|  |  |
| --- | --- |
|  | **2018** |
|  | IDM  K.P.I Shenesh Perera |

|  |
| --- |
| **[Data structures: Unit 19]** |
|  |

Contents

[Introduction 2](#_Toc532075796)

[Insertion vs. Merge Sort 3](#_Toc532075797)

[Selection vs. Quick Sort 4](#_Toc532075798)

[Bubble vs. Heap sort 5](#_Toc532075799)

[Conclusion 6](#_Toc532075800)

# Introduction

This report will consist of constructive comparison and contrasting between mainly 3 elementary sorting algorithms, namely; insertion, bubble and selection sort with 3 advanced sorting algorithms namely; merge, quick and heap sort. Each algorithm from the elementary categorization will be respectively compared with the mentioned complementary advanced algorithm. Their complexity and efficiency will be assessed using the Big (O) algorithm.

Figure 1.0, Shenesh Perera, 19/11/2018

# Insertion vs. Merge Sort

|  |  |  |  |
| --- | --- | --- | --- |
|  | Time Complexity | | |
| **Worst** | **Average** | **Best** |
| **Insertion Sort** | O(n2 ) | O(n) | O(n2 ) |
| **Merge Sort** | O(n\*log(n)) | O(n\*log(n)) | O(n\*log(n)) |

Insertion sort is a simple elementary algorithm that takes the incremental approach, as in it goes through each element till the sorting is complete. As such this takes quite the time, so insertion sort is only a recommended approach if the set of data is sufficiently of a small size. Regardless, this technique of sorting can be significantly improved by having the data set pre-sorted. This technique is called the binary insertion sort, in which the number of comparisons in the normal insertion sort is reduced by simply performing binary search.

Merge sort is an advanced algorithm commonly categorized as a “Divide & Conquer” algorithm. This is because the technique employs the procedure of splitting the data set by half, then sorting each half and finally merging the 2 halves together.

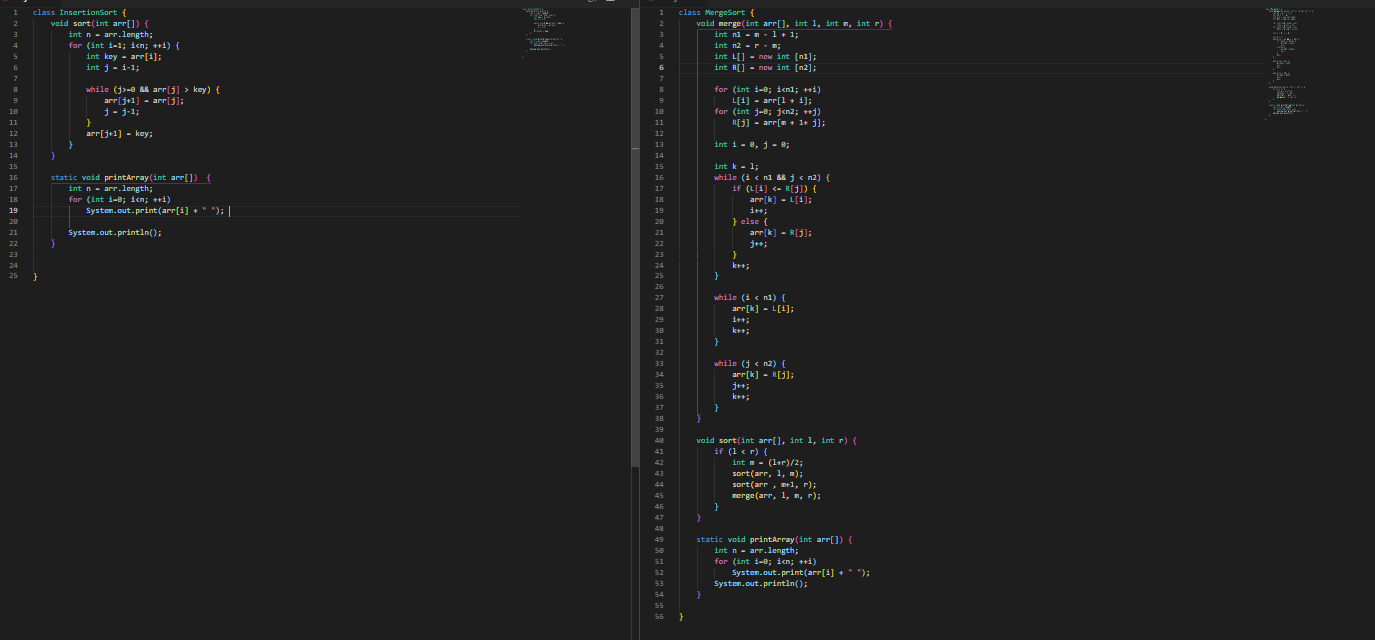
Comparing the 2 algorithms, per the stated big (o) factors, we can say that merge sort performs significantly better as O (n\*log(n)) > O(n2 ). However in terms of the complexity during implementation, insertion sort is desirable since merge sort involves code for partitioning then performing a merge in the end. Binary insertion sort however using linked lists performs at O (n) which is faster but this technique involves the implementation of binary search as well.

