**DAA PROJECT REPORT**

**IMPLEMENTATION AND COMPARISON BETWEEN DIFFERENT SORTING ALGORITHMS**

**HARSH SHAH**

**STUDENT ID: 1002057387**

**ABOUT THE PROJECT**

In this project, I have used Python language to implement and compare various sorting algorithms.

I have made a main menu driven program, from which the user can select an option to perform a particular type of sorting or can also compare the run time complexity.

Different functions are defined for different sorting algorithms. For instance, bubble\_sort() is used to define Bubble Sort Algorithm.

**INTRODUCTION**

In this project, I have studied about various sorting algorithms used to sort an array which is an input given by the user. I implemented and compared their time complexity and how they behaved with different data sizes. The following sorting algorithms were used in this project :

* Mergesort
* Heapsort
* Quicksort (Regular quick sort and quick sort using 3 medians)
* Insertion sort
* Selection sort
* Bubble sort

Now let’s talk about what is Time Complexity and then I am going to implement and compare the above sorting algorithms.

**TIME COMPLEXITY**

Time complexity of an algorithm quantifies the amount of time taken by an algorithm to run as a function of the length of the input. The algorithm that performs a particular task in the smallest amount of time is considered as the most efficient one in terms of the time complexity.

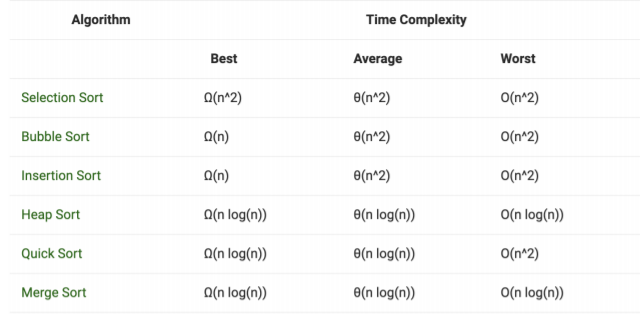
There are 3 cases for determining time complexity of an Algorithm:

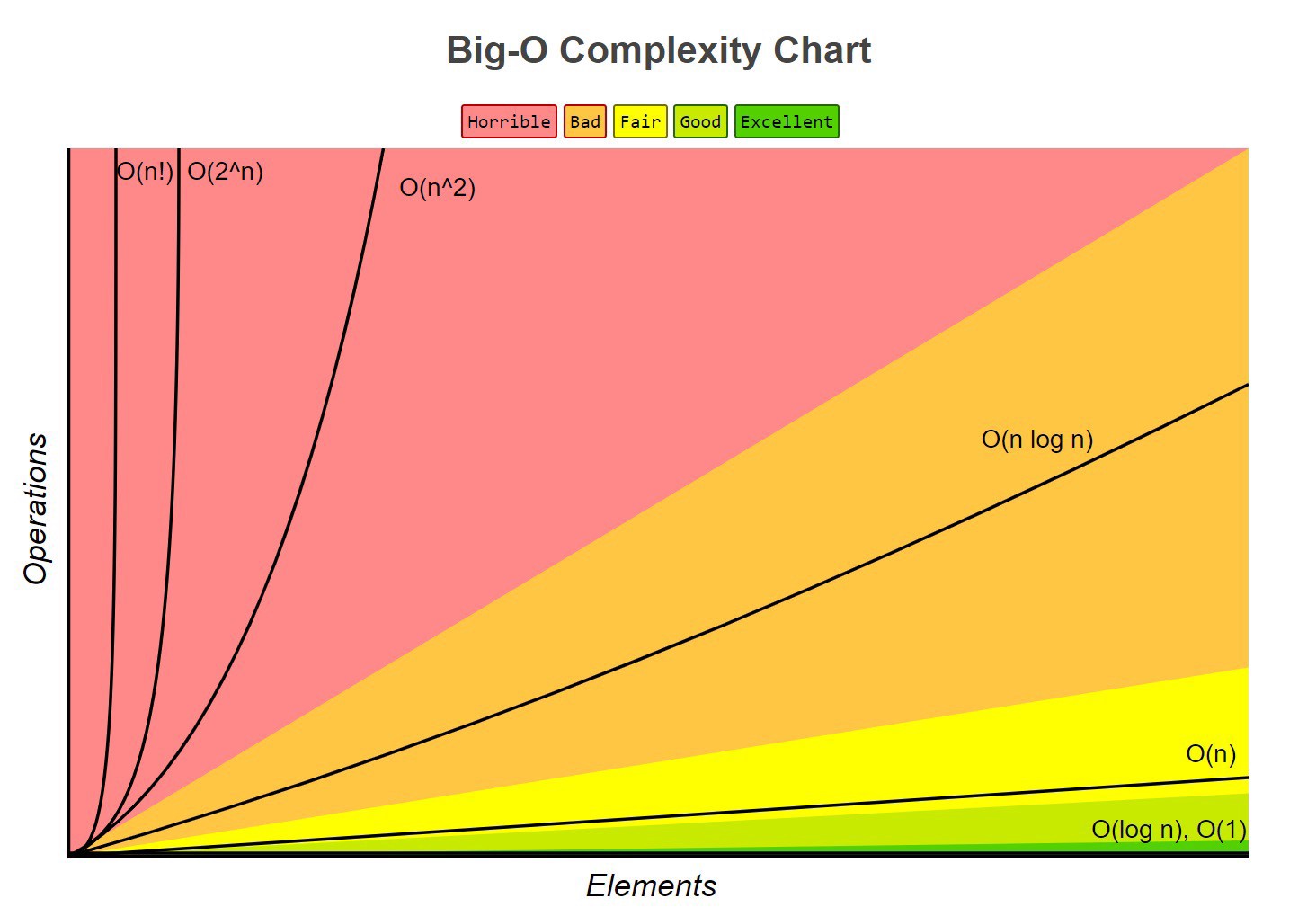
1. Best Case

2. Average Case

3. Worst case

* Best Case : Function defined by the minimum number of steps taken on any instance of size n.
* Average Case : Function defined by the average number of steps taken on any instance of size n.
* Worst Case : Function defined by the maximum number of steps taken on any instance of size n. It represents the curve passing through the highest point of each column.





**SORTING ALGORITHMS**

1. **MERGESORT**

Merge sort is a divide-and-conquer algorithm based on the approach of breaking down a list into several sub-lists until each sublist consists of a single element and then merging those sublists in a way that results into a sorted list of elements. Time complexity of Merge Sort in all 3 cases is the same O(N Log(N)) because merge sort divides the array into two halves and take linear time to merge two halves to get a sorted list of elements.

**Main Components:**

In the merge sort algorithm, there is only one method that accepts the array as the input and there are 3 while loops and if condition and the left list and right list is used to split the array into two parts.

**Main Data Structure:**

Merge Sort is mostly performed on an array.

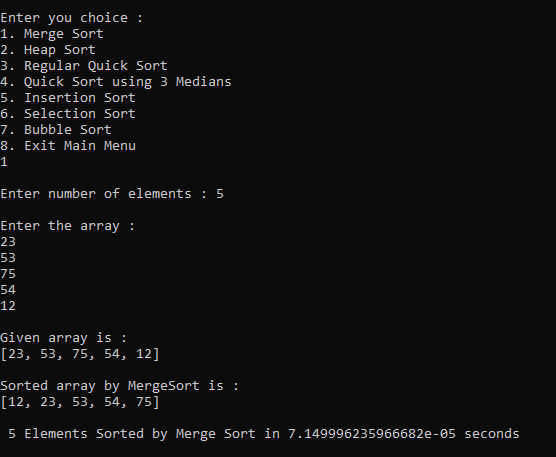
**How to Improve Runtime:**

The tiled merge sort algorithm stops partitioning subarrays when subarrays of size S are reached, where S is the number of data items fitting into a CPU's. Time complexity is O(NLog(N)),Space complexity is O(N).

**How merge sort is Better:**

If data is in linked list form, it will work better.

**Example**: It took 7.149996235966682e-05 seconds to sort 5 elements using merge sort.



1. **HEAPSORT**

Heap sort is a comparison based sorting technique. Heaps can be used in sorting an array. In max-heaps, maximum element will always be at the root. Heap Sort uses this property of heap to sort the array given by the user. Overall time complexity of Heap Sort is O(NLog(N)).

**Main Components:**

Heap sort consists of only two functions, one is heap\_sort and heapify function. The heapify function performs the rearrangement and moving the elements to the root place and the comparison and swapping is done in the heap\_sort function.

**Main Data Structure:**

Heap Sort is mostly performed on an array.

