Project 1
Sorting Algorithms

Implement and compare the following sorting Algorithm

Made by:

Vishwa Chetankumar Naik |CSE-5311-004 DSGN & ANLY Algorithms

UTA ID: 1001871311

- Title:

Implement and compare the following sorting Algorithm

1. Merge Sort
2. Heap Sort
3. Selection Sort
4. Insertion Sort
5. Bubble Sort
6. Quick Sort
7. Quick Sort (Using 3 medians)

- **Index:**

- **What is Sorting Algorithm?**

    - A Sorting Algorithm is used to rearrange a given array or list elements according to a comparison operator on the elements. The comparison operator is used to decide the new order of element in the respective data structure.
    - Since sorting can often reduce the complexity of a problem, it is an important algorithm in Computer Science. These algorithms have direct applications in searching algorithms, database algorithms, divide and conquer methods, data structure algorithms, and many more

- **Some of most common sorting Algorithms are:**

    - Selection Sort
    - Bubble Sort
    - Quick Sort
    - Merge Sort
    - Heap Sort
    - Insertion Sort
    - Bucket Sort
    - Radix Sort

- **Sorting Algorithms are often classified Based on the following parameters.**

    - Computational Time
    - Memory
    - Computational Complexity
    - Stability
    - Recursion

- **Here we are going to study, observe and calculate the execution time of the different algorithm.**

1. Merge Sort:

- Merge sort is a sorting technique based on divide and conquer technique. With worst-case time complexity being O(n log n), it is one of the most respected algorithms.
- Merge sort first divides the array into equal halves and then combines them in a sorted manner.
- What is Divide and Conquer Approach?
- **Divide:** This involves dividing the problem into some sub problem.
- **Conquer:** Sub problem by calling recursively until sub problem solved
- **Combine:** The Sub problem Solved so that we will get find problem solution.
- **Algorithm**

  Step 1: If it is only one element in the list it is already sorted return.
  Step 2: Divide the list recursively into two halves until it can no more be divided.
  Step 3: Merge the smaller lists into new list in sorted order.
  (Ref: https://www.geeksforgeeks.org/merge-sort/)

2. Heap Sort:

- Heap sort is a comparison-based sorting technique based on Binary Heap data structure. It is like selection sort where we first find the minimum element and place the minimum element at the beginning. We repeat the same process for the remaining elements.
- A comparison sort is a type of sorting algorithm that only reads the list elements through a single abstract comparison operation (often a "less than or equal to" operator or a three-way comparison) that determines which of two elements should occur first in the final sorted list.
- **Algorithm:**

  Step 1: Build a max heap from the input data.

Step 2: At this point, the largest item is stored at the root of the heap. Replace it with the last item of the heap followed by reducing the size of heap by 1. Finally, happify the root of the tree.
Step 3: Repeat step 2 while size of heap is greater than 1.
(Ref: https://www.geeksforgeeks.org/heap-sort/)

3. Selection Sort:

- The selection sort algorithm sorts an array by repeatedly finding the minimum element (considering ascending order) from unsorted part and putting it at the beginning. The algorithm maintains two subarrays in each array.
- **Algorithm:**
  Step 1: Set MIN to location 0.
  Step 2: Search the minimum element in the list.
  Step 3: Swap with value at location MIN.
  Step 4: Increment MIN to point to next element.
  Step 5: Repeat until list is sorted.
  (Ref: https://www.geeksforgeeks.org/selection-sort/)

4. Insertion Sort:

- Insertion sort is a simple sorting algorithm that works similar to the way you sort playing cards in your hands. The array is virtually split into a sorted and an unsorted part. Values from the unsorted part are picked and placed at the correct position in the sorted part.
- **Algorithm:**
  Step 1: If it is the first element, it is already sorted. return 1.
  Step 2: Pick next element.
  Step 3: Compare with all elements in the sorted sub-list.
  Step 4: shift all elements in the sorted sub-list that is greater than the value to be sorted.
  Step 5: Insert the value.
  Step 6: Repeat until list is sorted.
  (Ref: https://www.geeksforgeeks.org/insertion-sort/)

