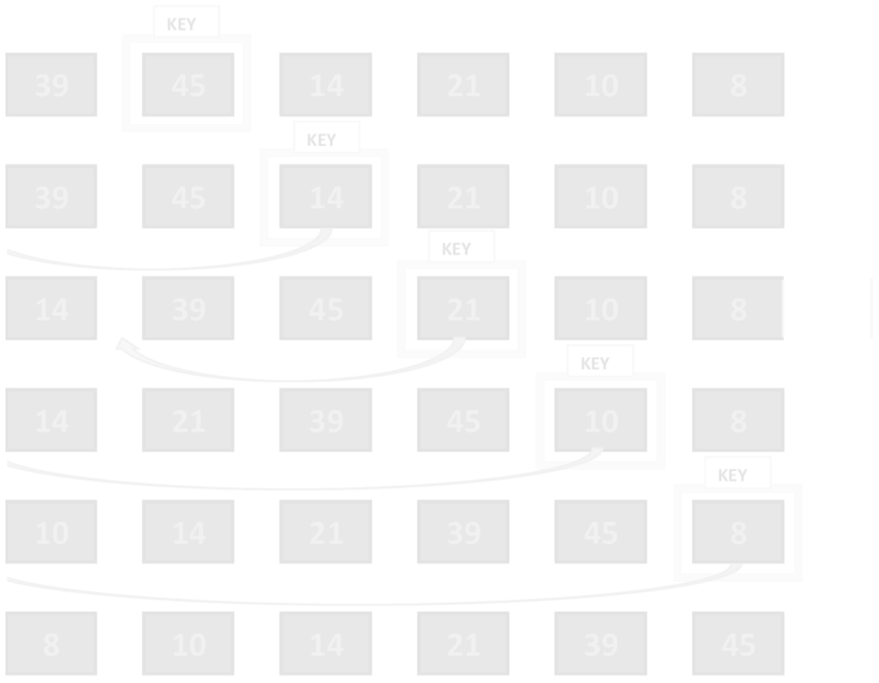
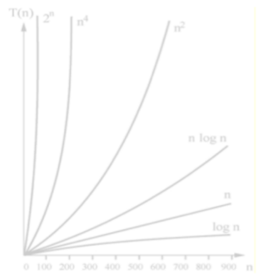
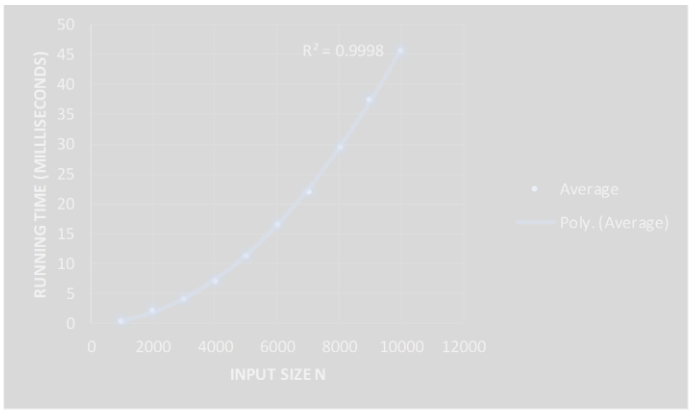






Project – Benchmarking Sorting Algorithms

Project Report

STUDENT No. G00376187

5/13/20

Computational Thinking with Algorithms

Table of Contents

Project Report

Introduction

The purpose of this report is to examine and discuss the results of a Java application which was used to benchmark five sorting algorithms.

This section of the report will discuss sorting algorithms, focusing on topics such as performance, in-place sorting and stable sorting.

Sorting Algorithms

A Sorting Algorithm is used to rearrange a given array or list elements into a specific order according to a comparison operator on the elements. The purpose of the comparison operator is to decide the new order of element within the respective data structure. (Sorting Algorithms - GeeksforGeeks, 2020)

Sorting algorithms take input data, perform specific operations on those lists and output arrays in the specified output. Examples of the uses of sorting algorithms include organizing items by price on a retail website and ordering results on a search engine results page. (What is sorting algorithm? - Definition from WhatIs.com, 2020)

The benefits of sorting algorithms include making computations and tasks simple by sorting information in advance. The search for efficient sorting algorithms dominated the early days of computing. (Heineman, Pollice and Selkow, 2016)

Performance

''Algorithm analysis provides theoretical estimation for the required resources of an algorithm to solve a specific computational problem. Analysis of algorithms is the determination of the amount of time and space resources required to execute it. Usually, the efficiency or running time of an algorithm is stated as a function relating the input length to the number of steps, known as time complexity, or volume of memory, known as space complexity. Analysis of algorithm is the process of analysing the problem-solving capability of the algorithm in terms of the time and size required (the size of memory for storage while implementation). The main concern of analysis of algorithms is the required time or performance. Generally, we perform the following types of analysis . Complexity theory is the study of algorithm performance. Big O Notation is used to measure performance of an algorithm. This method allows programmers to gauge the resources needed to solve a problem first and allows selection of an appropriate algorithm.'' (extracted from Data Structures Tutorials - Performance Analysis with examples, 2020)

Best-case – The minimum number of steps taken on any instance of size a. Defines a class of problem instances for which an algorithm exhibits its best runtime behaviour. For these instances , the algorithm does the least work.

Average case – An average number of steps taken on any instance of size a. Defines the expected behaviour when executing the algorithm on random problems instances. This measure describes the expectation an average user of the algorithm should have.

Worst-case – The maximum number of steps taken on any instance of size a. Defines a class of problem instances for which an algorithm exhibits its worst runtime behaviour. Instead of trying to identify the specific input, algorithm designers typically describe the properties of the input that prevent an algorithm from running efficiently.

Best, Average and Worst-case adapted from following sources;(Heineman, Pollice and Selkow, 2016) (DAA - Analysis of Algorithms - Tutorialspoint, 2020)

Time Complexity

Time complexity describes the amount of time needed to execute an algorithm in relation to the size of the input. Time can be defined as the amount of memory accesses performed, the number of comparisons between integers, the number of times an inner loop is executed, or any other natural unit related to the amount of real time the algorithm will take. (Algorithm Efficiency, 2020) (DAA - Analysis of Algorithms - Tutorialspoint, 2020)

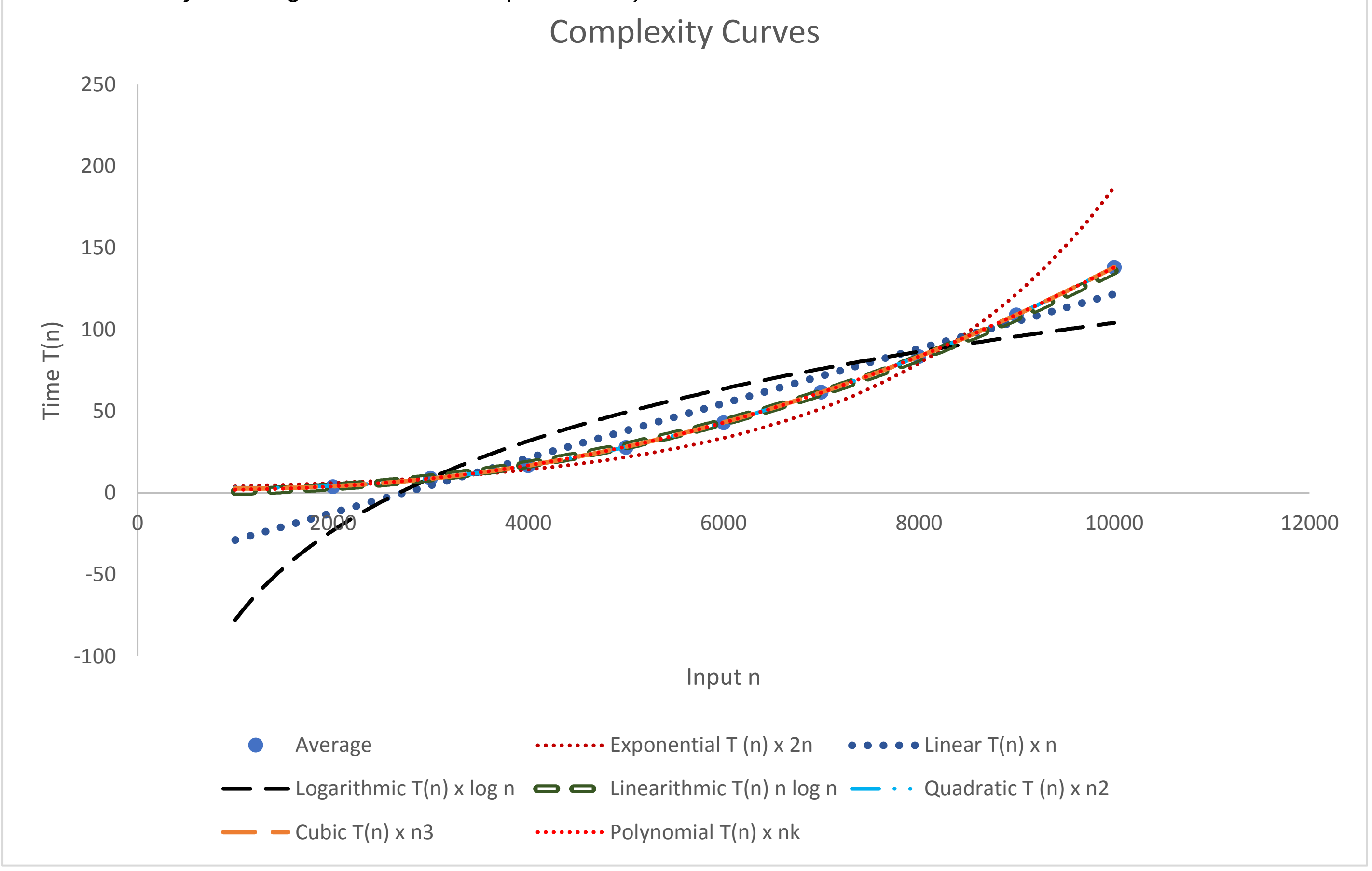


Figure 1 Time Complexity Curves (Sandra Rawat)

Space Complexity

Space complexity describes the amount of memory an algorithm takes to execute in relation to the size of input in relation to the algorithm. Fixed length units are used to measure this. Space complexity can be viewed as less relevant than time complexity because the space used is minimal or obvious, however as it increases it can be relevant an issue as time. (Algorithm Efficiency, 2020) (DAA - Analysis of Algorithms - Tutorialspoint, 2020)

In-place sorting

In-place sorting algorithms are memory efficient because they do not require extra space. It produces an output in the same memory that contains the data by transforming the input 'in-place' A small constant extra space for variables can be used. (In-Place Algorithm - GeeksforGeeks, 2020) Usually this space is O(log n) however space which measures in o(n) is allowed.

In-place algorithms :

- Bubble Sort
- Selection Sort
- Insertion Sort
- Heapsort
- Quicksort(except for recursive calls)

Not in place algorithms :

- Merge Sort
- Bucket Sort

Stable sorting

Sorting algorithms that can guarantee to maintain the relationship between two equal elements in an unordered collection are considered to be (Heineman, Pollice and Selkow, 2016, p.55) An unstable sorting algorithm does not consider between element locations in the original location, however it may or may not maintain relative ordering. Any sorting algorithm can be made stable by considering indexes as comparison parameter.

Comparator functions:

The method used to sort objects will be dependent on the type of objects.

- Integers can be compared by comparator functions: $< > = \leq$
- Characters and strings can be compared by lexicographical ordering.
- Custom made ordering schemes can also be applied. (Mannion, 2019)

Comparison-based sorts

A comparison sort uses comparator operators only to determine the order of the sorted list, A comparator is required to compare elements. The comparator defines the ordering e.g. numerical order, dictionary order to arrange elements. The best performance for comparison-based sorting algorithms to sort *n* elements is O(*n log n*) .

Comparison-based sorts include:

- Bubble Sort
- Insertion Sort
- Selection Sort
- Merge Sort
- Quicksort,
- Heapsort

Non-comparison-based sorting

Non-comparison-based sorting algorithms use integer arithmetic on keys rather than rely on comparison, making certain assumption about the data they are going to sort.

Non comparison sorting algorithms include :

- Counting sort; sorts using key-value,
- Radix sort; examines individual bits of keys, and
- Bucket Sort; examines bits of keys.

