Lecture 5: Linear time selection, summary of divide-and-conquer

---

Divide and conquer: high level bits

The last few lectures have been on the divide-and-conquer paradigm for algorithm design. The main idea is to divide a problem instance into smaller instances (often by simply partitioning the input), solving the smaller instances recursively, then "stitching together" the solutions to find a solution to the full instance.

Analyzing divide and conquer algorithms. Typically, we analyze the running time via an inductive analysis -- if $T(I)$ is the running time of the algorithm on an instance I, the run time is the time of the "divide" step, plus the time taken to solve the subproblems ($T(I_1) + \cdots + T(I_r)$, where $I_1, I_2, \ldots$ are the sub-problems), plus the time for the "conquer" or combination step. If the instances are parametrized by their size, then this sort of an analysis results in a "recurrence relation", i.e., the running time on an instance of size n (denoted $T(n)$ for simplicity) can be expressed as some function of $n, T(n-1), T(n-2), \ldots, T(1)$.

Proofs of correctness are also typically done via an inductive argument.

As I emphasized, there's no silver bullet for solving recurrences. There are many techniques, we saw a few of them in action.

Previous lectures

We have seen a couple of examples in the past two lectures. The first was one on Merge Sort, and the second was Integer Multiplication.

In Merge Sort, we used the idea that two sorted arrays of size $n/2$ can be merged in $O(n)$ time to conclude that the running time satisfies $T(n) = 2T(n/2) + O(n)$. We saw using a couple of different methods that this can be "solved" to $T(n) = O(n \log n)$.

In integer multiplication, we saw that the naïve divide and conquer algorithm results in a running time that satisfies $T(n) = 4T(n/2) + O(n)$. Unfortunately this only gives $T(n) = O(n^2)$ (which is the same as the "elementary school algorithm"). We then observed that by an algebraic manipulation, we can only perform three recursive calls, thus resulting in $T(n) = 3T(n/2) + O(n)$. This results in a much nicer running time of $O(n^{\log_2 3}) \approx O(n^{1.57})$.

Indeed, this rough idea can be pursued further (divide into more pieces, combine more cleverly), and this results in a running time of $O(n^{1+\epsilon})$ for any constant $\epsilon > 0$. For more details, look up Toom's algorithm (also see HW1 from 2018). The reason this algorithm isn't too well-known is because it is superseded by one that's based on the Fast Fourier Transform (see the book chapter linked on the homepage for details).

The selection problem

We will see one final example of divide and conquer. Consider the following problem:

Problem. suppose we are given an *unsorted* array $A[]$ with n elements (all distinct), and a parameter k. The goal is to find the kth smallest element of A.

The naïve solution to this problem is to sort the array first and then read off the kth smallest element. This takes time $O(n \log n)$. (It is also known ---as we might see later in the course--- that sorting cannot be done faster than $n \log n$, thus this approach hits a roadblock.)

Can we avoid sorting? Rather surprisingly, the answer turns out to be yes. The algorithm we describe was proposed by Blum, Floyd, Pratt, Rivest and Tarjan in 1973, and is popularly known as the "median of medians" algorithm.

Before getting there, here's the rough intuition for why divide and conquer can be helpful for this problem.

Approximate median. A number x is said to be an approximate median for the array $A[]$ if x is the rth smallest element in the array, for some $r \in [n/4, 3n/4]$.

I.e., an approximate median is an element of the array that's "roughly in the middle" in sorted order.

Key idea. Suppose we have a procedure that can take an array and return an approximate median in time $O(n)$. Then, we claim that selection can be done in $O(n)$ time.

Proof. Let ApproxMedian(A) be the procedure as above. Now, consider the following algorithm:

```
procedure Select(A, k):
  if sizeof(A) = 1 return the (only) element
  find x = ApproxMedian(A)
  define arrays B, C to be empty to start with
  for (i = 0, 1, ..., n-1):
    if A[i] <= x, add A[i] to array B, else add it to array C
  if k <= sizeof(B), return Select(B, k), else return Select(C, k - sizeof(B))
```

Let us start with proving the correctness, i.e., that the procedure correctly outputs the kth smallest element. This is clear if the size of the array is 1 (to be very precise, we need to have $k = 1$, otherwise we need to output an error; for now, let us ignore this technicality). The observation is that no matter how we choose x (whether an approximate median or not), computing the arrays B and C as described and making the appropriate recursive calls produces the right answer, assuming the recursive calls produce the right answer. Since we assumed that as the inductive hypothesis, the proof of correctness follows. (Convince yourself via examples in case you don't see this!)

