Comparing InsertionSort Run Times for Different Array Sizes

Homework #2

By

John Kiyak

CS 303

September 11, 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 insertion sort with the increasing array sizes.

**Program Design**

This program required a driver method [main()], and another method for the Insertion Sort algorithm [ (insertionsort() ]. The driver method was used to read the file, store the numbers from the text file into the array, call upon the Insertion Sort method, and calculated how long each instance of the Insertion Sort method takes with each differing file input.

The following steps were required to develop this program:

Write the driver method to read the file, store the numbers from the text file into the array, call upon the insertionsort method, and calculate the time elapsed that each array took to read.

Write the insertionsort method to sort the array that was given from the parameters, and return the array to be used in the driver method.

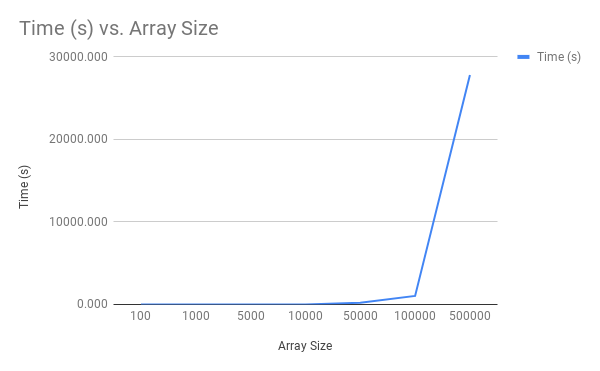
**Testing Plan**

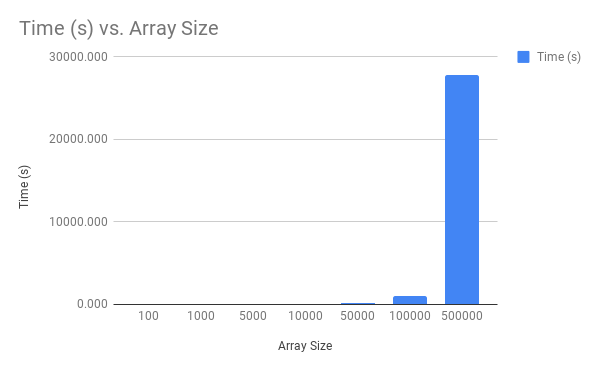
The testing plan was pretty straight forward. I would put all of the test cases from canvas (input\_100.txt through input\_500000.txt’) into the project file. Then, I would put the name of the file that I would wish to test into the main driver, under the with open(x) as inputfile, with x being the input 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 insertionsort 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:

|  |  |  |  |
| --- | --- | --- | --- |
| 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 |
| 8 | ‘input\_400000.txt’ | 400000 | 0 (file input error) |

****

****

**Analysis and Conclusions**

From looking at the graph and the numbers, we can see that as the size of the array text file increases, the time it takes to sort it increases as well. On the first 4 input text files, it takes hardly any time at all, the data doesn’t even show on the bar graphs. Once we get to 50000 and 100,000, they start to have some substance in runtime. In 10000, it takes 7 seconds, but on the 50000 trial, it goes up to 208 seconds, almost a 26 times increase from the one before. The 100000 one takes significantly more time, coming in at 1029 seconds while the 500,000 one takes the longest by far, coming in at 27816 seconds, almost 5 hours. An interesting thing to note, is that the biggest time multiplicativity seems to be between the 10000 and 50000 mark, with a 26 times difference between them, compared to almost 3 or 4 times for the previous test cases.

**References**

Since I am not too familiar with python, I needed some extra references to help with setting up my program. I used the pseudocode from the book to set up the insertionsort method, and I used some posts from stackoverflow and the python wikia to aid me in making my driver class (listed below):

<https://stackoverflow.com/questions/7370801/measure-time-elapsed-in-python>

https://en.wikibooks.org/wiki/Python\_Programming/Input\_and\_Output