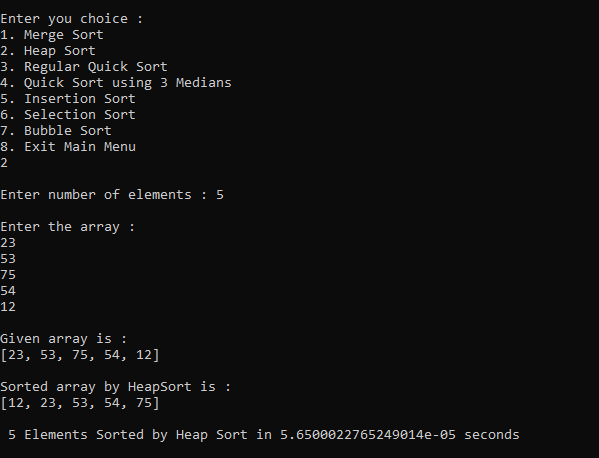
**How to improve Run time:**

In order to optimize the performance of heap sort we are proposing the optimized heap sort technique (OHS) by using Two-swap method. The Two-swap method sorts two-elements at a time for each heap construction, and it reduces 30-50% of the time complexity of heap sort in the construction of heap as well as in sorting of array elements. Time complexity is O(NLog(N)),Space complexity is O(1).

**How Heap sort is Better:**

It does not need extra storage. That makes it good for situations where you have a very large array.

**Example:** It took 5.6500022765249014e-05 seconds to sort 5 elements using heap sort.



1. **QUICKSORT**

Quick sort is based on the divide-and-conquer approach which works on the idea of choosing one element as a pivot element and partitioning the array around it such that the left side of the pivot element contains all the elements that are less than the pivot element and the right side contains all elements greater than the pivot element.

**Main Components:**

In the quick sort algorithm, there are two functions quicksort and partition function. partition function performs the partition of the array elements. And the comparison is done in the quicksort function and this function itself will call the partition function to perform the operation.

**Main Data Structure:**

Quick Sort is mostly performed on an array.

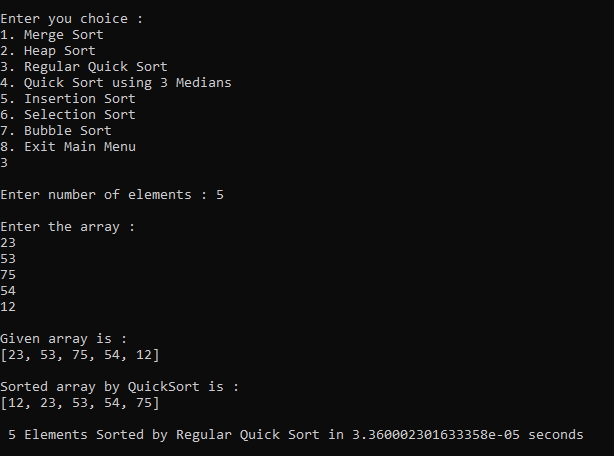
**How to Improve Runtime:**

Median-of-three partitioning. A second easy way to improvethe performance of quicksortis to use the median of a small sample of items taken from the array as the partitioning item. Doing so will give a slightly better partition, but at the cost of computing the median. Time complexity is O(NLog(N)), Space complexity is O(1).

**How Simple Quick sort is Better:**

There are certain reasons due to which quicksort is better especially in case of arrays, quicksort requires little space and exhibits good cache locality. Quick sort is an in-place sorting algorithm.

**Example:** It took 3.360002301633358e-05 seconds to sort 5 elements using quick sort.



1. **QUICK SORT USING 3 MEDIANS**

Quicksort with median-of-three partitioning works nearly the same as the normal quicksort does, with the only difference being how the pivot item is selected. In normal quicksort, generally the first element is selected as the pivot item.

In quicksort with median-of-three partitioning, the pivot item is selected as the median between the first element, the last element, and the middle element. In the cases of already sorted lists, this should take the middle element as the pivot and hence reducing the inefficiency found in normal quicksort.

Worst case: O(N2), Average Case: O(N log(N)), Best Case: O(N) time.

**Main Components:**

Quicksort with 3 median consists of 4 functions as quick\_sort\_3median, median\_of\_three, quicksort\_3median, and partition. In median\_of\_three, there is if and else condition. In partition, there are two while loops and also if and else condition.

**Main Data Structure:**

Quick Sort with 3 medians is mostly performed on an array.

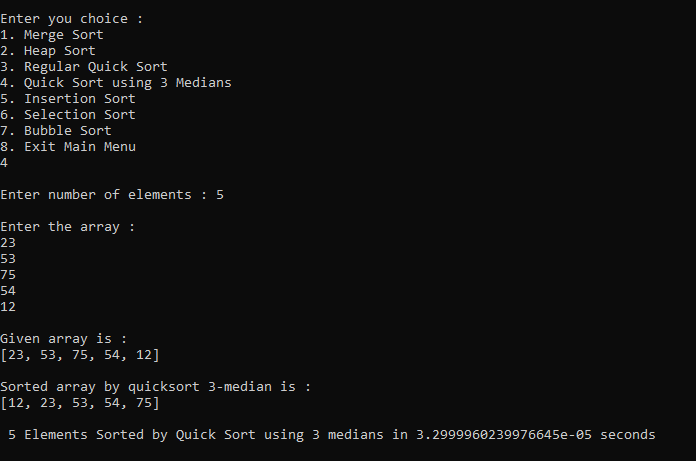
**How to Improve Runtime:**

This is already an improved version of quick sort. Time complexity is O(NLog(N)), Space complexity is O(1).

