Data Structures and Algorithms

Homework # 2

Eren Erdogan

Question 1:

Please run code, refer to instructions on how to run on Sakai. Also please view testing results txt file in folder to see test input cases and results.

No discussion was asked for this question.

Question 2:

For this problem we had to implement two different ways to solve the HPAir problem. One was done with a recursive solution and one with the use of Stacks.

The only difference between the two when running is the order of it's flight path checking. The recursive solution checks the adjacent airport alphabet character with the lowest value before going to the next lowest.

The Stack checks the highest value adjacent airport path first before proceeding to the next. This can be seen in the two different tests running in the testing results txt file. This is due to a stack being LIFO.

For running of the code, please refer to instructions on how to run on Sakai. Also view testing results txt file in folder to see test cases and results.

Question 3:

For this question we had to implement two different versions of the MergeSort algorithm, a Top-Down approach and a Bottom-Up approach, and keep track of the number of comparisons for each. Both algorithms had the same number of comparisons, which can be seen below.

Comparisons for Top-Down and Bottom Up MergeSort		
Number of Inputs (N)	**Top-Down**	**Bottom-Up**
1024	10240	10240
2048	22528	22528
4096	49152	49152
8192	106496	106496
16384	229376	229376
32768	491520	491520

In theory, both methods of MergeSort on a random set of data should take

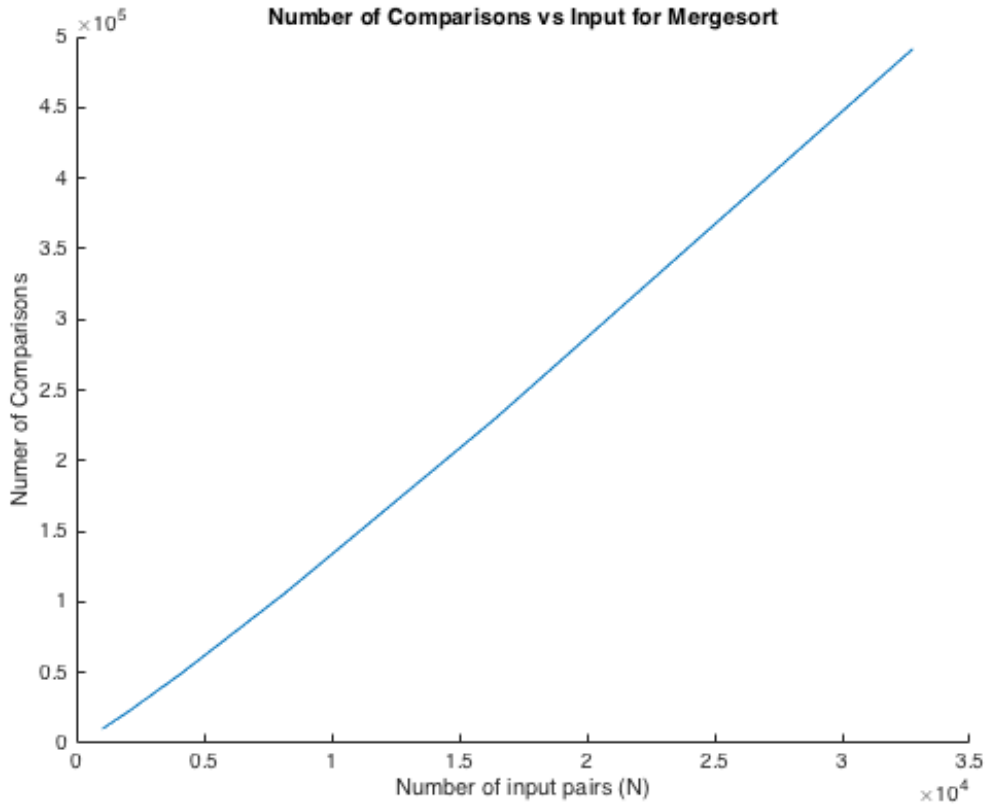
$$C(N) = N \cdot log(N)$$

We can take the input sizes (N) plug them into C(N) and see if they are indeed the same as our recorded data.

Comparisons vs NlogN			
Number of Inputs (N)	**Top-Down**	**Bottom-Up**	**NlogN**
1024	10240	10240	10240
2048	22528	22528	22528
4096	49152	49152	49152
8192	106496	106496	106496
16384	229376	229376	229376
32768	491520	491520	491520

We see that all these values are the same and confirm our theory that MergeSort has NlogN comparisons for random input data.

Lastly, we can plot out data points and see how the graph looks. The graph can be seen below.



We see it does indeed resemble a plot for NlogN, which is not perfectly linear but close.

Question 4:

For this problem we were asked to find out and describe when MergeSort would show the best-case performance. This best-case performance is going to come from the merge operation.

The minimum number of comparisons for the merge step is N/2. This case occurs when two already sorted lists are being passed into the merge method and the last element of the first list is bigger than the first element of the second one.

So specifically, if two lists that meet the criteria above are being merged then the first member of the larger list is compared N/2 times with the smaller list until the smaller list is exhausted. Once this is completed then the larger list can be copied over without further comparisons.

We'll provide a trace of an example of one of these merge operations and detail when we make a comparison.

Input lists: [0, 1, 2, 3] [4, 5, 6, 7]

So let's begin.

List 1: [0, 1, 2, 3]; List 2: [4, 5, 6, 7]; Merged List: [];

Now we do our first comparison.

Comparison 1: 0 < 4?

Yes, so we put that in the Merged List. Now we get,

List 1: [1, 2, 3]; List 2: [4, 5, 6, 7]; Merged List: [0];

Then our next comparison is,

Comparison 2: 1 < 4?

Yes, so we add it to the merged list.

List 1: [2, 3]; List 2: [4, 5, 6, 7]; Merged List: [0, 1];

Now, our third comparison is,

Comparison 3: **2 < 4?**

Yes, so we add it to the merged list.

List 1: [3]; **List 2: [4, 5, 6, 7];** **Merged List: [0, 1, 2];**

One more comparison left, which is,

Comparison 4: **3 < 4?**

Yes, so we add it to the merged list.

List 1: []; **List 2: [4, 5, 6, 7];** **Merged List: [0, 1, 2, 3];**

Now that the first list is empty and exhausted, we can add all the elements in the second list into merged list and return that.

List 1: []; **List 2: [];** **Merged List: [0, 1, 2, 3, 4, 5, 6, 7];**

And we're done with our Merge operation.

We started with an initial size of N = 8 and did 4 comparisons so N/2 total comparisons.

When actually running this there would be log N total instances of N/2 comparisons due to the cutting in half with recursive calls. Therefore, we can conclude that the total best case run time is specifically $\frac{1}{2}NlogN$, which occurs when passing in an entirely already sorted list like we demonstrated above.

Question 5:

Please run code, refer to instructions on how to run on Sakai. Also please view testing results txt file in folder to see test input cases and results.

We can see that for the testing of this, the iterative and recursive results generate almost identical results, as they should since a recursive call is using stacks internally and the iterative method uses stacks.