1 A Deterministic Linear-time Selection Algorithm

We now present a selection algorithm whose worst-case running time is $O(n)$. The strategy is the same as for the randomized algorithm: select a pivot element that splits the array in an approximately balanced way, throw away the part that is guaranteed to not contain the sought-out element, and recursively solve a selection problem (perhaps with a different index) in the kept part.

The difference is that rather than randomly choosing an element to serve as the pivot, we do a bit more work to isolate an element which is *guaranteed* to be an *approximately balanced separator*, i.e., an element which is guaranteed to be bigger than a constant fraction of all elements *and* smaller than a constant fraction of all elements. Below we describe how to find such a good pivot element for an arbitrary set S of size n. Note that in our divide-and-conquer algorithm we will be executing this "search for a good pivot" in each level of the recursion.

1.1 Finding a Good Pivot

We begin by dividing S into sets of size $2k+1$ arbitrarily (except perhaps for one set which might contain fewer elements). It is important to note that k is independent of the size, $|S|$, of the set S, i.e., it is a hard-wired constant in our code (and, therefore, same in all levels of the recursion).

Next, we compute the median of each set. Since k is fixed, we do this by "brute force", e.g., by sorting each set. Let c_k be the number of steps needed to find the median of a set by brute force (this is no more than $O(k \log k)$).

Finally, we form a set S' containing the

$$\left\lceil \frac{|S|}{2k+1} \right\rceil$$

medians. We then fire-off another instance of our selection algorithm, i.e., we employ recursion, asking for the median of this set S'. The answer, call it p, is our pivot for splitting S. Is p an approximately balanced separator? Let's see.

As a thought experiment, think of each set as a brick labeled by its median, and sort the bricks by decreasing label. Also, think of each brick as internally sorted. Clearly, the brick labeled by p is "the middle brick". Moreover, if you pick a brick B that is to the left of p's brick, and you pick any element from the left half of B (including B's median), that element will be strictly greater than p. Call all such elements "big". Similarly, if you pick a brick C that is to the right of p's brick, and you pick any element from the right half of C (including C's median), that element will be strictly smaller than p. Call all such elements "small". Now, when we use p as a pivot, we are guaranteed that we will eliminate from consideration either all the small elements or all the big elements. So, overall, the number of elements we will be eliminating is at least

$$\left\lceil \frac{|S|}{2k+1} \right\rceil \cdot \frac{1}{2} \cdot (k+1) - O(k) \ ,$$

where the $O(k)$ term reflects the fact that one brick might have fewer than $2k+1$ elements and that the total number of bricks might be even. So, after pivoting, our recursive call will need to solve a selection problem on no more than

$$|S| - \left(\left\lceil \frac{|S|}{2k+1} \right\rceil \cdot \frac{1}{2} \cdot (k+1) - O(k) \right) \approx |S| \cdot \frac{3k+1}{4k+2}$$

elements, where the approximation $\approx$ is valid as long as $|S| \gg k$. Therefore, counting all the work we did, we get that to solve a selection problem on a set of size n, we will need time (ignoring floors/ceilings)

$$\begin{aligned} T(n) &\leq c_k \cdot \frac{n}{2k+1} + T\left(\frac{n}{2k+1}\right) + n + T\left(\frac{3k+1}{4k+2} \cdot n\right) && (1) \\ &= T\left(\frac{n}{2k+1}\right) + T\left(\frac{3k+1}{4k+2} \cdot n\right) + O_k(n) \ , && (2) \end{aligned}$$

where the notation $O_k()$ serves as a reminder that the constant implicit in our big-O notation depends on k.

For $k = 1$, we have,

$$\begin{aligned} T(n) &= T(2n/3) + T(n/3) + O(n) \\ &= T(4n/9) + T(2n/9) + T(2n/9) + T(n/9) + O(n) + O(n) \\ &= \dots \end{aligned}$$

It is easy to see that if we keep unfolding the recursion, after k unfoldings we will have k terms of $O(n)$ and still have a $T(n/3^k)$ term (and many others). Therefore, the recursion will not bottom out until $k = \log_3 n$, implying $T(n) = \Omega(n \log n)$.

Can $k = 2$ do the trick? Plugging $k = 2$ we get

$$T(n) = T(7n/10) + T(n/5) + O(n) \ .$$

If $T(n) = O(n)$ in this case, this means that there exists an integer n_0 and a constant C such that $T(n) \leq Cn$ for all $n \geq n_0$. We can try to prove that this is indeed the case by induction. Here's how we proceed.

First we work informally, i.e., what is written below is "scratch work". We only show it here to see what you need to do to develop the "formal" solution, which we present next. We need to prove that if $T(n) \leq Cn$, then $T(n+1) \leq C(n+1)$. Recall that when we write that $T(n) = \text{blah} + O(n)$ this means that there exists D, n_0 such that $T(n) \leq \text{blah} + Dn$ for all $n \geq n_0$. Applying this we get that for $n \geq n_0$,

$$\begin{aligned} T(n) &\leq T(7n/10) + T(n/5) + Dn && \text{by the definition of } O(\cdot) \text{ applied to the recurrence} \\ &\leq C \cdot 7n/10 + C \cdot n/5 + Dn && \text{by the inductive hypothesis} \\ &= (9C/10 + D)n && \text{factorizing } n \\ &\leq Cn && \text{No reason. That's what we need.} \end{aligned}$$

In particular, in order to have $9C/10 + D \leq C$ we need that $D \leq C/10$, i.e., $C \geq 10D$. We are now ready to write the formal proof.

The fact that $T(n) = T(7n/10) + T(n/5) + O(n)$ means that there exist n_0, D such that $T(n) \leq T(7n/10) + T(n/5) + Dn$ for all $n \geq n_0$. Let $Z = T(7n_0/10) + T(n_0/5)$. We will prove:

1. $T(n_0) \leq (11D + 2Z)n_0$.
2. For every $s \geq n_0$: if $T(s) \leq (11D + 2Z)s$, then $T(s+1) \leq (11D + 2Z)(s+1)$.

The two assertions above imply that $T(n) \leq (11D + 2Z)n$, for every $n \geq n_0$. The first assertion is trivial by the definition of Z since, in fact, $T(n_0) \leq Dn_0 + Z$. For the second assertion observe that

$$\begin{aligned} T(s) &\leq T(7s/10) + T(s/5) + Ds && \text{by the definition of } O(\cdot) \text{ applied to the recurrence} \\ &\leq (11D + 2Z) \cdot (7s/10) + (11D + 2Z) \cdot (s/5) + Ds && \text{by the inductive hypothesis} \\ &= ((9/10)(11D + 2Z) + D)s \\ &= (109D/10 + 18Z/10)s \\ &\leq (11D + 2Z)s \end{aligned}$$

Naturally, the question arises: what is special about $k = 2$? While for $k = 1$ the amount of work in each level of the recursion tree is the same, for $k = 2$, the amount of work is shrinking geometrically. We might be tempted to increase the value of k further so as to reduce the constant factors in the running time. While this works for small k (2, 3, 4), for higher values of k, the benefit of rapidly shrinking subproblems is offset by the increase in the time (c_k) taken to compute the median of each bucket.