistic?

Of course there is a naive method that will certainly work: sort the n elements, and pick the k^{th} one. This will work in $O(n \log n + n) = O(n \log n)$ time.

Can we do better? We don't after all, really need all the information that we get by doing a complete sort, so it's reasonable to guess that if all we care about is getting the k^{th} smallest element, we might be able to do better. And in fact, we can.

Some cases are easy:

- We can obviously get the minimum (i.e., the first order statistic) in linear time, just by performing a linear scan. The same holds for the maximum.
- What if we want to get the second smallest element? We could do this
  1. Find the minimum.
  2. Throw it away.
  3. Find the minimum of what's left.

This would also run in linear time, although the multiplicative constant would be about twice as large.

We could also do this in one pass, keeping track of the two smallest elements we have found at each step. This would also work in linear time, although again the multiplicative constant would be greater than that for simply finding the minimum.

It's easy to see that the cost of finding the k^{th} order statistic using either of these methods is $\Theta(kn)$. If k is fixed, this is $\Theta(n)$.

But 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 so the cost by either of these methods is $\Theta(n^2)$. That's awful – we'd do better just by sorting the whole array to begin with.

- Actually, we can do somewhat better than this. It's not hard to see that using heaps, we can find the k^{th} order statistic in time $O(n + k \log n)$. This is a little better than $\Theta(n^2)$. But it's still no good for finding the median – it gives us a cost of $O(n \log n)$, which again is no better than simply sorting the whole array.

3 A better algorithm

In fact, there is an algorithm that allows us to find the k^{th} order statistic in average-case time $O(n)$. That is, the average time needed is completely independent of k.

The algorithm is a modification of QUICKSORT. The key idea is this: we use PARTITION repeatedly, just as before. But now at each step, we only have to recurse on one side of the partitioned set. That's where the cost savings comes from.

And for this to work, we hope that in the "average case", we recurse on a subarray that is about half the size of the previous subarray. If that is the case, then the total cost will be the cost of the partitions, which will be roughly some constant times

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

and so the total cost on average should be $O(n)$. We'll see that this is actually true.

Here is the algorithm. As before, A is an array, and the function finds the i^{th} smallest element in the subarray $A[p \mathinner{..} r]$. (Thus to find the k^{th} order statistic of the elements of A, the original call will be RANDOMIZED-SELECT$(A, 1, n, k)$.)

To understand the notation in the algorithm it helps to study the diagram in Figure 1.

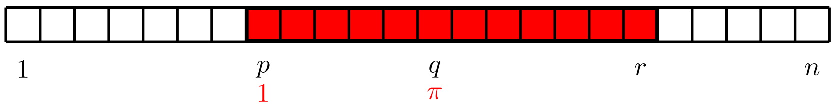

Figure 1: Notation used in the algorithm Randomized-Select. p, q, and r are indices in the original array A. π is the 1-based index of the pivot $A[q]$ in the subarray $A[p\,..\,r]$. (*Be careful—π here is used to denote just an ordinary variable.*)

```
Randomized-Select($A, p, r, i$)
    **if** $p = r$ **then**
        **return** $A[p]$
    $q \leftarrow$ Randomized-Partition($A, p, r$)
    $\pi \leftarrow q - p + 1$ // this is just the 1-based position of the pivot in $A[p\,..\,r]$
    **if** $i < \pi$ **then**
        **return** Randomized-Select($A, p, q - 1, i$)
    **else if** $i = \pi$ **then**
        **return** $A[q]$
    **else** // $i > \pi$
        **return** Randomized-Select($A, q + 1, r, i - \pi$)
```

The idea of this algorithm is simple: Say after the partition step, the position of the pivot element in the subarray $A[p\,..\,r]$ (when considered as a 1-based array) is π. There are three cases:

Case 1: $i < \pi$**.** In this case just find the i^{th} smallest element of the left-hand partition $A[p\,..\,q-1]$.

Case 2: $i = \pi$**.** In this case we are done. Just return $A[q]$.

Case 3: $i > \pi$**.** In this case find the $(i-\pi)^{\text{th}}$ smallest element in the right-hand partition $A[q+1\,..\,r]$.

Now we can show that the average cost of this algorithm is linear in the number of elements in the array, and is independent of i.

First of all, let us note that the cost of Randomized-Select(A, p, r, i) is the same as the cost of Randomized-Select($A, 1, r - p + 1, i$). That is, it really doesn't matter how we index the array. It's only the size of the array that is important.

