Abiola Gabriel Olofin and Omar Drira

CS 150

Professor Frank Xia

03/23/2020

Project 1 Report

Introduction

For this project, we are asked to compare theoretical analysis of different sorting algorithms to the actual performances of these different sorting algorithms. The sorting algorithms we studied are selection sort, bubble sort, insertions sort, merge sort, and three versions of quicksort based on the choice of the pivot element and those choices were the first element, the median, and a random element in the array. However, we had to create our own implementation of each sorting algorithm using abstract classes and interfaces. Before we started coding, we came up with a theoretical hypothesis that the sorting algorithm with the better time complexity based on big O notation would be the sorting algorithm with the best run-time and we decided that Merge sort would be the best algorithm because it has a run-time of $O(nlogn)$.

Approach

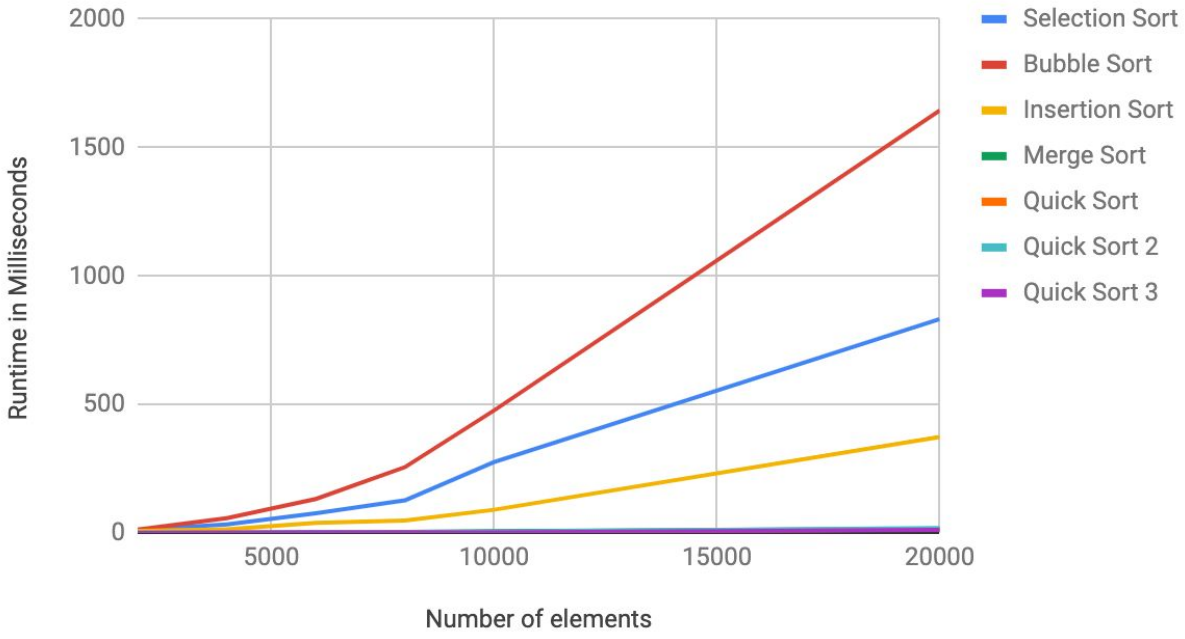
When designing the program, we followed the project directions that were given. This means that we created a Sorter interface that our selection, bubble, insertion, and merge sort would use to sort the array. After that, we implemented each of the sorting algorithms. We then created a quicksorter abstract class that each of the quicksorts

would use to implement their own quicksort based on the pivot. As a result, QuickSort1 uses a pivot of the first element, QuickSort2 uses a pivot of a random element, and QuickSort3 uses a pivot of the median of the first, middle, and last elements in the array. After all the implementations, the test cases we used which are shown in the experiment controller are 1) to test different lengths of arrays from 2000 to 20000, 2) to test the comparisons of runtimes of the quicksort implementations, 3) to test the runtime if a sorted array was used, 4) to test a sorted array in reversing order, 5) to test the runtime with repeated elements in the array, 6) to test Java's quicksort and merge sort and compare it to our implementation, and finally 7) to test the runtime if a percentage of the array was sorted.

