***CSE-5311-004-DSGN & ANLY ALGORITHMS***

***Project***

**Definition:**

Implement and compare the following sorting algorithms:● Merge sort● Heap sort● Quick sort (Regular quicksort & quicksort using 3 medians)● Insertion sort● Selection sort

● Bubble sort

**INDEX**

|  |  |
| --- | --- |
| **Items** | **Page No.** |
| Theory | 2 |
| Implementation Details | 5 |
| Experimental Result | 6 |
| Analysis | 9 |
| Conclusion | 10 |

**THEORY**

An algorithm is a limited arrangement or rationale of all-around characterized PC implementable guidelines. Algorithms are unambiguous determinations for performing computations, sorting, information handling, thinking and numerous different tasks.

Sorting is any process of arranging items systematically, it also means to arrange the given data into some order by providing some criteria or arranging them into some groups of some category. Sorting is very important in coding because run time of any program depends on what type of data is provided. For example, if we need to find and remove duplicity in data of large size. Here we have two scenarios, first if data is sorted, we can do all operations directly and in second case we have unsorted data, in this case we need to first sort whole data and then we can do operation. In both the case runtime of first one will be very low as compared to the runtime in the second case, also difference of both run time will be significant.

From the above example we come to very important topic which is Time Complexity. Time complexity is the computational multifaceted nature that describes the measure of time it takes to run an algorithm. Another significant theme in algorithm is Space Complexity. Space complexity nature is a proportion of the measure of working stockpiling an algorithm needs. That implies how much memory, in the worst-case scenario, is required anytime in the algorithm. Likewise, with time complexity, we're for the most part worried about how the space needs develop, in Big-O terms, as the size N of the information issue develops.

Below are a few sorting algorithms which we will discussed in this project:

1. Bubble Sort,
2. Insertion Sort,
3. Selection sort,
4. Merge Sort,
5. Heap Sort,
6. Quick Sort (Regular quicksort and quicksort using 3 Median).

These algorithms are the most commonly used. Let’s describe all the above algorithms briefly:

1. **Bubble Sort:**

This algorithm is comparison-based algorithm. It is the simplest sorting algorithm that works by repeatedly swapping the adjacent elements if they are in wrong order.

(Ref: <https://www.geeksforgeeks.org/bubble-sort/> )

1. **Insertion Sort:**

This algorithm is also comparison-based algorithm but it builds final sorted array one element at a time. This sorting algorithm works the way we sort playing cards in our hands. On each iteration, it finds the position of one element in array.

(Ref: <https://www.geeksforgeeks.org/insertion-sort/> )

1. **Selection Sort:**

This sorting algorithm is an in-place comparison-based algorithm where the list is divided into two sub parts. At the start the sorted part is empty list and unsorted part is the entire input list. In every iteration the minimum element from the unsorted list is picked and moved to the sorted list.

(Ref: <https://www.geeksforgeeks.org/selection-sort/> )

1. **Merge Sort:**

This algorithm is Divide and Conquer based algorithm. It means that it will divide the arrays in two parts and sort the elements of both parts in such manner that after merging both parts, resulting array is also in sequence.

(Ref: <https://www.geeksforgeeks.org/merge-sort/> )

1. **Heap sort:**

Heap sort is a comparison-based sorting technique based on Binary Heap data structure. Binary heap means it is complete binary tree where items are stored in a special order such that value in a parent node is greater (or smaller) than the values in its two children nodes. The former is called as max heap and the latter is called min heap. First of all make max heap from the given data. At this point, largest item will be stored at root. Replace the root with last item of heap by reducing the size. Heapify the root and repeat this until you get heap size greater than 1.

(Ref: <https://www.geeksforgeeks.org/heap-sort/> )

1. **Quick Sort (Regular quicksort and quicksort using 3 Median):**

Like Merge sort, Quick sort is also a Divide and Conquer algorithm. It picks an element as pivot and segments the given array around the picked pivot. There are a wide range of forms of quickSort that pick pivot in various manners.

* Pick first element as pivot.
* Pick last element as pivot.
* Pick random element as pivot.
* Pick median as pivot.