5. Bubble Sort:

- Bubble sort is simplest sorting algorithm that works by repeatedly swapping the adjacent elements if they are in wrong order.
- **Algorithm:**
  Step 1: Starting with the first element (index = 0), compare the current element with the next element of the array.
  Step 2: If the current element is greater than the next element of the array, swap them.
  Step 3: If the current element is less than the next element, move to the next element. Repeat step 1

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

6. Quick Sort:

- Quick sort is a highly efficient sorting algorithm and is based on partitioning of array of data into smaller arrays. A large array is partitioned into two arrays one of which holds values smaller than the specified value, say pivot, based on which the partition is made, and another array holds values greater than the pivot value.
- **Quick Sort Pivot Algorithm**
  Step 1: Choose the highest index value has pivot.
  Step 2: Take two variables to point left and right of the list excluding pivot.
  Step 3: Left points to the low index.
  Step 4: Right to the high.
  Step 5: While value at left is less than pivot move right.
  Step 6: While value at right is greater than pivot move left.
  Step 7: If both step 5 and step 6 does not match swap left and right.
  Step 8: If left ≥ right, the point where they met is new pivot.
- **Algorithm:**
  Step 1: Make the right-most index value pivot.
  Step 2: Partition the array using pivot value.
  Step 3: Quicksort left partition recursively.
  Step 4: Quicksort right partition recursively.
  (Ref: https://www.geeksforgeeks.org/quick-sort/)

7. Quick Sort (Using 3 median):

- In Quick sort median is for choosing pivot the first, middle, and last element are sorted. From the sorted elements the middle element is taken as pivot
  (Ref: https://walkccc.me/CLRS/Chap07/Problems/7-5/)

- **Time and Space complexity of the above algorithms:**

Time complexity

Algorithm	**Best Case**	**Average Case**	**Worst Case**
Merge sort	O(n*log n)	O(n*log n)	O(n*log n)
Heap sort	O(n*log n)	O(n*log n)	O(n*log n)
Selection sort	O(n^2)	O(n^2)	O(n^2)
Insertion sort	O(n)	O(n^2)	O(n^2)
Bubble sort	O(n)	O(n^2)	O(n^2)
Quick sort	O(n*log n)	O(n*log n)	O(n^2)
Quick(3 median) sort	O(n*log n)	O(n*log n)	O(n^2)

Space Complexity:

Algorithm	**Complexity**
Merge sort	**O(n)**
Heap sort	**O(1)**
Selection sort	**O(1)**
Insertion sort	**O(1)**
Bubble sort	**O(1)**
Quick sort	**O(1)**

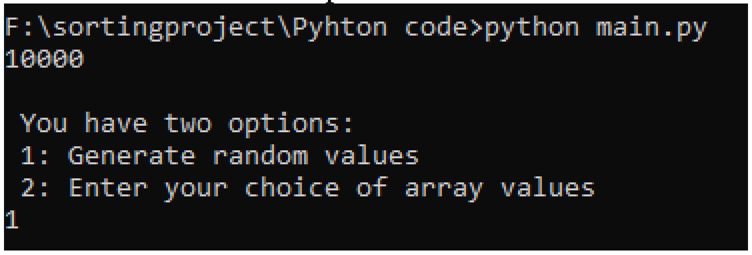
- **<u>Project Specification:</u>**

  - This project depicts the sorting algorithms working.
    1. Sorting Algorithm.
    2. Manual as well as user Input.
    3. Execution time.
    4. Graph Visualization.

