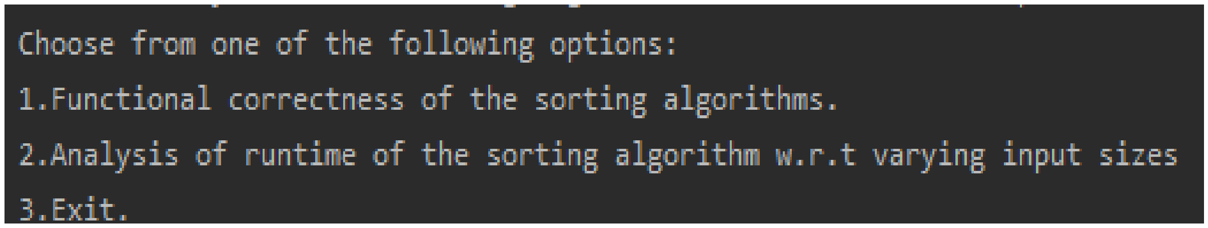
Implement and compare the following sorting algorithm :

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

– **Megha Vijendra**

The project interface provides three options to choose from and each option performs the following tasks:

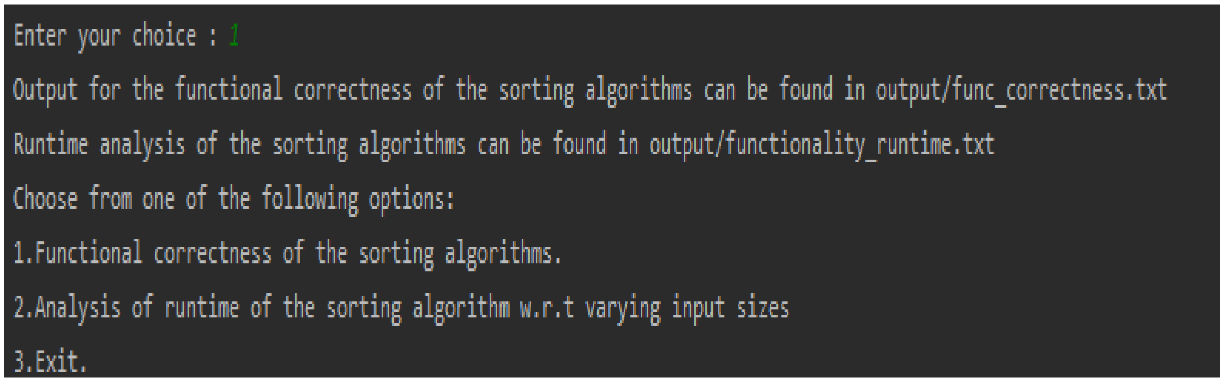


```
Choose from one of the following options:
1.Functional correctness of the sorting algorithms.
2.Analysis of runtime of the sorting algorithm w.r.t varying input sizes
3.Exit.
```

User Interface of the project

1. **Functional correctness of the sorting algorithms:**

This function takes in an input file which contains 50 random values and returns two output files which display the functional correctness i.e the output of the sorting algorithm after it sorts the input in the ascending/sorted order and the functionality runtime for the sorting algorithms. The latter analyses the runtime (in microsecond) for each of the sorting algorithms while the former checks for the accuracy and displays the output of the sort functions.



```
Enter your choice : 1
Output for the functional correctness of the sorting algorithms can be found in output/func_correctness.txt
Runtime analysis of the sorting algorithms can be found in output/functionality_runtime.txt
Choose from one of the following options:
1.Functional correctness of the sorting algorithms.
2.Analysis of runtime of the sorting algorithm w.r.t varying input sizes
3.Exit.
```

Steps to check the functional correctness of the sorting algorithms

- **func_correctness.txt**

Input string: [91, 28, 87, 49, 100, 14, 38, 26, 27, 20, 68, 19, 69, 13, 47, 95, 76, 88, 57, 97, 62, 1, 79, 8, 39, 7, 51, 96, 16, 70, 77, 99, 63, 53, 15, 41, 22, 72, 83, 24, 89, 98, 6, 46, 52, 11, 3, 75, 67, 90]

Output of Bubble Sort is:

[1, 3, 6, 7, 8, 11, 13, 14, 15, 16, 19, 20, 22, 24, 26, 27, 28, 38, 39, 41, 46, 47, 49, 51, 52, 53, 57, 62, 63, 67, 68, 69, 70, 72, 75, 76, 77, 79, 83, 87, 88, 89, 90, 91, 95, 96, 97, 98, 99, 100]