These are also known as Linear sorting algorithms because they sort in O(n) time

(Difference between Comparison (QuickSort) and Non-Comparison (Counting Sort) based Sorting Algorithms, 2020)

Sorting Algorithms

1.A simple comparison-based sort: Bubble Sort

Bubble Sort is a simple sorting algorithm that repeatedly swaps adjacent elements that are in the wrong order until the whole list of items is in sequence. The below diagram depicts the first iteration in sorting the array on display. The algorithm will continue with further iterations until the array is fully sorted.

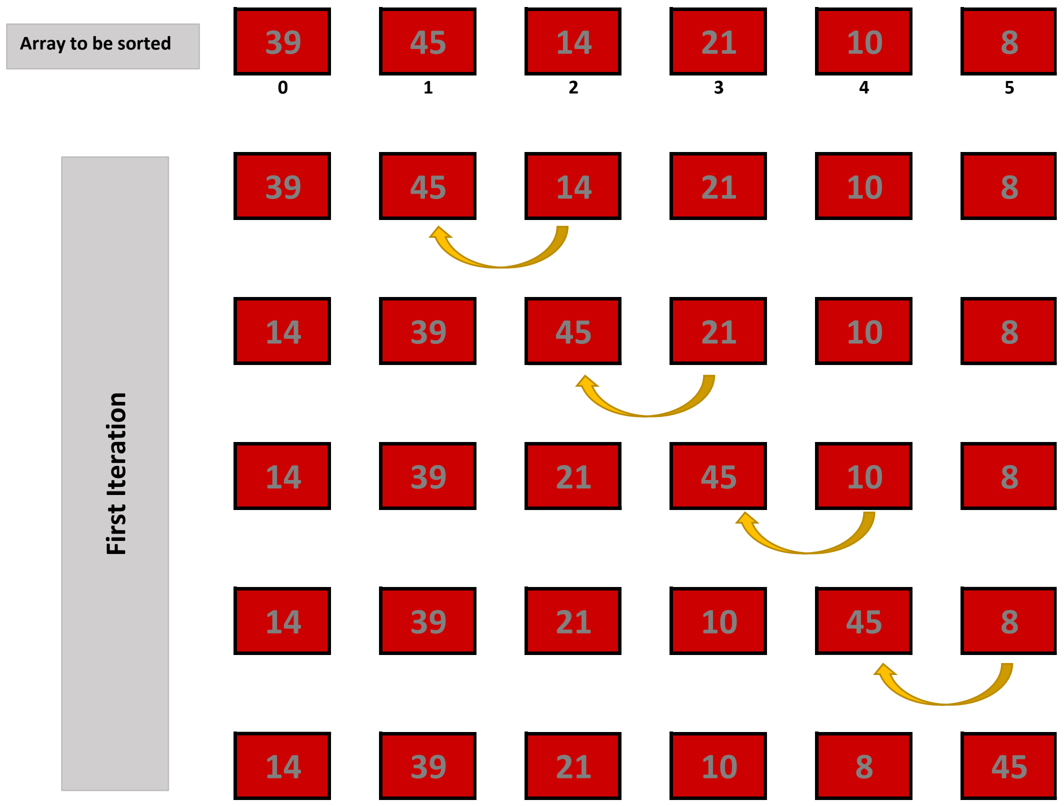


Figure 2 Diagram depicting Bubble Sort 1st iteration

Advantages:

- Bubble sort is easy to implement and understand.
- Elements are swapped in-place without using additional storage; therefore the space complexity can be low.

Disadvantages :

- does not perform efficiently with lists of a large input size. This is due to bubble sort requiring n-squared processing steps for every n number of elements to be sorted. Therefore, it is more suitable for academic teaching rather than supporting general use applications. (An introduction to Bubble Sorts, 2020)

Sorting In-Place: Yes
Stable: Yes
Average Case Time Complexity: Quadratic T(n) x n^2
Worst Case Time Complexity: Quadratic T(n) x n^2. The worst-case scenario occurs when the array needs to be 'reverse sorted'
Best Case Time Complexity: Linear T(n) x log n . The best case occurs when the array is already sorted.

2. An efficient comparison-based sort: Heap Sort

The Heap sort algorithm is popular due to its efficiency. Heap sort works by transforming the list of items to be sorted into a binary tree with heap properties. In a binary tree, every node has, at most, two descendants. Heapify() converts the binary tree into a heap data structure, where it satisfies the heap-order property; the value stored at each node is greater than or equal to its children. (Heapify – Wordpress, 2012)

The largest element of the heap is removed and inserted into the sorted list. The remaining sub-tree is transformed into a heap again. This process is repeated until no elements remain. Successive removals of the root node after each rebuilding of the heap produces the final sorted list of items. (HeapSort - GeeksforGeeks, 2020)

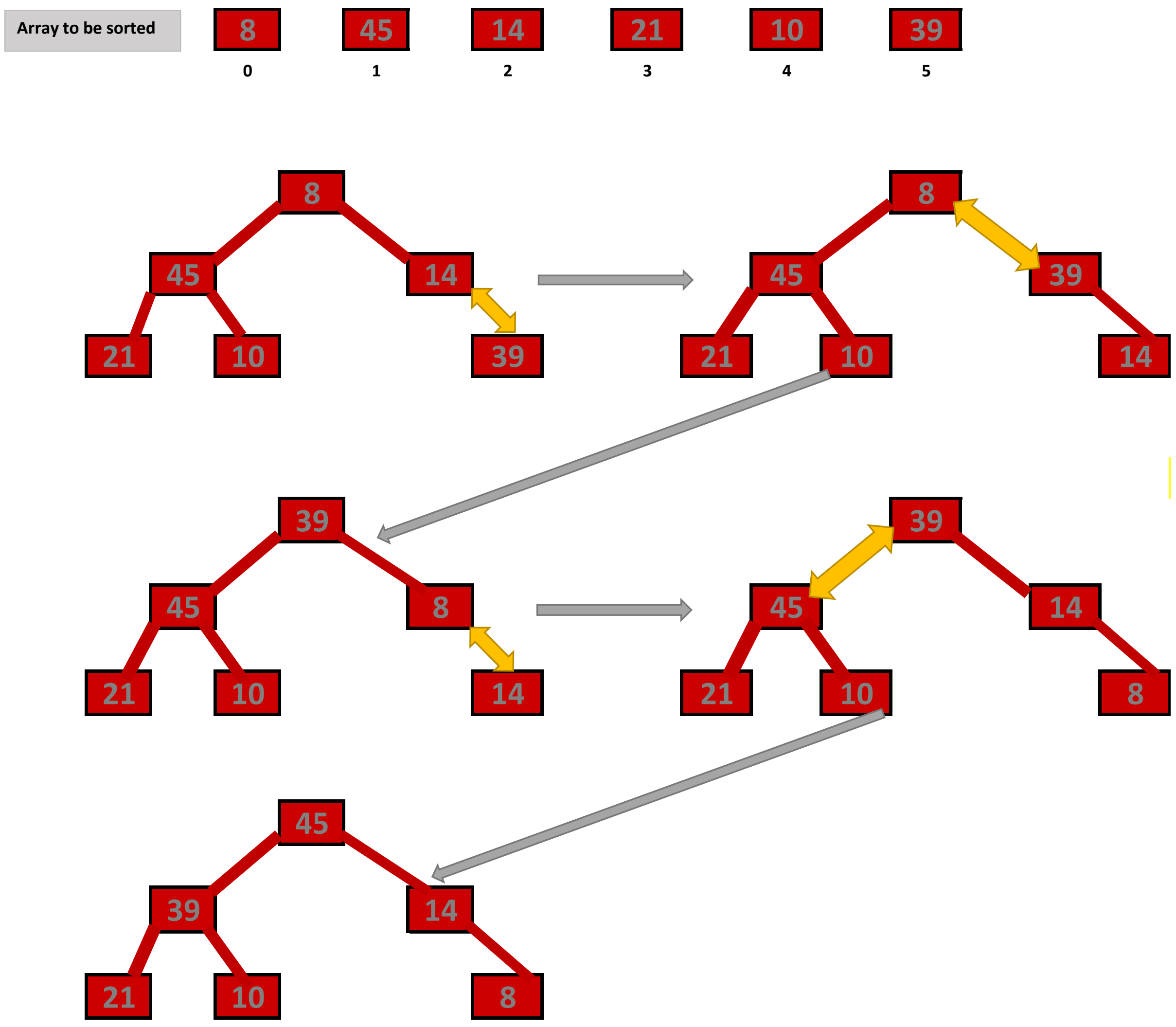


Figure 3 Diagram depicting Heap Sort

Advantages:

- Memory usage: the Heap sort algorithm can be implemented as an in-place sorting algorithm. This means that its memory usage is minimal because it needs no additional memory space to work. The Quicksort algorithm requires more stack space due to its recursive nature. (The Advantages of Heap Sort, 2020)

- Efficiency :other sorting algorithms may grow exponentially slower as the number of items to sort increase, the time required to perform Heap sort increases logarithmically making Heap sort particularly suitable for sorting a huge list of items. The performance of Heap sort is optimal which implies that no other sorting algorithms can perform better in comparison. (The Advantages of Heap Sort, 2020)

Disadvantages :

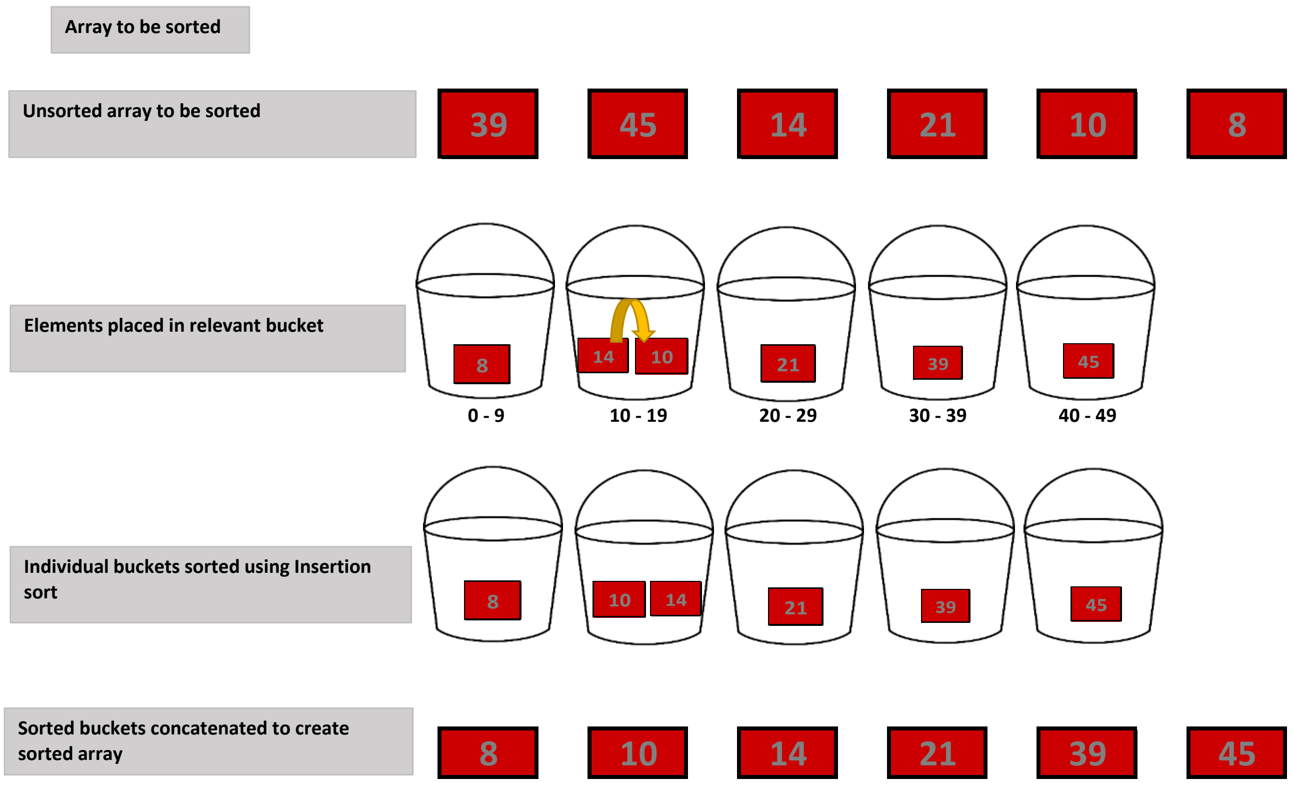
- Storing data on the heap is slower than storing it on the stack (Heap advantage and disadvantage | Practice | GeeksforGeeks, 2020)

Sorting In-Place: Yes
Stable: No, but can be made stable.
Average Case Time Complexity: Linear T(n) x log n
Worst Case Time Complexity: Linear T(n) x log n
Best Case Time Complexity: Linear T(n) x log n

3. A non- comparison-based sort: Bucket Sort

Bucket sort is mainly useful when input is uniformly distributed over a range, so that buckets are created to evenly partition the input range. Bucket sort constructs a set of n ordered buckets into which the elements of the input set are partitioned using a hash function, given a set of n elements. (extracted from Heineman, Pollice and Selkow, 2016) (Bucket Sort, 2020)

Once all elements to be sorted are inserted into the buckets, bucket sort extracts the values from left to right using Insertion sort on the contents of each bucket. As the values from the buckets are extracted from left to right to repopulate the original array the elements are ordered in each respective bucket. (extracted from Heineman, Pollice and Selkow, 2016)



Array to be sorted
Unsorted array to be sorted
39
45
14
21
10
8
Elements placed in relevant bucket
8
14
10
21
39
45
0 - 9
10 - 19
20 - 29
30 - 39
40 - 49
Individual buckets sorted using Insertion sort
8
10
14
21
39
45
Sorted buckets concatenated to create sorted array
8
10
14
21
39
45

Figure 4 Diagram depicting Bucket Sort

Advantages:

- bucket sort is faster to run than bubble sort. Inserting data into small buckets that are sorted individually reduces the number of comparisons to be carried out.