**How quick sort with 3 medians is Better:**

There are certain reasons due to which quicksort is better especially in case of arrays, quicksort requires little space and exhibits good cache locality. Quick sort is an in-place sorting algorithm. Quick sort with 3 medians works better than simple quick sort.

**Example:** It took 3.2999960239976645e-05 seconds to sort 5 elements using quick sort with 3 median.



1. **INSERTION SORT**

Insertion sort is based on the approach that one element from the input elements given by the user is consumed in each iteration to find its correct position i.e., the position to which it belongs in a sorted array. It is an “in-place” sorting algorithm.

**Main Components:**

Insertion sort consists of only one function which accepts input as an entire array of elements. After taking inputs as an array of elements there is one for loop which will traverse to the entire array and there is one while loop.

**Main Data Structure:**

Insertion Sort is mostly performed on an array.

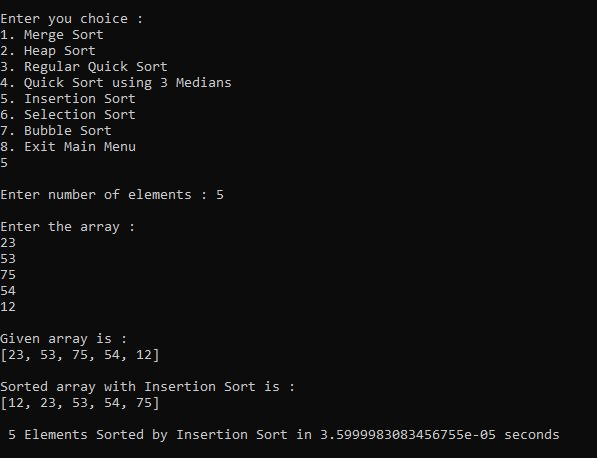
**How to improve Run time:**

The performance can be improved by doing binary searching for the insertion points. By removing the key comparison from the loop that shifts elements up.

The worst case time complexity of insertion sort is O(N2).

The space complexity of insertion sort is O(1).

**Example**: It took 3.5999983083456755e-05 seconds to sort 5 elements using insertion sort.



1. **SELECTION SORT**

The Selection sort algorithm is based on finding the minimum or maximum element in an unsorted array and then putting it in its correct position in a sorted array. The list is divided into two parts, the sorted part at the left end and the unsorted part at the right end. This sorting algorithm is an in-place comparison-based algorithm.

Selection sort has runtime complexity of O(N2).

**Main Components:**

Selection sort consists of only one function which accepts entire array of elements as input first it will calculate the length of array it consists of two nested for loops which will traverse through entire array set.

**Main Data Structure:**

Selection Sort is mostly performed on an array.

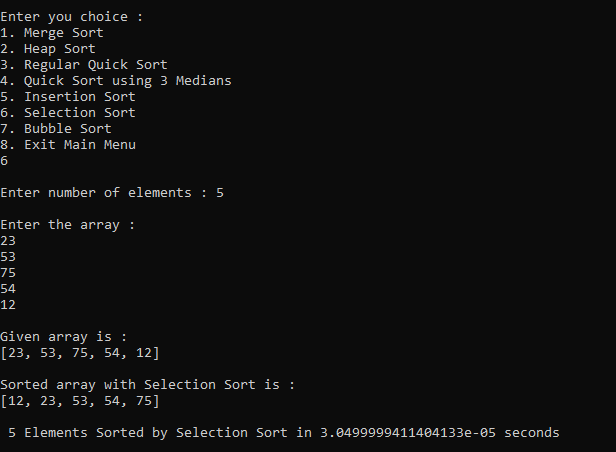
**How to improve Run time:**

Time complexity is O(N2). Space complexity is O(1).Only way improve runtime is If the imported array is already in sorted form.

**How Selection sort is Better:**

Selection sort can be good at checking if everything is already sorted. It is also good to use when memory space is limited. The good thing about selection sort is it never makes more than O(N) swaps and can be useful when memory write is a costly operation.

