Andrew Letz
CIS 313
Professor Proskurowski
Written Homework #3

Question 1

SS(i, j), i <= j denotes the set of all stack-sortable permutations of numbers i through j.

Prove SS(1, n) consists of all string obtained by concatenating
SS(i, n-1) and SS(1, i-1) separated by n for all i = 1...n
example: SS(6) <3, 4, 5, **| 6, |** 1, 2>
i -> n-1 n 1 -> i-1

I will start by showing that the concatenation of any two stack-sortable strings will result in a stack sortable string. If you examine the process of sorting a given permutation, you will see that it must start with a push and end with a pop. In other words, the stack will always be empty at the end of the sort. This means that the permutation that is concatenated onto the other will have an empty stack to "start" with, allowing it to terminate without issues. In summary, a string sortable permutation next to another will also be stack sortable.

With inserting a number (in this case n) in between two stack sortables, we can use the previous fact that the first permutation must leave the stack empty to prove it will have no effect on the sortability. Sorting the second half of the concatenated string will be exactly the same with the difference of one push at the start (pushing n) and one pop at the end (popping n). As we know n is larger than all other elements, it being at the end will not change the final ascending order. Because i is arbitrary, and every concatenated string will be stack sortable, we can definitively say that the result will be equal to simply performing SS(1, n).

This situation can easily be seen in the form of balanced brackets, where adding two well formed strings will also be well formed, and where adding an opening bracket to the start and a closing bracket to the end will retain sortability.

Question 2

Question 3

Question 4

The complexity of insertion into a binomial queue with n elements is O(logn) because the merge is constant time O(1) but must be executed for each queue (which there are logn of).

Question 5

The BuildBinomialHeap procedure would require an insertion into the binomial heap n times. As shown in question 4, the complexity of this insertion is O(logn), so the final complexity would be O(n) * O(logn) = O(nlogn).