14. (2 points for the first 3 subproblems plus 1 point if you do the last subproblem) We consider several variations on the standard Subset Sum and Knapsack problems. For each problem give a recursive algorithm to solve the problem, then convert it into an iterative, array-based, bottom-up algorithm with the requisite running time. For the recursive algorithms, make sure to specify the initial call(s) from the main program that would initiate the computation.

   (a) The problem is defined as follows:
   Input: positive integers $v_1, \ldots, v_n$, with $L = \sum_{i=1}^{n} v_i$.
   Output: Determine where a solution to the equation $\sum_{i=1}^{n} (-1)^{x_i} v_i = 0$ exists, where each variable x_i can be either 0 or 1.
   The running time should be polynomial in $n + L$ where $L = \sum_{i=1}^{n} v_i$.

   (b) The problem is defined as follows:
   Input: positive integers $v_1, \ldots, v_n$ and L.
   Output: Determine where a solution to the equation $(\sum_{i=1}^{n} x_i v_i) \bmod n = L \bmod n$ exists, where each variable x_i can be either 0 or 1. Here $x \bmod y$ means the remainder when x is divided by y.
   The running time should be polynomial in $n + \log L$ (alternatively you can assume L fits in one word of memory, and bound the running time by a polynomial in n).

   (c) The problem is defined as follows:
   Input: positive integers $w_1, \ldots, w_n$, $v_1, \ldots, v_n$ and L.
   Output: The maximum possible value of $\sum_{i=1}^{n} x_i v_i$ subject to $\sum_{i=1}^{n} x_i w_i \leq L$ and each x_i is a nonnegative integer.
   The running time should be polynomial in $n + L$.

   (d) The problem is defined as follows: a
   Input: positive integers $v_1, \ldots, v_n$
   Output: Determine whether subset S of these integers exists such that $\sum_{v_i \in S} v_i^3 = \prod_{v_i \in S} v_i$.
   The running time should be polynomial in $n + L$, where $L = \max(\sum_{i=1}^{n} v_i^3, \prod_{i=1}^{n} v_i)$.

15. (4 points) The input consists of a collection n intervals over the real line. The output should be the maximum aggregate lengths of the intervals in any subcollection X of non-overlapping intervals. So you want to consider the collection of non-overlapping intervals where the aggregate lengths of the intervals is maximum.

   (a) Give a recursive algorithm A that considers all possible intervals as the next interval I to be added to X and then recurses on the remaining intervals that might feasibly be added to X in the future. So in the recursive call tree the root would have n children.

   (b) Explain why one can not readily convert A into an iterative, array-based bottom up algorithm that runs in polynomial time.

   (c) Give a recursive algorithm B that considers all possible intervals as the next interval I to be added to X and then makes 2 recursive calls on the remaining intervals that might feasibly be added to X in the future. So in the recursive call tree the root would have $2n$ children.

   (d) Convert B into an iterative array-based bottom-up algorithm that runs in time $O(n^3)$.

   (e) Give recursive algorithm C for this problem that converted into an $O(n^2)$ time iterative array-based bottom-up algorithm.
   Hint: Pick a particular interval to initially focus on.

   (f) Give the $O(n^2)$ time iterative array-based bottom-up algorithm that results from converting C.

   (g) Explain how one could recover the actual intervals in X by tracing back through the array constructed by the iterative algorithm.

16. (6 points) The input for this problem consists of n keys $K_1, \ldots, K_n$, with $K_1 < K_2 < \ldots, K_n$, and associated probabilities $p_1, \ldots, p_n$. The problem is to find the AVL tree for these keys that minimizes the expected depth of a key. An AVL tree is a binary search tree with the property that every node has balance factor -1, 0, or 1. The balance factor or a node is the height of its right subtree minus the height of its left subtree. Give a polynomial-time dynamic programming algorithm for this problem. It is sufficient for your algorithm to compute the minimum expected depth, your algorithm need not actually compute the binary tree on which this minimum is achieved.