Comparing MergeSort Run Times for Different Array Sizes

Homework #3

By

John Kiyak

CS 303

September 18, 2018

**Problem Specification**

The goal of this assignment was to implement a method that will sort arrays of differing sizes from text files. When the arrays were sorted, the amount of time taken to sort the arrays were also calculated as well, and were used to evaluate the performance of merge sort with the increasing array sizes. The results from mergesort and insertion sort were compared for productivity/time taken.

**Program Design**

This program required a driver method [main()], one for the mergesort method, and another for the merge method. The driver method was used to read the file, store the numbers from the text file into the array, call upon the mergesort method, The merge sort method would calculate the middle array value for the array, and then it would run mergesort two more times, and then calling upon the merge function to merge them all together. A temporary array was placed in order to hold all of the values before being paced into the mergesort function. The merge sort was called once to combine the low and middle (left), and another time to combine the middle and high (right side), and merge was called to combine the two together and return the merged array.

**Testing Plan**

The testing plan was pretty straight forward. I would put all of the test cases from canvas (input\_100.txt through input\_500000.txt’) into the project file. Then, I would put the name of the file that I would wish to test into the main driver, under the with open(x) as inputfile, with x being the input file that I would wish to test. I would run the file until it was done, and the print statement with the associated run time was printed off on the screen. I ran each of the input files and recorded how long they each took to sort. If I wanted to print the array off, I could add a print statement around the mergesort call, but it would make the test time longer as the arrays got bigger.

**Test Cases**

The test cases and a graph of the result are shown in the table below:

InsertionSort

|  |  |  |  |
| --- | --- | --- | --- |
| Test Case | “File.txt” input | Array Size | Time Elapsed (s) |
| 1 | ‘input\_100.txt’ | 100 | 0.00 |
| 2 | ‘input\_1000.txt’ | 1000 | 0.078 |
| 3 | ‘input\_5000.txt’ | 5000 | 2.094 |
| 4 | ‘input\_10000.txt’ | 10000 | 7.973 |
| 5 | ‘input\_50000.txt’ | 50000 | 208.100 |
| 6 | ‘input\_100000.txt’ | 100000 | 1029.719 |
| 7 | ‘input\_500000.txt’ | 500000 | 27816.000 |

**MergeSort**

|  |  |  |  |
| --- | --- | --- | --- |
| Test Case | “File.txt” input | Array Size | Time Elapsed (s) |
| 1 | “input\_100.txt” | 100 | 0.00 |
| 2 | “input\_1000.txt” | 1000 | 0.0 |
| 3 | “input\_5000.txt” | 5000 | .031 |
| 4 | “input\_10000.txt” | 10000 | .0624 |
| 5 | “input\_50000.txt” | 50000 | .4149 |
| 6 | “input\_100000.txt” | 100000 | .91714 |
| 7 | “input\_500000.txt” | 500000 | 5.14 |

**Analysis and Conclusions**

From looking at the tables, they have around the same elapsed time until the 5000 array size. Then, the insertion sort has a considerable more running time than the mergesort method, which is especially clear in the 500000 array size, coming in at a almost 5000 multiple run time than the mergesort method compared to the insertionsort method.

**References**

To help streamline my input method for arrays, I used the lab 2 solution on the CS303 Lab canvas page to make my filereader more efficient and concise. I also used geeksforgeeks to help in my introduction to merge and mergesort.