**Problem description:**   
Given a set of boxes, find the greatest number of boxes that can be nested together.

**Inputs:**   
boxes {b1,b1,…,bn}, which consist of a height, width, and depth measurement.

**Output:**  
 A set of boxes that represents the greatest number of boxes that can be nested.

**Assumptions:**  
A box cannot fit within a box of equal height, width, and depth.

**Strategy:**Use a directional graph to show the relationships between boxes, then analyze the graph to find the longest chain of nestable boxes.

**Description:**  
Create an n by n array to represent a directional graph D. Fill in the graph using the following expression:  
D[i,j] = { 1 fitsIn(bi,bj)  
 { 0 bi = bj  
 { -1 fitsIn(bj,bi)

Alongside this, create an array L of length n to represent the greatest size set of nestables each box can be part of.  
Scan the array and find all columns that contain no -1s. For each of these columns, process each di,j such that i is column number and j is row number. For each di,j, set dj,i equal to di,j + 1. If L[j] < dj,i then set L[j] to di,j + 1 as well.

Repeat this process until the graph contains no -1s. Create solution array S. Find the index di,j with the largest value in the graph (if there are more than one, just pick the first one). Add bi to S, then find index dj,k that represents the maximum value in column j. Add bj to S, etc. until reaching column m such that L[m] = 1. Add bm to S, and then return S.

**Time Complexity:**  
Array: O(n2)  
  
Scan: O(n)  
Process: O(n)  
Repeat up to n times: O(n2)  
Overall time complexity: O(n2+n2) = O(n2)

**Problem description:**   
Given a set of doors, find the number of unique ways the doors can be locked such that the number of secured doors = S. Secured is defined as being locked and either being the leftmost door or having a locked door to the left.

**Inputs:**   
A number of doors n and a target integer S.

**Output:**  
The number of unique configurations of locked doors that give S secured doors.

**Assumptions:**  
None

**Strategy:**Use a binary tree to create a map of possible configurations. Use the branch and bound methodology to keep the time complexity below 2n.

**Description:**  
Each node in the tree has a K value and a Boolean ‘securable.’ The height of the tree is equal to n (assuming the tree’s root starts at height 0). The root node has K = 0 and ‘securable’ = true. From there, every left child inherits the K value of its parent, and has ‘securable’ set to false. Every right child’s K = its parent’s K + 1 and has ‘securable’ set to true. If a child has an K value either greater than S, or less than S-(n-height(child)), delete that child and do not calculate any further children on that branch. If K = S, keep the child, but do not calculate any further children on that branch. Return the number of leaves.

**Problem description:**   
Given a list of integers A = {a1,a2,…,an}, and integers K and S, return the number ways K integers from A can sum to S.

**Inputs:**   
A set of integers A, and integers K and S.

**Output:**  
The number of ways K integers from A can sum to S.

**Assumptions:**  
None

**Strategy:**Use a binary tree to create a map of possible configurations. Use the branch and bound methodology to keep the time complexity below 2n.

**Description:**  
Each node has properties k and s, where k represents the number of values from A that have been added, and s is the sum of those numbers. At the root, k = 0 and s = 0. For each level i in the tree, each left child inherits its parent’s k and s values. Each right child’s k = kparent+1, and s = sparent+ai. Do not add a child and stop calculating along that branch if schild > S or kchild = K and schild < S. Each time a child’s s value equals S, increment a counter of combinations. Once the tree is complete, return that counter.

**Problem description:**   
Given a list of ski runs and a number of minutes per day, divide the ski runs up over the minimum number of days that they can be skied in order. Ties are settled by time wasted dissatisfaction (twd), which is calculated as follows:

twd(t) = { 0 t = 0  
 { -C 1 <= t <= m  
 { (t-m)2 otherwise

**Input:**  
A set of ski runs R = {r1,r2,…,rn}, where each ri is a positive integer representing the number of minutes to complete the run and return to the top of the mountain. An integer L that represents the number of minutes in a day that can be skied with. C is a constant that is added to twd when t < m. m the number of minutes in the day that can be wasted before major dissatisfaction with the plan arises.

**Output:**  
A schedule that minimizes the number of days required to ski all the slopes in order, and that secondarily minimizes twd.

**Assumptions:**  
None

**Strategy:**Use a binary tree to create a map of possible configurations. Use the branch and bound methodology to keep the time complexity below 2n.

**Description:**  
Each node has properties t and twd, and d. At the root, twd = 0, t = L-r1, and d = 1. Each level of the tree represents the next run (assuming root is at height ‘1’), with the left child representing the decision to ski the next slope on the same day, and the right representing the decision to ski the next slope on the next day. Each left node’s (at level i) twd and d are inherited from its parent, and its t = L-tparent-ri. If t < 0, do not add the child, and do not calculate further nodes on that branch. For each right node, its twd = twdparent + twd(tparent), its t = L-ri, and its d = dparent+1. When the bounded tree is complete, find the nodes with the minimum d value at height n. Of those nodes, find the node Ns with the minimum twd value. Create a solution array S, of length d where d = dN. Traverse up the tree from node Ns, adding each run to the solution array in the location indicated by that node’s d value.