Figure 1.1, Shenesh Perera, 19/11/2018

Figure 1.1, right; MergeSort & left; InsertionSort

# Selection vs. Quick Sort

|  |  |  |  |
| --- | --- | --- | --- |
|  | Time Complexity | | |
| **Worst** | **Average** | **Best** |
| **Selection Sort** | O(n2) | O(n2) | O(n2) |
| **Quick Sort** | O(n2) | O(n\*log(n)) | O(n\*log(n)) |

Selection sort is an elementary sorting algorithm that separates the data set into 2 sub data sets, while one being the sorted and the rest that is unsorted, the algorithm repeatedly finds the minimum value and puts it in the beginning of the sorted array.

Quick sort is yet another divide and conquer algorithm, just like merge sort it also partitions the data set before performing any action on it. The first element of the data set is picked, then place it in its accurate position in the data set, then all elements smaller than that value are put before it while the larger elements go after it. The common opinion is that quick sort is an unstable algorithm, as such it is usually recommended to modify it.

Similar to the previous comparison, the selection sort is a simpler algorithm to implement but is not as efficient as quick sort; however a barebones quick sort implementation is considered unstable therefore in order to make the algorithm stable you should treat indexes as the comparison parameter. This adds to the inconvenience during implementation.

That said, if the data set involved is a linked list, then it is highly recommended to move away from a quick sort and do a merge sort instead, as linked lists are sequential and merge sort does it’s sorting sequentially as well while quick sort requires random access which is better suited if the data set is an array.

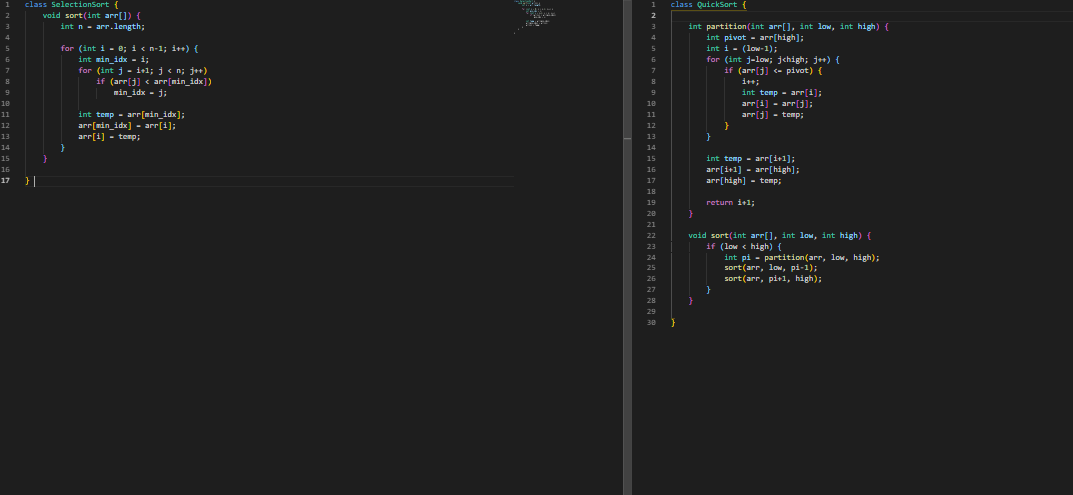


Figure 1.2, Shenesh Perera, 19/11/2018

Figure 1.2, left; SelectionSort & right; QuickSort

# Bubble vs. Heap sort

|  |  |  |  |
| --- | --- | --- | --- |
|  | Time Complexity | | |
| **Worst** | **Average** | **Best** |
| **Bubble Sort** | O(n2) | O(n2) | O(n) |
| **Heap Sort** | O(n\*log(n)) | O(n\*log(n)) | O(n\*log(n)) |

Bubble sort is the simplest sorting algorithm available, its procedure is simple; to swap the elements till the whole data set is in the correct order. This algorithm is the introductory to sorting for new programmers. However it is used widely in the gaming industry because the algorithm can detect subtle changes like the misplaced element with as less complexity possible.