Now let us denote the average cost of Randomized-Select($A, 1, n, i$) by $C(n, i)$. It turns out that computing $C(n, i)$ directly is very tricky. So we will compute the maximum value of this over i instead: let us define

$$T(n) = \max\{C(n, i) : 1 \le i \le n\} \tag{1}$$

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

We have seen above some reason to believe that $T(n) = O(n)$. We can now prove this:

3.1 Theorem

$$T(n) = O(n)$$

Proof. First, we know that the cost of Partition is $O(n)$. Say the cost of Partition on an array of size $n \geq 1$ is $\leq an$ for some fixed constant $a > 0$. (Of course, we really should say this is true for some $n > n_0$, but by making a large enough, we can assume it is true for all $n \geq 1$.)

Then, based on the algorithm, we can construct a recursive inequality.

$$\begin{aligned}
C(n,i) &\leq an + \frac{1}{n}\left(\sum_{\pi=1}^{i-1} C(n-\pi, i-\pi) + \sum_{\pi=i+1}^{n} C(\pi-1, i)\right)\\
&\leq an + \frac{1}{n}\left(\sum_{\pi=1}^{i-1} T(n-\pi) + \sum_{\pi=i+1}^{n} T(\pi-1)\right)\\
&\leq \max\left\{an + \frac{1}{n}\left(\sum_{\pi=1}^{i-1} T(n-\pi) + \sum_{\pi=i+1}^{n} T(\pi-1)\right) : 1 \leq i \leq n\right\}\\
&= an + \max\left\{\frac{1}{n}\left(\sum_{\pi=1}^{i-1} T(n-\pi) + \sum_{\pi=i+1}^{n} T(\pi-1)\right) : 1 \leq i \leq n\right\}
\end{aligned}$$

The only line in this computation you really need to be careful about is the first one. After that, the computations are completely straightforward. The first line encapsulates the knowledge we have from the algorithm. Here's how to understand it:

The average cost $C(n, i)$ of finding the i^{th} element in an n-element array is composed of two parts:

- The cost of the partition. This is just an, and doesn't vary.
- The cost of the recursive call. This varies, depending on where the pivot of the partition winds up compared with i. In the notation we are using here, π is the position of the pivot.

  We assume that the pivot is equally likely to wind up in any of the n positions in the array, and we take the 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. As before, they fall into three cases:

  Case 1: the pivot falls to the left of the i^{th} term in the array. That is to say, $\pi < i$. This accounts for the first sum inside the big parentheses, where π goes from 1 to $i - 1$. For each such π, we need to compute the average cost of finding the $i - \pi^{\text{th}}$ element of the subarray $A[\pi + 1 \mathinner{..} n]$. And this is just $C(n - \pi, i - \pi)$.

  Case 2: the pivot falls on the i^{th} term in the array. That is, $\pi = i$. There is nothing to do in this case, since we return immediately. So there is no term inside the parenthesis corresponding to this case.

  Case 3: the pivot falls to the right of the i^{th} term in the array. That is to say, $\pi > i$. This accounts for the second sum inside the big parentheses, where π goes from $i + 1$ up to n. For each such π, we need to compute the average cost of finding the i^{th} element of the subarray $A[1 \mathinner{..} \pi - 1]$. And this is just $C(\pi - 1, i)$.

So that explains how the first line of the computation is derived.

Now once we have taken the maximum over i in the last line, we can 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 equation 1) and get

$$T(n) \leq an + \max\left\{ \frac{1}{n}\left(\sum_{\pi=1}^{i-1} T(n-\pi) + \sum_{\pi=i+1}^{n} T(\pi-1) \right) : 1 \leq i \leq n \right\} \tag{2}$$

So what we have done so far is this: we started with a really messy recursive inequality for $C(n,i)$, and derived from it a rather less messy recursive inequality for $T(n)$. And while it might not seem all that easier to deal with this new recursive inequality, in fact it is a lot easier.

We will in fact use (2) to prove by induction that $T(n) = O(n)$. As usual, our inductive hypothesis is that there is a fixed constant $C > 0$ such that

$$T(k) \leq Ck \tag{3}$$

for $1 \leq k < n$.

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

So now let us prove that the inductive hypothesis remains true for $k = n$.

