

Project Big Oh Exploration

Spring 2019

Jill Eliceiri

Table of Contents

<u>INTRODUCTION</u>	<u>2</u>
<u>METHODS</u>	<u>2</u>
<u>RESULTS</u>	<u>3</u>
<u>DISCUSSION</u>	<u>5</u>
<u>CONCLUSION.....</u>	<u>6</u>
<u>REFERENCES</u>	<u>7</u>

Introduction

The focus of this project is to establish, analyze, and compare the actual running times of four sorting algorithms with their theoretical running times. The four sorting algorithms include: heap sort, merge sort, insertion sort, and selection sort, with selection sort being selected as the one not covered in class. This application is written in Java using the BlueJ IDE.

Methods

For the program design, the main controlling class in this project is `BigOhExploration.java`. When objects of this class are instantiated, randomly generated arrays are initialized. The four concrete classes that implement the `Sort` interface have their own unique implementation of the `sortArray()` method. In the main class, before the `sortArray()` method is called for each type of `Sort`, a `Time` object is instantiated. The starting time is set directly before the `sortArray()` method is called, and the ending time is set right after the `sortArray()` method finishes.

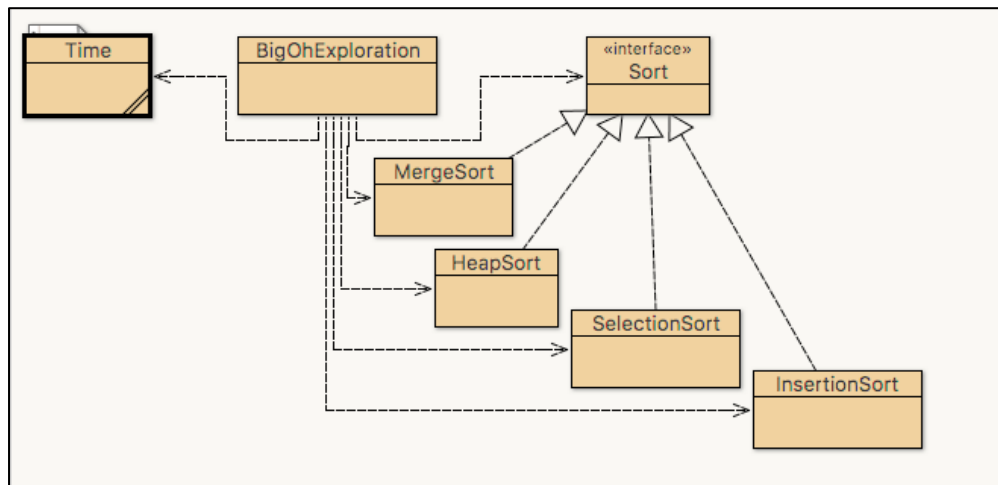


Figure 1: Big Oh Exploration Program Design

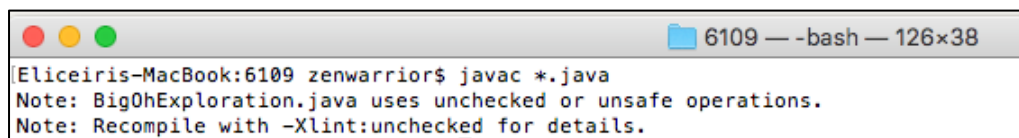
In order to establish trends from sufficient amounts of data, multiple runs were executed on each data set. In this program, 10 different trials were executed for each different sized array, on each type of `Sort`. The program also calculates the mean average of each set of runs.

Java code for the main class is inspired and portions are taken from Derek Banas Big O Notation Tutorial (Banas, 2013). For each of the sorting classes, java code is from the Geeks for Geeks

website and was contributed by Rajat Mishra (Mishra, n.d.). The Time class utilizes the `System.nanoTime()` method and casts it to a variable of type double, which is named seconds. Code for the Time class was also inspired by the Geeks for Geeks website, in addition to two searches on the Stack Overflow website for measuring elapsed time in Java (Bourrillion, 2009), and converting nanoseconds to seconds (Rosenfield, 2009).