Output of Insertion Sort:

[1, 3, 6, 7, 8, 11, 13, 14, 15, 16, 19, 20, 22, 24, 26, 27, 28, 38, 39, 41, 46, 47, 49, 51, 52, 53, 57, 62, 63, 67, 68, 69, 70, 72, 75, 76, 77, 79, 83, 87, 88, 89, 90, 91, 95, 96, 97, 98, 99, 100]

Output of Merge Sort:

[1, 3, 6, 7, 8, 11, 13, 14, 15, 16, 19, 20, 22, 24, 26, 27, 28, 38, 39, 41, 46, 47, 49, 51, 52, 53, 57, 62, 63, 67, 68, 69, 70, 72, 75, 76, 77, 79, 83, 87, 88, 89, 90, 91, 95, 96, 97, 98, 99, 100]

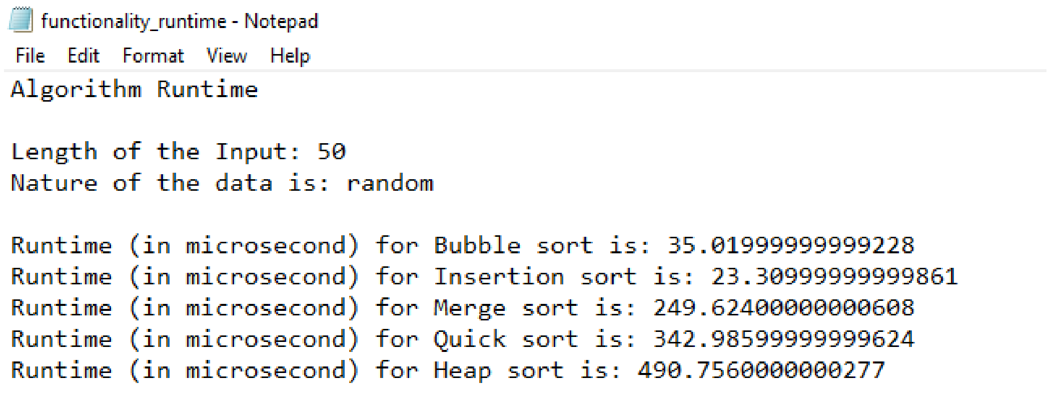
Output of Quick Sort:

[1, 3, 6, 7, 8, 11, 13, 14, 15, 16, 19, 20, 22, 24, 26, 27, 28, 38, 39, 41, 46, 47, 49, 51, 52, 53, 57, 62, 63, 67, 68, 69, 70, 72, 75, 76, 77, 79, 83, 87, 88, 89, 90, 91, 95, 96, 97, 98, 99, 100]

Output of Heap Sort:

[1, 3, 6, 7, 8, 11, 13, 14, 15, 16, 19, 20, 22, 24, 26, 27, 28, 38, 39, 41, 46, 47, 49, 51, 52, 53, 57, 62, 63, 67, 68, 69, 70, 72, 75, 76, 77, 79, 83, 87, 88, 89, 90, 91, 95, 96, 97, 98, 99, 100]

