CS303 Lab 3 – Insertion Sort PDF

Problem Specification:

The goal is to take in multiple input files ranging from 100 to 500000 random numbers and use merge sort to sort each file and record the time that it takes to sort each array. All numbers are assumed to be integers greater than 0. This assignment differs from the previous one because we are implementing a new sorting algorithm that has a time complexity of O(nlog(n)).

Program Design:

This program requires to main files, one being Merge.java and the other being MergeDriver.java. Merge.java has the algorithm that sorts and merges an array that continuously splits itself till it only has one element left. Then using the algorithm, it compares each element together and inserts the number where it should be numerically in the array. MergeDriver.java just takes in 7 different text files reads them into an array. Eventually parsing them into integers and placing them into a new array. Using a temporary array with the same contents of the integer array, we can compare every element. Finally, using System.nanoTime(), we can calculate the time it takes to merge sort each file.

Testing Plan:

The plan for testing is to take each array that has all the numbers from each input file and to sort them. We start by defining file variables that hold all the data of numbers. Then use a scanner to read each number in and placing it into an array of strings. Turning each input into the string array into an integer. Afterwards, you use this new array of integers and call the insertion sort on it. Using System.nanotime() before and after the call we can figure out the time that it takes.

Test Cases:

A picture containing text, scoreboard, plaque

Description automatically generated

Text

Description automatically generated

Text

Description automatically generated

Text

Description automatically generated

Text

Description automatically generated

|  |  |  |  |
| --- | --- | --- | --- |
| Array Size | Merge Sort Time(ns) |  | Insertion Sort Time(ns) |
| 100 | 96700 |  | 15700 |
| 1000 | 1449600 |  | 75800 |
| 5000 | 872600 |  | 321401 |
| 10000 | 1597800 |  | 631501 |
| 50000 | 5869600 |  | 2915000 |
| 100000 | 12756400 |  | 1516200 |
| 500000 | 69877500 |  | 5961101 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Array Size | Min = 5 | Min = 10 | Min = 20 | Min = 30 |
| 100 | 140300 | 117400 | 144500 | 117200 |
| 1000 | 1690200 | 1205700 | 1057900 | 1173300 |
| 5000 | 1044100 | 1490400 | 1492100 | 1711300 |
| 10000 | 2535700 | 3082100 | 1237400 | 1505900 |
| 50000 | 6188700 | 6422800 | 6333300 | 6176100 |
| 100000 | 12340400 | 12719100 | 12866300 | 13678900 |
| 500000 | 61920400 | 64267400 | 65587200 | 77606900 |

Analysis/Conclusion:

Based on the data that I have; I would honestly say that something is wrong with the code because the merge sort seems to be talking a lot longer than the insertion sort no matter the array size. The fact of the matter is that the time complexity of merge sort is O(nlog(n)) whether it is the best, average, or worst case. The time complexity of insertion sort is O(n^2) for both average and worst case, while the best case is O(n). Given this information, I would have to assume that these inputs that we have been given have to be better suited for insertion sort. The input arrays must have more sorted arrays that sort faster given insertion instead of merge sort.

Adding the insertion sort, has made everything much longer in the time that it takes to sort each array. I have no idea if my implementation is wrong or if this is how it is supposed to be but either way the merge sort with the insertion sort is a lot longer than it is merge sort. There is also some weird time variations between each test, I assume that it is based on system resource problems.

References:

I did not use any outside resources; all the code came from my own creation.

Below are the pictures of my code.