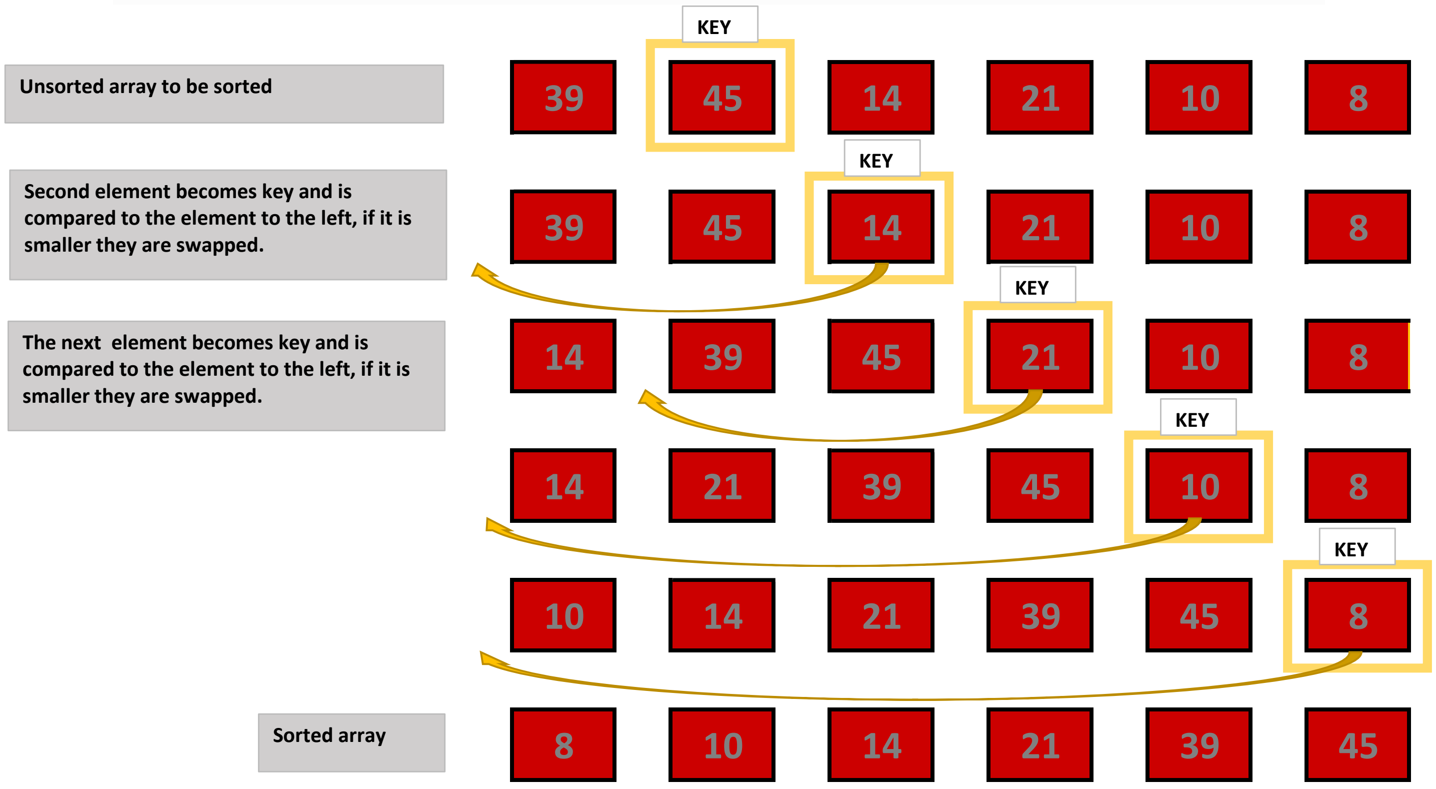
Disadvantages :

- bucket sort is a more complicated algorithm than bubble sort to describe for a computer. This is data type dependent, integers can be easily sorted into buckets, whereby letters can be more challenging to divide, due to some being more frequently used than others.

Sorting In-Place: No
Stable: Yes
Average Case Time Complexity: O(n + k)
Worst Case Time Complexity: Quadratic T(n) x n^2
Best Case Time Complexity: O(n + k)

4.A sorting algorithm of choice: Insertion Sort

Insertion sort is a comparison sort algorithm that works the way we sort playing cards in our hands. Insertion sort repeatedly scans the list of items, each time inserting the item in the unordered sequence into its correct position. (Heineman, Pollice and Selkow, 2016)


Unsorted array to be sorted
Second element becomes key and is compared to the element to the left, if it is smaller they are swapped.
The next element becomes key and is compared to the element to the left, if it is smaller they are swapped.
Sorted array
KEY
39 45 14 21 10 8
39 45 14 21 10 8
14 39 45 21 10 8
14 21 39 45 10 8
10 14 21 39 45 8
8 10 14 21 39 45

Advantages:

Figure 5 Diagram depicting Insertion Sort

- The main advantage of the insertion sort is its simplicity.
- It also exhibits a good performance when dealing with a small list.
- The insertion sort is an in-place sorting algorithm so the space requirement is minimal.

Disadvantages :

- The disadvantage of the insertion sort is that it does not perform as well as other, better sorting algorithms. With n-squared steps required for every n element to be sorted, the insertion sort does not deal well with a huge list. Therefore, the insertion sort is particularly useful only when sorting a list of few items. It can also be useful when input array is almost sorted, only few elements are misplaced in complete big array.

(The Advantages & Disadvantages of Sorting Algorithms, 2020)

Sorting In-Place: Yes
Stable: Yes
Average Case Time Complexity: Quadratic T(n) x n2
Worst Case Time Complexity: Quadratic T(n) x n^2
Best Case Time Complexity: Quadratic T(n) x n^2

5. A sorting algorithm of choice : Quick Sort

Introduced by C.A.R. Hoare in 1960, Quicksort selects an element or a pivot in the collection, either randomly, leftmost or middle to partition an array into sub arrays. There are numerous enhancements and optimizations launched for Quicksort that have made Quicksort the most efficient of any algorithm. Sedgewick (1978) suggests that a combination of median of three for selecting the pivot point and using insert sort for small sub arrays offers a speedup of 20m – 25% over pure Quicksort . (Heineman, Pollice and Selkow, 2016, p.73)

Given an array and an element x of array as the pivot, x is placed at its correct position in the sorted array and each element is sorted recursively by placing all elements smaller than x before x, and placing all elements greater than x after x. This should occur in linear time. (QuickSort - GeeksforGeeks, 2020)

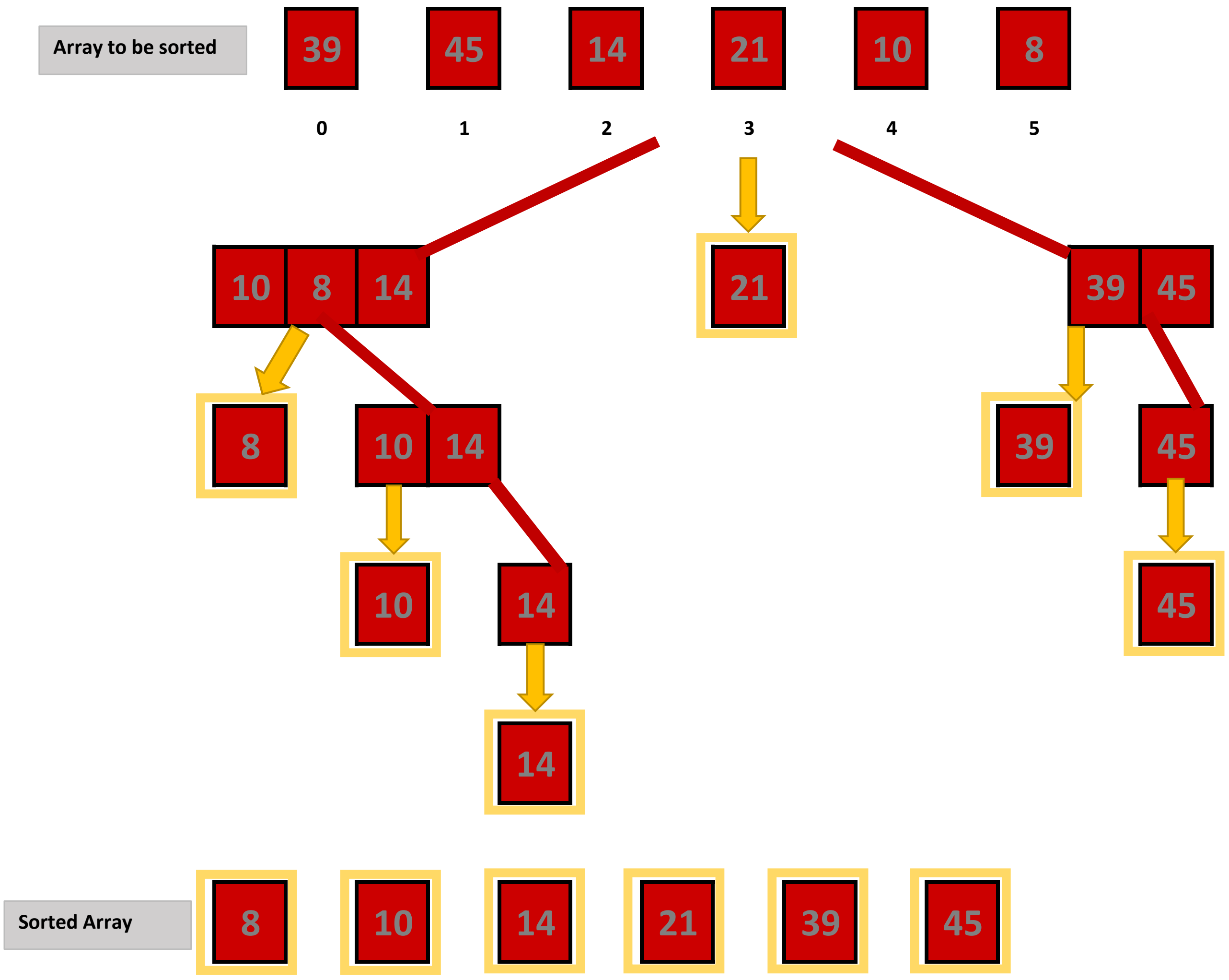


Figure 6 Diagram depicting Quicksort

Advantages:

- QuickSorts' advantage in terms of efficiency because it is able to deal well with a huge list of items. Because it sorts in place, no additional storage is required. QuickSort can be implemented in different ways by changing the choice of pivot, so that the worst case rarely occurs for a given type of data. (QuickSort - GeeksforGeeks, 2020)
- In general, the quick sort produces the most effective and widely used method of sorting a list of any item size.

Disadvantages :

- its worst-case performance is similar to the average performances of bubble, insertion or selections sorts. Although the worst case time complexity of QuickSort is O(n^2) which is more than many other sorting algorithms, QuickSort is faster in practice, because its inner loop can be efficiently implemented on most architectures, and in most real-world data. (QuickSort - GeeksforGeeks, 2020)

Sorting In-Place: Yes
Stable: No, although can be made stable
Average Case Time Complexity: O(n Log n) The average case occurs when a random element of pivot is selected, enabling Quicksort to provide an average case performance that usually outperforms other sorting algorithms.