- **<u>Project Steps:</u>**

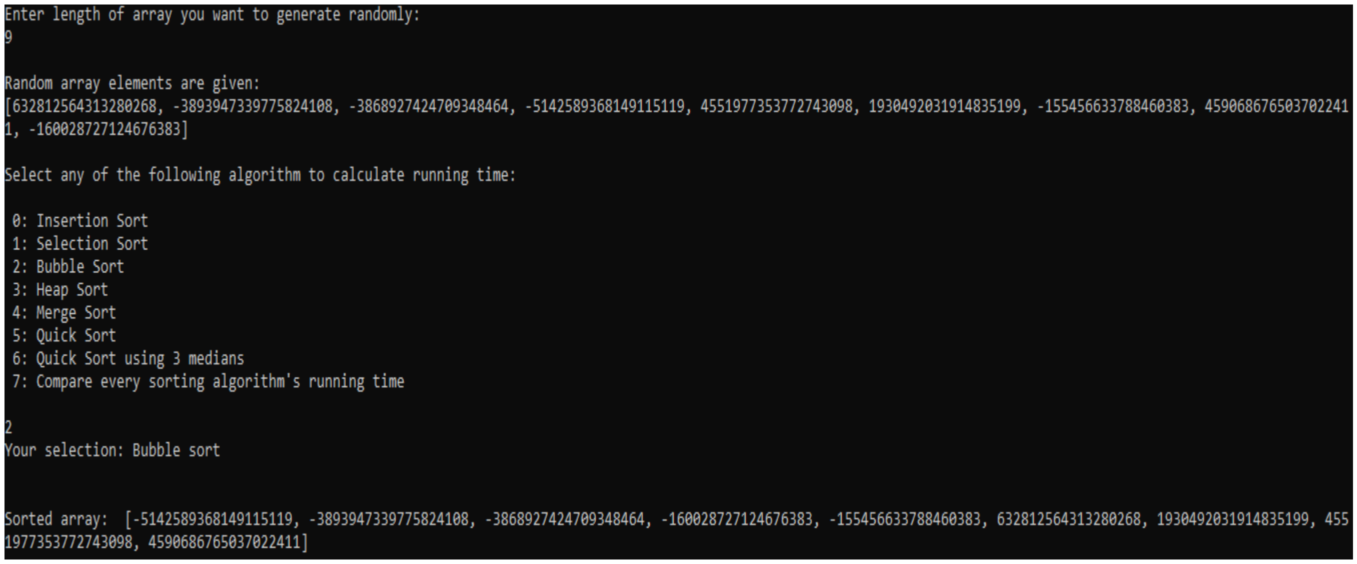
  - **<u>Step 1:</u>**

    Select form the two options:



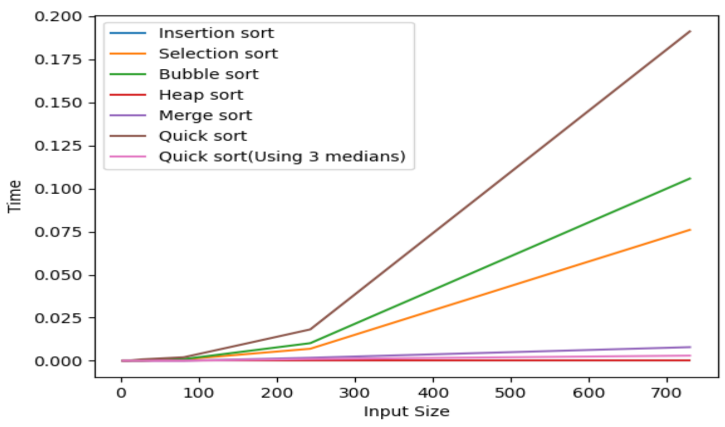
```
F:\sortingproject\Pyhton code>python main.py
10000

 You have two options:
 1: Generate random values
 2: Enter your choice of array values
1
```

