Glen Correia
1001980331

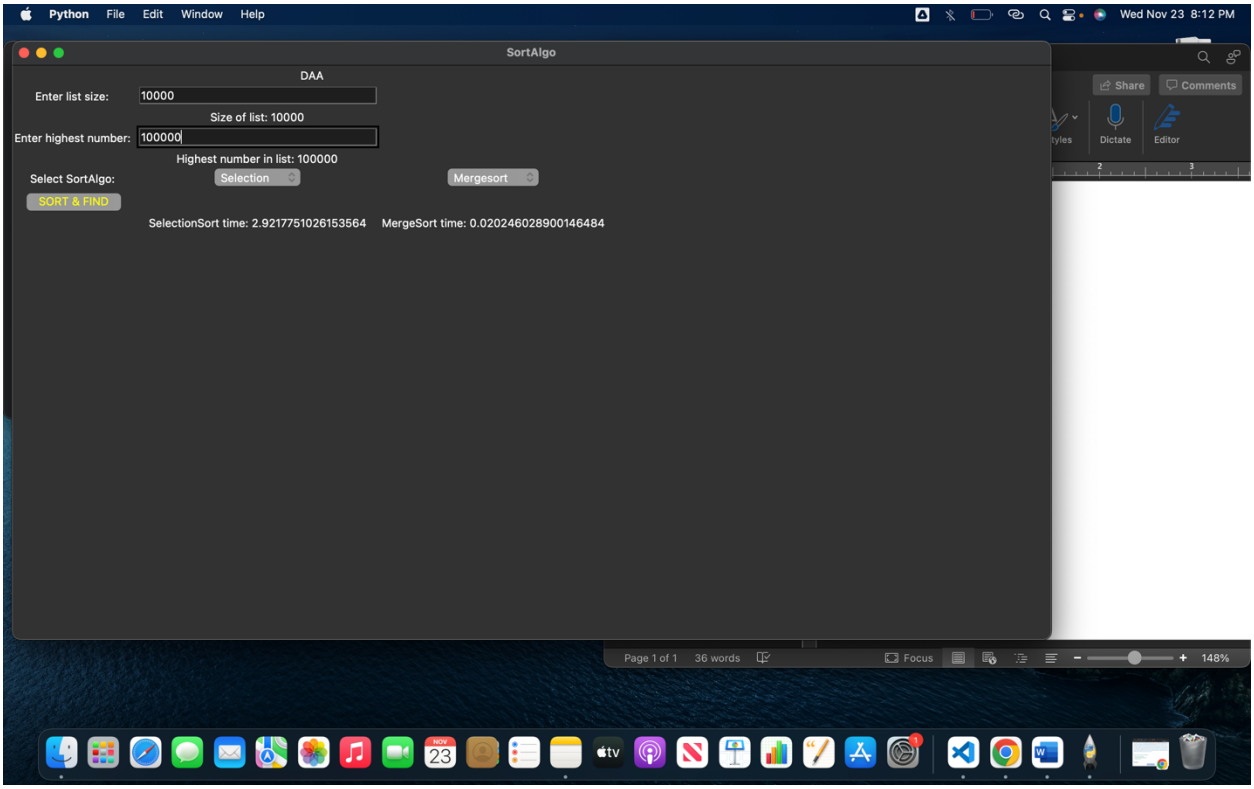
REPORT FOR PROJECT

Steps to include :
Open main.py in Editor (Vscode)
run main.py
A GUI will be. created

GUI:

GUI is created using tkinter library in python. On execution of py main.py command the user interface will open in new tab.

For Bubblesort, Selectionsort, Insertionsort maximum list size was 1000 for testing and for remaining algorithm list size was 10^7
All execution time is in seconds.



Input to be provided:

Enter the size of list: 10(size of list)
Enter highest number: 99 (list will have numbers range from 1 to value entered by user)

Program will generate random value of specified list size using random library

Note: For Bubblesort, Selectionsort, Insertionsort the sorted list is provided in output but not for other algorithms. Please limit your input to 100.