Worst Case Time Complexity: Quadratic T(n) x n^2 Quick sort exhibits worse case quadratic behaviour if partitioning at each recursive step only divides a collection of n elements into an empty and large set, by picking the greatest or smallest point as the pivot, where one of these sets has no elements and the other has n-1. (Analysis of quicksort (article) | Quick sort | Khan Academy, 2020)

Best Case Time Complexity: O(n Log n) The best case occurs when the partition process always picks the middle element as pivot.

Implementation & Benchmarking

Process

The author created this benchmarking project to focus on the average running times for each of the above-mentioned Sorting Algorithms. The author also calculated the worst and best times to gain a greater view of each sorting algorithms. Process followed to examine the average times for the sorting algorithms:

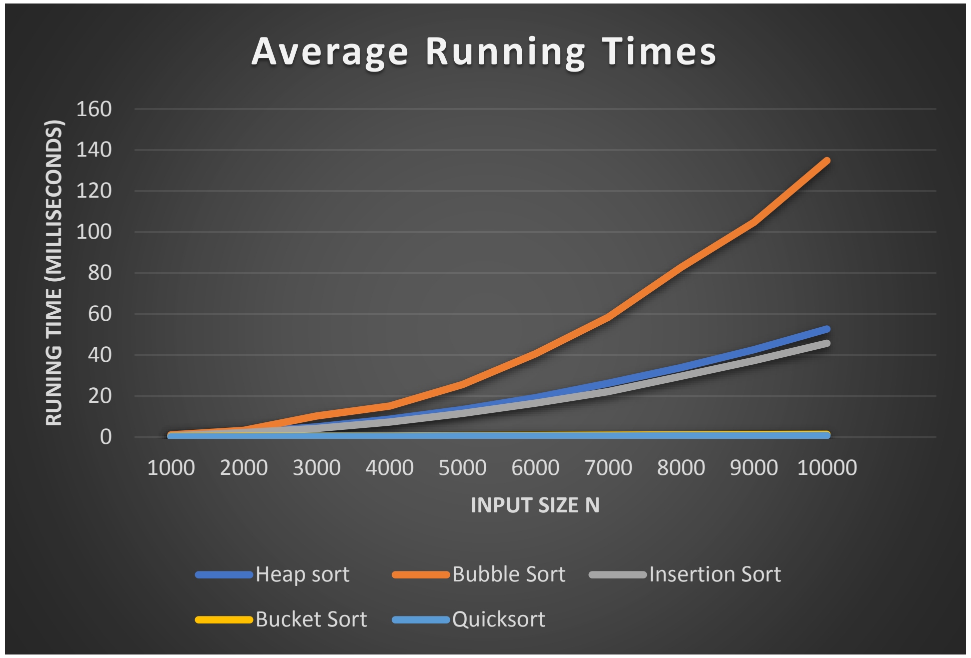
1. Generate random arrays of the input sizes in increments of ‘000 [1000, 2000, 3000, 4000, 5000, 6000, 7000, 8000, 9000,10000].
2. Carry out warmUp runs x 10
3. Run 10 times, divide results by 10 to calculate the average time
4. Convert results to milliseconds (ms)
5. Output the results to the console, followed by graphing.

Average	**1000**	**2000**	**3000**	**4000**	**5000**	**6000**	**7000**	**8000**	**9000**	**10000**
Heap sort	0.662	2.306	4.985	8.718	13.449	19.192	26.224	33.871	42.59	52.653
Bubble Sort	1.009	3.147	10.189	15.018	25.539	40.597	58.382	82.757	104.86	134.862
Insertion Sort	0.469	2.121	4.082	7.272	11.489	16.555	22.11	29.608	37.364	45.71
Bucket Sort	0.214	0.332	0.346	0.463	0.601	0.717	0.875	1.018	1.166	1.303
Quicksort	0.048	0.116	0.183	0.248	0.325	0.395	0.469	0.536	0.619	0.703

Worst	**1000**	**2000**	**3000**	**4000**	**5000**	**6000**	**7000**	**8000**	**9000**	**10000**
Heap sort	0.635	2.312	5.013	8.691	13.466	19.392	26.039	33.782	42.607	52.491
Bubble Sort	0.776	3.105	7.85	14.971	25.779	43.011	58.726	81.787	107.269	139.138
Insertion Sort	0.466	1.904	4.184	7.397	11.32	16.277	22.673	29.52	36.795	45.809
Bucket Sort	0.054	0.09	0.115	0.154	0.193	0.236	0.307	0.355	0.375	0.437
Quicksort	0.244	0.934	2.309	3.924	5.825	8.141	Stack Overflow			

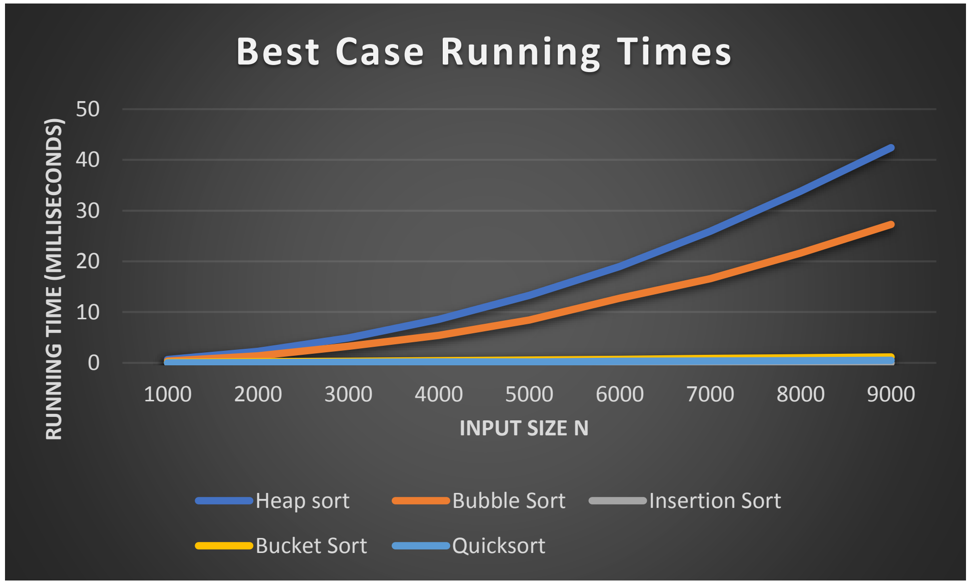
Best	**1000**	**2000**	**3000**	**4000**	**5000**	**6000**	**7000**	**8000**	**9000**	**10000**
Heap sort	0.657	2.258	4.899	8.579	13.344	18.97	25.924	33.811	42.409	52.254
Bubble Sort	0.359	1.379	3.258	5.415	8.465	12.725	16.626	21.649	27.288	33.811
Insertion Sort	0.002	0.003	0.005	0.006	0.008	0.009	0.017	0.012	0.014	0.015
Bucket Sort	0.119	0.224	0.328	0.463	0.585	0.71	0.892	1.013	1.165	1.259
Quicksort	0.023	0.059	0.107	0.158	0.212	0.288	0.362	0.425	0.507	0.596

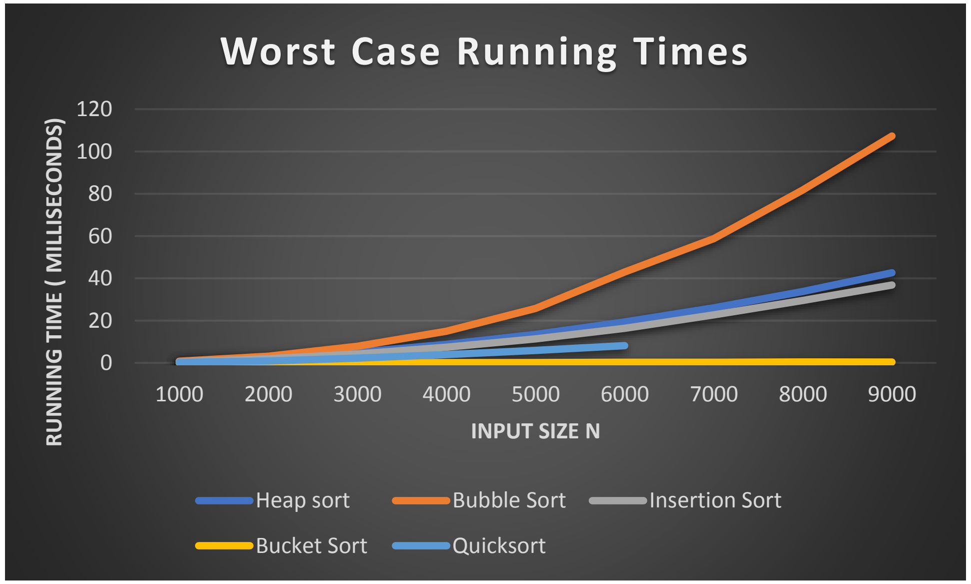
Table 1 Results of Benchmarking Sorting Algorithms



*Note: Microsoft Excel Scatter Plots were used in order to plot results against respective trendlines. The closer an R^2 is to 1, the better the specified trendline fits the data.

As shown on the Average Running Times graph, Quicksort and Bubble are the most efficient algorithms, sorting in O(n log n). Insertion sort and Heap sort are efficient, running in O(n log n), however research dictates that Heap Sort is a better sorter algorithm than Insertion Sort. Bucket and Bubble Sort is the are least efficient, both with a Time Complexity of n^2.