Next, let us consider the running time. Since x is an approximate median, we must have $\text{sizeof}(B) \leq 3n/4$ as well as $\text{sizeof}(C) \leq 3n/4$. This means that irrespective of how k compares with the size of B, the recursive problem is on an array of size at most $3n/4$. This yields a running time of $T(n) \leq O(n) + T(3n/4)$.

We can simplify this via the "plug-n-chug" method, as follows:

$T(n) \leq cn + T(3n/4) \leq cn + cn(3/4) + T((3/4)^2 n) \leq cn + cn(3/4) + cn(3/4)^2 + \cdots + cn(3/4)^{r-1} + T((3/4)^r n)$
.

Setting r such that $(3/4)^r n = 1$, i.e., $r = \log n / \log(4/3)$, and using the fact that the infinite geometric series $1 + (3/4) + (3/4)^2 + \ldots$ sums to 4, we have that $T(n) \leq 4cn + T(1)$, which means that $T(n) = O(n)$, as desired.

Caveat -- approximate median. The above computation holds assuming that the approximate median can be computed in $O(n)$ time. We describe two methods now. The first is a randomized procedure -- it finds the approximate median in $O(n)$ time, "with high probability". The second is a procedure that is not "a priori" $O(n)$, but turns out to be so.

Probabilistic algorithm: approximate median in practice. In practice, the simplest procedure is to pick a small sample of the array (say 11 elements), and pick the sample-median (in this case the 6th smallest element). Indeed, even if we take a single sample, there is a probability of $1/2$ that we have an approximate median. (Do you see why?)

In fact, one can prove that if $2r + 1$ elements are sampled at random and the $r + 1$'th smallest element in the sample is chosen, the probability that we FAIL to get an approximate median is only $\exp(-r/4)$. This decays really quickly with r.

Deterministic algorithms. While one may prefer the randomized algorithm in practice, the purist might insist that there's still a small probability of failure. Is there a good *deterministic* procedure for the approximate median?

Blum et al. consider the following "median of medians" procedure:

- Suppose we partition the array A into (n/5) sub-arrays of size 5 (i.e., the first five elements, the next five elements, and so on).
- Next, suppose we sort each of the sub-arrays (they are of constant size, so any standard procedure takes constant time per sub-array).
- Let $B[0], B[1], \ldots B[n/5 - 1]$ be the array in which $B[i]$ is the middle (i.e., the third) element of the ith sub-array.
- Finally, return the median (i.e., the $(n/10)$'th smallest element of $B[]$.

Note that for the last step, one actually needs a recursive call to the Select() procedure (albeit with an array of size $n/5$).

The key observation is the following: for any array $A[]$, the median of the array $B[]$ produced as above is an approximate median of $A[]$.

The proof is actually really simple and elegant! In class, we saw a "proof by picture"; let us see here a more detailed proof. Let x be the median of the array $B[]$. By definition, there are $n/10$ elements of $B[]$ that are $\leq x$. Now, the elements $B[i]$ are, in fact, the 3rd smallest elements of some sub-array of $A[]$. Thus we have that at least $3n/10$ elements of $A[]$ are $\leq x$ (three for each $B[i]$ that is $\leq x$). Likewise, we can argue that there are at least $3n/10$ elements of $A[]$ that are $\geq x$. Thus, since $3n/10 < n/4$, if x is the rth smallest element of $A[]$, then $n/4 \leq r \leq 3n/4$. In other words, x is an almost median.

Putting everything together. The main difference between the probabilistic algorithm and the median-of-medians procedure is that the latter itself used Select() as a sub-routine (with an instance of size $n/5$). Thus, if we plug the latter into the procedure Select() described above, using the fact that the *other* steps in the procedure take $O(n)$ time, we end up with a recurrence: $T(n) \leq T(3n/4) + T(n/5) + cn$.

One can try solving this via plug-n-chug, but it becomes fairly messy fairly quickly. However, it can be solved relatively easily via guess-n-prove. We can, indeed, prove that $T(n) \leq 20cn$, assume it is true for $n = 1$, and show the bound by a simple induction.
