CSE 202 NOTES FOR OCTOBER 3, 2002

Randomized Algorithms

Randomized algorithms are algorithms which rely on some source of randomized bits to make decisions in such a way that no one input can systematically elicit worst-case behavior. Typically you analyze the asymptotic behavior of the average case of input for a randomized algorithm, unlike the analysis for deterministic algorithms where you consider worst-possible case input. Doing this average case analysis raises the possibility of some common fallacies:

Average Time is Time for Half the Maximum: One might be tempted to think that $AT(n) = AT(n/2) + O(n)$. After all, if the input is a list and you're selecting a sublist from it to operate on, on average you'll select half the list. Right?

This wrong because you will rarely, if ever, select half the list.

Average Time is the Time For the Average Input: Inputs vary. The random decisions vary. What do we mean by *average*? We could take an average input and see how long the algorithm might take, but this is inherently meaningless (I'd go so far as to say that *average input* is meaningless). Instead, we want to know how our algorithm will behave averaged over all possible random bits it could use on the worst possible input.

More subtly, perhaps, but more important, is that the expected value of a function is not necessarily the same as that function of the expected value. This is true only for certain kinds of functions, most of which are uninteresting. That is, $E[f(x)] = f(E[x])$ is usually false, unless $f(\cdot)$ is of a special form.

A brief introduction into average case analysis can be found in CLRS. Generally one uses so-called *indicator random variables* which take on a value of 1 or 0 depending on if an event happens, then finds the expected value of the summation of some set of these indicator random variables. Two important lemmas are mentioned in appendix C of that book—linearity of expectation, and a calculation of the expected value of an indicator random variable. Apparently some statisticians call indicator random variables *dummy variables*, so don't be confused if you encounter inconsistent terminology.

Order Statistics

We occasionally need to calculate the median of a list of numbers, a recent example of this showed in our discussion of the Closest-Pair-Of-Points algorithm. Our approach there was to sort the numbers and then choose the element in the middle of the list, but this requires $O(n \log n)$ time. We'd like to find a way to calculate the median in linear time, even though this won't help us for the CPoP algorithm, and it seems feasible since we really only need the $n/2$-th smallest element in the list of n things. We know we can find the 1st smallest and the nth smallest with

straightforward functions. Oddly, though, this problem perplexed computer scientists in two ways: first, to solve the problem at all, and second to solve the problem without using a randomized technique. Both algorithms have been found, so you can sleep easy at night knowing that you can find any of the i-th largest elements in linear time.

Consider an approach like QuickSort—we could randomly partition the array such that we pivot on, say, the k-th element and the array is set so that all elements less than k are adjusted to lie to the left of k, and all elements greater or equal to k t the right of k. In this case, we choose k randomly from the list; if i (the rank of the number we're looking for) is less than k then we simply repeat this procedure looking for the i-th smallest element of elements $1..k$, and if i is greater than k then we repeat this procedure looking for the $k-i$-th smallest element of $k..n$. The algorithm is shown in algorithm 1.

Algorithm 1 Linear-time average case median finding algorithm

RSELECT(i, $A[1..n]$): returns i-th largest element of A

```
if n=1 then
  return A[1]
end if
j ← rand(1, n);
small ← {k : A[k] < A[j]};
big ← {k : A[k] ≥ A[j]};
if |big| < i then
  return RSELECT(i − |big|, small);
else if |big| = i then {the i-th smallest element has i things larger than it}
  return A[j]
else
  return RSELECT(i, big);
end if
```

As an illustration of what not to do, we'll apply the fallacies above to this algorithm. What are the average value of |big| and |small|? $n/2$. Therefore, $AT(n) = AT(n/2) + O(n)$ and clearly $T(n) \in O(n)$. This approach is wrong—the average case time really is $O(n)$, but for different reasons. Don't be right for the wrong reasons. Mathematicians hate that.

You might have figured out by now that the worst case time for this algorithm is $O(n^2)$: you could just be extraordinarily unlucky and pick $j = n$, then $j = n-1$, then $j = n-2$, etc. until you get to i. If $i \approx 1$ then this behaves like $T(n) = T(n-1) + O(n)$ which is clearly some function of n^2. However, this would be one of those times where the worst-case analysis really isn't very useful—you could not in any way conspire against the algorithm to produce the worst case result, unless you knew something about the random number generator, which we're assuming is really random and uniformly distributed. Thus, we really do need an asymptotic time analysis for the average case in order to make any meaningful statement about the algorithm.

How should we proceed? We will use an approach that's essentially like the guess-and-prove-by-induction method that we saw earlier. We can break down the

time as

$$AT(n) = \frac{1}{n}\sum_{k=1}^{j-1} AT(n-k) + \frac{1}{n}\sum_{j+1}^{n-1} AT(k) + cn$$

The first term represents the average time for operating on the left half of the list, the second term represents the average time for operating on the right half of the list, and the third term represents both the work done in the algorithm and that one time that we get the perfect value of j the first time. Note that we're taking a summation of average times and dividing by $1/n$ because we're calculating an expectation. (The expectation of a function of a discrete random variable X, $f(x)$ is $E[f(X)] = \sum_{x_i \in U} f(x_i)p(X = x_i)$ where U is the sample space.)

We can generate a guess that $AT(n) \leq c'n$, and then assume for the purposes of induction that $AT(m) \leq c'm, \forall 1 \leq m < n$. Then

$$AT(n) = \frac{1}{n}\sum_{k=1}^{i-1} AT(n-k) + \frac{1}{n}\sum_{k=i+1}^{n-1} AT(k) + cn \tag{0.1}$$

$$\leq \frac{1}{n}\sum_{k=1}^{i-1} c'(n-k) + \frac{1}{n}\sum_{k=i+1}^{n-1} AT(k) + cn \tag{0.2}$$

$$= \frac{i}{n}c'(n - \frac{i}{2}) + (\frac{n-i}{n}c'\frac{n+i}{2} + cn \tag{0.3}$$

$$\leq c'n \tag{0.4}$$

Where 0.1 is a restatement of above, 0.2 is due to the inductive assumption, 0.3 is because of the fact that the average of a *linear* function is its midpoint, and 0.4 is after some algebraic simplification that I won't go into here. (Hint: substitute $\alpha = \frac{i}{n}$ in 0.3 and arrive at an expression that is less than $c'n - \frac{c'n}{4} + cn$ and choose $c' = 4c$).

What we have shown here is that on *any* input, the average of the random decisions is $O(n)$, which is analogous to the worst case behavior that we've been looking at so far, at least in spirit. This is much different than saying "for a randomly chosen input, the execution time is such-and-such": the first is a property of the algorithm, and the latter is a property of the inputs. Of course, in some circumstances it may make sense to look at the distribution of inputs, but that's relatively infrequent.

Previously I alluded to the fact that both a randomized algorithm and a deterministic algorithm for finding the linear-time order statistic exist. The algorithm for doing so is tackled in Knuth and in CLRS, so I won't go into too much detail. The basic idea is that you divide the list of numbers into groups of 5 (yes, you must do this in groups of 5). For each of the groups, you find the medians in constant time (you have 5 elements, so you could just use one big switch statement to figure out the medians). Now, recursively find the median of these $n/5$ medians, and do the partitioning that way. This allows us to prune our search space a little bit better than we were before and leads to a recursion relation of $T(n) \leq T\left(\frac{n}{5}\right) + T\left(\frac{7n}{10}\right) + cn$, which turns out to be linear only because $1/5 + 7/10 = .9$. Again, for more detail, see Knuth and/or CLRS.