CS 161 W19: Recitation 2 Problems

January 2019

Exercise 0

Recall from lecture that the algorithm SELECT finds the k^{th} smallest element in an array of n elements in $O(n)$ time. In class, we split the array into groups of 5 elements, and used the "median of medians" as the pivot, as illustrated in the following pseudocode.

```
Algorithm 1: SELECT(A, $k$)
  if $len(A) \leq 50$ then
    A = MergeSort(A)
    return A[$k$-1]
  pivot = GETPIVOT(A)
  L, R = PARTITION(A, pivot)
  if $len(L) == k - 1$ then
    return pivot
  else if $len(L) > k - 1$ then
    return SELECT(L, $k$)
  else
    return SELECT(R, $k$-len(L)-1)
```

```
Algorithm 2: GETPIVOT(A)
  Split A into groups of 5 elements each ($n/5$ groups)
  for $i \in [0, n/5)$ do
    m$_i$ = SELECT(group $i$, 3)
  M = [m$_1$, m$_2$, ..., m$_{n/5}$]
  return SELECT(M, $n/10$)
```

In this problem, we consider what happens to our algorithm if we modify GETPIVOT to use groups of a different size. (To simplify your argument, feel free to ignore rounding issues and floors/ceilings.)

a. Suppose that we instead divide A into $n/3$ groups of size 3, find the median of each group, and then let p be the median of these medians. Derive an upper bound on the number of elements of A that are greater than p. (See the end of the lecture 4 slides, or CLRS Ch. 9 for how this is done with groups of size 5—the proof was skipped in lecture.)

b. Derive a recurrence relation $T(n)$ for the worst-case runtime of SELECT using the modified GETPIVOT with groups of size 3.

c. Using the substitution/"guess and check" method, solve the recurrence from part b to compute the big-O runtime of this version of SELECT. (Hint: The runtime may not be $O(n)$.)

d. Repeat parts a–c where we instead modify GETPIVOT to divide A into $n/7$ groups of size 7.

e. Why do you think we used groups of size 5 in lecture?

f. Advanced (Take Home) - What happens if the group size is not a constant? For instance, what if we modify GETPIVOT to divide A into $\sqrt{n}$ groups of size $\sqrt{n}$?

Exercise 1

You are given a collection of n differently-sized light bulbs that have to be fit into n flashlights in a dark room. You are guaranteed that there is exactly one appropriately-sized light bulb for each flashlight and vice versa; however, there is no way to compare two bulbs together or two flashlights together as you are in the dark and can barely see! (You are, however, able to see where the flashlights and light bulbs are.) You can try to fit a light bulb into a chosen flashlight, from which you can determine whether the light bulb's base is too large, too small, or is an exact fit for the flashlight. If the bulb fits exactly, it will flash once, in which case you have a correct match. (Note that the flashing light does not allow you to visually compare bulbs/flashlights to other bulbs/flashlights.)

Suggest a (possibly randomized) algorithm to match each light bulb to its matching flashlight. Your algorithm should run strictly faster than quadratic time in expectation. Give an upper bound on the worst-case runtime, then prove your algorithm's correctness and expected runtime.

Exercise 2

Suppose you're given n distinct ordered pairs of integers $(x_1, y_1), (x_2, y_2), ..., (x_n, y_n)$, where for all i, j, $x_i \neq x_j$ and $y_i \neq y_j$. Recall that two points uniquely define a line, $y = mx + b$, with slope m and intercept b. (Note that choosing m and b also uniquely defines a line.) We say that a set of points S is *collinear* if they all fall on the same line; that is, for all $(x_i, y_i) \in S$, $y_i = mx_i + b$ for fixed m and b. In this question, we want to find the maximum cardinality subset of the given points which are collinear. Assume that given two points, you can compute the corresponding m and b for the line passing through them in constant time, and you can compare two slopes or two intercepts in constant time.

a. Design an algorithm to find a maximum cardinality set of collinear points in $O(n^2 \log n)$ time. If there are several maximal sets, your algorithm can output any such set. Since we haven't covered hashing yet, your algorithm should not use any form of hash table.

b. It is not known whether we can solve the collinear points problem in better than $O(n^2)$ time. But suppose we know that our maximum-cardinality set of collinear points consists of exactly n/k points for some constant k. Design a randomzed algorithm that reports the points in some maximum-cardinality set in expected time $O(n)$. *(Hint: Your running time may also be expressed as $O(k^2 n)$.)* Prove the correctness and runtime of your algorithms.

c. Is your algorithm from part b guaranteed to terminate?