Results

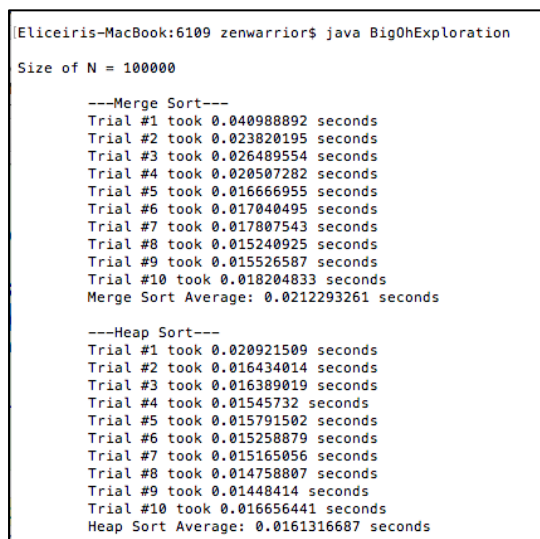
Figure 2 displays the command to compile the program, which compiles without error and does have two notes that are listed.



```
Eliceiris-MacBook:6109 zenwarrior$ javac *.java
Note: BigOhExploration.java uses unchecked or unsafe operations.
Note: Recompile with -Xlint:unchecked for details.
```

Figure 2 Compile the program

While running the program, the various array sizes, which are represented in the figure below by “Size of N =”, may be altered in the main class by changing the parameters of the `BigOhExploration` objects upon instantiation. In addition, various types of Sort classes may be isolated for analyzation by commenting out the appropriate Sort objects.



```
Eliceiris-MacBook:6109 zenwarrior$ java BigOhExploration
Size of N = 100000

---Merge Sort---
Trial #1 took 0.040988892 seconds
Trial #2 took 0.023820195 seconds
Trial #3 took 0.026409554 seconds
Trial #4 took 0.020507282 seconds
Trial #5 took 0.016666955 seconds
Trial #6 took 0.017040495 seconds
Trial #7 took 0.017807543 seconds
Trial #8 took 0.015240925 seconds
Trial #9 took 0.015526587 seconds
Trial #10 took 0.018204833 seconds
Merge Sort Average: 0.0212293261 seconds

---Heap Sort---
Trial #1 took 0.020921509 seconds
Trial #2 took 0.016434014 seconds
Trial #3 took 0.016389019 seconds
Trial #4 took 0.01545732 seconds
Trial #5 took 0.015791502 seconds
Trial #6 took 0.015258879 seconds
Trial #7 took 0.015165056 seconds
Trial #8 took 0.014758807 seconds
Trial #9 took 0.01440414 seconds
Trial #10 took 0.016656441 seconds
Heap Sort Average: 0.0161316687 seconds
```

Figure 3 Run the program

The table in Figure 4 lists the actual mean running time in seconds for each of the four sort types. The input size (N) is listed in the left column. The “-” symbol in a table cell represents running times that were not established because of the very long expected running times. For both

insertion sort and selection sort, a second set of mean running times were established on the input sizes in the table below from 100,000 to 1,000,000, of which the results are noted in italics.

	O(N log N) Merge sort	O(N log N) Heap sort	O(N²) Insertion sort	O(N²) Selection sort
N = 1,000	0.003	0.003	0.059	4.127
N = 10,000	0.002	0.002	0.352	4.111
N = 100,000	0.157	0.122	1.792 <i>2st set: 1.4221</i>	4.394 <i>2st set: 4.265</i>
N = 200,000	0.132	0.119	19.693 <i>2st set: 7.843</i>	135.611 <i>2st set: 21.360</i>
N = 500,000	0.142	0.139	51.691 <i>2st set: 48.809</i>	131.557 <i>2st set: 137.651</i>
N = 1,000,000	0.166	0.177	260.567 <i>2st set: 175</i>	512.740
N = 2,000,000	0.391	0.442	-	-
N = 4,000,000	0.775	1.089	-	-
N = 8,000,000	1.596	2.837	-	-
N = 16,000,000	3.566	6.012	-	-

Figure 4: Actual Running Mean Time Results in seconds

Discussion

Figure 5 illustrates the graphed results, which are zoomed out. Selection sort is displayed in red, insertion sort in orange, heap sort in blue, and merge sort in green. The last two are difficult to see at this view.

