CSE 203 – Data Structure and Algorithms I Assignment

Due: 09/06/2017 (Friday)

You are allowed to consult with upto three classmates and online resources for hints but the write-up has to be your own. Any copying of answers may lead to no marks for all relevant parties.

1. (5 points)

Suppose you are choosing between the following three algorithms:

- Algorithm A solves problems by dividing them into five subproblems of half the size, recursively solving each subproblem, and then combining the solutions in linear time.
- Algorithm B solves problems of size n by recursively solving two subproblems of size $n-1$ and then combining the solutions in constant time.
- Algorithm C solves problems of size n by dividing them into nine subproblems of size $n/3$, recursively solving each subproblem, and then combining the solutions in $O(n^2)$ time.

What are the running times of each of these algorithms (in big-O notation), and which would you choose?

2. (5 points)

An array $A[1 \ldots n]$ 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.

(a) 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.)

(b) 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 and only if A does.)

3. (5 points)

Show that the greedy algorithm for Huffman encoding is optimal. In a text the characters a, b, c, d, e, f appear 68, 13, 26, 35, 102, 18 times respectively. What is the Huffman encoding of the alphabet?

4. (5 points)

Provide dynamic programming algorithm for one of the following problems. What is the running time of your algorithm?

(a) **Weighted interval scheduling:** (Kleinberg and Tardos 6.1) We have n requests labeled $1, \ldots, n$, with each request i specifying a start time s_i and a finish time f_i. Each interval i also has a value, or weight v_i. Two intervals are compatible if they do not overlap. The goal of our problem is to select a subset $S \subseteq \{1, \ldots, n\}$ of mutually compatible intervals, so as to maximize the sum of the values of the selected intervals, $\sum_{i \in S} v_i$.

(b) **0-1 Knapsack:** (Dasgupta *et al.* 6.4) We have a set of items with value and weight of i-th item given by v_i and w_i respectively. We want to pick a subset of items such that total value of items picked is maximized with the restriction that total weight of items picked cannot exceed W. (Note: you can pick each item either 0 or 1 time).

(c) **Change making:** You are given n types of notes given by their denominations $d_1, \ldots, d_n$ and a value, V. Give an algorithm to make change for V using fewest number of notes. You have unlimited number of notes for each type. (Note: the coin system may not be canonical i.e. greedy algorithm may not give an optimal solution)

5. (10 points)

Design and implement a dynamic programming algorithm for one of the following problems. You must use the bottom-up approach.

(a) **Edit distance:** (Dasgupta *et al.* 6.3) You are given two strings s and t of lengths m and n respectively. Find the minimum number of edits - insertions, deletions, and substitutions of characters - needed to transform the first string into the second as well as the sequence of edits. Your algorithm should run in $O(mn)$ time.

For example for the sample input,

```
EXPONENTIAL
POLYNOMIAL
```

your output should be (actual edits may vary)

```
6
EXPONEN-TIAL
--POLYNOMIAL
```

(b) **Longest common subsequence:** (Cormen *et al.* 15.4) You are given two strings s and t of lengths m and n respectively. Find the longest common subsequence of the two strings as well as its length. Your algorithm should run in $O(mn)$ time.
For example for the sample input,

```
ABCBDAB
BDCABA
```

your output should be (actual subsequence may vary)

```
4
BCBA
```

(c) **Matrix chain multiplication:** (Cormen *et al.* 15.2) You are given a sequence of matrices $A_1, A_2, \ldots, A_n$. Find the order in which the matrices should be multiplied to minimize the total number of multiplications as well as the total number multiplications needed. Your algorithm should run in $O(n^3)$ time.
For example for the sample input,

```
A 40 20
B 20 30
C 30 10
D 10 30
```

your output should be

```
26000
((A (B C)) D)
```

6. (5 points)

A mission-critical production system has n stages that have to be performed sequentially; stage i is performed by machine M_i. Each machine M_i has a probability r_i of functioning reliably and a probability $1 - r_i$ of failing (and the failures are independent). Therefore, if we implement each stage with a single machine, the probability that the whole system works is $r_1 \cdot r_2 \cdots r_n$. To improve this probability we add redundancy, by having m_i copies of the machine M_i that performs stage i. The probability that all m_i copies fail simultaneously is only $(1 - r_i)^{m_i}$, so the probability that stage i is completed correctly is $1 - (1 - r_i)^{m_i}$ and the probability that the whole system works is $\prod_{i=1}^{n}(1 - (1 - r_i)^{m_i})$. Each machine M_i has a cost c_i, and there is a total budget B to buy machines. (Assume that B and c_i are positive integers.)

Given the probabilities $r_1, \ldots, r_n$, the costs $c_1, \ldots, c_n$, and the budget B, find the redundancies $m_1, \ldots, m_n$ that are within the available budget and that maximize the probability that the system works correctly.