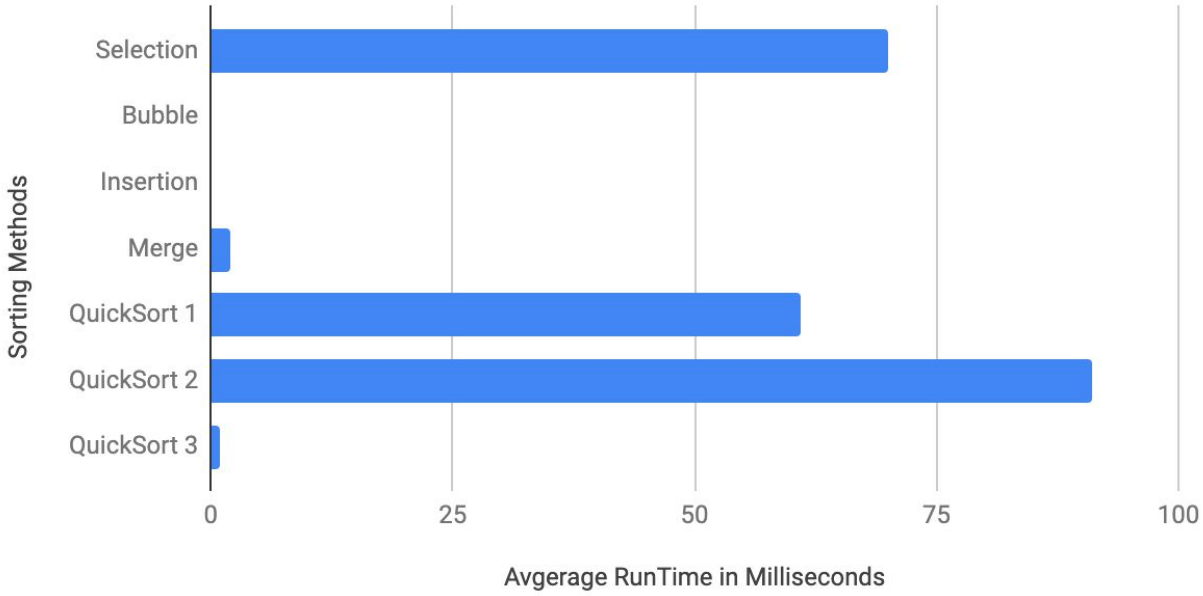
Data

We collected data to graph by running 10 trials of each tested method and averaged all the runtimes. Through the data, we found that QuickSort 3 was the fastest quicksort because it used the median and that merge sort was the fastest non-quicksort method. We also found that Bubble sorting's runtime only got worse as the number of elements increased. The following are the graphs of each test we ran:

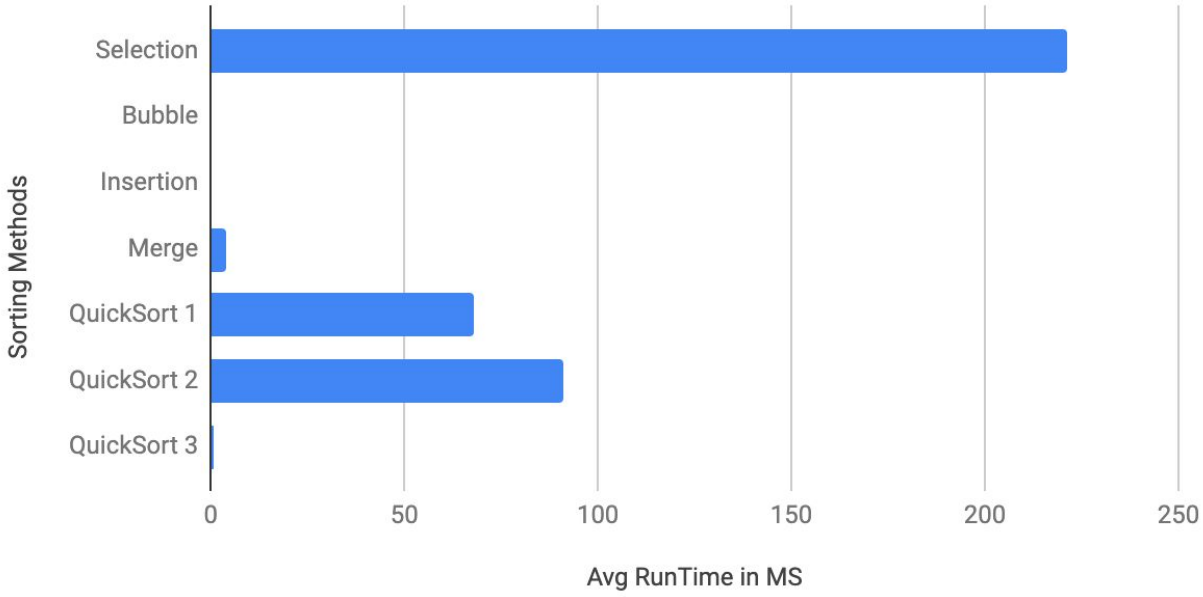
Runtime of Sorting Algorithims Based on Array Sizes