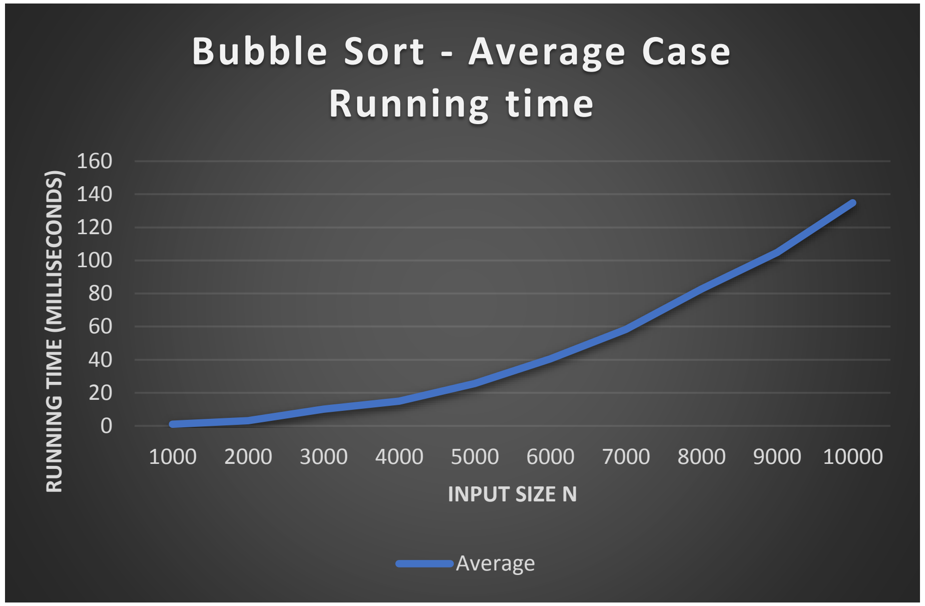


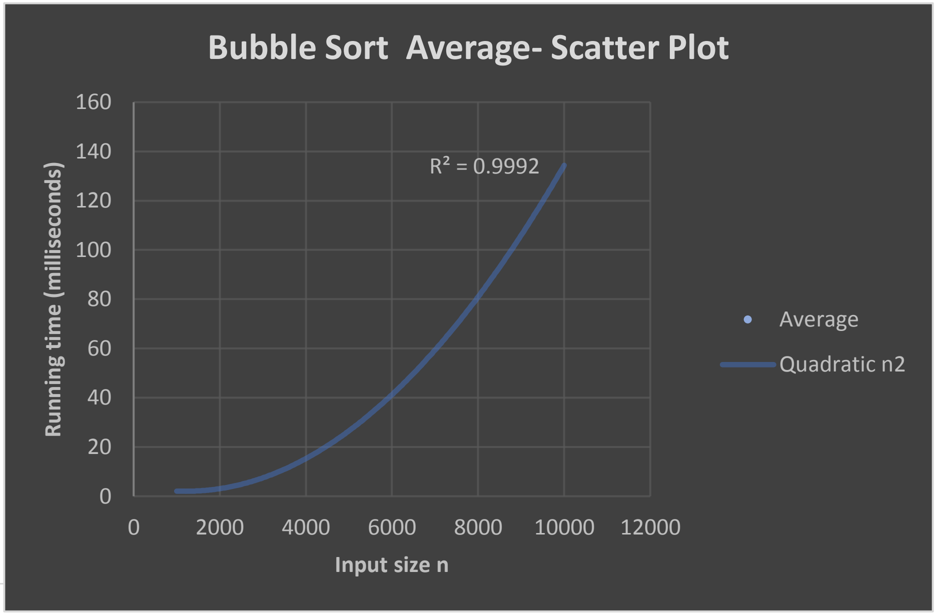


Results- individual sorting algorithms

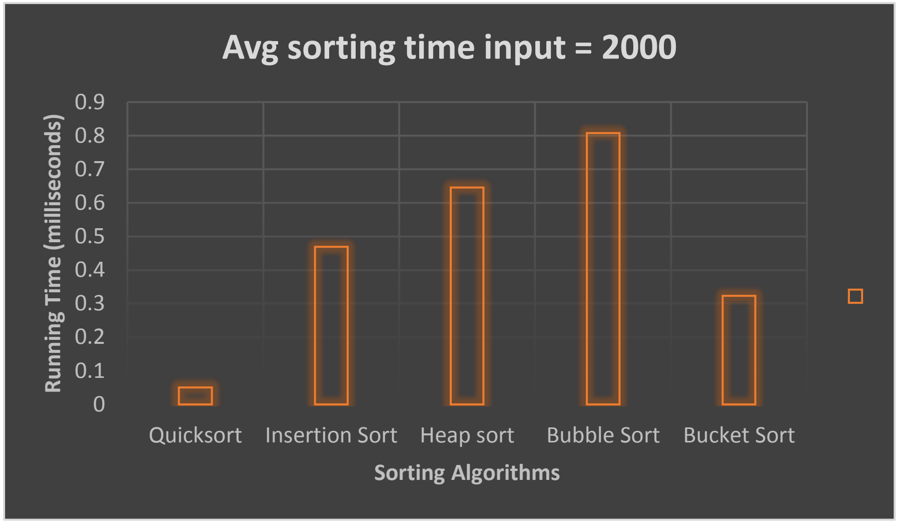
Bubble sort

The below scatter plot displays $R^2 = 0.9992$ to a Quadratic n^2 trendline, therefore the algorithm is a good fit to Time Complexity of n^2. As mentioned in literature regarding Bubble sort, the algorithm does not perform well with large list, as seen in the adjacent graph, as input size increases, time increases quadratically.



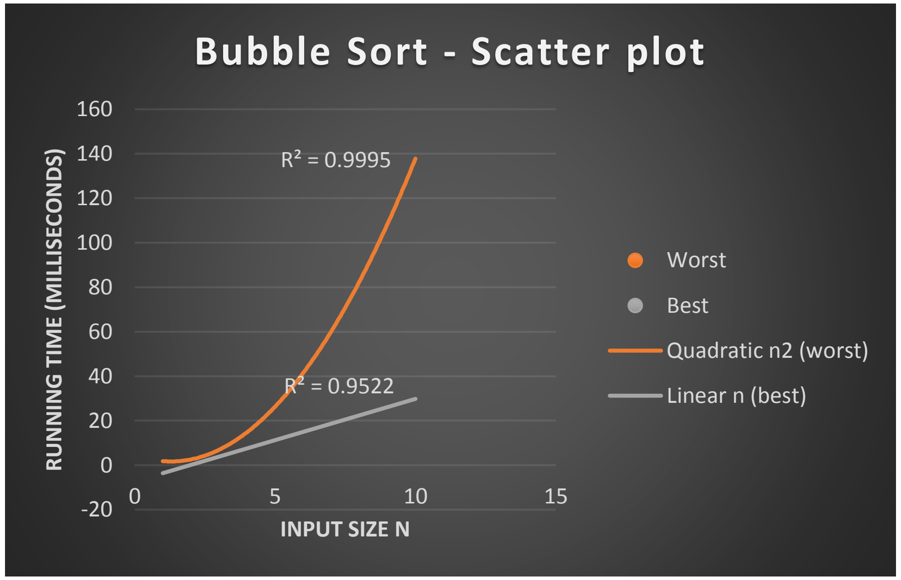


With a smaller input size of 2000 Bubble sort obtains a much more competitive sorting time compared to the other sorting algorithms in the adjacent bar chart.



The adjacent scatter plot plots the Bubble sort as Worst Case = quadratic $n^{2,}$ and Best Case = linear n, both are in line with the expected time complexity.

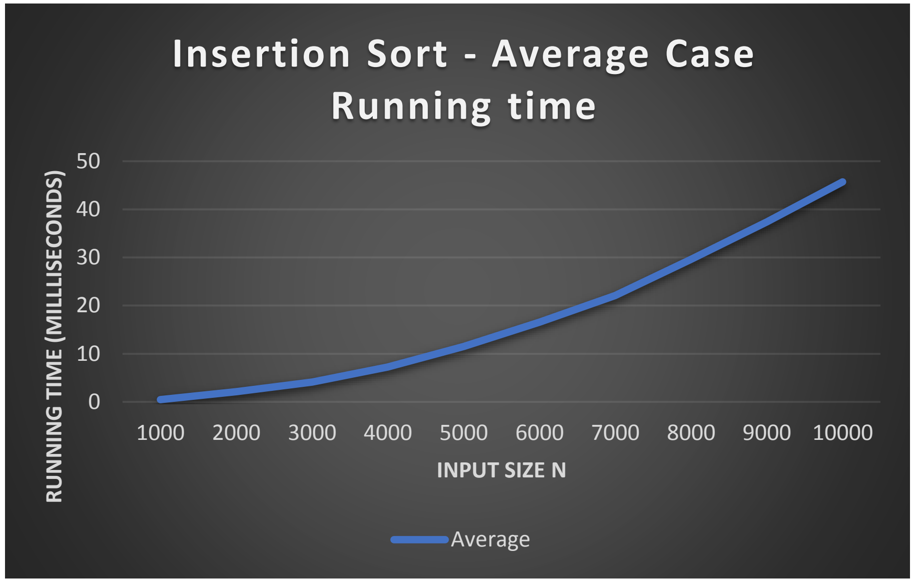
To calculate worst case, an inverted array was used, and for best case a sorted array was used

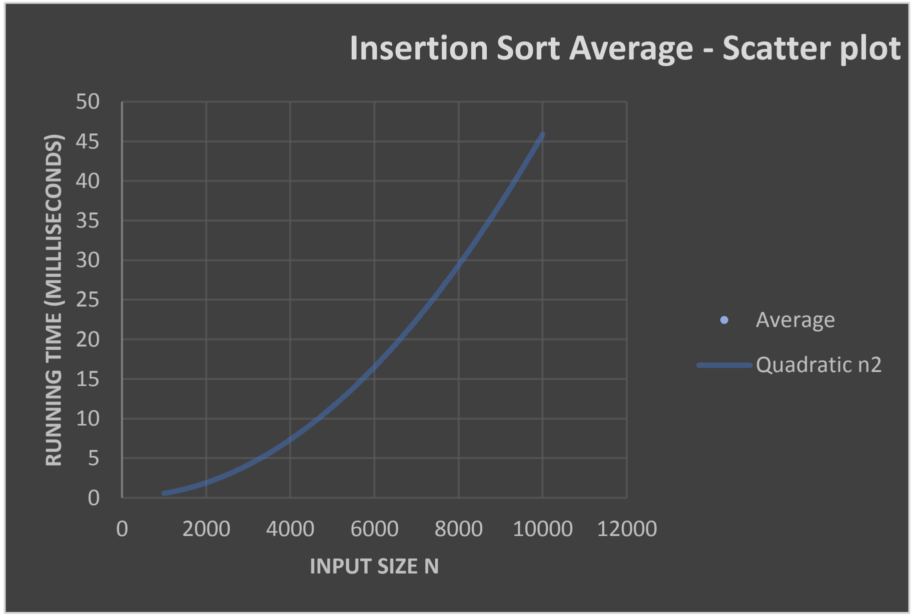


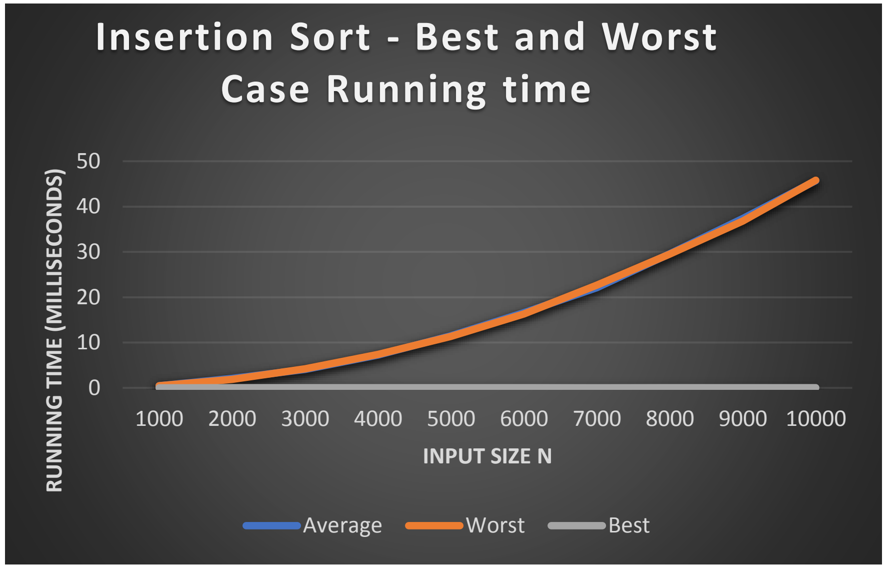
Insertion sort

The below graph displays $R^2 = 0.998$ to a Quadratic n^2 trendline, confirming the algorithm is a good fit to Time Complexity of n^2.

As insertion sort is reported to not perform well with large lists, it is evident from the data that the time increased as the array became larger.

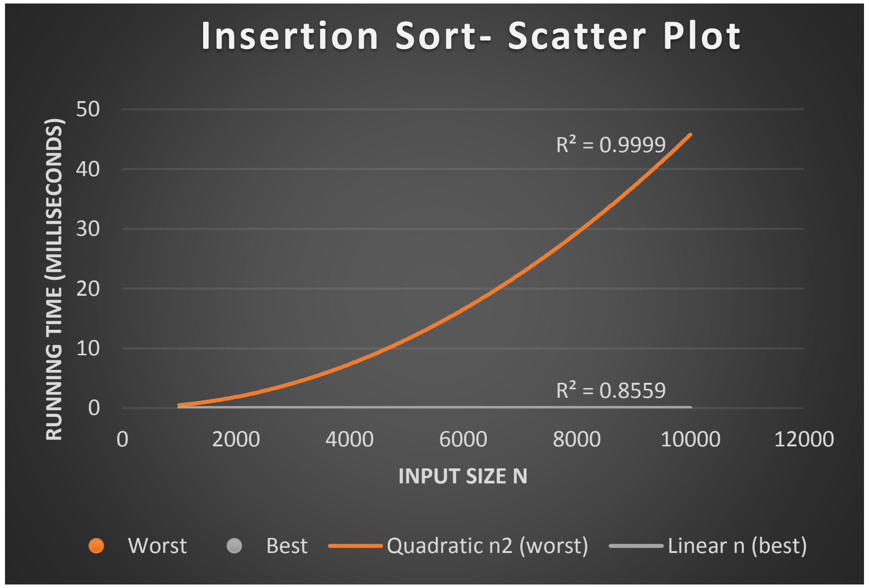






The below scatter plot plots the results as Worst Case = n^{2,} and Best Case = linear n, both are in line with the expected time complexity.