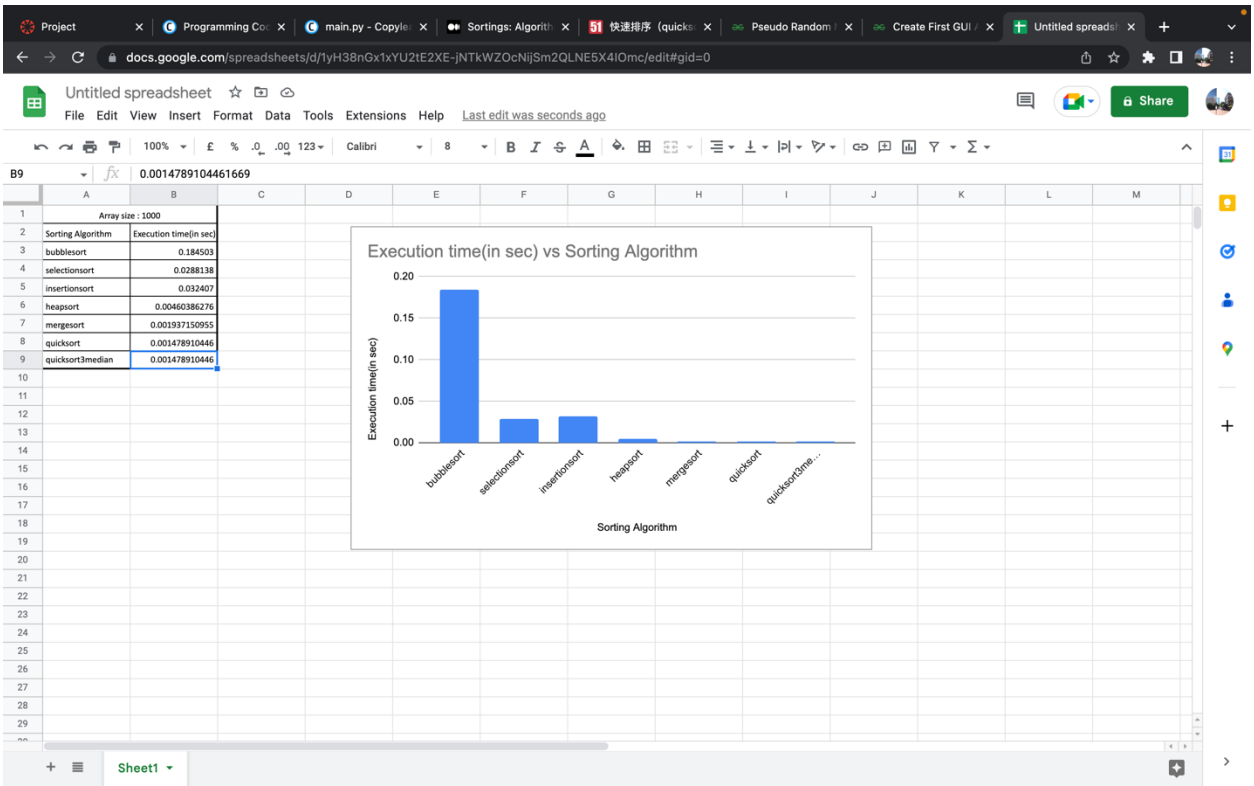
Data structure used: list (in python)

Attached excel sheet has graph associated to below execution data sorting algorithm vs execution time of it.