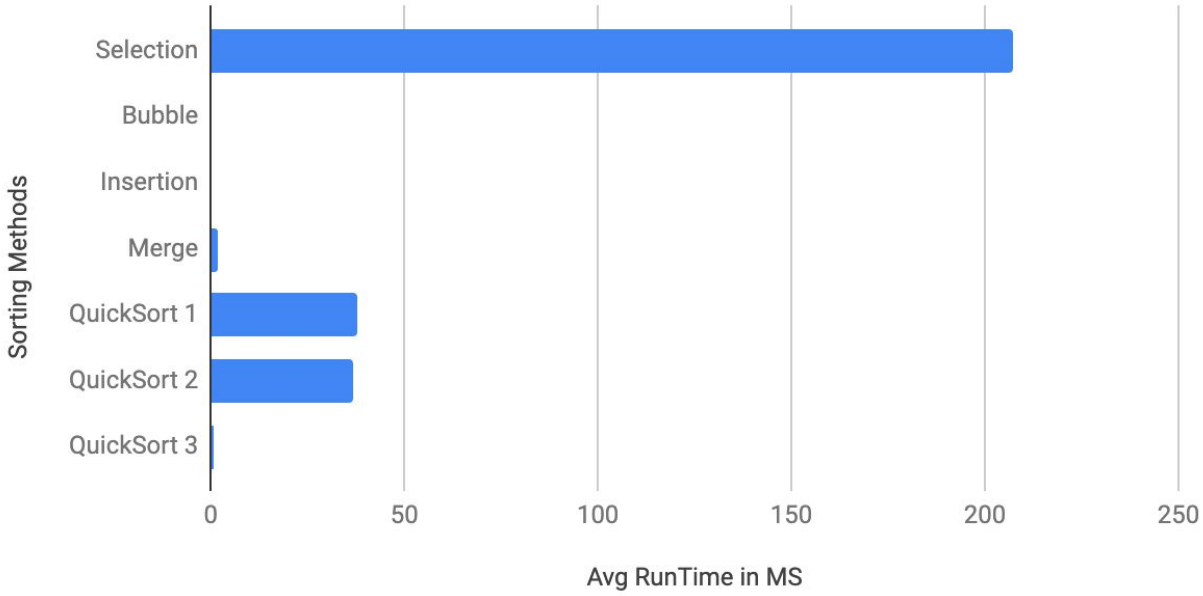
Runtime of Sorting Algorithims Based on Sorted Array With Size of 10000 elements