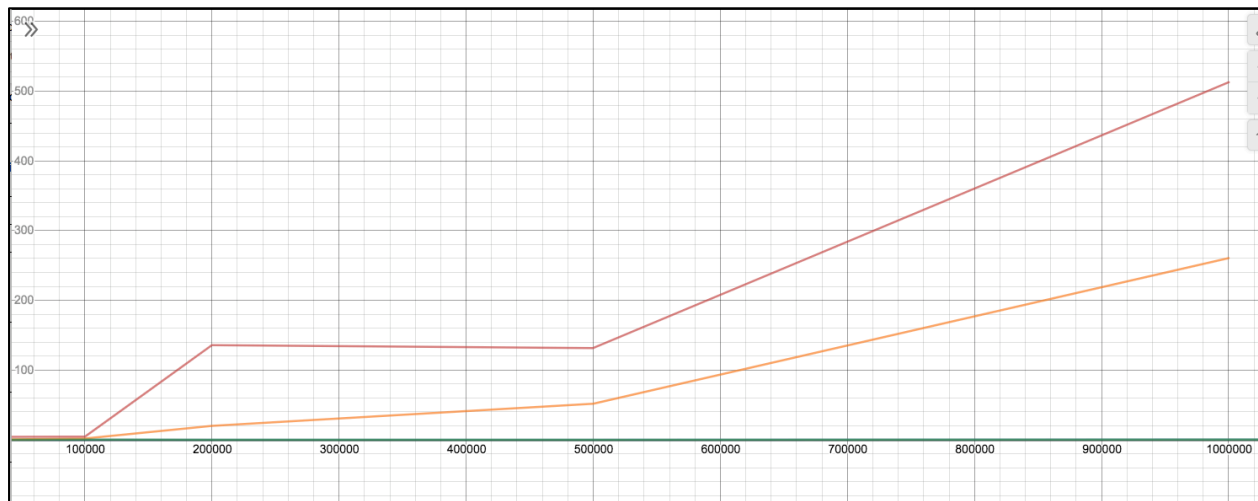


Figure 5: Graphed Running Time Results

Selection sort and insertion sort both share a running time of $O(N^2)$. As seen in the graph, once the value of the input N increases, the running time for selection sort and insertion sort grows rapidly. Because of the quadratic running time, when N is doubled, the running time is approximately four times larger, theoretically. In this particular analysis, this is observed only in the largest values of N . While insertion sort swaps and shifts values when needed, selection sort always performs a consistent number of comparisons. Therefore, because insertion sort is adaptive, it does have a faster best-case running time than selection sort. This is similar to what is seen in these particular results, where insertion sort is faster than selection sort with very large values of N , based on a randomly generated array.

It is seen from the results that merge sort and heap sort share similar growth rates. This is expected, as they both have a theoretical time of $O(N \log N)$. In this particular analysis, it appears that merge sort starts to outperform heap sort for large values of N , say greater than 1,000,000, as seen in the graph in Figure 6. This was an interesting result, as the worst-case space complexity for merge sort is $O(N)$, and only $O(1)$ for heap sort. According to a related

Stack Overflow question and answer (Heap Sort vs Merge Sort in Speed, n.d.), many factors can contribute to this actual performance such as memory and cache. Since heap sort is swapping the data and merge sort is moving the data, merge sort may be faster on machines where the extra space needed to hold the temporary array is not an issue. This is what is observed in this particular analysis. On computers without the extra space, heap sort may be more advantageous than merge sort.