We have two things we can use:

- the inductive hypothesis (3), which we can assume is true for $1 \leq k < n$,and
- the recursive inequality (2).

We start with the recursive inequality:

$$\begin{aligned}
T(n) &\leq an + \max\left\{ \frac{1}{n}\left(\sum_{\pi=1}^{i-1} T(n-\pi) + \sum_{\pi=i+1}^{n} T(\pi-1) \right) : 1 \leq i \leq n \right\} \\
&\leq an + \max\left\{ \frac{C}{n}\left(\sum_{\pi=1}^{i-1} (n-\pi) + \sum_{\pi=i+1}^{n} (\pi-1) \right) : 1 \leq i \leq n \right\} \qquad \text{by the inductive hypothesis} \\
&= an + \max\left\{ \frac{C}{n}\left((i-1)n - \frac{(i-1)i}{2} + \frac{(n-1)n}{2} - \frac{(i-1)i}{2} \right) : 1 \leq i \leq n \right\} \\
&= an + \max\left\{ \frac{C}{n}\left((i-1)n - (i-1)i + \frac{(n-1)n}{2} \right) : 1 \leq i \leq n \right\}
\end{aligned}$$

Now

$$(i-1)n - (i-1)i = -i^2 + (n+1)i - n$$

and we have to find the maximum value of this 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

$$i = \frac{n+1}{2}$$

(and this is, by the way, between $i = 1$ and $i = n$). So the maximum value of the expression $(i-1)n - (i-1)i$—which is also $(i-1)(n-i)$—is

$$\begin{aligned}\Bigl(\frac{n+1}{2} - 1\Bigr)\Bigl(n - \frac{n+1}{2}\Bigr) &= \frac{n-1}{2}\frac{n-1}{2} \\ &= \frac{(n-1)^2}{4}\end{aligned}$$

and so we have

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

So we can fix C once and for all so that

- $C \ge a$, and
- $a + (3/4)C \le C$

(for instance, $C = 4a$ would work), then we get $T(n) \le Cn$ and we are done. □

Note that since this algorithm works in $O(n)$ average time regardless of i, it enables us to find the median in expected time $O(n)$—that is, we can just set $i = \lceil (n+1)/2 \rceil$.

There is actually an algorithm that finds the median in *worst-case* time $O(n)$. The text presents it in Section 9.3. In practice it's not used, however. It's pretty complicated to program, and the algorithm we just analyzed works quite well when the pivoting is randomized, as we did here.

4 Appendix: Computing $C(n, i)$ directly

This is really an appendix. I want to reproduce here (with some minor and inessential changes) Donald Knuth's derivation of exact bounds on $C(n, i)$. It's not a hard argument to follow, but it certainly took some cleverness to come up with it, and a great deal of confidence as well[1]. He published this in the paper "Mathematical Analysis of Algorithms" in the journal *Information Processing* (pages 19–27) in 1971—this contains the proceedings of the 1971 IFIP conference.

Here's how he did it: he starts just as we did above, with the recursion

$$C(n,i) = (n-1) + \frac{1}{n}\left(\sum_{\pi=1}^{i-1} C(n-\pi, i-\pi) + \sum_{\pi=i+1}^{n} C(\pi-1, i)\right)$$

The only difference between this and what we wrote above is

- The first term on the right is explicitly $(n-1)$—we had written an, but Knuth is using a model where the cost is exactly $n-1$.
- He has an equality here (whereas we had an inequality) for pretty much the same reason.

Actually, Knuth finds it a little more convenient to write it like this (only the second sum is changed somewhat):

$$\begin{aligned} C(n,i) &= (n-1) + \frac{1}{n}\left(\sum_{\pi=1}^{i-1} C(n-\pi, i-\pi) + \sum_{\pi=i}^{n-1} C(\pi, i)\right) \\ &= (n-1) + \frac{1}{n}\big(A(n,i) + B(n,i)\big) \end{aligned}$$

Multiplying through by n, this becomes

$$nC(n,i) = n(n-1) + A(n,i) + B(n,i) \tag{4}$$

Now he wants to simplify this. It doesn't really look like much can be done here, but he plows ahead: First he notices that

$$\begin{aligned} A(n+1, i+1) &= A(n,i) + C(n,i) \\ B(n+1, i) &= B(n,i) + C(n,i) \end{aligned}$$

He uses these two relations to write four equations:

$$A(n+1, i+1) - A(n,i) = C(n,i) \tag{5}$$

$$A(n, i+1) - A(n-1, i) = C(n-1, i) \tag{6}$$

$$B(n+1, i+1) - B(n, i+1) = C(n, i+1) \tag{7}$$

$$B(n,i) - B(n-1, i) = C(n-1, i) \tag{8}$$

[1]He gives this derivation as a problem in Volume 1 of *The Art of Computer Programming*, but the sketch of the solution he provides there is pretty opaque. Here I am following the original paper.