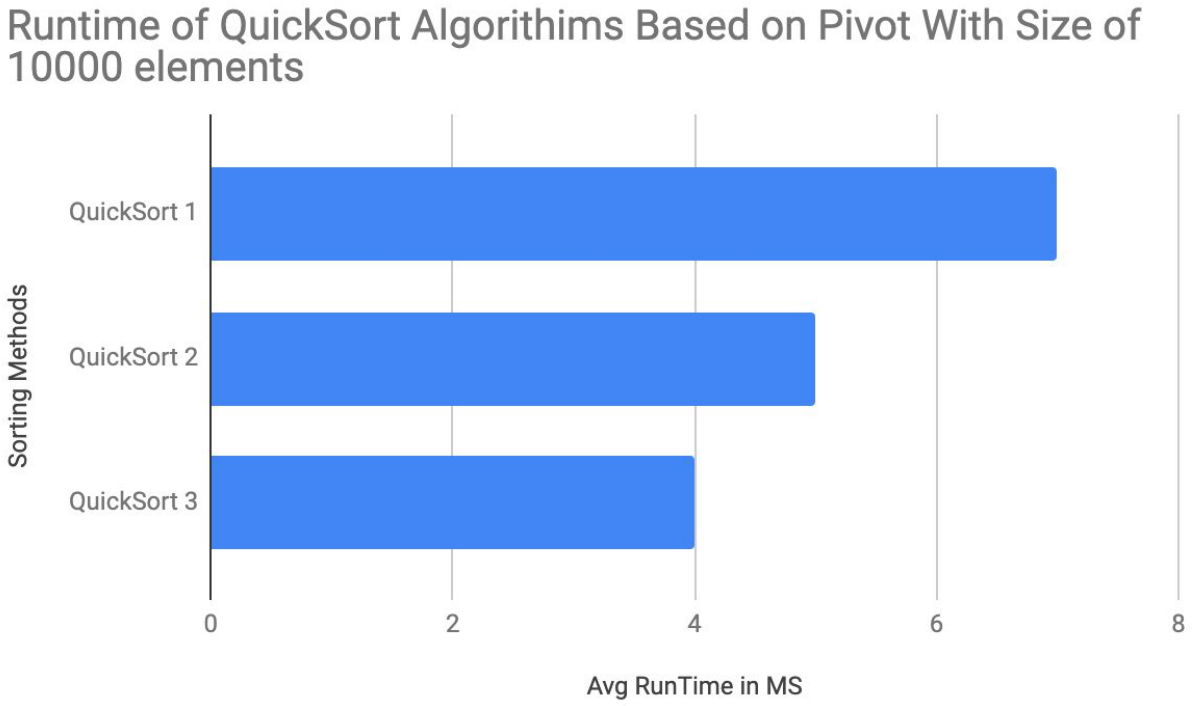


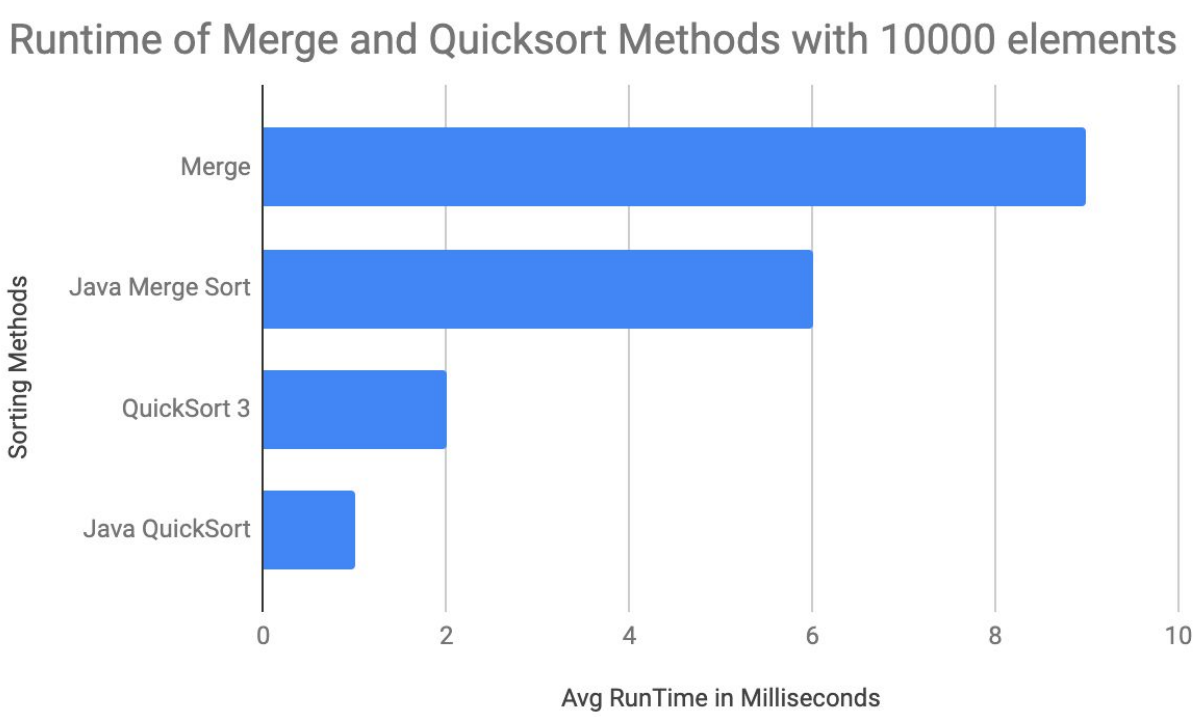
Runtime of Sorting Algorithims Based on Reversed Sort Array With Size of 10000 elements