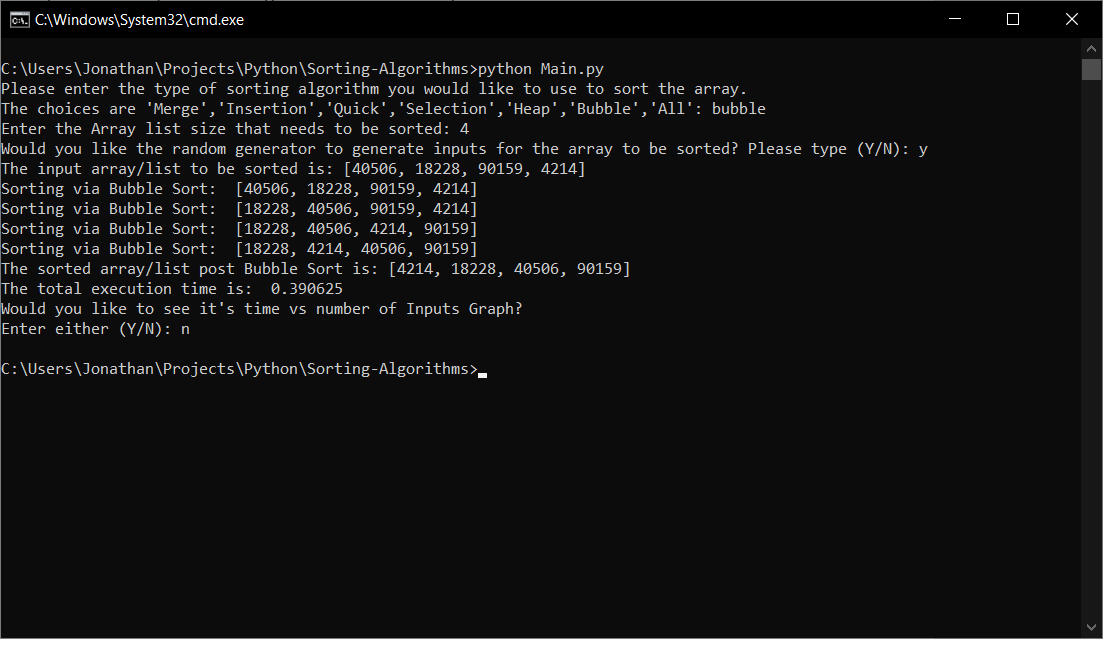
The key process in this algorithm is partition of array. Target of partitions is, given an array and an element x of array as pivot, put x at its correct position in sorted array and put all smaller elements (smaller than x) before x, and put all greater elements (greater than x) after x.

In Quick Sort using 3 Median. The pivot item is selected as the median between the first element, the last element and the middle element.

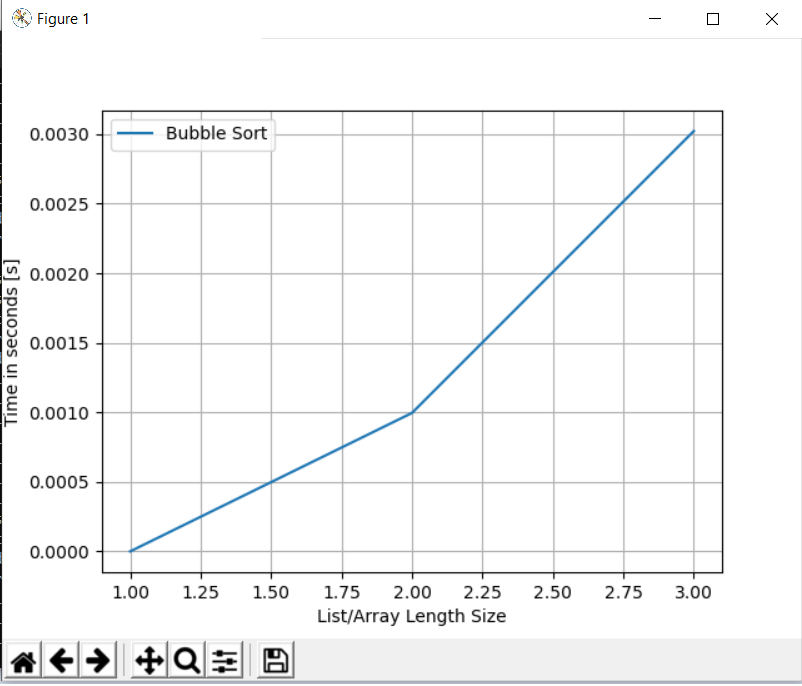
(Ref: <https://www.geeksforgeeks.org/heap-sort/> )