Subtracting (6) from (5) and (8) from (7) and then adding the two results, he arrives at

$$\begin{aligned} &A(n+1,i+1) - A(n,i+1) - A(n,i) - A(n-1,i) \\ &+ B(n+1,i+1) - B(n,i+1) - B(n,i) + B(n-1,i) \\ &= C(n,i) - C(n-1,i) + C(n,i+1) - C(n-1,i) \end{aligned} \tag{9}$$

And—here is the (inspired or lucky, depending on your view of these things) key step[2]. From (4) (for the first equality) followed by (9) (for the second), he gets

$$\begin{aligned} &(n+1)C(n+1,i+1) - nC(n,i+1) - nC(n,i) + (n-1)C(n-1,i) \\ &= (n+1)n - n(n-1) - n(n-1) + (n-1)(n-2) \\ &+ A(n+1,i+1) - A(n,i+1) - A(n,i) + A(n-1,i) \\ &+ B(n+1,i+1) - B(n,i+1) - B(n,i) + B(n-1,i) \\ &= 2 + C(n,i) - C(n-1,i) + C(n,i+1) - C(n-1,i) \end{aligned} \tag{10}$$

and the result of this simplifies to

$$(n+1)C(n+1,i+1) - (n+1)C(n,i+1) - (n+1)C(n,i) + (n+1)C(n-1,i) = 2$$

which even Knuth remarks is "an extraordinary coincidence that $n+1$ [is] a common factor in each of the C's". Dividing by $n+1$ then, he gets

$$C(n+1,i+1) - C(n,i+1) - C(n,i) + C(n-1,i) = \frac{2}{n+1} \tag{11}$$

Now this is a recurrence that Knuth will find easy to solve. However, first, he has to deal with the boundary cases $i = 1$ and $i = n$. First we see what happens when $i = 1$:

$$C(n,1) = (n-1) + \frac{1}{n}\big(C(1,1) + C(2,1) + \ldots + C(n-1,1)\big)$$

Writing the same equation for $n+1$ instead of n and then subtracting one equation from the other, we get

$$(n+1)C(n+1,1) - nC(n,1) = (n+1)n - n(n-1) + C(n,1)$$

which is just

$$C(n+1,1) - C(n,1) = \frac{2n}{n+1} = 2 - \frac{2}{n+1} \tag{12}$$

and we know also that $C(1,1) = 0$. Therefore we can solve (12) immediately to get

$$C(n,1) = 2n - 2H_n \tag{13}$$

and by symmetry, we also have

$$C(n,n) = 2n - 2H_n \tag{14}$$

[2]It's the only part of the derivation that's not fairly standard.

Now we can go back to the recurrence (11). We can write it as

$$\big(C(n+1,i+1)-C(n,i)\big)-\big(C(n,i+1)-C(n-1,i)\big)=\frac{2}{n+1}$$

From this, we get immediately that

$$\begin{aligned}C(n+1,i+1)-C(n,i)&=\frac{2}{n+1}+\frac{2}{n}+\ldots+\frac{2}{i+2}+C(i+1,i+1)-C(i,i)\\&=2(H_{n+1}-H_{i+1})+2-\frac{2}{i+1}\end{aligned}$$

We can telescope this equation, adding up the following instances of it:

$$\begin{aligned}C(n,i)-C(n-1,i-1)&=2\Big(H_n-H_i+1-\frac{1}{i}\Big)\\C(n-1,i-1)-C(n-2,i-2)&=2\Big(H_{n-1}-H_{i-1}+1-\frac{1}{i-1}\Big)\\C(n-2,i-2)-C(n-3,i-3)&=2\Big(H_{n-2}-H_{i-2}+1-\frac{1}{i-2}\Big)\\&\vdots\\C(n-i+2,2)-C(n-i+1,1)&=2\Big(H_{n-i+2}-H_2+1-\frac{1}{2}\Big)\end{aligned}$$