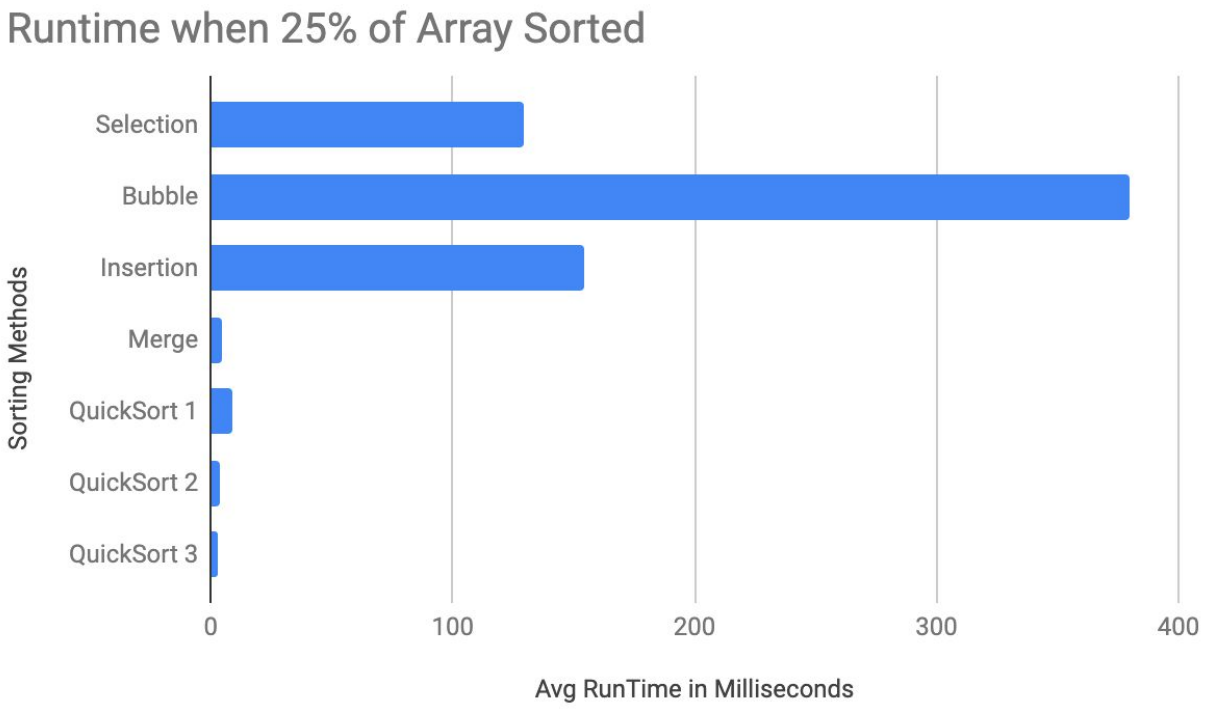


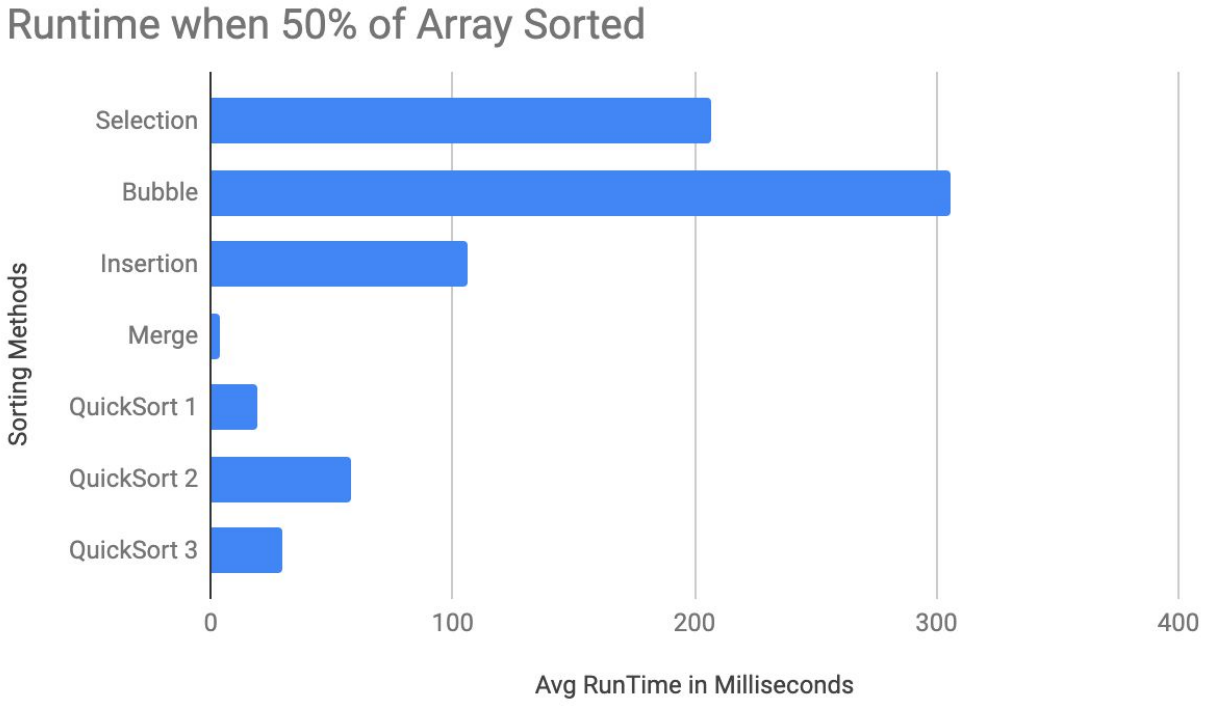
Runtime of Sorting Algorithims Based on Repeated Elements in Array With Size of 10000 elements

