CS 141, Spring 2019 Midterm 2/B

First name:

Last name:

Student ID:

- This exam is **closed book, closed notes**, 80 minutes long
- Read the questions carefully
- No electronic equipment allowed (cell phones, tablets, computers, . . .)
- Write legibly. What can't be read will not be graded
- Use pseudocode, Python, or English to describe your algorithms (no C/C++/Java)
- When designing an algorithm, you are allowed to use any algorithm or data structure we explained in CS 141 or CS 14, without giving its details, unless the question specifically requires that you give such details
- Always remember to analyze the time complexity of your solution
- If you have a question about the meaning of a question, come to the front of the class

Problem 1. (15 points [greedy])

Draw the Huffman tree and find the optimal prefix code for the symbols in the following frequency table

symbol	frequency	code
A	10	
B	5	
C	4	
D	3	
E	20	
F	1	
G	2	

Problem 2. (15 points [greedy])

In the fractional knapsack problem we discussed in class, we are supposed to choose among n items, where each item i has a positive benefit b_i and a positive weight w_i; we are also given the size of the knapsack W. The problem is to find the amount x_i of each item i which maximizes the total benefit $\sum_{i=1}^{n} x_i(b_i/w_i)$ under the condition that $0 \leq x_i \leq w_i$ and $\sum_{i=1}^{n} x_i \leq W$.

Write the pseudo-code for the greedy algorithm for fractional knapsack we discussed in class.

Problem 3. (15 points [greedy proof])

You are given two unsorted arrays $A = \{a_1, a_2, \ldots, a_n\}$ and $B = \{b_1, b_2, \ldots, b_n\}$ of n distinct positive integers. The objective is to find an ordering of A and B so that $W = \prod_{i=1}^{n} a_i^{b_i}$ is maximized. Consider the following greedy algorithm.

Algorithm GREEDY(A : array, B : array)
  sort A in decreasing order
  sort B in decreasing order
  return (A, B)

We claim that the ordering computed by GREEDY is optimal. Prove that GREEDY has the greedy choice property for this problem.

Problem 4. (15 points [dynamic programming])

We want to extend the LCS dynamic programming algorithm we covered in class to find the longest common subsequence between three strings X, Y and Z. Let X_i be a prefix of string X of length i, Y_j be a prefix of string Y of length j, and Z_k be a prefix of string Z of length k. If we define $C[i, j, k]$ to store the length of the longest common subsequence between X_i, Y_j and Z_k, then

$$C[i,j,k] = \begin{cases} & \text{if } i = 0 \text{ or } j = 0 \text{ or k=0} \\ & \\ & \text{if} \\ & \\ & \text{if} \end{cases}$$

Problem 5. (15 points [dynamic programming - black box])

A string Y is a *palindrome* if $Y^R = Y$, where Y^R is the reverse of Y. Given a string X of length n, we want to find the minimum number of characters that need to be inserted in X to make X a palindrome. For instance, X=`Ab3bd` can become `dAb3bAd` or `Adb3bdA` by inserting two characters (one `d`, one `A`). Give a $O(n^2)$-time dynamic programming algorithm for this problem.
Hint: Compute X^R and use one of the algorithms we discussed in class a black-box.

Problem 6. (15 points [dynamic programming])

A string Y is a *palindrome* if $Y^R = Y$, where Y^R is the reverse of Y. Given a string X a partitioning of X is a *palindrome partitioning* if every substring of the partition is a palindrome. For example, `aba|bbb|a|bb|a|b|aba` and `aba|b|bbabb|ababa` are two palindrome partitioning of $X =$ `ababbbabbababa`. Design a dynamic programming algorithm to determine the coarsest (i.e., fewest cuts) palindrome partitioning of X. In the example, the second partition (3 cuts) is optimal. Remember to analyze the space- and time-complexity of your solution.

Hint: Define the dynamic programming table $C[i]$ to be number of cuts in the best palindrome partition of X_i, where X_i is the prefix of X of length i.

$$C[i] = \begin{cases} \\ \\ \\ \\ \\ \end{cases}$$

The time complexity is

The space complexity is