**CIS 320: Algorithms** September 15, 2021

# Lecture 5

*Lecturer: Aaron Roth* *Scribe: Aaron Roth*

## Divide and Conquer II: Medians and Other Order Statistics

Suppose we are given an array of length $n$ and we would like to find the $k$'th smallest element in the array. For example, if $k = 1$, this corresponds to finding the min; if $k = n$, this corresponds to finding the max; if $k = n/2$, this corresponds to finding the median. How can we do it? We can define the problem that we want to solve by means of a naive but transparent algorithm. Given an array $A$, we want to return the entry derived as follows:

1. Sort $A$ in ascending order, and then
2. Return $A[k]$.

This algorithm is simple and works, but is potentially wasteful. If we use MergeSort or QuickSort, this takes time $\Theta(n \log n)$, and no comparison based sorting algorithm can do any better.

If $i = 1$ or $n$ (i.e. we want to find the min or max), its clear we can solve the problem in linear time — we saw this in the lecture on insertion sort, and recall the simple algorithm here:

**Algorithm 1** FindMin(A)

```
Let MinIndex = 1.
for i = 2 to n do
  if A[i] < A[MinIndex] then
    MinIndex = i.
  end if
end for
Return A[MinIndex].
```

For $k = 2$ or $k = 3$ you could do something similar by maintaining variables to remember 2 or 3 indices, but this does not scale well to super-constant $k$.

It turns out there is a general randomized *linear time* divide and conquer solution that looks a lot like QuickSort. The idea is pretty simple:

1. Pick a random pivot $z$ and partition $A$ to have the form $B \circ z \circ C$ such that for every pair of indices $i, j$: $B[i] \leq z \leq C[j]$.
2. If $|B| > k$, then we know the $k$'th smallest index is in $B$, and so we can recurse on $B$. Or, if $|B| < k - 1$, then we know the $k$'th smallest index is in $C$, so we can recurse on $C$ (and try to find the $k - |B|$'th smallest index). Finally, if $|B| = k - 1$, then $z$ is our answer!

This is very much like the QuickSort recursion — but when we partition $A$, we only ever need to recurse on *one* of the two parts, not both of them. This is where the improvement in running time will come from.

Lets recall our partition operation from our study of QuickSort:

**Algorithm 2** Partition$(A, z)$

**Let** $\ell = \text{Length}(A)$ and **initialize** new arrays $B$ and $C$.
**Initialize** indices $p_B = 1$, $p_C = 1$.
**for** $p_A = 1$ to $\ell$ **do**
  **if** $A[p_A] \leq z$ **then**
    $B[p_B++] = A[p_A]$
  **else**
    $C[p_C++] = A[p_A]$
  **end if**
**end for**
**Return** $B, C$

We can use it as a subroutine to implement our outline for a selection algorithm.

**Algorithm 3** QuickSelect$(A, k)$

**Let** $\ell_A = \text{Length}(A)$ and **Pick** $i \in \{1, \ldots, \ell_A\}$ uniformly at random. **Let** $z = A[i]$.
**Let** $A_{-i} = A[1, \ldots, i-1] \circ A[i+1, \ldots \ell_A]$
**Let** $(B, C) = \text{Partition}(A_{-i}, z)$, and **Let** $\ell_B = \text{Length}(B)$.
**if** $\ell_B = k - 1$ **then**
  **Return** $z$.
**else if** $\ell_B \geq k$ **then**
  **Return** QuickSelect$(B, k)$
**else**
  **Return** QuickSelect$(C, k - \ell_B - 1)$
**end if**

First we observe that QuickSelect actually does what it is supposed to do:

**Theorem 1** *For any array $A$ of length $\ell_A$ and any $1 \leq k \leq \ell_A$, QuickSelect returns the $k$'th largest element in $A$ in sorted order.*

**Proof** We prove this by induction on $\ell_A$: our inductive hypothesis is the theorem statement. In the base case $\ell_A = 1$, it must also be that $k = 1$. In this case, we will have that $\ell_B = 0 = k - 1$, and we return $z = A[1]$, as desired.