- **functionality_runtime.txt**

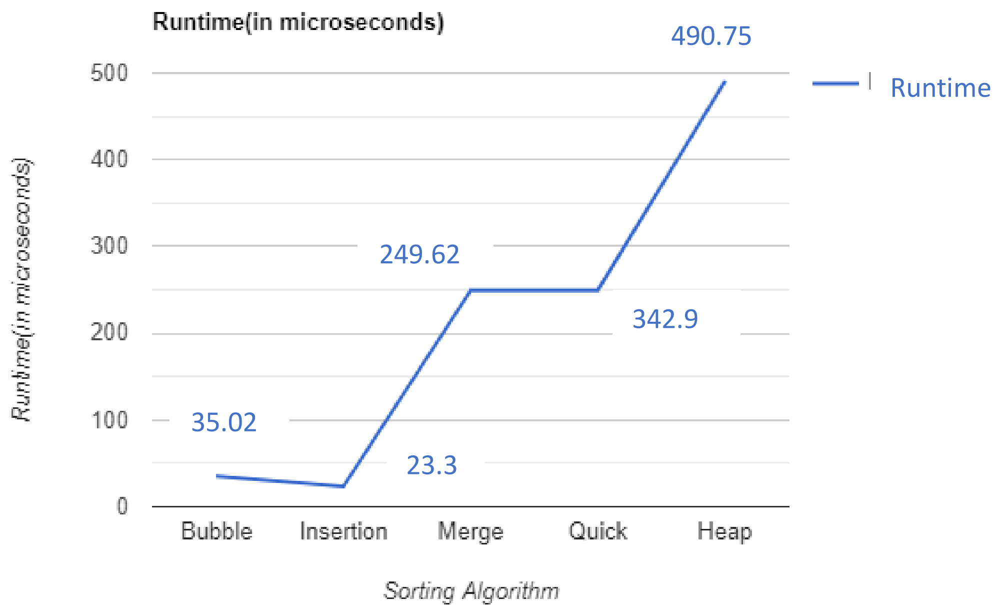


```
Algorithm Runtime

Length of the Input: 50
Nature of the data is: random

Runtime (in microsecond) for Bubble sort is: 35.01999999999228
Runtime (in microsecond) for Insertion sort is: 23.30999999999861
Runtime (in microsecond) for Merge sort is: 249.62400000000608
Runtime (in microsecond) for Quick sort is: 342.98599999999624
Runtime (in microsecond) for Heap sort is: 490.7560000000277
```

Output for Option1 in functionality_runtime.txt



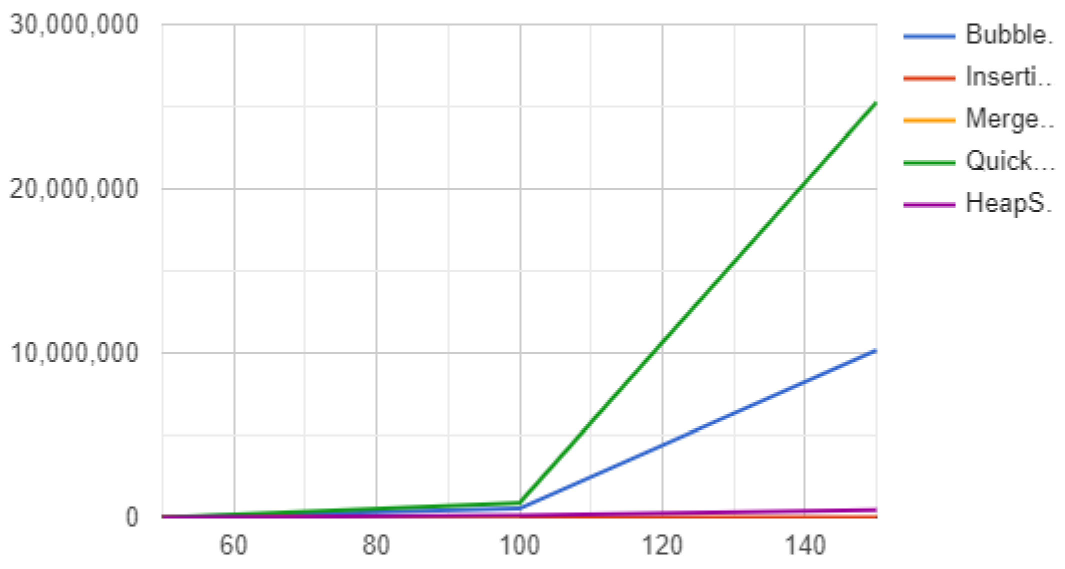
Graph depicting runtime(in microseconds) for the sorting algorithms

2. **Analysis of runtime of the sorting algorithm w.r.t varying input sizes:**

In this component, we check the runtime analysis of each of the sorting algorithms by giving different sizes of input integers that are either sorted or random. We have a user interface which gives us the option of choosing the input sizes. Inputs sizes vary from 1000 , 10000, 50000, 1000000, 10000000 integers. Next, we can choose either random(r) or sorted integers(s) to sort.

Input Size / Algorithm type	Bubble Sort	Insertion Sort	Merge Sort	Quick Sort	Heap Sort
1000	7193.46	522.69	4780.61	13719.82	9066.44
10000	539286.05	2869.49	65099.95	885391.61	121673.82
50000	10176564.8	11653.41	437377.85	25265691.66	468243.67
75000	35728939.82	24433.31	1199866.71	139057776.16	1033006.34

Tabular visualization of the runtime(in microseconds) for sorting



Graph representing the sorting algorithm runtime(in microseconds)

3. Exit: On selecting this option, the program breaks and gracefully exits from running.