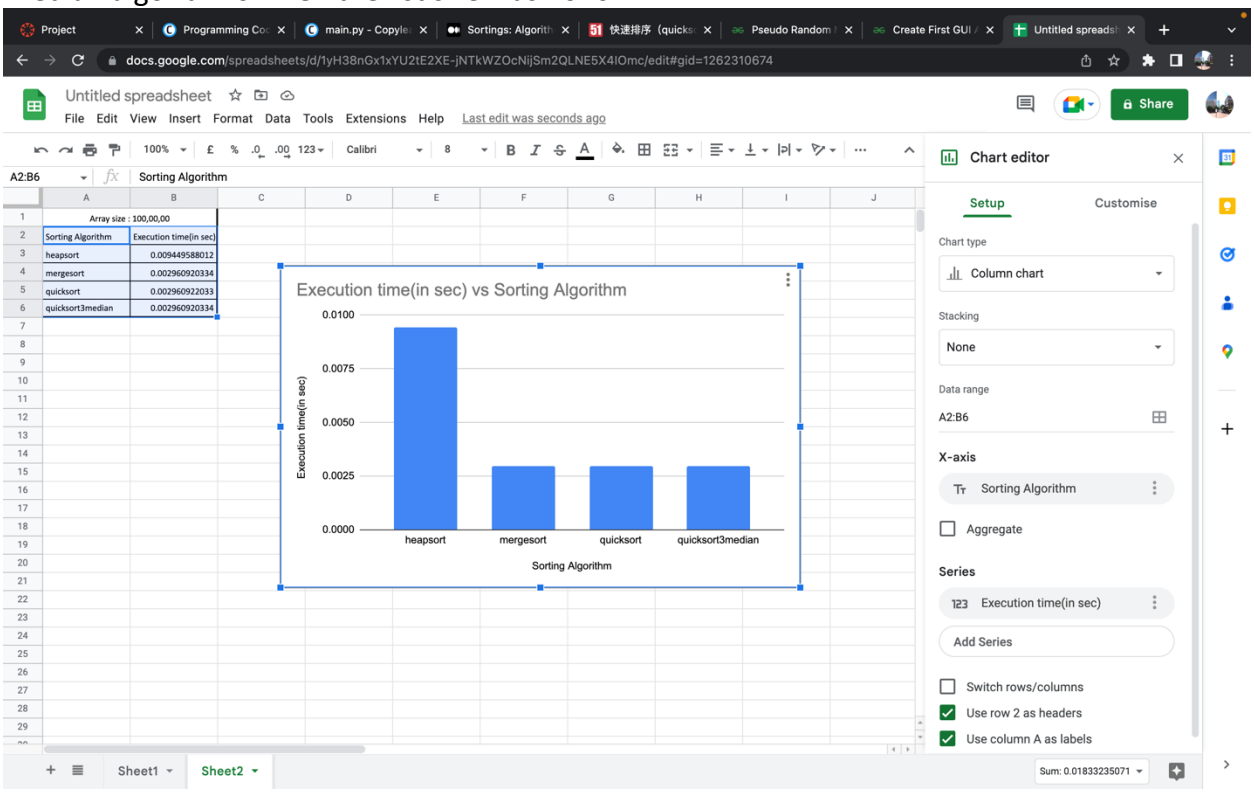
No	sheet-name	Description
1	All sorting algo execution time	For list size 10^3 it shows graph with the running time of all algorithms
2	Large dataset execution time	For list size 10^6 the graph contains only heapsort, mergesort, quicksort, quicksort3median as the system crashed for bubblesort and selectionsort and insertionsort

Graph:

1) Below graph shows the execution time of all the algorithms when the list size was 10^3.



2. Below graph shows the execution time of Heapsort, Mergesort, Quicksort and Quicksort using 3 median algorithms when the list size was 10^6



Improvement:
Quicksort in worst case has time complexity of O(n^2) when the pivot element is lowest or highest but we can improve time complexity using median of 3 method which has time complexity of O(nlog n)

Output of different algorithms, list might have repeated values

system>py bubblesort.py
Enter list size: 10
Enter highest number: 99
Random list [32, 45, 65, 98, 68, 38, 37, 6, 56, 10]
Running time 0.0
Sorted list [6, 10, 32, 37, 38, 45, 56, 65, 68, 98]

