B503 Final Exam Practice

Problem 1. Given n pieces of wood and a length array L, we want to cut them into small pieces to guarantee you could have **no less** than k pieces with the **same** length. What is the longest length you can get from the n pieces of wood? (You couldn't cut the wood into length in float. If you couldn't get $\geq k$ pieces, return 0.) For example, given $L = [232, 124, 456]$, $k = 7$, we return 114. Assume N is the maximum length of woods, what is the best time complexity you could finish the task.

Problem 2. Given an array of non-negative integers, you are initially positioned at the first index of the array. Each element in the array represents your maximum jump length at that position. Design an algorithm to determine if you are able to reach the last index.

For example, given $[2, 3, 0, 1, 4]$, we return true because we could jump 1 step from index 1 to 2, then 3 steps to the last index. Given $[3, 2, 1, 0, 4]$, we return false because we will always arrive at index 4 at last.

Problem 3. Consider the following statement, decide whether it is true or false. If it is true, give a short explanation. If it is false, give a counterexample.

Suppose we are given an instance of the Shortest $s - t$ Path Problem on a directed graph G. We assume that all edge costs are positive and distinct. Let P be a minimum-cost $s - t$ path for this instance. Now suppose we replace each edge cost c_e by its square, c_e^2 , thereby creating a new instance of the problem with the same graph but different costs.

True or false? P must still be a minimum-cost $s - t$ path for this new instance.

Problem 4. Given a graph G and its MST T, we construct a new graph $G' = G \cup \{e\}$ by adding an edge $e = (u, v)$ (suppose there is no edge between u, v in G). Design an algorithm to find the MST of G' with running time $O(n)$, where n is number of nodes in G.

Problem 5. Given n, k, design an efficient algorithm to compute n^k. What is the time complexity of your algorithm?

Problem 6. Given two **sorted** arrays A, B, each of which has n elements, design an algorithm to find the median of the two sorted arrays with $O(\log n)$ time complexity.

Problem 7. A palindrome string is a sequence of characters that reads the same backward as forward, such as "madam" or "racecar". Given a string s, find the longest palindromic (continuous) substring in s. For instance, given "babad", we return "bab" or "aba".

Problem 8. There are n coins with different values in a line. Two players take turns to take one or two coins from the left side until there are no more coins left. The player who takes the coins with the most value wins. Assume both players are smart enough and will choose the best strategy, could you please decide the **first** player will win or lose?

For example, given $[1, 2, 2]$, the first player will win since he can take 2 coins at first. Given $[1, 2, 4]$, the first player will lose no matter how many coins he takes in the first step.

Problem 9. Given an array A of integers, design an algorithm to return the length of the longest arithmetic subsequence in A. Recall that a subsequence of A is a list $A[i_1], A[i_2], \ldots, A[i_k]$ with $1 \le i_1 < i_2 < \ldots < i_k \le A.length$, and that a sequence B is arithmetic if $B[i+1] - B[i]$ are all the same value (for $1 \le i \le B.length$).

For example, given $[9, 4, 7, 2, 10]$, we return 3, since the longest arithmetic subsequence is $[4, 7, 10]$.

Problem 10. You have an array where the i-th element is the price of a given stock on the day i. Design an algorithm to find the maximum profit, when you may complete **at most two** transactions. Note that, you may not engage in multiple transactions on the same day (i.e., you must sell the stock before you buy again).

For example, given $[3, 3, 5, 0, 0, 3, 1, 4]$, we output the maximum profit 6, since we could buy on day 4 and sell on day 6, and then buy on day 7 and sell on day 8. Given $[7, 6, 4, 3, 1]$, we return 0, since the best solution is to make no transaction.

Problem 11. One of the (many) hard problems that arises in genome mapping can be formulated in the following abstract way. We are given a set of n markers $\{\mu_1, \ldots, \mu_n\}$–these are positions on a chromosome that we are trying to map—and our goal is to output a linear ordering of these markers. The output should be consistent with a set of k constraints, each specified by a triple (μ_i, μ_j, μ_k), requiring that μ_j lies between μ_i and μ_k in the total ordering that we produce. (Note that this constraint does not specify which of μ_i or μ_k should come first in the ordering, only that μ_j should come between them.)

Design an algorithm with the following property: If it is possible to satisfy k^* of the constraints, then the algorithm produces an ordering of markers satisfying at least $1/3k^*$ of the constraints. Your algorithm may be randomized; in this case it should produce an ordering for which the expected number of satisfied constraints is at least $1/3k^*$.