CS5050 Advanced Algorithms

Fall Semester, 2015

Assignment 3: Prune and Search

Due Date: 8:30 a.m., Friday, Oct. 9, 2015 (**at the beginning of CS5050 class**)

1. Suppose you are given an array A with n entries, with each entry holding a distinct number. You are told that the sequence of values $A[1], A[2], \ldots, A[n]$ is *unimodal:* For some index p between 1 and n, the values in the array entries increase up to position p in A and then decrease the remainder of the way until position n. (So if you were to draw a plot with the array position j on the x-axis and the value of the entry $A[j]$ on the y-axis, the plotted points would rise until x-value p, where they'd achieve their maximum, and then fall from there on.)

   You'd like to find the "peak entry" p without having to read the entire array – in fact, by reading as few entries of A as possible. Show how to find the entry p by reading at most $O(\log n)$ entries of A. In other words, design an $O(\log n)$ time algorithm to find the peak entry p. **(20 points)**

2. In the SELECTION algorithm we studied in class, the input numbers are divided into groups of five. Will the algorithm still work in linear time if they are divided into groups of seven? Please justify your answer. **(20 points)**

3. Here is a generalized version of the selection problem, called *multiple selection*. Let $A[1 \cdots n]$ be an array of n numbers. Given m integers $k_1, k_2, \ldots, k_m$, with $1 \leq k_1 < k_2 < \cdots < k_m \leq n$, the *multiple selection problem* is to find the k_i-th smallest number in A for all $i = 1, 2, \ldots, m$. For simplicity, we assume that no two numbers of A are equal.

   (a) Design an $O(n \log n)$ time algorithm for the problem. **(5 points)**

   (b) Design an $O(nm)$ time algorithm for the problem. Note that this is better than the $O(n \log n)$ time algorithm if $m < \log n$. **(5 points)**

   (c) Improve your algorithm to $O(n \log m)$ time, which is better than both the $O(n \log n)$ time and the $O(nm)$ time algorithms. **(15 points)**

4. Suppose we are given a "black-box" which is a program procedure such that given any real number x as the input, the procedure can tell whether x is a *feasible value* in constant time. Further, we have the following **basic rules**: If x is a feasible value, then any number *less than* x is also a feasible value; on the other hand, if x is not a feasible value, then any number *larger than* x is not a feasible value.

   Given an array $A[1, \ldots, n]$ of n real numbers, we want to find all feasible values of the elements of A by using the black-box. For each element x of A, we can call the black-box on x to determine whether x is feasible value. In this way, by calling the black-box n times, we

can find all feasible values of A in $O(n)$ time. However, by making use of the above basic rules, it is possible to find all feasible values of A by calling the black-box significantly less than n times. For simplicity, we assume that no two numbers of A are equal.

(a) Design an $O(n \log n)$ time algorithm to find all feasible values of A by calling the black-box at most $O(\log n)$ times. **(10 points)**

(b) Improve your algorithm to $O(n)$ time such that the total number of calls on the black-box is still at most $O(\log n)$. **(15 points)**

Note that you are not responsible for designing the black-box and just assume it is given as a program procedure for you to use.

Total Points: 90