Math 261C: Randomized Algorithms

Lecture topic: QuickSort & QuickSelect

Lecturer: Sam Buss
Scribe notes by: Michelle Bodnar
Date: March 31, 2014

Open question: Does randomization help substantially? For instance, is randomized polynomial time (BPP) equal to polynomial time?

First example: Finding medians, or more generally, finding the k^{th} smallest element of an unsorted array.

Our first approach will be a randomized QuickSort algorithm. Given an array A of length n, we can sort the array then pick out the k^{th} element. (Use $k = \frac{n}{2}$ for median). The number of pairwise comparisons needed is $O(n \log n)$.

Algorithm: QuickSort(A, n)

If $n = 1$, return.

Otherwise, choose a pivot $p \in \{0, 1, 2, \ldots, n-1\} = [n]$ uniformly at random.
Let $x = A[p]$.
Linearly scan A and put elements less than x in the first part, and greater than or equal to x in the seco
Run QuickSort on each of these parts.

Thus, the algorithm runs recursively until the array is sorted.

Let a_i^* denote the i^{th} sorted element of A in sorted order. Suppose $i < j$. Consider the question of when does QuickSort compare a_i^* to a_j^*? This happens exactly when a_i^* or a_j^* is the first pivot value picked in $\{a_i^*, a_{i+1}^*, \ldots, a_j^*\}$. Thus,

$$\text{Prob}(a_i^* \text{ and } a_j^* \text{ are compared}) = \frac{2}{j - i + 1}.$$

By linearity of expectation we have

$$\begin{aligned}\mathbb{E}[\#\text{ of comparisons}] &= \sum_{i=0}^{n-2}\sum_{j=i+1}^{n-1}\frac{2}{j-i+1}\\ &\le \sum_{i=0}^{n-2} H_n\\ &= n(\log(n)+O(1))\\ &= n\log(n)+O(n)\end{aligned}$$

where H_n is the n^{th} harmonic number. $H_n = \sum_{k=1}^{n}\frac{1}{k} = \log(n)+\gamma+o(1)$ where $\gamma \approx .577$ is the Euler-Mascheroni constant.

Thus, the expected runtime is approximately $n\log n$, which is good. However, the worst-case performance is $O(n^2)$. This occurs, for instance, if the pivot is chosen to be the next smallest element every time.

Now we return to the problem of finding the k-th element of in the input array A. As already mentioned, one possibility to first sort the array; this is wasteful, however, since it sorts parts of the array that do not contain the k-th element.

Instead, we use QuickSelect is a smarter algorithm. QuickSelect acts like QuickSort, but doesn't call the recursion on the "halves" of the array that aren't needed.

What is the probability that a_i^* is compared to a_j^* by QuickSelect? If $i<j<k$ and a pivot is chosen from the interval (i,j) or $(j,k]$ then a_i^* and a_j^* will never be compared. To see this, observe that if the pivot is in (i,j) then QuickSelect will put a_i^* and a_j^* in different halves of the array, and will never consider a_i^* again. If the pivot is in $(j,k]$ then both a_i^* and a_j^* will go into the half of the array that is never again considered, so they will never be compared. The other cases of $i<k<j$ and $k<i<j$ are handled similarly. Therefore

$$\text{Prob}(a_i^* \text{ and } a_j^* \text{ are compared}) \le \frac{2}{\max\{k-i, j-k, j-i\}+1}.$$

By linearity of expectation,

$$\begin{aligned}\mathbb{E}[\#\text{ of comparisons}] &\le \sum_{i=0}^{n-2}\sum_{j=i+1}^{n-1}\frac{2}{\max\{k-i,j-k,j-i\}+1}\\ &= 2n(1+\ln(2))+O(n)\end{aligned}$$

so the expected time is linear. The final equality is left as a homework assignment, and involves arguing separately the cases of $k<i$, of $i<k<j$, and of $j<k$.

The next lecture will take up the topic of how to do better than QuickSelect. As a preview, how could we do better? It is suggested that we could skew the distribution by randomly

choosing 2 elements. If k is small, we choose the smaller pivot. In general, if we pick $\sqrt{n}$ elements, we can get a good idea of where the k^{th} element might lie. This idea is used in the Floyd-Rivest algorithm which has expected runtime of $n + \min\{k, n-k\} + O(n)$. The Floyd-Rivest algorithm will be discussed in the next lecture.