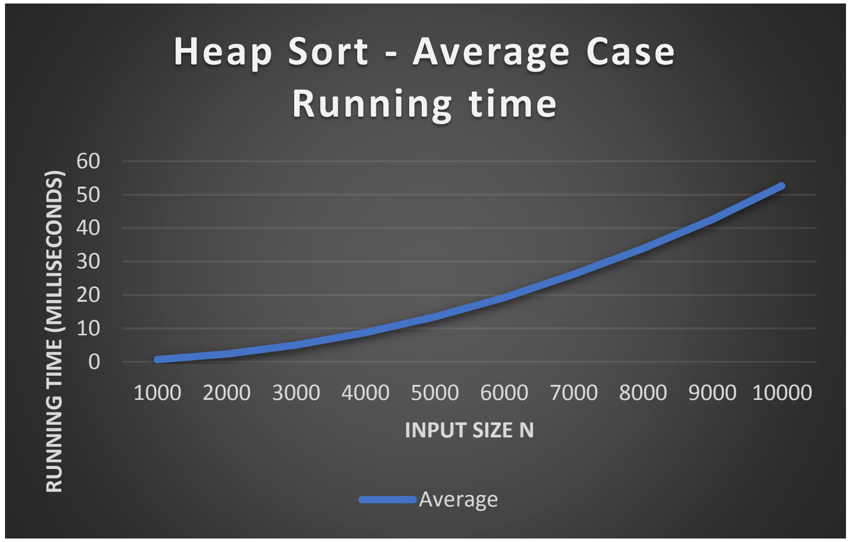
To calculate worst case, an inverted array was used, and for best case a sorted array.

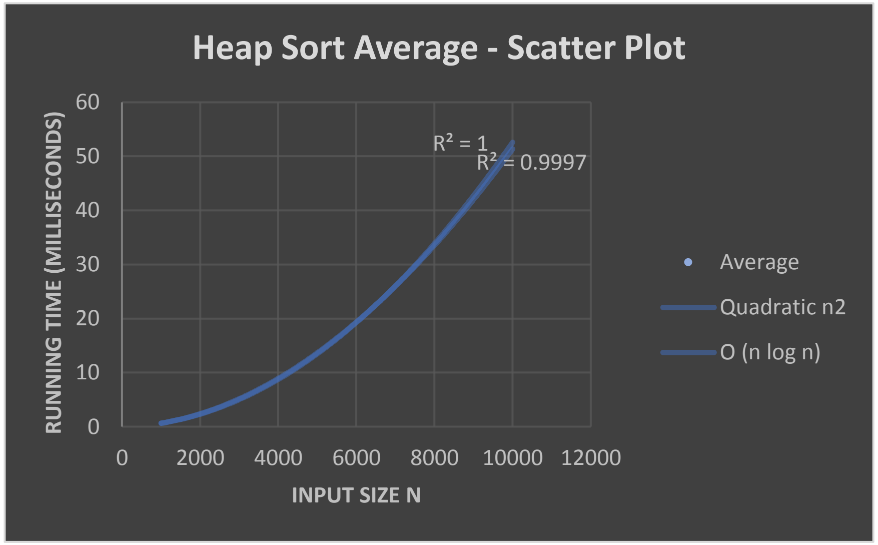


Heap sort

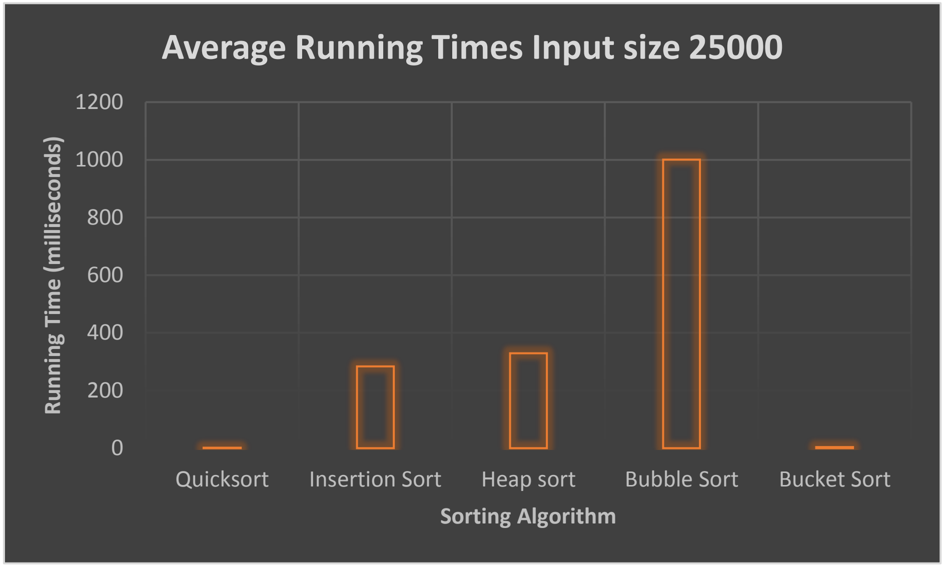
Heap sort is reported to have an Average Time complexity of O(n log n). However, in this project, a average time complexity of n^2 or quadratic is the best fit trendline, with an R^2 of 1 .When plotting to an O(n log n) trendline, R^2 was = to 0.997. This is a minor difference.

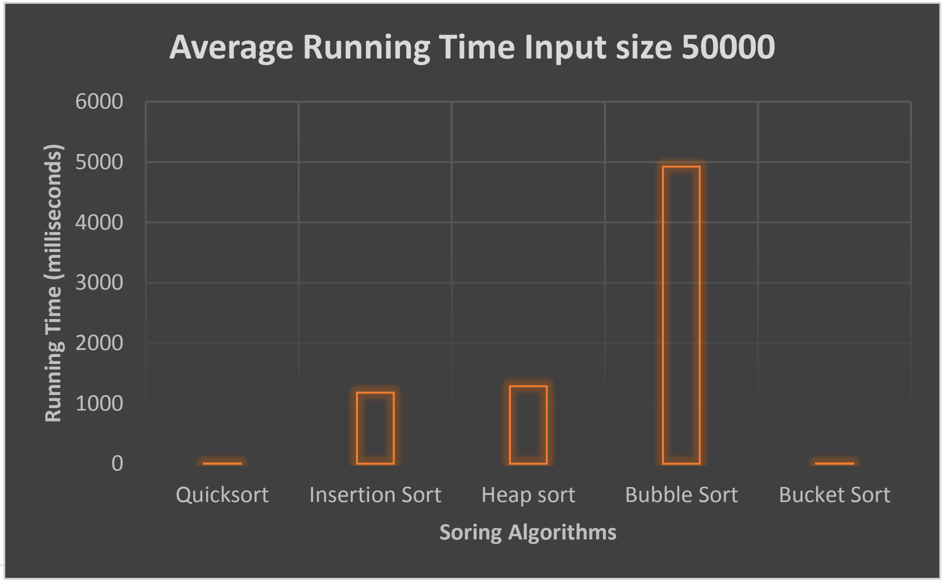
The author acknowledges that this may be due to other factors, such as code issues/differences or operating system differences. In this instance, heap sort did not increase logarithmically as mentioned in the Sorting Algorithms section of this report.





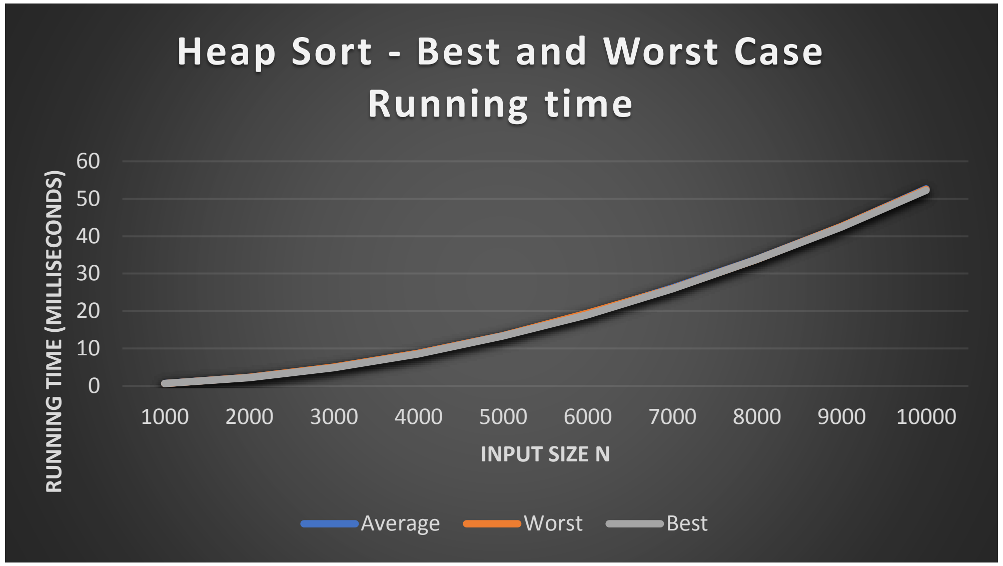
In order to investigate the research that states Heap Sort becomes more efficient as input increases, the author ran the programme with inputs of 25,000 and 50,0000. It is evident that as the input size increases, Heap Sort is able to perform very efficiently, the benefits of Heap sort are evident as it achieves sorting times of the same order of magnitude as Insertion Sort.

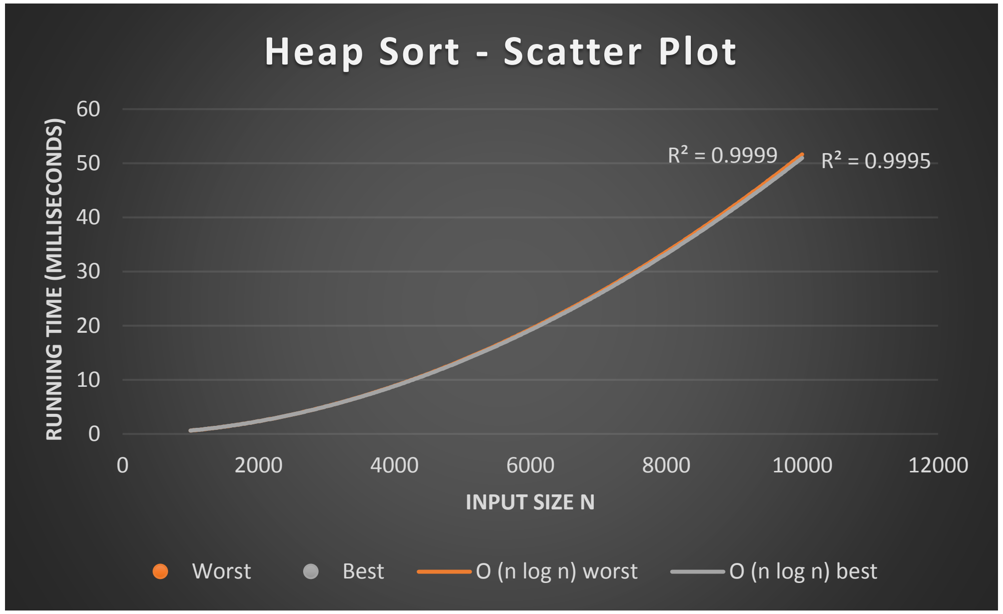




The below scatter plot plots the results as Worst Case = O (n log n) and Best Case O(n log n), both are in line with the expected time complexity for Heap sort.

The process carried out for Heap sort is the same for any arrangement of input data; inverted, sorted, or random. During the heapify procedure, the whole height of the heap is processed. So, it is not possible to generate sample manually to achieve different results for comparison for best case and worst case.