**IMPLEMENTATION DETAIL**

There is a main file provided which when executed will ask for the type of sorting algorithm the user wants to use. If the user selects ‘Quick Sort’ then it will again ask form 2 options i.e. Regular QuickSort or QuickSort using 3 median. After the preference is selected it will ask the user to provide the size of the list the user wants to sort. Here we have added a functionality of generating random numbers as well so if the user don’t what to enter list manually he/she can use the number generated randomly. Here now user can see the input list which is unsorted, a sorted list along with the time taken to sort the list and the time and space complexities of the particular sort algorithm.

****

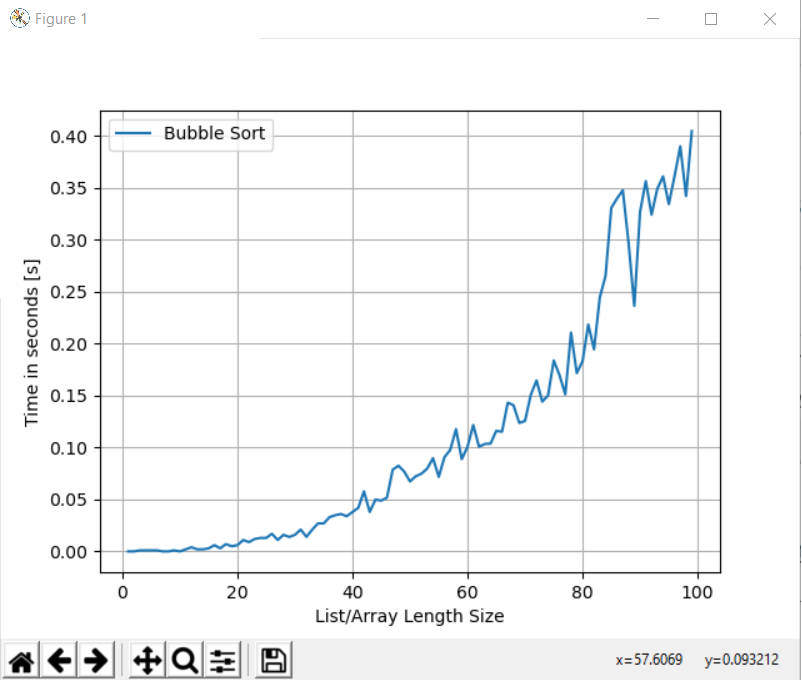
After the execution time is shown we can see that it asks the user ‘Would you like to see it’s time Vs number of input Graph?’. This functionality is only if we are generating numbers randomly. It the user says ‘y’ then a graph would be visible which will show the same for the number of inputs given by the user.



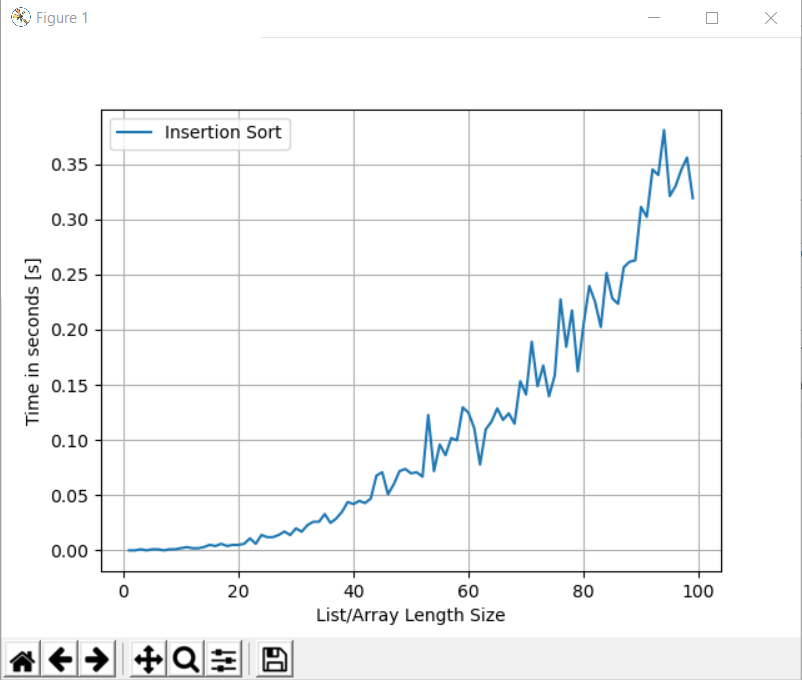
**Experimental Results**

The graph for each algorithm for input size [1 – 100] with randomly generated inputs is shown below:

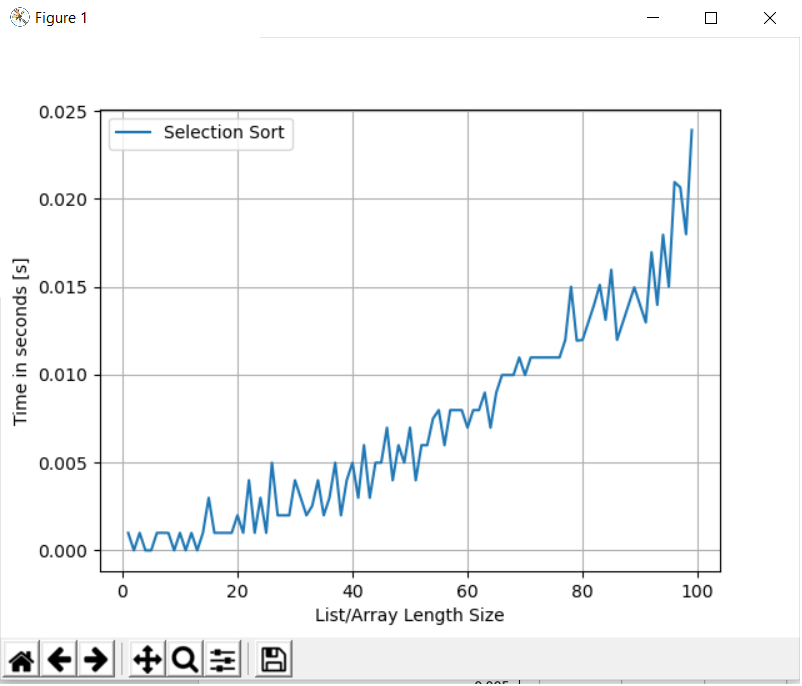
* BUBBLE SORT:



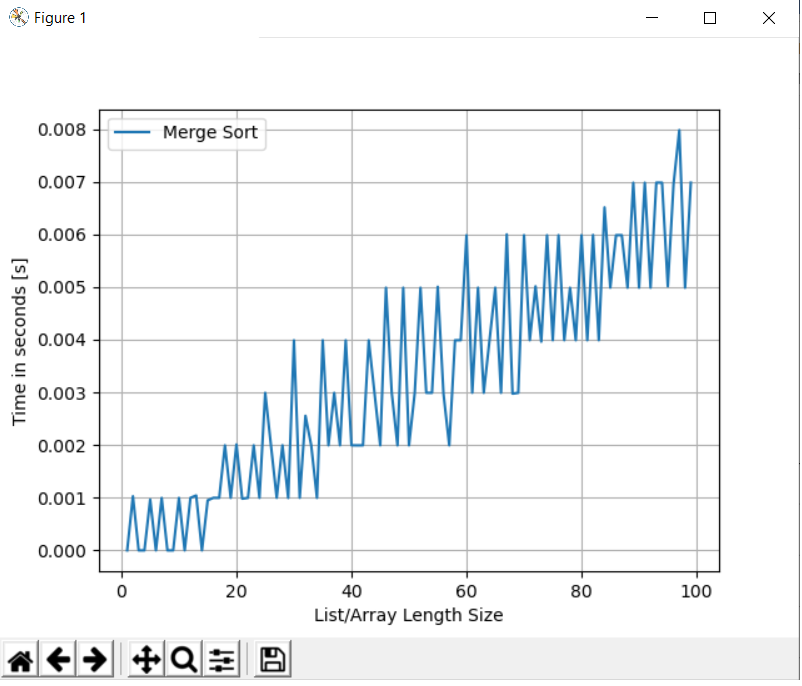
* INSERTION SORT:



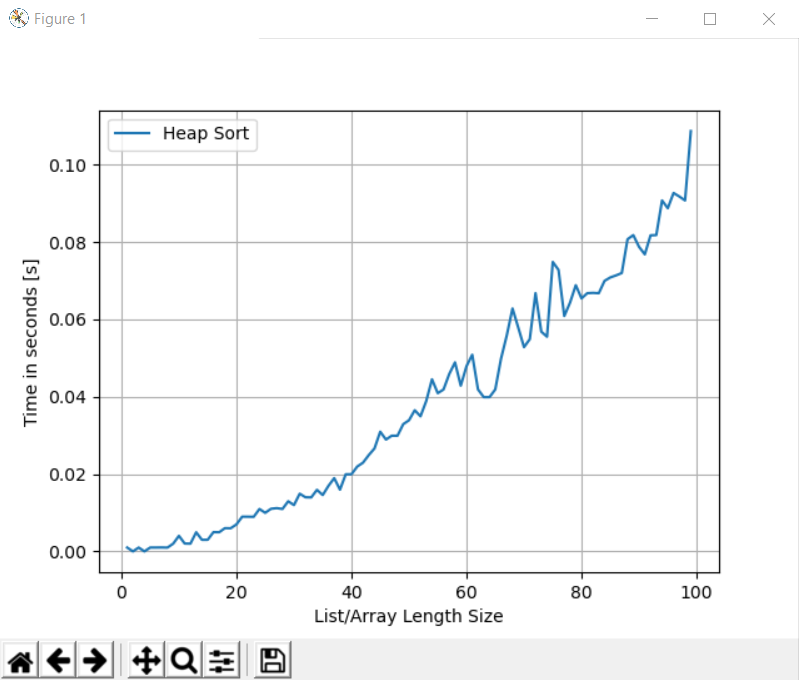
* SELECTION SORT:



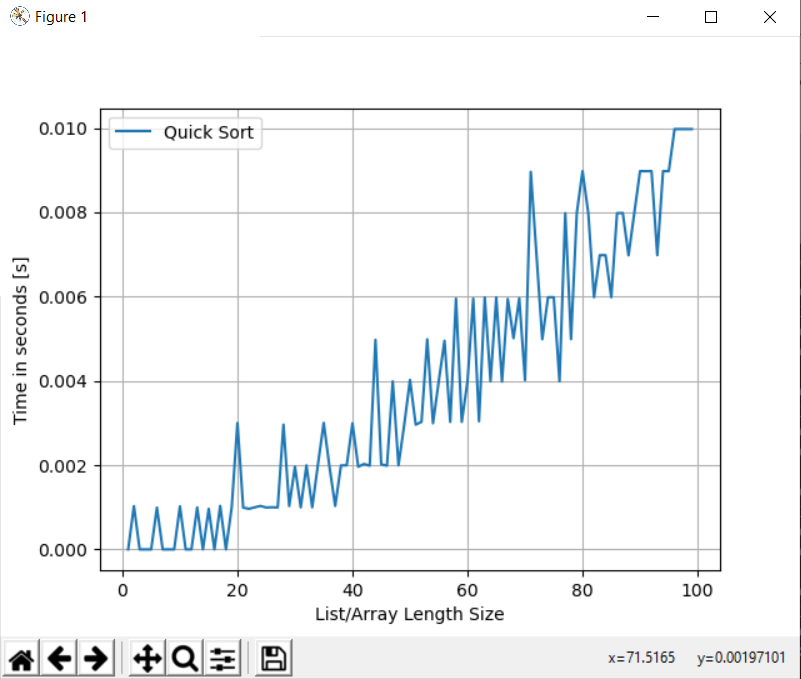
* MERGE SORT:

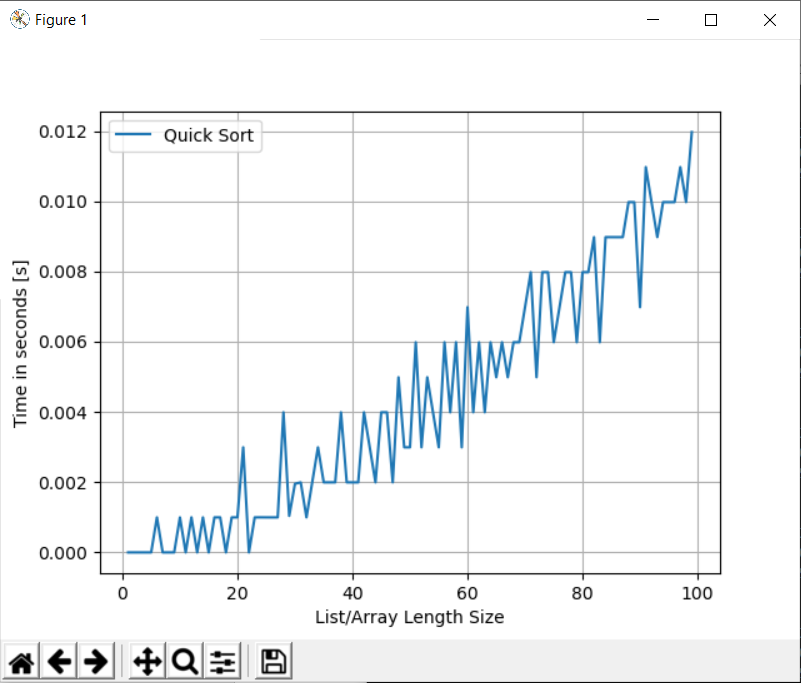


* HEAP SORT:



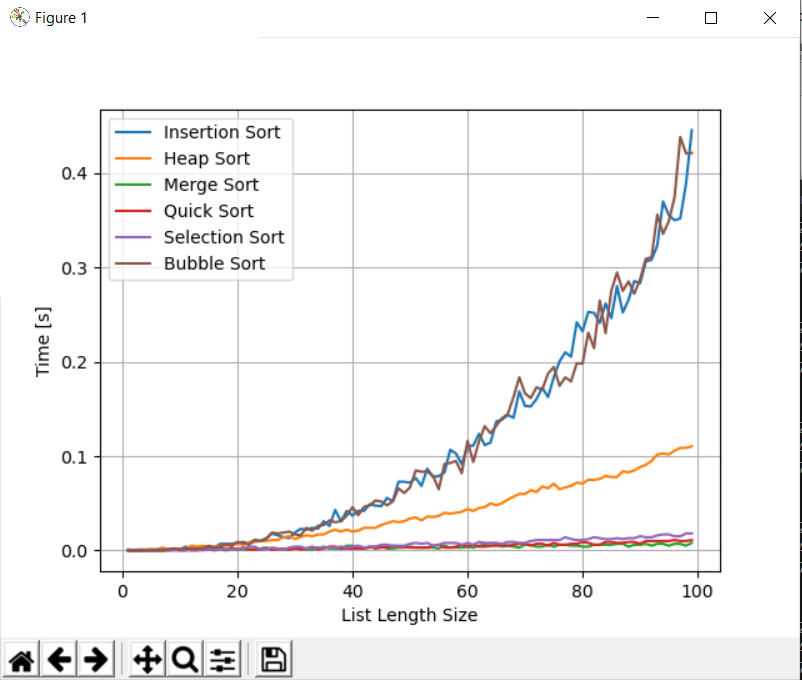
* QUICK SORT (REGULAR QUICKSORT AND QUICKSORT USING MEDIAN OF 3)





**Analysis**

Let us now compare all the algorithms in a single graph.



Above result shows that as number of input increases, execution time also increases. Quick sort gives best result in all above mentioned algorithms.

Time Complexity of Algorithms:

|  |  |
| --- | --- |
| Bubble Sort | O(n^2) |
| Insertion Sort | O(n^2) |
| Selection Sort | O(n^2) |
| Heap Sort | O(n\*logn) |
| Merge Sort | O(n\*logn) |
| Quick Sort | O(n\*logn) |

**CONCLUSION**

From the above information, we can conclude that as number of input size builds, time required to sort likewise increases. From over seven calculations, Quick sort gives best execution while bubble sort gives the most noticeably awful exhibition. Time required to sort the list is exponentially increments for bubble sort, insertion sort and selection sort while time increments step by step for heap sort, merge sort and quick sort.