EECS4020 Algorithms

Homework 2
(On a voluntary basis: No need to submit)

You are encouraged to work in groups. If you wish, you may submit your solution (of any question) to our tutors on or before March 24, 2022, so that they may have a chance to give feedback of your work.

I. Quicksort

1. What is the running time of Quicksort when all elements of array A have the same value?

2. Suppose that the splits at every level of quicksort are in the proportion α to $1 - \alpha$, where $0 < \alpha \leq 1$ is a constant.

   Show that the minimum depth of a leaf in the recursion tree is approximately $-\log n / \log \alpha$, and the maximum depth is approximately $-\log n / \log(1 - \alpha)$.

3. Let us recall an old story that we have talked about in the lecture:

   > *Cinderella's stepmother has bought n bolts and n nuts. The bolts and the nuts are of different sizes, each of them are from size 1 to size n. So, each bolt has exactly one nut just-fitting it.*
   >
   > *These bolts and nuts have the same appearance, so that we cannot distinguish a bolt from another bolt, or a nut from another nut, just by looking at them. Fortunately, we can compare a bolt with a nut by screwing them together, to see if the size of the bolt is too large, or just fitting, or too small, for the nut.*

   Cinderella wants to join the ball held by Prince Charming. But now, her wicked stepmother has mixed the bolts, and mixed the nuts, and then *taken out one of the nut*. Her stepmother tells her that she can go to the ball unless she can find the corresponding bolt. Cinderella knows that she can first use Quicksort to sort the bolts and nuts, and then find out the desired bolt, but this would take expected $O(n \log n)$ comparisons.

   However, the ball will start very soon, and time is not enough for Quicksort. Can you increase Cinderella's chance to meet Prince Charming, by designing a method that takes only expected $O(n)$ comparisons?

II. Lower Bound in Comparison Sorts

1. Suppose that you are given a sequence of n elements to sort. The input sequence consists of n/k subsequences, each containing k elements. The elements in a given subsequence are all smaller than the elements in the succeeding subsequence and larger than the elements in the preceding subsequence. Thus, all that is needed to sort the whole sequence of length n is to sort the k elements in each of the n/k subsequences. Show an $\Omega(n \log k)$ lower bound on the number of comparisons needed to solve this variant of the sorting problem.

   (Note: It is not rigorous to simply combine the lower bounds for the individual subsequences.)

2. Your uncle, Peter, is an owner of 25 horses, and he wants to find out the three fastest horses among these 25 horses.

   Unfortunately, Peter doesn't have any timing device. All he has is a racecourse that allows a competition of five horses each time. In particular, if any five horses are placed in the racecourse for a trial, then we will be able to tell the relative speed of these five horses.

   You want to help him to minimize the number of trials needed to determine the best three horses. Can you design a racing schedule which requires at most 8 trials? How about 7 trials? How about 6 trials?[‡]

III. Sorting in Linear Time

1. Let B be a set of n integers, with each integer taking a value between 0 and n^2.

   Show that the integers in B can be sorted in $O(n)$ time.

2. Show how to use RadixSort to sort the following list of English words:

   COW, DOG, SEA, RUG, ROW, MOB, BOX, TAB,

   BAR, EAR, TAR, DIG, BIG, TEA, NOW, FOX.

3. Let S be a set of n integers, where each integer is in the range 0 to k. We want to preprocesses that the set S, such that when a user gives us any integers ℓ and r with $\ell \leq r$, we can use $O(1)$ time to count how many of integers in S are within the range $[\ell..r]$.

   Show how to accomplish the task using $O(n + k)$ preprocessing time and space.

IV. Order Statistics

1. Let A be an array of n distinct numbers. In the lecture, we have learnt that finding the kth smallest number of A, for any k, can be done in $O(n)$ time.

   Suppose that we now want to know more about the array. In particular, we want to find out the 2^jth smallest numbers of A, for all $j = 0, 1, \ldots, \lfloor \log n \rfloor$. (That is, we simultaneously want know which numbers are respectively the 1st, the 2nd, the 4th, the 8th, ... smallest numbers of A.)

   Design an $O(n)$-time algorithm to accomplish the above task.

2. Similar to Question 1, but this time we want to get the first $\sqrt{n}$ smallest numbers of A in sorted order. Show that the above can also be done in $O(n)$ time.

3. Similar to Question 1, but this time we want to get the k numbers which are closest to the median. However, we are not required to sort these k numbers when we output them.

   For example, suppose the array A is as follows:

   $$A = \langle 2.3, 0.1, 2.1, 1.6, 0.5, 3.7, 2.2 \rangle.$$

   Then, the median of the array is 2.1, and the 3 numbers closest to the median would be $2.1, 2.2$, and 2.3.

   Design an $O(n)$-time algorithm to accomplish this task. Note that your algorithm needs to work for any value of k.[†]

[‡]This question was once asked during Kai's job interview ^o^;

[†]This question is a bit difficult; try to spend some more time on it. Add oil!