Choose the length of array and algorithm to sort array

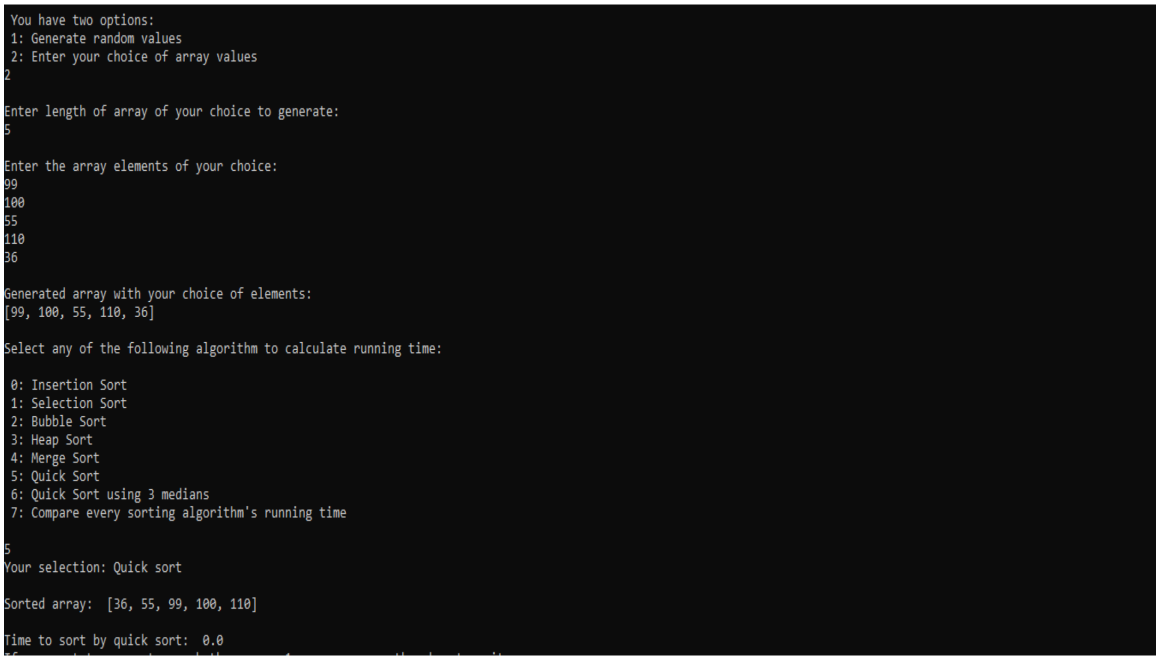


```
Enter length of array you want to generate randomly:
9

Random array elements are given:
[632812564313280268, -3893947339775824108, -3868927424709348464, -5142589368149115119, 4551977353772743098, 1930492031914835199, -155456633788460383, 459068676503702241
1, -160028727124676383]

Select any of the following algorithm to calculate running time:

 0: Insertion Sort
 1: Selection Sort
 2: Bubble Sort
 3: Heap Sort
 4: Merge Sort
 5: Quick Sort
 6: Quick Sort using 3 medians
 7: Compare every sorting algorithm's running time

2
Your selection: Bubble sort


Sorted array:  [-5142589368149115119, -3893947339775824108, -3868927424709348464, -160028727124676383, -155456633788460383, 632812564313280268, 1930492031914835199, 455
1977353772743098, 4590686765037022411]
```

Generate Graph:

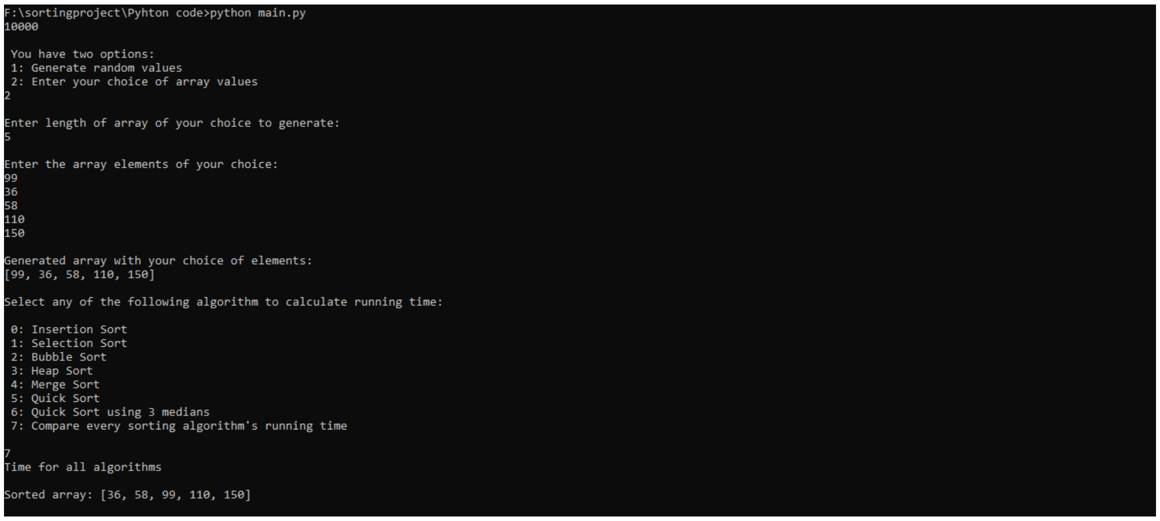


➢ **<u>Step 2:</u>**
Choose algorithm to sort array and enter your choice of value array.

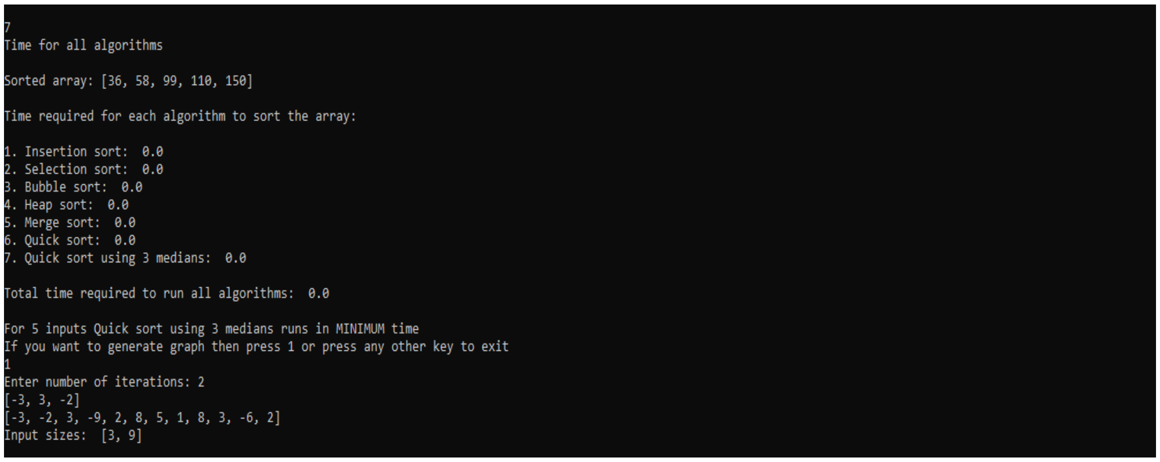


```
 You have two options:
 1: Generate random values
 2: Enter your choice of array values
2

Enter length of array of your choice to generate:
5

Enter the array elements of your choice:
99
100
55
110
36

Generated array with your choice of elements:
[99, 100, 55, 110, 36]

Select any of the following algorithm to calculate running time:

 0: Insertion Sort
 1: Selection Sort
 2: Bubble Sort
 3: Heap Sort
 4: Merge Sort
 5: Quick Sort
 6: Quick Sort using 3 medians
 7: Compare every sorting algorithm's running time

5
Your selection: Quick sort

Sorted array:  [36, 55, 99, 100, 110]

Time to sort by quick sort:  0.0
```

➢ Time calculation for all algorithms.