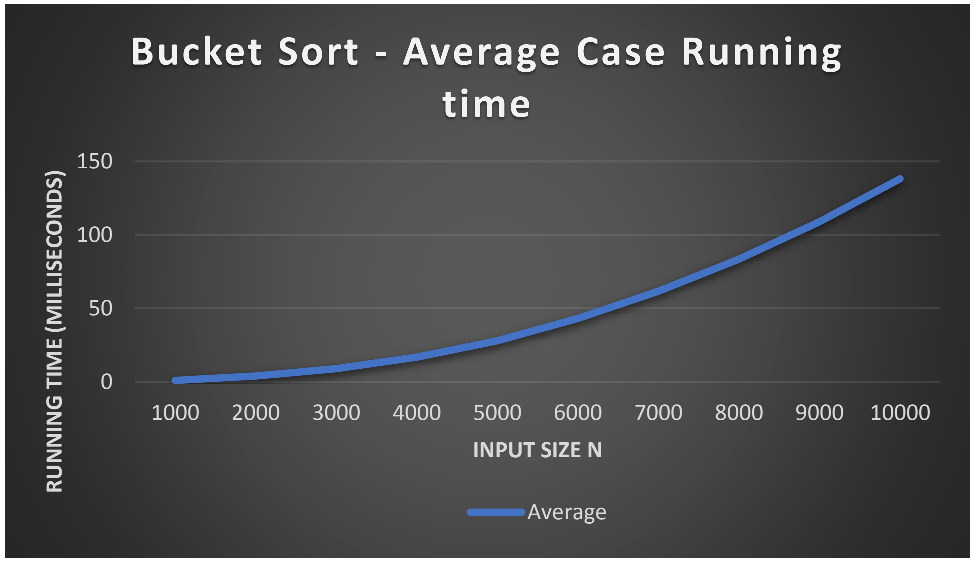


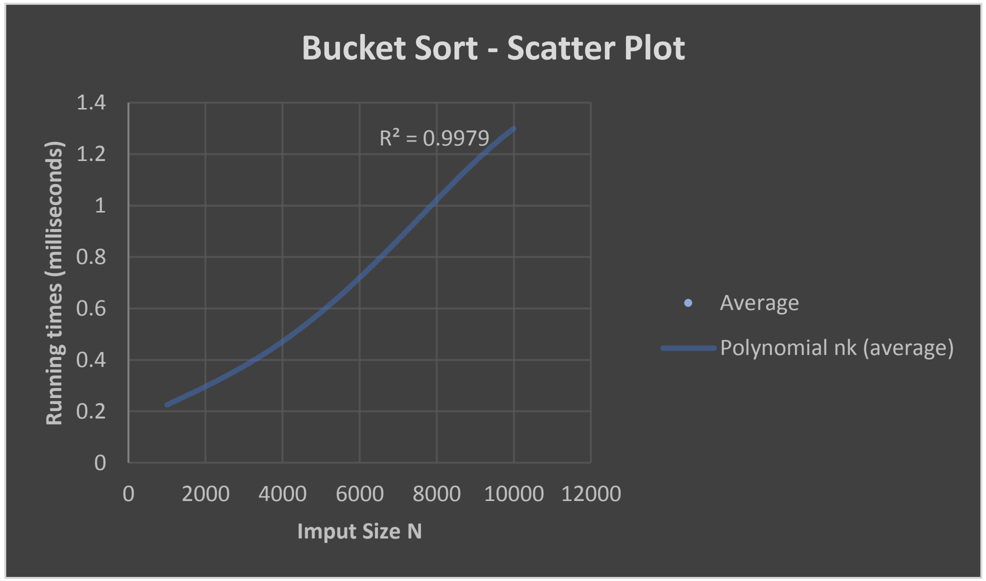


Bucket sort

Bucket sort is reported to achieve a Time Complexity of O(n +k) These results are confirmed by the below plot, by plotting to a polynomial trendline to the order of 4, with an $R^2 = 0.9979$.

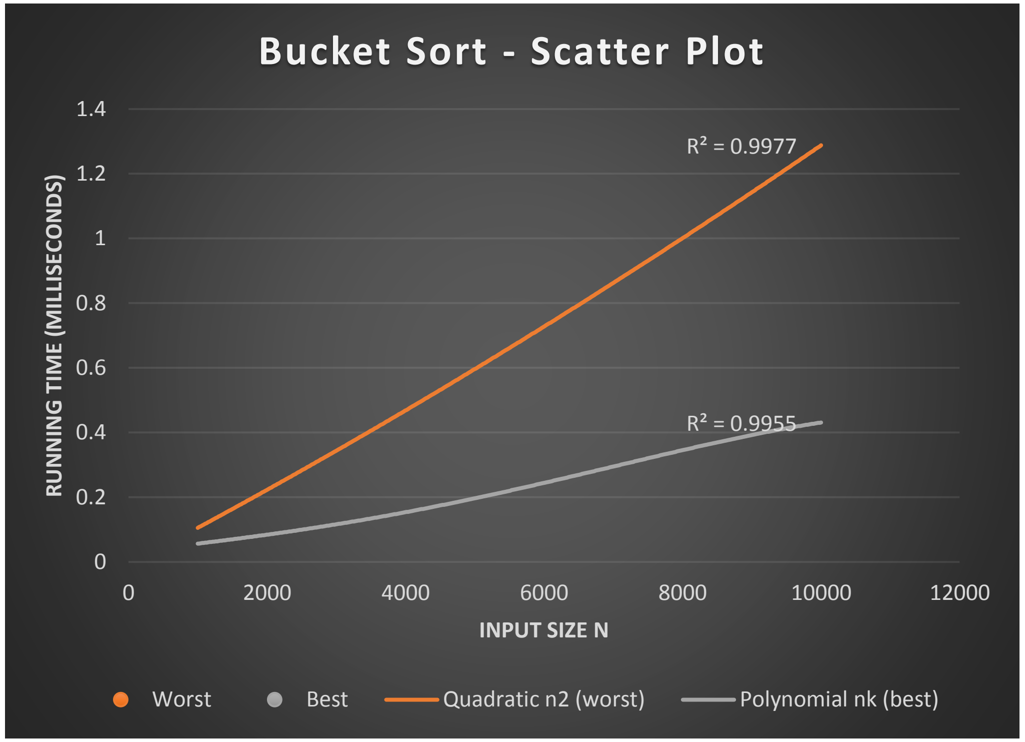
Bucket sort is reported to be faster than Bubble sort and this was confirmed during the benchmarking process. Bucket sort is a very efficient algorithm, competing with Quicksort for an efficient Time Complexity.





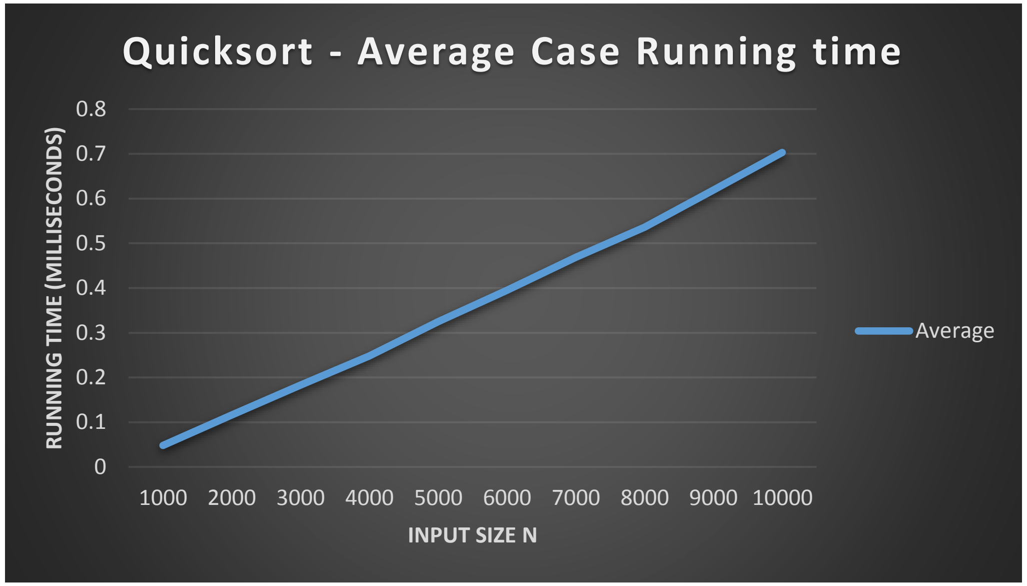
The adjacent scatter plot plots Bucket Sort as Worst Case = n^2 and Best Case nk, both are in line with the expected time complexity.

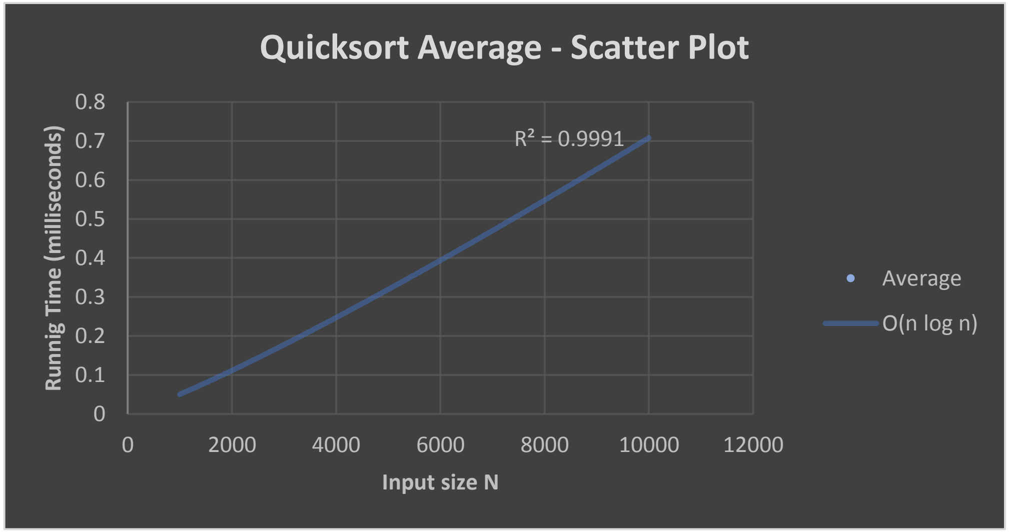
To calculate worst case, an inverted array was used, and for best case a sorted array was used.



Quick sort

Quicksort is a very efficient algorithm, as it has been the least time-consuming algorithm within the Benchmarking project. The average case time complexity of Quicksort is documented as O(n log n), which is evident on the scatter plot with a $R^2 = 0.991$ to the O(n log n) trendline.



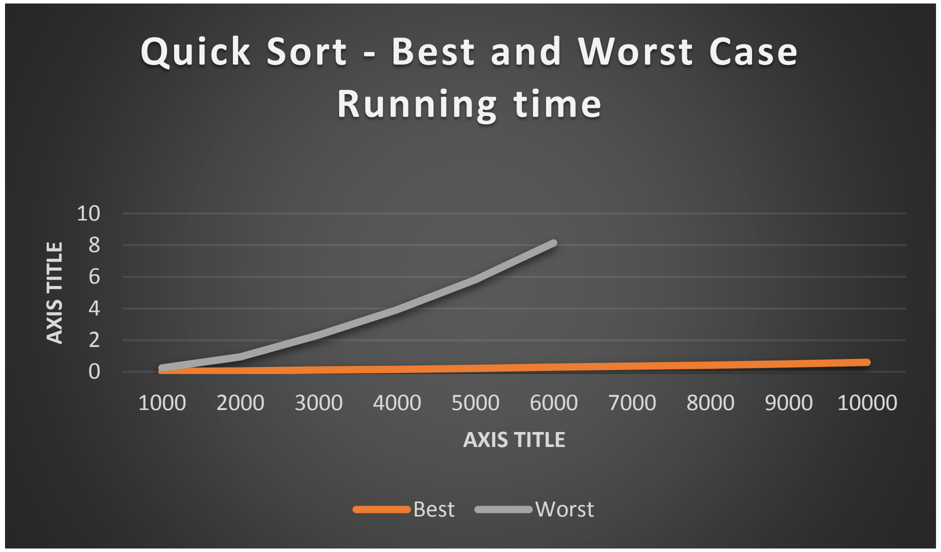


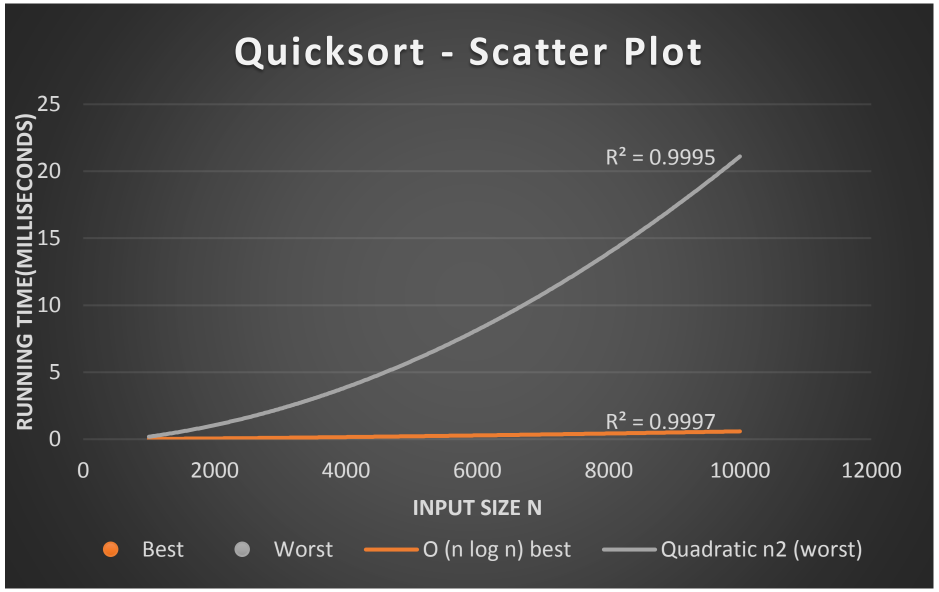
The adjacent scatter plot plots the results as Worst Case = n^2 and Best Case O (n log n) both are in line with the expected time complexity.

