CSE 202: Design and Analysis of Algorithms Spring 2015

Problem Set 2

Instructor: Fan Chung Graham **Due on:** Wed April 29

Instructions

- For your proofs, you may use any lower bound, algorithm or data structure from the text or in class, and their correctness and analysis, but please cite the result that you use.
- If you do not prove that your algorithm is correct, we will assume that it is incorrect. If you do not provide an analysis of the running time, we will assume you do not know what the running time is.

Problem 1

An array $A[1 \dots n]$ is said to have a majority element if more than half of its entries are the same. Given an array, the task is to design an efficient algorithm to tell whether the array has a majority element, and, if so, to find that element. The elements of the array are not necessarily from some ordered domain like the integers, and so there can be no comparisons of the form *is A[i] > A[j]?* (Think of the array elements as GIF files, say.) However you can answer questions of the form: *is A[i] = A[j]?* in constant time.

1. Show how to solve this problem in $O(n \log n)$ time. (Hint: Split the array A into two arrays A_1 and A_2 of half the size. Does knowing the majority elements of A_1 and A_2 help you figure out the majority element of A? If so, you can use a divide-and-conquer approach.)

   Grading: 10 points where it will be graded with the grading rubric and divided by 2.

2. Can you give a linear-time algorithm? (Hint: Here's another divide-and-conquer approach:

   - Pair up the elements of A arbitrarily, to get $n/2$ pairs
   - Look at each pair: if the two elements are different, discard both of them; if they are the same, keep just one of them

   Show that after this procedure there are at most $n/2$ elements left, and that they have a majority element if A does.)

   Grading: 10 points where it will be graded with the grading rubric and divided by 2.

Problem 2

Given an unlimited supply of coins of denominations $x_1, x_2, \dots, x_n$, we wish to make change for a value v; that is, we wish to find a set of coins whose total value is v. This might not be possible: for instance, if the denominations are 5 and 10 then we can make change for 15 but not for 12. Give an $O(nv)$ dynamic-programming algorithm for the following problem:

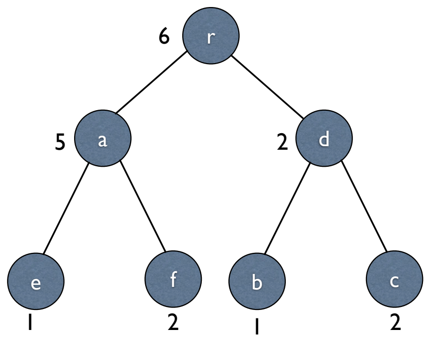
Given an input $x_1, \dots, x_n; v$, is it possible to make change for v using coins of denominations $x_1, \dots, x_n$?

Grading: 20 points as described in the grading rubric.

Problem 3

Suppose you are given a rooted binary tree T in which each node x has a score $s(x)$. A pruning of T is a set C of nodes in T with the property that C contains exactly one node on any root to leaf path. Thus, if a node x is in C, no ancestor or descendant of x can be in C.

The size of a pruning C is the number of nodes in C, and the score of a pruning is the sum of the scores of the nodes in it. For example, the following tree has 2 prunings of size 3: $C_1 = \{a, b, c\}$ with score 8 and $C_2 = \{d, e, f\}$ with score 5, and one pruning of size 2: $C_3 = \{a, d\}$, with score 7.



Given a binary tree T, with root r, and the scores $s(x)$ for each node x of T, design a dynamic programming algorithm to find the score of a pruning C of T such that (a) the size of C is exactly k (b) the score of C is as large as possible. To get any credit, the running time of your algorithm should be polynomial in n and k. (Hint: The exhaustive search algorithm that checks all possible prunings of size k has $n^{O(k)}$ running time.)

Grading: 20 points as described in the grading rubric.