**Example**: It took 3.0499999411404133e-05 seconds to sort 5 elements using selection sort.



**7.BUBBLE SORT**

Bubble Sort is a sorting algorithm which compares two adjacent elements and swaps them if they are not in the right order. To sort the entire array, the array is traversed n-1 time (array having n elements). These are called passes, In the first pass the largest element moves to the last position (sorting in ascending order).

**Main Components:**

bubble sort consists of only one function which accepts entire array of elements as input first it will calculate the length of array it consists of two nested for loops which will traverse through entire array set and while traversing it will compare two adjacent elements and evaluate which value is greater.

**Main Data Structure:**

Bubble sort commonly used array data structure.

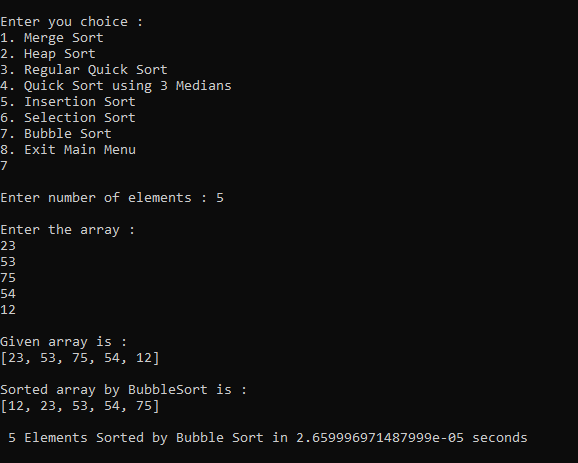
**How to improve Run time:**

Its performance can be improved by doing bidirectional (cocktail sort or shaker sort) bubble sort. As a result of which the average number of comparisons will be slightly less compared to the simple bubble sort. Time complexity of bubble sort is O(N2). Space complexity is O(1).

**How Bubble sort is Better:**

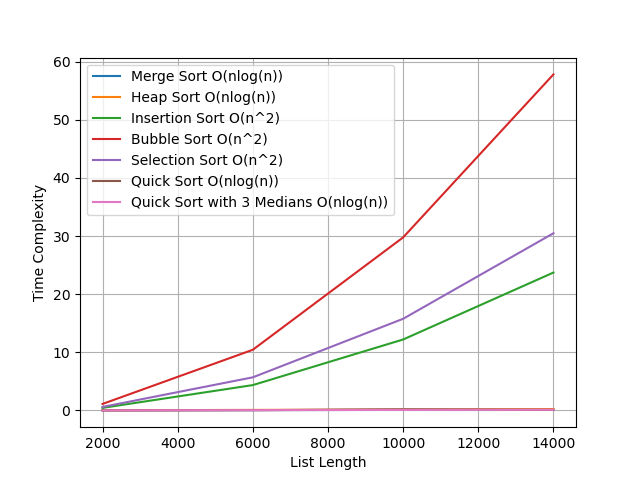
When data set is small and when the data is in all sorted form it works better.

**Example:** It took 2.659996971487999e-05 seconds to sort 5 elements using bubble sort.



**COMPARISONS**

Here are some comparisons with the above mentioned algorithms with different data sizes to measure each algorithm’s performance relatively to others.



1. Data Size = 2000 elements

When accounted for smaller datasets, there is a slight difference between the runtime between merge and quick sort which are 0.019 s and 0.014 s respectively. Followed by heap sort which took 0.035 s, Quick Sort using 3 medians taking 0.017s, Selection sort taking 0.61 s, Insertion Sort taking 0.44 s and lastly Bubble sort taking the highest amount of time, i.e., 1.12 s.

1. Data Size = 6000 elements

The merge sort is slightly faster than the heap sort. Merge Sort took 0.06 s while heap sort took 0.10 s. Quick sort took 0.051 s which is the fastest. While Bubble Sort took 10.44 s, Quick sort using 3 medians took 0.058 s, Insertion sort took 4.36 s and Selection Sort took 5.70 s respectively.

1. Data Size = 10000 elements

As the size of the elements increases, it becomes conspicuous that quick sort is the most efficient among all sorting algorithms. While Bubble sort being the slowest taking 29.7 s.

1. Data Size = 14000 elements

As expected with the growing size of data, Bubble Sort can be rendered as “Cumbersome and Useless” while Selection Sort and Insertion sort performed slower than expectations as well (30.47s and 23.72 s respectively). While Quick Sort, Quick Sort using 3 Medians, Heap Sort and Merge Sort were best performing algorithms for larger datasets.

**REFERENCES**

Professor Negin Fraidouni lecture notes/slides

wikipedia.com

tutorialspoint.com

javatpoint.com

geeksforgeeks.com

researchgate.net