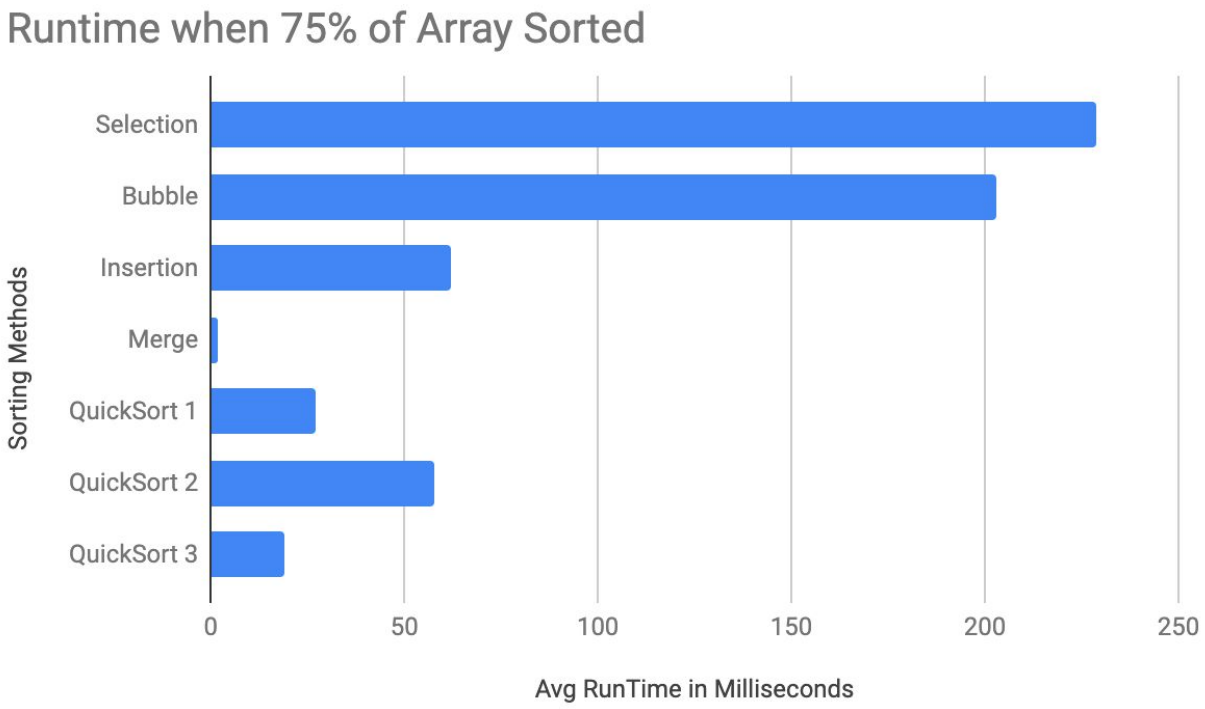



Runtime of QuickSort Algorithims Based on Pivot With Size of 10000 elements
QuickSort 1
QuickSort 2
QuickSort 3
Sorting Methods
0
2
4
6
8
Avg RunTime in MS

