**CS303 Lab 4 – Heap Sort**

**Problem Specification:**

The goal of this assignment was to create and test the heap sort algorithm. We read in 7 different files with random numbers ranging 1 – n, n being the number of random numbers in the files. The 7 files had the following number random numbers: 100, 1000, 5000, 10000, 50000, 100000, 500000. I expected for this to take longer than my merge and insertion sort due to my computer clearly having problems running this code.

**Program Design:**

The heap sort program consists of very few functions. It consists of a buildMaxHeap function, a maxHeapify function, a Left and Right function, and a sort function. BuildMaxHeap is a function that creates a max heap from the provided array by repeatedly calling the maxHeapify function. The maxHeapify function uses the Left and Right function to find the Left and Right leaves in order to turn the array into a max heap. In the end, the sorting array goes through and places the new largest number in the array and calls maxHeapify in order to create a new max heap.

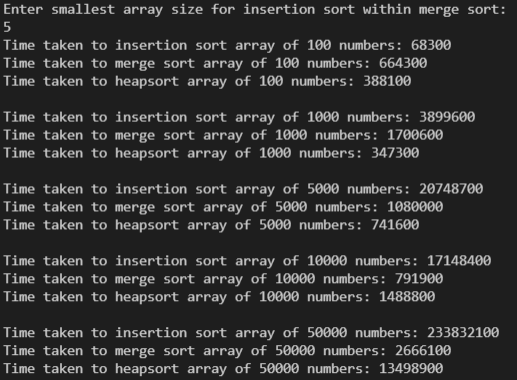
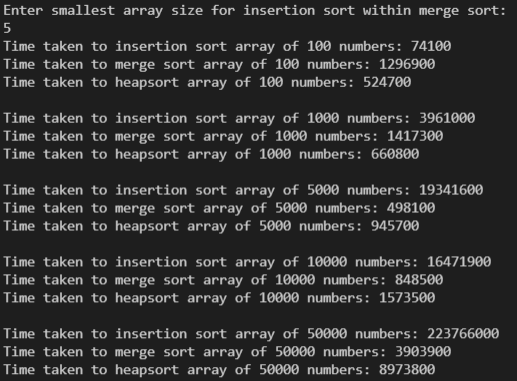
**Testing Plan:**

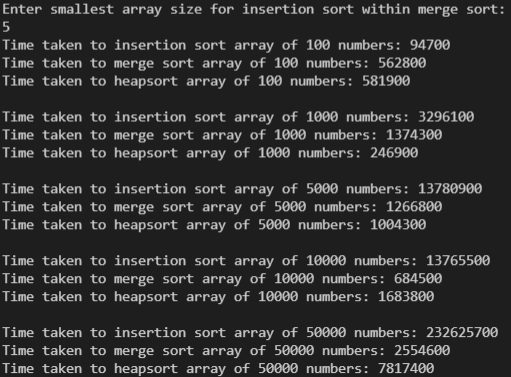
Just like both insertion sort and merge sort, we read in each of the files with random numbers, put it into an array, and then called the functions on each of the sorting algorithms. This time around we only tested size 100, 1000, 5000, 10000, and 50000 because after that the time would increase a lot more for insertion sort making the testing process much longer. The random numbers are the same as all of the other labs completed until now.

**Test Cases:**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Array Size | Heap Sort Time(ns) |  | Merge Sort Time(ns) |  | Insertion Sort Time(ns) |
| 100 | 830300 |  | 1029100 |  | 105000 |
| 1000 | 475400 |  | 2167000 |  | 6199000 |
| 5000 | 1285800 |  | 635000 |  | 28261500 |
| 10000 | 1786300 |  | 616000 |  | 19777800 |
| 50000 | 14825000 |  | 4595500 |  | 335266400 |

This is the test case that is mainly being used for analysis, the rest below are just showing example outputs.





**Analysis/Conclusion:**

Testing all the sorting algorithms created more logical results in terms of time. The insertion sort took the longest as the size became greater, and the merge and heap sort competed to work faster, and they both produced expected results.

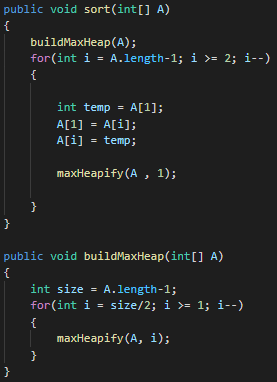
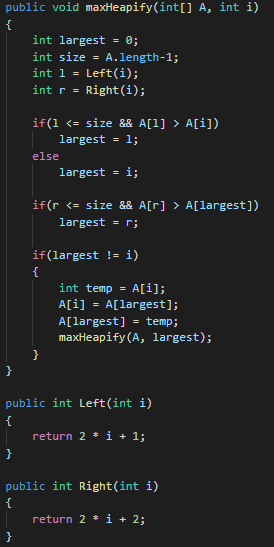
Heap sort produced faster times for smaller arrays while merge sort produced faster results for larger arrays. Due to the smaller size of the array, it makes sense that heap sort is faster as it sorts while dividing. The merge sort was able to sort more efficiently as the size of the array got larger.

**References**:

I did not use any outside resources; all the code came from my own creation. The lab report is based off my previous reports causing similarities between the two. I am citing my own previous work in order to avoid self-plagiarism.

All input files were provided to us with the lab instructions.

Below are the pictures of my code.



This is a snippet of the scanning the file input\_100, casting the string into integers for the input array of each sort function, and the timing and call of each sort function. This is just repeated for each file of random numbers. The only part exempt is the part asking for the minimum size the array has to be for the insertion sort in merge sort which is only asked once at the beginning.