Analyzation of Sorting Algorithms, Lab 5, Java II

Terry Speicher *Student, EPCC,* Jason Ivey, *Member, CSIT,*

Abstract—Sorting has become a sub-field of computer science. The ability to take a set of data and arrange it into a specific sequence of ascending or descending values has created a multitude of algorithms and analyses. Our challenge was to create a class from which we could run at least 6 different sorting algorithms coded in Java and compare the performance of each on a given set of data.

Index Terms—Computer Society, IEEEtran, journal, LaTeX, Force Push, nanosecond unreliability.

1 Introduction

OUR team created a class that contained 6 different sorting routines. We also created a separate class to handle the acquisition of the data to be sorted, the invoking of the Sort class to access the individual sort methods, and the recording of the time that it would take to perform each task. We also wanted to be able to provide different sizes of data sets to be able to compare sort times for varying values of n.

tjs
November 5, 2016

1.1 Tasking the Sorts

Our task was not to code each sorting routine from scratch, although we did so with the Bubble sort and our own Force Push sort. Our task was, instead, to assemble the sorts from other documented sources such as the textbook (in the case of the Quick Sort) or the Internet as noted in each method. Once the sorting routines were gathered and assembled, a methodology had to be developed to provide fair and accurate testing and recording. Each sorting algorithm was structured to receive a reference to an array of type Point. The class Point was defined in a separate point.java class file and consisted of two double values representing the (x,y) Cartesian graphing pair. The array would be sorted in ascending order based on the x coordinate. No return type was required since the reference passed as an actual parameter was a reference to the array from the invoking class. Thus, the original array was the one being sorted. System time was recorded in nanoseconds before and after each sorting routine was called. The difference of the two times was logged as the time that it took that sort algorithm to perform a sort on the array with n elements.

1.2 Methodology

Our methodology for testing the different sorting algorithms was to create a sort testing class that would have an array of integers representing the different sizes of arrays (different size n arrays) to be sorted. This gave us great flexibility in deciding the different sizes of the data sets. Given a file with 100,000 test elements, this array of integers was used to create sub arrays from the full list.

Each sort was invoked given an array of sizes varying from 10 elements to 100,000 elements. Each sort was timed and that time was recorded. Because run time of a program can be influenced by other tasks the computer may be

- *T. Speicher is a student in the Computer Science department of El Paso Community College and the University of Texas, El Paso. He is currently Owner of Timely Enterprises, Inc., a professional outsourced IT support provider.*
  E-mail: Terry@TimelyEP.com
- *Jason Ivey is a undergraduate in the field of Computer Science and member of the CSIT, El Paso division.*

Manuscript received November 5, 2016; revision is scheduled for summer 2021.

performing at the same time, we ran each test 10 times and took the average time for each test case.

We noticed that there were anomalies with the results when the test cases used

$$n < 10$$

, regardless of the sort routine called. In fact, the first test of each routine resulted in times that were higher than the second run of the same routine even when the same data was being used. To counter this anomaly, we disregarded the results of the first few tests i.e.

$$n = 2, n = 3, n = 4, ...n = 9$$

.

1.3 Hypothesis

We were working on a hypothesis that an

$$O(n^2)$$

sort might have better real time performance on lower n elements that an

$$O(nlogn)$$

sort. Thus, we structured our test cases to cover more tests of

$$n < 100$$

However, our tests did not show a significant benefit for using a non-recursive or

$$O(n^2)$$

sort for lower values of n. These were the averages of the test results for

$$n <= 100$$

based on the random data from the original file:

2 Conclusion

The conclusion goes here.

Appendix A
Proof of the First Zonklar Equation

Appendix one text goes here.

Appendix B

Appendix two text goes here.

Acknowledgments

The authors would like to thank...

References

[1] H. Kopka and P. W. Daly, *A Guide to LaTeX*, 3rd ed. Harlow, England: Addison-Wesley, 1999.

PLACE PHOTO HERE

Michael Shell Biography text here.

John Doe Biography text here.

Jane Doe Biography text here.