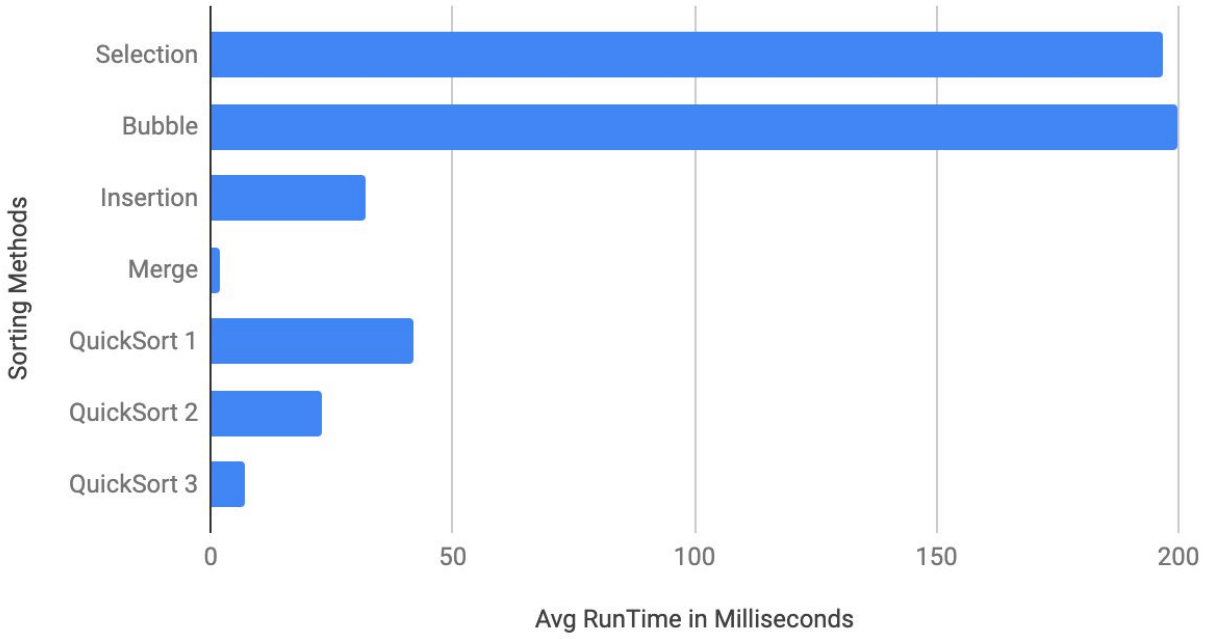

Runtime of Merge and Quicksort Methods with 10000 elements
Merge
Java Merge Sort
QuickSort 3
Java QuickSort
Sorting Methods
0
2
4
6
8
10
Avg RunTime in Milliseconds


