PSET 11 – Stoikos Stefanos

**Algorithm:** The algorithm that I have used uses dynamic programming. Initially, we find all the possible rotations for all the boxes. After storing all of them we sort them based on the base area. The area which sits on top of the box beneath it. We sort in in decreasing order.Now the problem becomes similar to the longest increasing subsequence. Using 2 arrays we will traverse the sorted rotations list. The maxHeight which will store the max possible height at each block. The blockSequence array which will store the indices pointing to where each block is placed in order to achieve the max height. The max height array works with 2 pointers i,j \* The optimal substructure optimal property that are going to use is the following:

-maxHeight(i) = Max(maxHeight(j) + height(i)), only when width(j)>width(i) & length(j)>length(i). (which is the given property of the problem).

-blockSequence(i), which will store the indices to the max height block proceeding it. (happens only when we find the max possible height stack for the block i)

Then we find the max in maxHeight to find the max height. To find the blocks necessary to find the max height, we find the corresponding index of the max at the maxHeight array, we go to the corresponding blockSequence array, and start following the indices given from there. We stop if the index becomes 0 or if the index is the same 2 times in a row. We store that in an array, and we write it to the file.

**Interesting design:** Instead of using a classic binary search to find an index, due to some issues with big arraylists, I decided to use an abstract layer introduced in Java 8, called Stream API. It allows me to do multiple lambda operations at once. The running time is more than the binary search, but given this problem that I encountered, it was an interesting solution.

For the total possible rotations, I used an arrayList instead of an array, because I wanted to be more flexible with the java objects, and it is more intuitive.

**Objects:** I used java, an object oriented language, so It was natural to create an object named block with 3 dimensions (Length,Width,Height). I included a to String method in order to see the block dimensions, for debugging. I also included a compareTo method in order to sort the blocks using the base area.

**Classes:**

**MaxHeight:** Using dynamic programming it find the max height

**GetBlocks:** Finds the sequence of blocks required for the max height

**WriteToFile:** Writes to file using filewriter

**FindIndex:** Finds the index of an element in a given array

**Running-Time: Θ(n^2)\*,** max Height is traversed using a nested forloop.

**Space-Requirement: Θ(n),** the arrays/arraylists we are going to be creating are: maxHeight,rotations,blockSequence. All of them will take up space Θ(n) given n elements.

**Testing:** I tested my code with all the given test cases the problem set gave us and I also came up with some edge cases.