For the inductive case, we may assume that $\ell_A \geq 2$. In this case, if $\ell_B = k - 1$, because we know that for all $1 \leq i \leq \ell_B$ we have $B[i] \leq z$ and for all $1 \leq i \leq \ell_C$ we have $z \geq C[i]$, then we have that $z$ is the $k$'th smallest element, and the algorithm returns it.

If $\ell_B \geq k$, then we similarly know that the $k$'th smallest element in $A$ is the $k$'th smallest element in $B$, and by our inductive hypothesis, we return the correct element (QuickSelect$(B, k)$).

Finally, in the remaining case, $\ell_B \leq k - 2$. In this case, we know that the $k$'th smallest element lies in $C$. However, since every element of $B$ as well as $z$ is smaller than every element of $C$, we know that the $K$'th smallest element in $A$ must be the $(k - \ell_B - 1)$'th smallest element in $C$. Since $\ell_B \leq k - 2$, we have that $1 \leq (k - \ell_B - 1) \leq \ell_C$, and so we may apply our inductive hypothesis to conclude that we return the correct element (QuickSelect$(C, k - \ell_B - 1)$). ■

And what can we say about the running time of QuickSelect? It is a randomized algorithm because we pick the pivot at random, so we will analyze its expected running time. It is a recursive algorithm, so as we saw with MergeSort, it will be tempting to write down its running time with a recurrance relation and solve it. Once we have a good guess for the running time, solving these is often a straightforward exercise in induction. But how might we come up with our guess that the running time is linear?

Because we pick our pivot element at random, in expectation, it divides $A$ in half. So what if it divided $A$ exactly in half? We can use this to build intuition. The partition operation takes $O(n)$ time, and we would recurse on a dataset of size $n/2$, and so if $T(n)$ denotes our running time on an instance of size $n$, we would have:

$$T(n) \leq T(n/2) + c \cdot n$$

for some constant $c$. Its easy to directly unroll this recurance:

$$T(n) = c \cdot n + c \cdot \frac{n}{2} + c \cdot \frac{n}{4} + \ldots + c \cdot 1 \leq c \cdot n \cdot \left( \sum_{i=0}^{\infty} 1/2^i \right) = 2cn$$

So perhaps linear time is a good guess for the expected running time.

**Theorem 2** *For any $k$, and on an input $A$ of length $n$, QuickSelect$(A, k)$ runs in time $O(n)$.*

**Proof** Suppose we select a pivot $z$ that is the $i$'th largest element of $A$. This divides $A$ into two pieces, of size $\ell_B = i - 1$ and $\ell_C = n - i$. In the worst case, we recurse on the larger of the two pieces.

Let $X_i$ denote the indicator of the event that we select a pivot $z$ corresponding to the $i'$th largest element of $A$. Since we select $z$ uniformly at random, we have $\Pr[X_i] = 1/n$ for all $1 \leq i \leq n$. Denote by $T(n)$ the expected running time of QuickSelect on an instance of size $n$. Let us make the inductive assumption that $T(n) \leq c \cdot n$ for some constant $c$. This is trivially satisfied in the base case when $n = 1$. Since we know that Partition runs in linear time (and assuming for simplicity that $n$ is even), we have that for some constant $d$:

$$\begin{aligned}
T(n) &\leq \left( \sum_{i=1}^{n} \Pr[X_i = 1] \cdot T(\max(i-1, n-i)) \right) + d \cdot n \\
&= \frac{1}{n} \left( \sum_{i=n/2}^{n-1} 2 \cdot T(i) \right) + d \cdot n \\
&\leq \frac{1}{n} \left( \sum_{i=n/2}^{n-1} 2 \cdot c \cdot i \right) + d \cdot n \\
&= \frac{2c}{n} \left( \sum_{i=n/2}^{n-1} i \right) + dn \\
&= \frac{2c}{n} \cdot \left( \frac{3n}{2} - 1 \right) \cdot \frac{n}{4} + d \cdot n \\
&= \frac{3}{4} \cdot cn - \frac{c}{2} + dn \\
&\leq cn
\end{aligned}$$

Here, the second inequality follows from our inductive assumption on $T(i)$ for all $i < n$. The last inequality follows if $d \leq \frac{c}{4}$. Since we are free to choose as large a $c$ as we want in our argument, we can without loss of generality always choose $c \geq 4d$. ■