Instructions: Please make sure that your solutions are well-thought-out, complete, concise, rigorous, and clearly written. Each solution sheet should be marked with your name, the course number, the homework number, the problem number and the date. You are allowed to discuss problems and exchange solution ideas with other students, but you must write up your own solutions. If you collaborated with other students on a problem, please acknowledge them in your write up. These are the fun problems to solve. Some of them require a bit of thinking. Please start early!

Problem 1: (6 + 7 + 7 = 20 Points)
In the class, we have studied three binary tree traversal algorithms, viz. PREORDER, INORDER and POSTORDER traversals, which visit all the nodes in the tree and print them in order specified by the respective algorithms. In this question, you are asked to do the opposite, that is, to reconstruct a binary tree from the traversal results. Note that a binary tree is not necessarily a binary search tree.

a. Suppose you are given the preorder traversal and the postorder traversal of a binary tree. Can you reconstruct the binary tree? If so, please give an algorithm to do so, and analyze its time complexity. If not, the give a counterexample.

b. Suppose you are given the preorder traversal and the inorder traversal of a binary tree. Can you reconstruct the binary tree? If so, please give an algorithm to do so, and analyze its time complexity. If not, the give a counterexample.

c. Suppose you are given the inorder traversal and the postorder traversal of a binary tree. Can you reconstruct the binary tree? If so, please give an algorithm to do so, and analyze its time complexity. If not, the give a counterexample.

Problem 2: (7 + 7 + 6 = 20 Points)
This question is about making the binary search dynamic. Binary search of a sorted array takes logarithmic search time, but the time to insert a new element is linear in the size of the array. We can improve the time for insertion by keeping several sorted arrays.

Specifically, suppose that we wish to support SEARCH and INSERT on a set of $n$ elements. Let $k = \lceil \log(n + 1) \rceil$, and let the binary representation of $n$ be $\langle n_{k-1}, n_{k-2}, \ldots, n_0 \rangle$. We have $k$ sorted arrays $A_0, A_1, \ldots, A_{k-1}$, where for $i = 0, 1, \ldots, k-1$, the length of array $A_i$ is $2^i$. Each array is either full or empty, depending on whether $n_i = 1$ or $n_i = 0$, respectively. The total number of elements held in all $k$ arrays is therefore $\sum_{i=0}^{k-1} n_i 2^i = n$. Although each individual array is sorted, there is no particular relationship between the elements in different arrays.

a. Describe how to perform the SEARCH operation for this data structure. Analyze its worst-case running time.

b. Describe how to insert a new element into this data structure. Analyze its worst-case and amortized running times.

c. Discuss how to implement DELETE.
Problem 3: (6 + 7 + 7 = 20 Points)
Suppose you have two lists of positive numbers \( \{ a_1, \ldots, a_n \} \) and \( \{ b_1, \ldots, b_n \} \), all \( a_i \) and \( b_j \) are distinct. You are asked to compute the product \( P = \prod_{1 \leq i, j \leq n} a_i b_j \), so that \( P \) is the maximum. Note that no repetition of \( a_i \) is allowed in the product. Also, no repetition of \( b_j \) is allowed. Give an algorithm to compute \( P \). What is the greedy choice you have used? Prove that your greedy choice property gives you optimal solution.

Problem 4: (4 + 4 + 4 + 3 = 15 Points)
Solve Exercise 3.3 from DPV. Note that \( \text{pre} \) and \( \text{post} \) numbers are respectively \text{start} and \text{finish} times.

Problem 5: (10 Points)
Solve Exercise 3.11 from DPV.

Problem 6: Undirected vs. directed connectivity (5 + 5 + 5 = 15 Points)
Solve Exercise 3.13 from DPV.

Problem 7: Euler tour (13 + 12 = 25 Points)
An Euler tour of a strongly connected, directed graph \( G = (V, E) \) is a cycle that traverses each edge of \( G \) exactly once, although it may visit a vertex more than once.

a. Show that \( G \) has an Euler tour if and only if in-degree(\( v \)) = out-degree(\( v \)) for each vertex \( v \in V \).

b. Describe an \( O(E) \)-time algorithm to find an Euler tour of \( G \) if one exists.