EECS 336 Fall 2015
Homework Problem 5.1

i) Opt(i) = the maximum number of nested boxes that I can bring to the boxing club meeting, including box i as the inner-most box, considering boxes (1,2,...,(i-1)). The maximum number of boxes that can be carried to the meeting is the max of Opt(i) for (i = 1...n)

ii) Opt(i) = max [1≤ j ≤(i-1)] of (Opt(j) + compatible(i,j))*compatible(i,j))
compatible(i,j) returns 1 if i can be nested within box j, and 0 otherwise, by checking $h_j > h_i$, $w_j > w_i$, and $l_j > l_i$

iii) Base Case: (i = 0) Opt(1) = 1 (the biggest box cannot be nested in any other boxes)

iv) Here is the algorithm:

Algorithm 1 Max_Number_Boxes ()

```
Sort Boxes by Height
Initialize memo[] of length n
memo[0] = 1
for all i ∈ [1...(n − 1)] do {loop over all of the boxes, in non-increasing sorted order}
  max = 1
  for all j ∈ [0...(i − 1)] do {loop over all prior boxes examined}
    if (compatible(i,j)) then {compatible iff box i can be nested within box j}
      val ← memo[j] + 1
      if (val >max) then {new max number of nestable boxes found including box i as inner}
        max ← val
    memo[i] ← max
return max(memo[0...(n-1)])
```

Correctness

The recurrence is correct because the maximum number of boxes that can be carried, including box i, can be found by checking all boxes with larger height and seeing whether box i fits within each one. Since we have already done this for each of the boxes with greater height, and kept track of how many boxes they can be nested into, if box i fits in this box, we can increment this count by one. A key feature here is that Opt(i) does not mean the max number of boxes that can be carried *up to i*, but rather *up to and including box i as the inner most box*. This allows us to scan over the boxes in $\mathcal{O}(n)$ to find the total max number of boxes that can be brought to the meeting. This may or may not include the box with the smallest height, which is why the algorithm must scan over the memo table at algorithm conclusion.

Runtime

The run-time is $\mathcal{O}(n^2)$ because the memo table is size n, and filling in each cell of the memo table takes n work. That is much faster than a brute force method of checking all possible combinations of boxes.

Tested in code (and code attached)

(expected,actual) = (1, 1) → boxes: [(3, 3, 2)]
(expected,actual) = (2, 2) → boxes: [(1, 2, 1), (3, 3, 2)]
(expected,actual) = (2, 2) → boxes: [(1, 2, 1), (2, 2, 2), (3, 3, 2)]
(expected,actual) = (2, 2) → boxes: [(1, 2, 1), (2, 2, 2), (2, 3, 2), (3, 3, 2)]
(expected,actual) = (2, 2) → boxes: [(1, 2, 1), (2, 2, 2), (2, 3, 2), (3, 3, 2), (3, 3, 2)]
(expected,actual) = (2, 2) → boxes: [(1, 1, 1), (1, 2, 1), (2, 2, 2), (2, 3, 2), (3, 3, 2), (3, 3, 2)]
(expected,actual) = (4, 4) → boxes: [(1, 2, 3), (2, 3, 4), (3, 4, 5), (4, 5, 6)]
(expected,actual) = (4, 4) → boxes: [(1, 2, 3), (2, 3, 4), (3, 4, 5), (4, 5, 6)]
(expected,actual) = (2, 2) → boxes: [(1, 1, 1), (3, 3, 2)]
(expected,actual) = (2, 2) → boxes: [(2, 1, 1), (3, 3, 2)]
(expected,actual) = (1, 1) → boxes: [(2, 1, 2), (3, 3, 2)]
(expected,actual) = (2, 2) → boxes: [(2, 2, 1), (3, 3, 2)]
(expected,actual) = (2, 2) → boxes: [(1, 2, 1), (3, 3, 2)]