Runtime when 25% of Array Sorted
Selection
Bubble
Insertion
Merge
QuickSort 1
QuickSort 2
QuickSort 3
Sorting Methods
0
100
200
300
400
Avg RunTime in Milliseconds

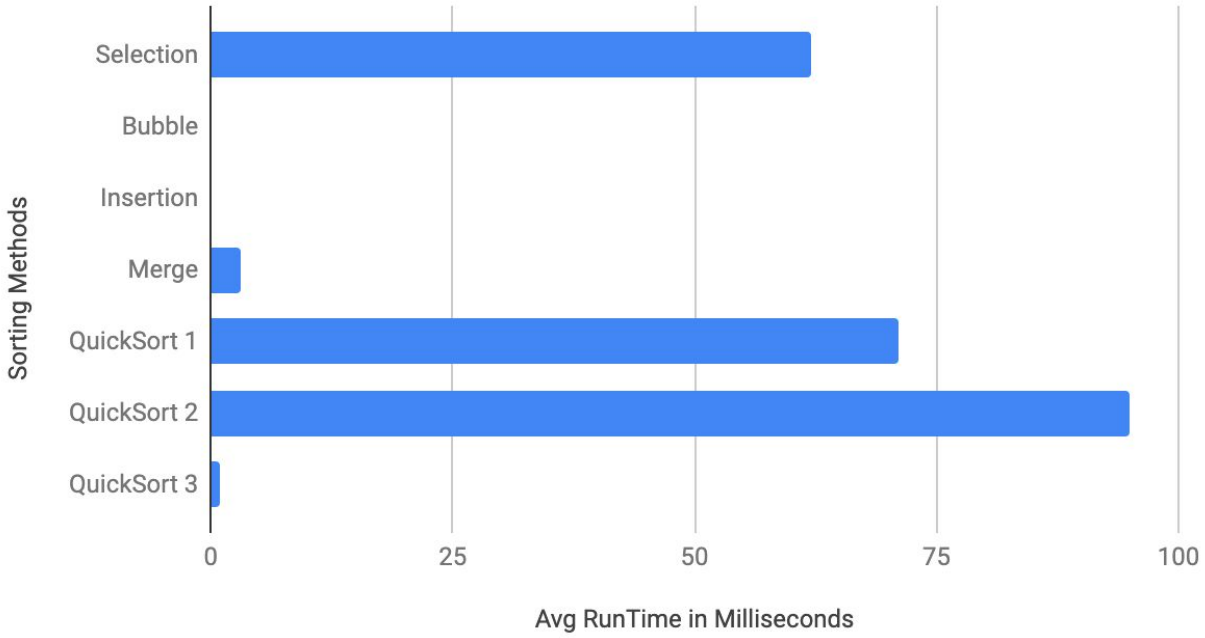

Runtime when 50% of Array Sorted
Selection
Bubble
Insertion
Merge
QuickSort 1
QuickSort 2
QuickSort 3
Sorting Methods
0
100
200
300
400
Avg RunTime in Milliseconds


Runtime when 75% of Array Sorted
Selection
Bubble
Insertion
Merge
QuickSort 1
QuickSort 2
QuickSort 3
Sorting Methods
0
50
100
150
200
250
Avg RunTime in Milliseconds

Runtime when 95% of Array Sorted



Runtime when 100% of Array Sorted



Conclusion

Overall, we see that QuickSort 3 was the better quicksort implementation because it used the median as its pivot. We also noticed that the Merge sort and QuickSort 3 tended to be the fastest sorting algorithms even when the number of elements was 20000. Java's implementation of the Merge algorithm was faster than our implementation but our implementation of the quicksort with the median pivot is faster than java's implementation of the quicksort algorithm. With all this data, our theoretical analysis was right because merge had the highest time complexity of big O of $O(nlogn)$.

References

Arrays API

Random API