**Name: Qi Zhao GWID: G41261092 Gmail: zhaoqi@gwu.edu**

**Box Stacking**  
You are given a set of n types of rectangular 3-D boxes, where the i^th box has height h(i), width w(i) and depth d(i) (all real numbers). You want to create a stack of boxes which is as tall as possible, but you can only stack a box on top of another box if the dimensions of the 2-D base of the lower box are each strictly larger than those of the 2-D base of the higher box. Of course, you can rotate a box so that any side functions as its base. It is also allowable to use multiple instances of the same type of box.

**Dynamic programming**

**Notation:** tallestStack[i]. Represent the length of the tallest stack.

**Optimality:** if tallestStack[i] is an optimal solution, then tallestStack[i-1] is also an optimal solution. Because if the tallestStack[i-1] is not an optimal solution, we assume there exist another better solution X for i-1 types, then we should get another better solution Y based on the X for i types. Which will contradict the beginning condition.

**Recursion:**

Assume A = {base : h(i)\*w(i) , h(i)\*d(i),w(i)\*h}

tallestStack[i] = tallestStack[i-1] + the height of the boxes in A which are not belong to the tallestStack[i-1] + the difference between the maximum height of the boxes in A and that of tallestStack[i-1] boxes.

Time Complexity: O(n)

**Algorithm:**

Boxes[][3] represent the boxes’ height, long and width. Build a new array Bases[][3] to store the bases of the box, Bases[i][j] represent the i-type boxes whose height is boxes[i][j].

set Bases[i][j] as key, the height boxes[i][j] as a value, put this pair into a HashMap

for i from 1 to n:

if(!Hashmap.contains(Bases[i][j])

put the pair to the hashMap, add the value to the height of the stack.

else{

if Hashmap.get(Bases[i][j]) < boxes[i][j]

Hashmap.put(Bases[i][j],boxes[i][j])

Minus the former value of key bases[i][j] from the height of the stack, then add the new value to the height of the stack.

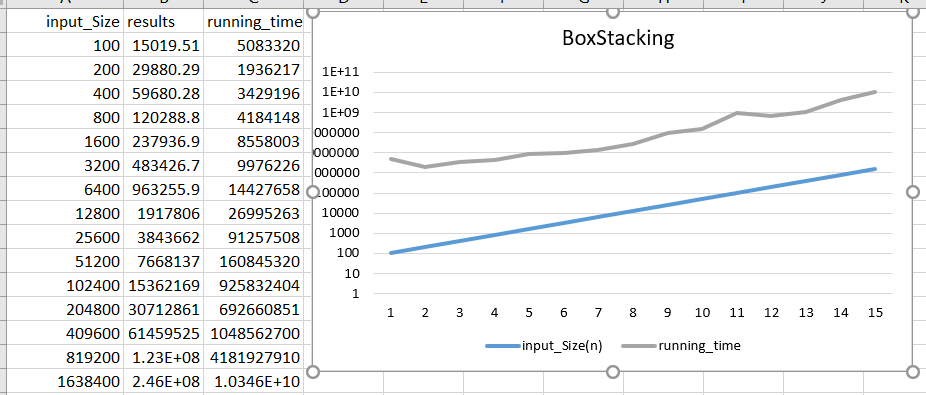
**Key implementation:**

Rebuild am array storing the box bases.

Use HashMap to store the box base- height pairs.

Next steps were illustrated in **algorithm part**.

**Numerical results**



Conclusion:

In the above paragraph, the lower represent the input size and the higher line represent the running time. We could conclude that these two variable are linearly related. this diagram could prove that the time complexity of this algorithm is O(n).