1. **Consider a variation of the Knapsack problem where we have two knapsacks, with integer capacities W1 and W2. As usual, we are given n items with positive values and positive integer weights. We want to pick subsets S1,S2 with maximum total value (i.e., ∑i∈S1vi+∑i∈S2vi) such that the total weights of S1 and S2 are at most W1 and W2, respectively. Assume that every item fits in either knapsack (i.e., wi≤min{W1,W2} for every item i). Consider the following two algorithmic approaches. (1) Use the algorithm from lecture to pick a a max-value feasible solution S1 for the first knapsack, and then run it again on the remaining items to pick a max-value feasible solution S2 for the second knapsack. (2) Use the algorithm from lecture to pick a max-value feasible solution for a knapsack with capacity W1+W2, and then split the chosen items into two sets S1,S2 that have size at most W1 and W2, respectively. Which of the following statements is true?**
2. **Recall the dynamic programming algorithms from lecture for the Knapsack and sequence alignment problems. Both fill in a two-dimensional table using a double-for loop. Suppose we reverse the order of the two for loops. (I.e., cut and paste the second for loop in front of the first for loop, without otherwise changing the text in any way.) Are the resulting algorithms still well defined and correct?**
3. **Consider an instance of the optimal binary search tree problem with 7 keys (say 1,2,3,4,5,6,7 in sorted order) and frequencies w1=.05,w2=.4,w3=.08,w4=.04,w5=.1,w6=.1,w7=.23. What is the minimum-possible average search time of a binary search tree with these keys?**

Solution : 2.18

1. **The following problems all take as input two strings X and Y, of length m and n, over some alphabet Σ. Which of them can be solved in O(mn) time? [Check all that apply.]**
2. **Recall our dynamic programming algorithms for maximum-weight independent set, sequence alignment, and optimal binary search trees. The space requirements of these algorithms are proportional to the number of subproblems that get solved: Θ(n) (where n is the number of vertices), Θ(mn) (where m,n are the lengths of the two strings), and Θ(n2) (where n is the number of keys), respectively.**

**Suppose we only want to compute the value of an optimal solution (the final answer of the first, forward pass) and don’t care about actually reconstructing an optimal solution (i.e., we skip the second, reverse pass over the table). How much space do you then really need to run each of three algorithms?**