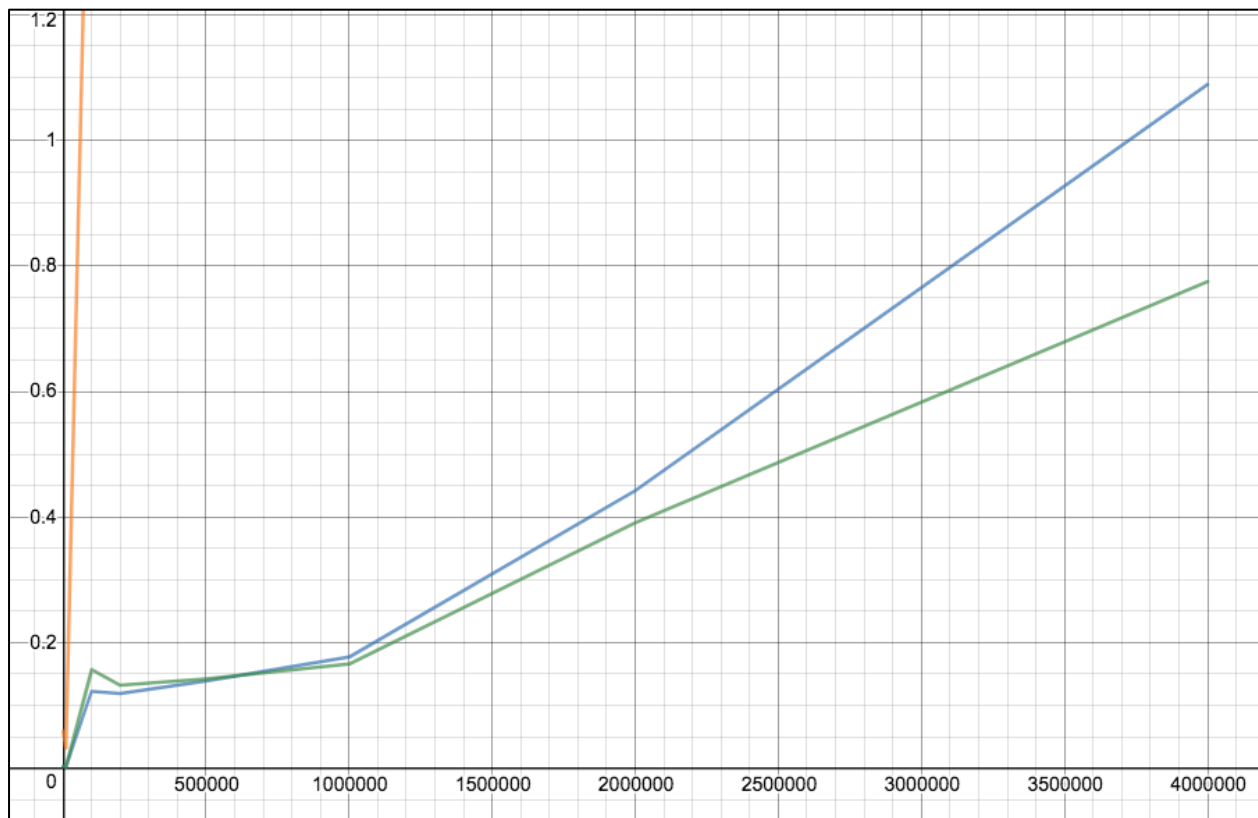


Figure 6: Merge sort (green) and Heap sort (blue)

Conclusion

This project established the actual running times of four sorting algorithms and compared them to the theoretical running times. While the results were similar to the theoretical running times, there were variations among results when running some sets a second time. These variations could be due to a number of factors including cache, memory, and processes running in the background. Overall, precision increased between the actual running time and theoretical

running time as the value of N increased. For an improved future project, the number of trials could be increased, and outliers could be removed from the results to increase accuracy.

References

- Banas, D. (2013, March 8). *New Think Tank*. Retrieved February 2019, from Big O Notations: <http://www.newthinktank.com/2013/03/big-o-notations/>
- Bourrillion, K. (2009, November). *How do I measure elapsed time in Java?* Retrieved February 2019, from Stack Overflow: <https://stackoverflow.com/questions/1770010/how-do-i-measure-time-elapsed-in-java>
- Heap Sort vs Merge Sort in Speed*. (n.d.). Retrieved February 2019, from Stack Overflow: <https://stackoverflow.com/questions/53269004/heap-sort-vs-merge-sort-in-speed>
- Mishra, R. (n.d.). *merge-sort, heap-sort, selection-sort, insertion-sort*. Retrieved February 2019, from Geeks for Geeks: <https://www.geeksforgeeks.org/>
- Rosenfield, A. (2009, May). *How to convert nanoseconds to seconds using the TimeUnit enum?* Retrieved February 2019, from Stack Overflow: <https://stackoverflow.com/questions/924208/how-to-convert-nanoseconds-to-seconds-using-the-timeunit-enum>