to yield

$$C(n,i)-C(n-i+1,1)=2\sum_{j=2}^{i}\Big(H_{n-i+j}-H_j+1-\frac{1}{j}\Big)$$

Now we can apply the identity

$$\sum_{k=1}^{n}H_k=(n+1)H_n-n$$

(together with the fact that $H_1=1$) to get

$$\begin{aligned}C(n,i)-C(n-i+1,1)&=2\Big(\big((n+1)H_n-n\big)-\big((n-i+2)H_{n-i+1}-(n-i+1)\big)\Big)\\&\quad-2\Big(\big((i+1)H-i-i\big)-(2H_1-1)\Big)\\&\quad-2\sum_{j=2}^{i}\Big(1-\frac{1}{j}\Big)\\&=2\big((n+1)H_n-(n-i+2)H_{n-i+1}-(i+1)H_i-n+n-i+1+i-1\big)\\&\quad+2(i-1)-2(H_i-1)\\&=2(n+1)H_n-2(n-i+2)H_{n-i+1}-2(i+2)H_i+2i-4\end{aligned}$$

and since we have already seen above that

$$C(n-i+1,1) = 2(n-i+1) - 2H_{n-i+1}$$

we end up with

$$C(n,i) = 2(n+1)H_n - 2(n-i+3)H_{n-i+1} - 2(i+2)H_i + 2(n-1) \tag{15}$$

So this is Knuth's result. It's interesting, mainly because it's not at all obvious that one could solve the original recurrence at all. I don't, however, think it's all that useful. It looks, for instance, like the terms in the final expression are $O(n \log n)$. In fact, there is a lot of cancellation that goes on, and so one actually gets $O(n)$ behavior, as we know one should. But doing all that probably gets you no more than our original derivation did.

To be precise, here's how one could go about bounding the expression we have just arrived at. Note that we can write the right-hand side as the sum of two terms:

$$\begin{gathered} 2\bigl(nH_n - (n-i+1)H_{n-i+1} - (i-1)H_{i-1}\bigr) \\ +2\bigl(H_n - 2H_{n-i+1} - 3H_i + (n-1)\bigr) \end{gathered}$$

Since $H_n = O(\log n)$, it is clear that the second term is $O(n)$. So it's the first term we have to deal with, and that's where the cancellation happens. We know that

$$\log n < H_n < \log n + 1$$

and so we can bound the first term as follows[3]:

$$\begin{aligned} nH_n - (n-i+1)H_{n-i+1} - (i-1)H_{i-1} &< n(\log n + 1) - (n-i+1)\log(n-1+1) - (i-1)\log(i-1) \\ &= n\log n - (n-i+1)\log(n-i+1) - (i-1)\log(i-1) + n \end{aligned}$$

We can ignore the final term, since we are just going to prove that this whole thing is $O(n)$. As for the rest, it is just $f(i-1)$, where we we define

$$f(t) = n\log n - (n-t)\log(n-t) - t\log t$$

$f(t)$ clearly equals 0 when $t = 0$ and also when $t = n$. And in fact, $f(t) = f(n-t)$. Further, one can, simply by taking the derivative, see that f is increasing on the interval $[0, n/2]$.

4.1 Exercise *Prove this fact*[4].

[3] Actually, we'll ignore the factor of 2, since it clearly won't matter for what we're doing.

[4] By the way, you might be wondering how I thought of extracting this expression in the first place. The answer is this: I noticed that if we replace H_n by $\log n$ in the expression that Knuth wound up with in (15), we get something that is reasonably close to

$$\log \frac{n^n}{(n-i+1)^{n-i+1}(i-1)^{i-1}}$$

which in turn is the value of

$$\log \frac{n^n}{t^t(n-t)^{n-t}}$$

for $t = i-1$. So I went from there. What the argument here shows in fact is that $\frac{n^n}{t^t(n-t)^{n-t}}$ has its greatest value when $t = n/2$, at which point the value is 2^n. This is kind of a nice result in itself, I think.

Therefore, $f(t)$ is bounded by its maximum value, which must be $f(n/2)$, and

$$\begin{aligned} f\left(\frac{n}{2}\right) &= n\log n - 2\frac{n}{2}\log\frac{n}{2} \\ &= n\log n - n(\log n - \log 2) \\ &= n\log 2 \end{aligned}$$

So we see that in fact $C(n,i) = O(n)$.