Problem Set 8

Assigned: July 15
Due: July 22

Problem 1

A. Give an $O(n^2)$ algorithm that, given a sequence S, finds the longest subsequence that first increases then decreases. For instance, in the sequence $S = [10, 4, 5, 11, 2, 7, 4, 3, 9]$ the longest such subsequence is [4,5,11,7,4,3]. The subsequence does not have to be consecutive. (Hint: Use two arrays, one for increasing subsequences and the other for decreasing subsequences.)

B. Consider the modification of (A) which requires that you find the longest subsequence that first increases, then decreases, and does not repeat any values. For instance, given the input sequence, 10,4,5,11,2,7,4,3,9 the subsequence 4,5,11,7,4,3 would be disallowed because the value 4 is repeated. The longest valid subsequence would 4,5,11,7,3. Explain why the kind of dynamic programming technique used in (A) *cannot* be adapted to solve this problem.

Problem 2

The KNAPSACK problem is defined as follows: You are given a collection of objects. Each object `X` has a value `X.value` and a weight `X.weight`. You are packing a knapsack and there is a maximum weight W that you can carry. The problem is to choose the objects so that their total weight is at most W, and their total value is as large as possible.

In general, if the weights are floating point numbers or large integers, then the problem is believed to be intractable (that is, there is no efficient solution.) However, if all the weights involved are small integers, then there is a solution which is polynomial time in W.

Find an efficient dynamic programming solution to the problem, on the assumption that the weights and W are all small positive integers. State the running time of your algorithm as a function of n, the number of objects, and W.

Write the algorithm so that the optimal set (not just the optimal value) can be easily recovered, and describe how the set is recovered.

Hint: For $k = 1$ to W, for $j = 1$ to n, find the most valuable subset of the first j objects whose total weight is exactly k.

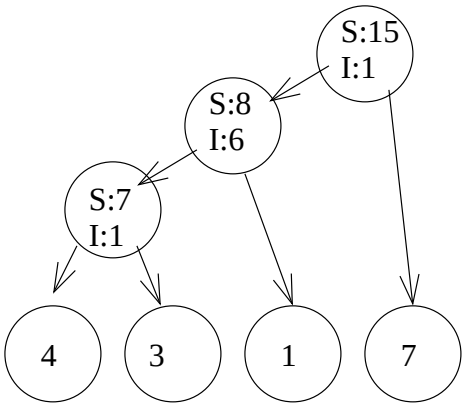
Problem 3

Let T be a binary tree whose leaves are labelled with numbers. For any internal node N of T, we define the *imbalance* of N to be the absolute value of the difference between the sum of values in the left subtree and the sum in the right tree. Define the *overall imbalance* of T to be the maximum imbalance of all internal nodes of T.

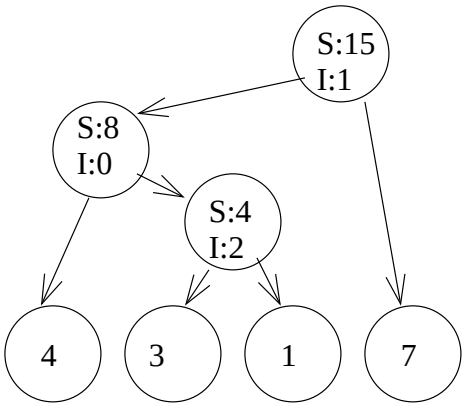
Given a sequence of numbers S, the *most balanced* tree for S is the tree whose leaves are S in the specified order with the smallest overall imbalance.

Write a dynamic programming algorithm that computes the most balanced tree for any sequence S. Hint: Use two arrays. $S[i, j]$ is the sum of elements i through j. $B[i, j]$ is the overall imbalance of the best subtree spanning elements i through j.

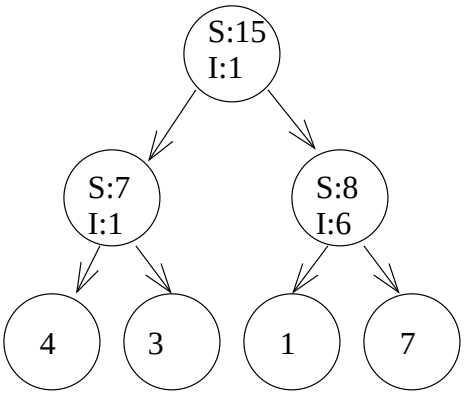
For instance, for the sequence $S = 4, 3, 1, 7$ there are five possible trees. The picture below shows the tree, with each node labelled with its sum S and its imbalance I. The most balanced tree is T2, with an overall imbalance of 2.



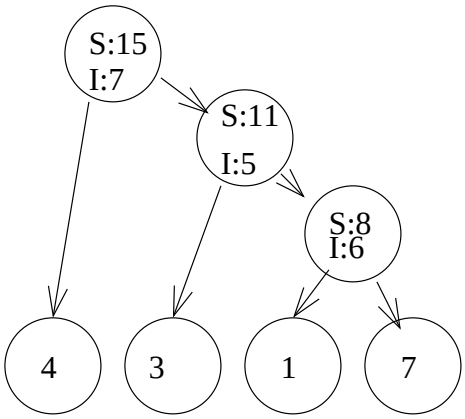
T1. Overall Imbalance:6



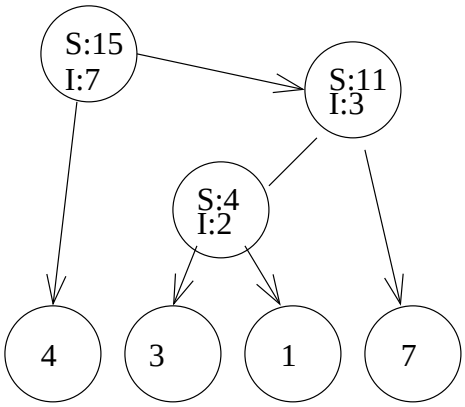
T2. Overall Imbalance:2



T3:. Overall Imbalance:6



T4. Overall Imbalance:7



T5. Overall Imbalance:7