• Describe the algorithm that you implemented, argue that it is correct, and argue its expected running time.

We implemented the algorithm based on the recursive definition of the following function:

A(i) = max height using blocks 1...i which are all of the permutations of the blocks sorted by decreasing length

0	i = 0
$\max_{0<j<i} (B_i + A(i - j), A(i - 1))$	otherwise

Constraint: Block A(i - j) fits over Block i and the number of blocks available have not been used

This algorithm is correct in terms of finding the maximum tower height because it will consider every possible valid tower solution and then return the tower with the maximum height over all solutions. It also is valid because of the restraint on the max function. All towers constructed will have some blocks, i, stacked on top of some other blocks j. The block i will always have smaller length and width than block j and that holds true for all blocks i and j in the tower. Our algorithm will also never exceed the given block amount because it restricts the tower from using more than the given amount of blocks.

The runtime of our algorithm is O(2n) = O(n) where n is the total number of permutations of the different blocks, since we do not consider all permutations for each block.

• **Describe an interesting design decision that you made (i.e. an alternative that you considered for your algorithm and why you decided against it).**

We were considering treating each type of block as an individual type of block and rotating it (permuting the dimensions) when we validated that it can be used. However, doing this just adds more layers of complexity and does not allow us to do the sorting which optimizes the runtime.

• **An overview of how the code you submit implements the algorithm you describe (e.g. highlights of central data structures/classes/methods/functions/procedures/etc . . .)**

To compute the maximum height, the A(i) function finds all the possible towers that we can build using the blocks in all rotations. Our code finds all of the possible towers that can be built and then finds the maximum.

What else do we need? We need to argue why our algorithm is correct

• **How you tested your code and the results of sample tests**

We tested our code by looking at multiple cases.

One case was where there were a different number of blocks to be used in our tower. We considered the sample input first limiting the amount of blocks to be three. For this case our code worked correctly and used the max amount of blocks available.

This meant that if we were to change the amount of blocks available, we would get different results. So we changed the amount of blocks available for use to be 2 while keeping the same three block dimension available. Again, our results came back correct and we results that differed from when we were able to use three blocks.

We also considered the case of blocks that had similar dimension of length and width. We wanted to if choosing blocks with dimensions that were similar would yield a solution that was optimal, but not valid because a length or width of one block was bigger than one underneath it. To confirm our code worked, we tested multiple blocks with multiple dimensions and manually went through to find the optimal solution by hand and compared it with our algorithms results.

We also continued to test our code with the sample files provided, and the results are reasonable:

Blocks10.1.in = The tallest tower has 5 blocks and a height of 3132

Blocks10.1.out = The tallest tower has 7 blocks and a height of 3344

Blocks100.1.in = The tallest tower has 10 blocks and a height of 5947

Blocks100.1.out = The tallest tower has 22 blocks and a height of 13878