Heap sort is an advanced sorting algorithm that revolves around the binary heap data structure, it employs a technique that of similar to selection sort, the maximum element is found and it’s placed at the end, then repeat for the rest of the elements. Heap sort is yet another unstable algorithm, to fix this issue the previously mentioned method of considering index as the comparison parameter is used.

Bubble sort is definitely not meant for particularly large data sets, since it goes through the data set sequentially multiple times for each swap, which takes a lot of time. However though bubble sort is an algorithm that has flexibility, in the game industry the algorithm is often modified with the help of a queue data structure to boost its efficiency.

Heap sort cannot be properly compared against bubble sort as sheer complexity of the implementation of a heap sort is enough to conclude that bubble sort shouldn’t be compared to it. However though, in terms of usability heap sort is effective for extremely large data sets, while its instability is inconvenient there are a plenty of workarounds.

Unlike previous advanced algorithms, heap sort does not do any partitioning.

Figure 1.2, Shenesh Perera, 19/11/2018

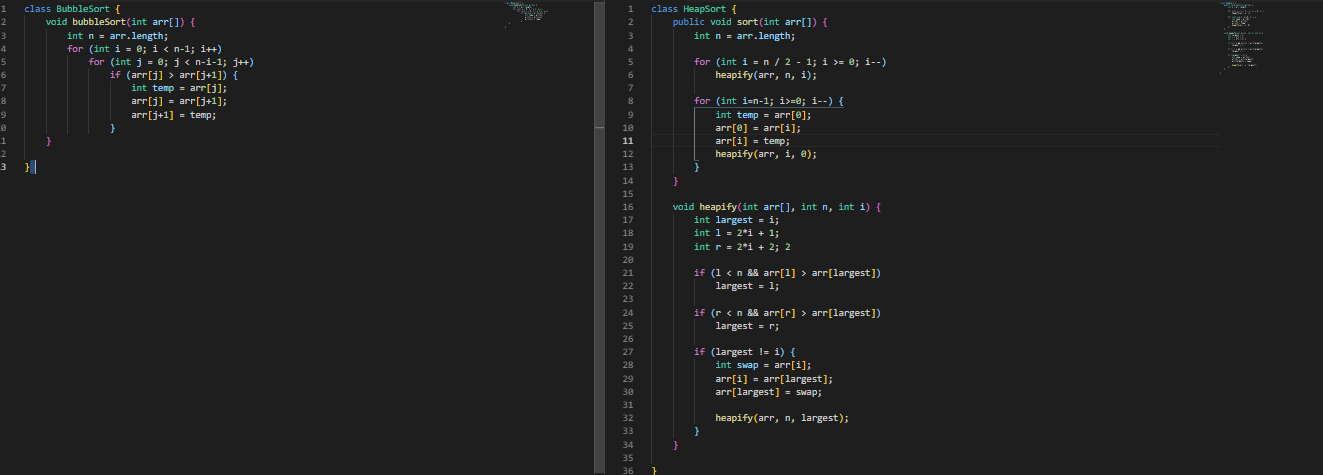


Figure 1.2, left; BubbleSort & right; HeapSort

# Conclusion

From among the 6 discussed sorting algorithms, from personal experience, big (o) evidence and research the merge sort algorithm implemented on a linked list data structure has a wide use case in the industry. However in terms of how efficient the algorithm is, then heap sort comes a ring above but as the implementation requires the knowledge of binary heap data structure it is practically inconvenient.

Below is a composition of my benchmark tests on a 2.4GHz intel i3 on the 6 algorithms;

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Time | Size of array | | | |
| **1,000** | **5,000** | **10,000** | **50,000** |
| Insertion | 0.002 | 0.400 | 1.023 | 5.928 |
| Selection | 0.005 | 0.560 | 2.052 | 7.833 |
| Bubble | 0.100 | 0.890 | 5.984 | 12.629 |
| Merge | 0.003 | 0.039 | 0.972 | 3.991 |
| Quick | 0.034 | 0.590 | 1.005 | 8.385 |
| Heap | 0.015 | 0.088 | 2.592 | 7.221 |

With the above results, performance of all the algorithms can be properly captured. With all the points that have been discussed plus this performance description, then it is possible to come to a conclusion depending on the situation which sorting algorithm must be used.