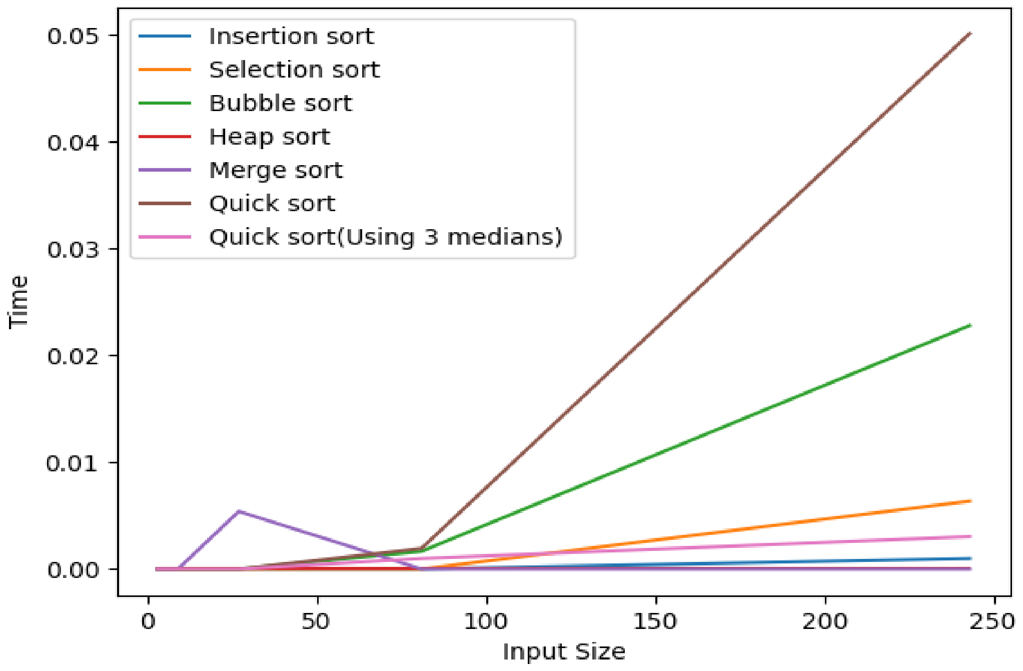


```
F:\sortingproject\Pyhton code>python main.py
10000

 You have two options:
 1: Generate random values
 2: Enter your choice of array values
2

Enter length of array of your choice to generate:
5

Enter the array elements of your choice:
99
36
58
110
150

Generated array with your choice of elements:
[99, 36, 58, 110, 150]

Select any of the following algorithm to calculate running time:

 0: Insertion Sort
 1: Selection Sort
 2: Bubble Sort
 3: Heap Sort
 4: Merge Sort
 5: Quick Sort
 6: Quick Sort using 3 medians
 7: Compare every sorting algorithm's running time

7
Time for all algorithms

Sorted array: [36, 58, 99, 110, 150]
```

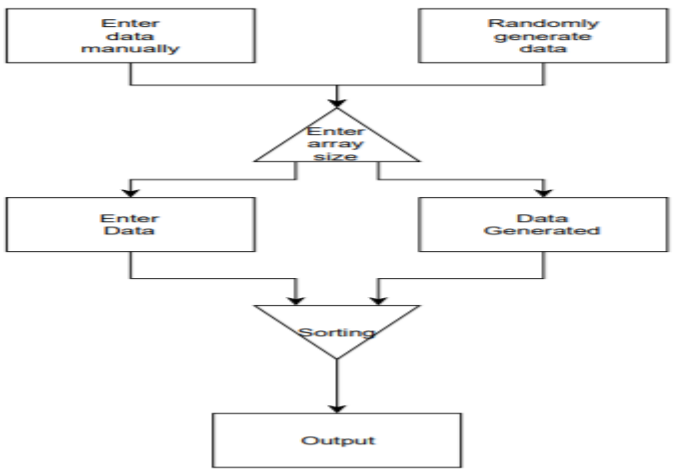


```
7
Time for all algorithms

Sorted array: [36, 58, 99, 110, 150]

Time required for each algorithm to sort the array:

1. Insertion sort:  0.0
2. Selection sort:  0.0
3. Bubble sort:  0.0
4. Heap sort:  0.0
5. Merge sort:  0.0
6. Quick sort:  0.0
7. Quick sort using 3 medians:  0.0

Total time required to run all algorithms:  0.0

For 5 inputs Quick sort using 3 medians runs in MINIMUM time
If you want to generate graph then press 1 or press any other key to exit
1
Enter number of iterations: 2
[-3, 3, -2]
[-3, -2, 3, -9, 2, 8, 5, 1, 8, 3, -6, 2]
Input sizes:  [3, 9]
```

- Graph for Input size vs Time:



- **<u>Project Flow:</u>**



- **Conclusion:**
    - From the above result I can conclude that Quick sort takes minimum time to run.
- Link to my web page for GUI representation for the project.

  Algorithms (https://vishwanaik15.github.io/algorithms/).