Due: June 5, in tutorial

Note: Assignments are due at the *beginning* of tutorial, and since solutions are discussed in tutorial, late assignments are *not* accepted. The work that you submit *must* be your own. You may discuss problems with each other; however, you should take no notes during these discussions, and you should prepare written solutions alone.

1. Suppose the runtime of an algorithm is $T(n)$. Indicate which of the expressions below for $T(n)$ would make the algorithm polytime.

a) 3^{n^2}	d) $1000n^{17} + n^2 \log_2 n$	g) $2^{\sqrt{n}}$
b) $3^{\log_2 n}$	e) $n!$	h) $(0.5)^n$
c) $n^{1/n}$	f) $2^{(\log_2 n)^2}$	i) $2^{1000}n$

2. (a) Suppose that the edge costs are all distinct in an input G to the MST problem. Prove that G has a unique minimum cost spanning tree. Do this by considering the proof that Kruskal's algorithm always finds a minimum cost spanning tree.

   (b) Now suppose that all edge costs are distinct, except $c_1 = c_2$, where $c_1 = c(e_1)$ and $c_2 = c(e_2)$ are the two minimum edge costs. Prove or give a counter-example: G has a unique minimum cost spanning tree.

   (c) Finally suppose that all edge costs are distinct, except $c_1 = c_2 = c_3$, where $c_1 = c(e_1)$, $c_2 = c(e_2)$, $c_3 = c(e_3)$ are the three minimum edge costs. Prove or give a counter-example: G has a unique minimum cost spanning tree.

3. The Make-Change Problem consists in paying a given amount using the least number of coins, using some fixed denominations, and an unlimited supply of coins of each denomination. Consider the following greedy algorithm to solve the change making problem, where the denominations are $\{1, 10, 25, 100\}$:

function MakeChange(N) (*makes change for N units using coins from C*)
    const $C \leftarrow \{1, 10, 25, 100\}$
    $L \leftarrow \emptyset$ (*L is the list holding the solution*)
    $s \leftarrow 0$ (*s is the sum of the items in L*)
    while $s < N$ do
        $x =$ largest item in C such that $s + x \leq N$
        $L \leftarrow L, x$
        $s \leftarrow s + x$
    return L

   (a) Show that this greedy algorithm (with the given denominations) does *not* necessarily produce an optimal solution. That is, present an N for which the output L of MakeChange(N) contains more coins than the optimal solution.

(b) Suppose now that the available denominations are $\{1, p, p^2, \ldots, p^n\}$, where $p > 1$ and $n \geq 0$ are integers. That is, "$C \leftarrow \{1, 10, 25, 100\}$" in the above algorithm is replaced by "$C \leftarrow \{1, p, p^2, \ldots, p^n\}$." Show that with this series of denominations (for some fixed p, n) the greedy algorithm above always finds an optimal solution.

4. Consider the following variation of the Longest Monotone Subsequence problem:

Input: $d, a_1, a_2, \ldots, a_d \in \mathbb{N}$.

Question: What is the length of the longest subsequence of $a_1, a_2, \ldots, a_d$, where any two consecutive members of the subsequence differ by at most 1?

For example, the longest such subsequence of $\{7, 6, 1, 4, 7, 8, 20\}$ is $\{7, 6, 7, 8\}$, so in this case the answer would be 4.

Give a dynamic programming solution to this problem by doing the following three steps:

(a) Define an array R of subproblems. Explain how to obtain the solution from R.

(b) Find a recurrence (including initial conditions) that could be used to compute the values in the array R.

(c) Write an algorithm, at a high level, for computing the values in the array R.

5. Consider the following String Matching Problem:

Input: Integers $m, m \in \mathbb{N}$ and strings $s, t_1, \ldots, t_m$ such that $|s| = n$.

Question: Can string s be expressed as a concatenation of some of the t_i's (allowing repetition)?

For example, if $n = 9, m = 3, s = 011000101, t_1 = 0000, t_2 = 100, t_3 = 01$, then the answer is YES because $s = t_3 t_2 t_3 t_3$. On the other hand, if $s = 001110$ with the same t_1, t_2, t_3, then the answer is NO because no concatenation of the t_i's can produce the three 1s in a row.

Give a dynamic programming solution to this problem by doing the following three steps:

(a) Define an array R of subproblems. Explain how to obtain the solution from R.

(b) Give a recurrence (including initial conditions) that could be used to compute the values in the array R.

(c) Give, at a high level, an algorithm for computing the values in the array R. Estimate the runtime of your algorithm using big-Oh notation.