*During the running of the worst case, a stack overflow error occurred at values over 7000. To check the worst case for Quicksort the author selected a pivot point where there was 1 element on one side of the pivot. The other side of the pivot would contain 6,999. This increases the time.

Additionally, this method makes a recursive call to *generateQuickSortBestSampleStep* to fill the array. This step increases the amount of memory needed to perform the sorting, as it is not sorting in place anymore, causing a stack overflow error.

*For best case, an array was created that would guarantee equal parts on either side of the pivot for each iteration of the algorithm.





References

GeeksforGeeks. 2020. *Sorting Algorithms - Geeksforgeeks*. [online] Available at: <https://www.geeksforgeeks.org/sorting-algorithms/> [Accessed 14 May 2020].

WhatIs.com. 2020. *What Is Sorting Algorithm? - Definition From Whatis.Com*. [online] Available at: <https://whatis.techtarget.com/definition/sorting-algorithm> [Accessed 14 May 2020].

Heineman, G., Pollice, G. and Selkow, S., 2016. *Algorithms In A Nutshell*. 2nd ed.

Btechsmartclass.com. 2020. *Data Structures Tutorials - Performance Analysis With Examples*. [online] Available at: <http://www.btechsmartclass.com/data_structures/performance-analysis.html> [Accessed 14 May 2020].

Tutorialspoint.com. 2020. *DAA - Analysis Of Algorithms - Tutorialspoint*. [online] Available at: <https://www.tutorialspoint.com/design_and_analysis_of_algorithms/analysis_of_algorithms.htm> [Accessed 14 May 2020].

Jcsites.juniata.edu. 2020. *Algorithm Efficiency*. [online] Available at: <http://jcsites.juniata.edu/faculty/rhodes/cs2/ch12a.htm> [Accessed 14 May 2020].

GeeksforGeeks. 2020. *In-Place Algorithm - Geeksforgeeks*. [online] Available at: <https://www.geeksforgeeks.org/in-place-algorithm/> [Accessed 14 May 2020].

Javarevisited.blogspot.com. 2020. *Difference Between Comparison (Quicksort) And Non-Comparison (Counting Sort) Based Sorting Algorithms?*. [online] Available at: <https://javarevisited.blogspot.com/2017/02/difference-between-comparison-quicksort-and-non-comparison-counting-sort-algorithms.html#ixzz6LrgT4utt> [Accessed 14 May 2020].

FutureLearn. 2020. *An Introduction To Bubble Sorts*. [online] Available at: <https://www.futurelearn.com/courses/programming-102-think-like-a-computer-scientist/0/steps/53111> [Accessed 14 May 2020].

Sciencing. 2020. *The Advantages Of Heap Sort*. [online] Available at: <https://sciencing.com/the-advantages-of-heap-sort-12749895.html> [Accessed 14 May 2020].

Wordpress, 2012, Heapify [online] Available at: <https://johnderinger.wordpress.com/2012/12/28/heapify/ > [Accessed 12 May 2020]

Practice.geeksforgeeks.org. 2020. *Heap Advantage And Disadvantage | Practice | Geeksforgeeks*. [online] Available at: <https://practice.geeksforgeeks.org/problems/heap-advantage-and-disadvantage> [Accessed 14 May 2020].

Sciencing. 2020. *The Advantages & Disadvantages Of Sorting Algorithms*. [online] Available at: <https://sciencing.com/the-advantages-disadvantages-of-sorting-algorithms-12749529.html> [Accessed 14 May 2020].

GeeksforGeeks. 2020. *Quicksort - Geeksforgeeks*. [online] Available at: <https://www.geeksforgeeks.org/quick-sort/> [Accessed 14 May 2020].

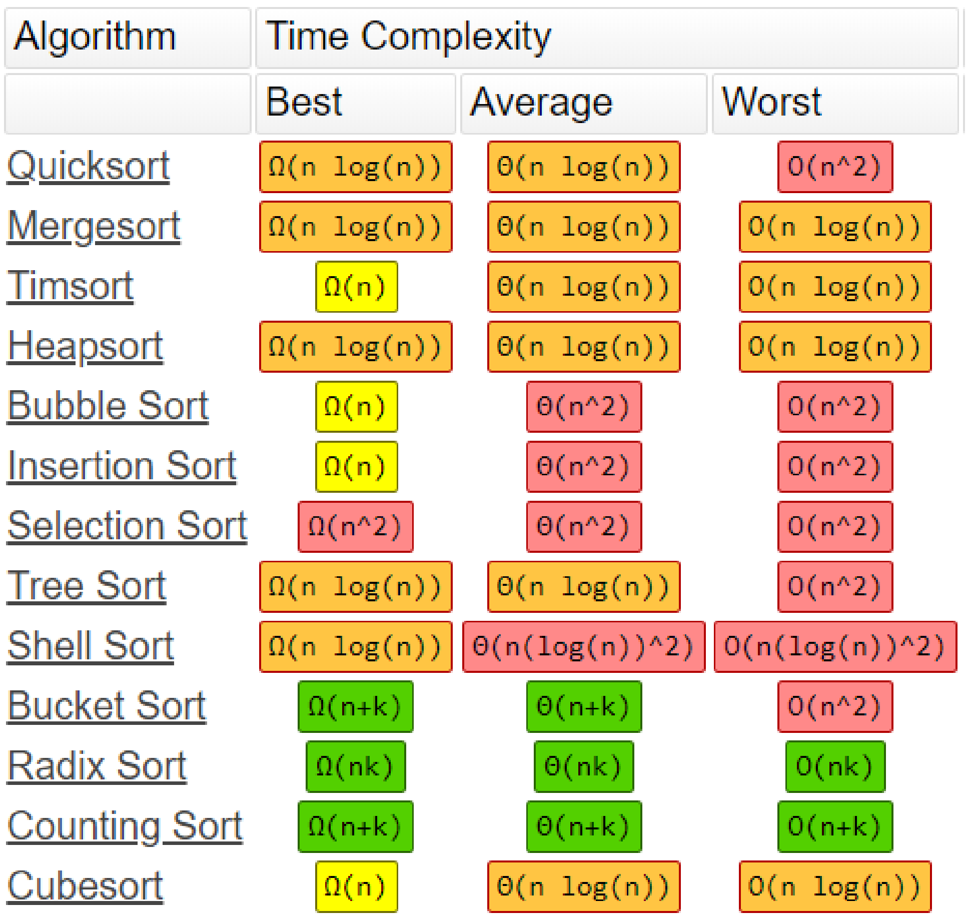
Khan Academy. 2020. *Analysis Of Quicksort (Article) | Quick Sort | Khan Academy*. [online] Available at: <https://www.khanacademy.org/computing/computer-science/algorithms/quick-sort/a/analysis-of-quicksort> [Accessed 14 May 2020].

2020, Array Sorting Algorithms <https://www.bigocheatsheet.com/>[Accessed 14 May 2020].

Wikipedia, 2020, Time Complexity Graph, <https://en.wikipedia.org/wiki/Time_complexity>[Accessed 14 May 2020].

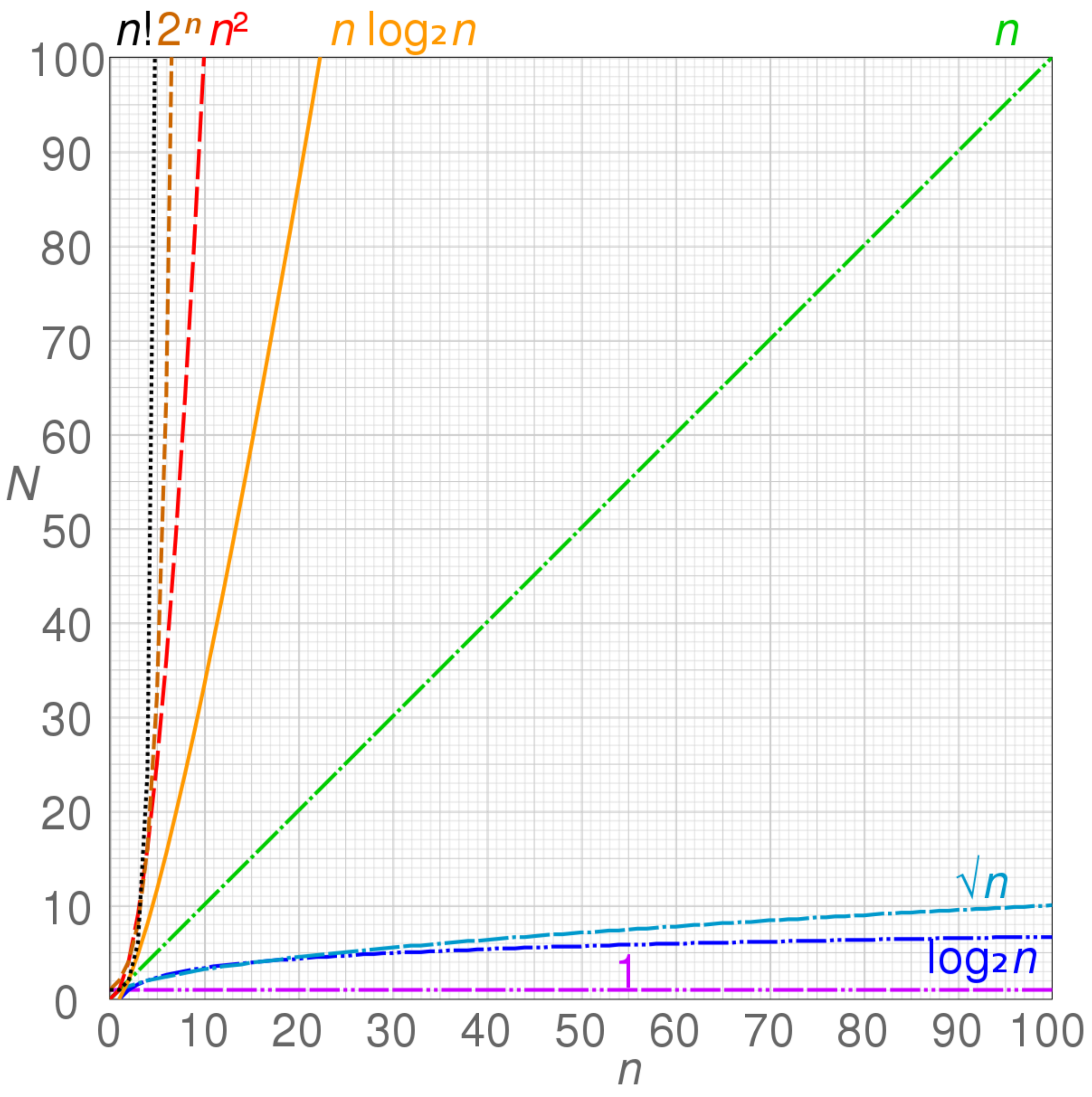
Geekviewpoint.com. 2020. *Bucket Sort*. [online] Available at: <http://www.geekviewpoint.com/java/sorting/bucketsort> [Accessed 14 May 2020].

Appendices



Algorithm	Time Complexity		
	Best	Average	Worst
Quicksort	Ω(n log(n))	Θ(n log(n))	O(n^2)
Mergesort	Ω(n log(n))	Θ(n log(n))	O(n log(n))
Timsort	Ω(n)	Θ(n log(n))	O(n log(n))
Heapsort	Ω(n log(n))	Θ(n log(n))	O(n log(n))
Bubble Sort	Ω(n)	Θ(n^2)	O(n^2)
Insertion Sort	Ω(n)	Θ(n^2)	O(n^2)
Selection Sort	Ω(n^2)	Θ(n^2)	O(n^2)
Tree Sort	Ω(n log(n))	Θ(n log(n))	O(n^2)
Shell Sort	Ω(n log(n))	Θ(n(log(n))^2)	O(n(log(n))^2)
Bucket Sort	Ω(n+k)	Θ(n+k)	O(n^2)
Radix Sort	Ω(nk)	Θ(nk)	O(nk)
Counting Sort	Ω(n+k)	Θ(n+k)	O(n+k)
Cubesort	Ω(n)	Θ(n log(n))	O(n log(n))

Appendices 1 Table depicting Algorithm Time Complexity BigOCheatSheet.com



Appendices 2 Graph Depicting Time Complexity, Wikipedia