CS 161 Fall 2017: Section 2

Randomized Algorithms: Majority Element

Suppose we are given an array, A, of length n, with the promise that there exists some number, x, that occurs at least $n/2 + 1$ times in the array. Additionally, we are only allowed to check whether two elements are equal (no comparisons).

Last week, we developed an algorithm to find this in $O(n \log n)$ using divide and conquer. This week, we will develop a randomized algorithm that is expected to run in linear time.

(a) Complete the following pseudo-code for a randomized algorithm that returns the majority element of A.

```
MajorityElement(Input: array A of length n)
  While True
      i = ChooseRandomNumber(1,...,n)   //uniformly random number between 1 and n
```

(b) Is there a bound on the worst-case runtime of the above algorithm?

(c) Prove that the *expected* number of equality checks is at most $2n$.

Select with Different Group Sizes

The algorithm Select finds the k^{th} smallest element in an array of n elements in $O(n)$ time. In class, we chose to split the n numbers into groups of 5 elements, and used the "median of the medians" as a pivot. In this problem, we consider what would have happened if we had split the elements into groups of 3, rather than groups of 5. To simplify your arguments, feel free to assume that all array sizes are divisible by 3 at all recursive stages of the algorithm, and ignore rounding issues. The pseudocode for this new Select algorithm is given below—the only difference between this and the algorithm we saw in class is that the number "3" in ChoosePivot was replaced by "5".

Algorithm 1: SELECT(A, n, k)

```
if n == 1 then
    return A[1];
p ← ChoosePivot(A, n);
A_< ← {A(i) | A(i) < p};
A_> ← {[A(i) | A(i) > p};
if |A_<| = k − 1 then
    return p;
else if |A_<| > k − 1 then
    return Select(A_<, |A_<|, k);
else if |A_<| < k − 1 then
    return Select(A_>, |A_>|, k − |A +_< | − 1);
```

Algorithm 2: CHOOSEPIVOT(A, n)

```
Split A into n/3 groups s_1, ..., s_{n/3} of 3 elements each;
for i = 1 to n/3 do
    m_i = median(s_i); (this takes a constant number of operations, since |s_i| = 3
M = {m_1, ..., m_{n/3}};
return Select(M, n/3, n/6); (this returns the median of M)
```

(a) Suppose we divide a list A of n distinct numbers into $n/3$ groups of 3, and let p be the median of the medians of the sets. (This is the behavior of ChoosePivot, as described above.) Derive an upper bound on the number of elements of A that are greater than p. (This same bound also applies to the number of elements that are less than p, right?)

(b) (2 points) Derive a recurrence for the worst-case run time $T(n)$ for this variant of Select.

(c) (2 points) Using the substitution/"guess and check" method, solve the recurrence from part b) to compute the runtime of this version of the Select algorithm. (*Hint:* The run time may *not* be $O(n)$.)