system>py selectionsort.py
Enter list size: 10
Enter highest number: 99
Random list [30, 14, 89, 47, 56, 43, 76, 9, 79, 50]
Running time 0.0
Sorted list [9, 14, 30, 43, 47, 50, 56, 76, 79, 89]

```
system>py insertionsort.py
Enter list size: 10
Enter highest number: 99
Random list [38, 78, 92, 43, 18, 92, 91, 73, 3, 7]
Running time 0.0
Sorted list [3, 7, 18, 38, 43, 73, 78, 91, 92, 92]

system>py heapsort.py
Enter list size: 10
Enter highest number: 99
Random list [51, 10, 98, 21, 80, 35, 65, 23, 18, 45]
Running time 0.0
Sorted list [10, 18, 21, 23, 35, 45, 51, 65, 80, 98]

system>py mergesort.py
Enter list size: 10
Enter highest number: 99
Random list [11, 25, 93, 48, 82, 72, 35, 69, 34, 4]
Running time 0.0
Sorted list [4, 11, 25, 34, 35, 48, 69, 72, 82, 93]

system>py quicksort.py
Enter list size: 10
Enter highest number: 99
Random list [99, 91, 20, 41, 59, 54, 54, 2, 26, 35]
Running time 0.0
Sorted list [2, 20, 26, 35, 41, 54, 54, 59, 91, 99]

system>py quicksort3median.py
Enter list size: 10
Enter highest number: 99
Random List [86, 11, 37, 55, 81, 36, 44, 43, 74, 8]
Running time 0.0
Sorted List [8, 11, 36, 37, 43, 44, 55, 74, 81, 86]
```

List having duplicate elements

```
system>py quicksort3median.py
Enter list size: 10
Enter highest number: 8
Random List [2, 1, 6, 4, 6, 4, 8, 7, 4, 8]
Running time 0.0
Sorted List [1, 2, 4, 4, 4, 6, 6, 7, 8, 8]
```

system>py quicksort.py
Enter list size: 10
Enter highest number: 8
Random list [1, 4, 1, 3, 4, 3, 8, 2, 6, 1]
Running time 0.0
Sorted list [1, 1, 1, 2, 3, 3, 4, 4, 6, 8]

system>py mergesort.py
Enter list size: 10
Enter highest number: 8
Random list [8, 7, 7, 3, 7, 1, 7, 6, 4, 5]
Running time 0.0
Sorted list [1, 3, 4, 5, 6, 7, 7, 7, 7, 8]

system>py insertionsort.py
Enter list size: 10
Enter highest number: 8
Random list [6, 2, 6, 2, 2, 5, 2, 7, 2, 8]
Running time 0.0
Sorted list [2, 2, 2, 2, 2, 5, 6, 6, 7, 8]

system>py selectionsort.py
Enter list size: 10
Enter highest number: 8
Random list [1, 4, 6, 2, 4, 6, 8, 6, 4, 2]
Running time 0.0
Sorted list [1, 2, 2, 4, 4, 4, 6, 6, 6, 8]

system>py heapsort.py
Enter list size: 10
Enter highest number: 8
Random list [6, 3, 2, 1, 6, 7, 8, 4, 2, 1]
Running time 0.0
Sorted list [1, 1, 2, 2, 3, 4, 6, 6, 7, 8]

system>py bubblesort.py
Enter list size: 10
Enter highest number: 8
Random list [4, 6, 6, 2, 3, 1, 8, 6, 6, 7]
Running time 0.0
Sorted list [1, 2, 3, 4, 6, 6, 6, 6, 7, 8]

List size:1000
Highest number in the list: 9999

system>py bubblesort.py
Enter list size: 1000
Enter highest number: 9999
Running time 0.3116610050201416

system>py selectionsort.py
Enter list size: 1000
Enter highest number: 9999
Running time 0.05501532554626465

system>py insertionsort.py
Enter list size: 1000
Enter highest number: 9999
Running time 0.07621550559997559

system>py heapsort.py
Enter list size: 1000
Enter highest number: 9999
Running time 0.010993003845214844

system>py mergesort.py
Enter list size: 1000
Enter highest number: 9999
Running time 0.004204273223876953

system>py quicksort.py
Enter list size: 1000
Enter highest number: 9999
Running time 0.00299835205078125

system>py quicksort3median.py
Enter list size: 1000
Enter highest number: 9999
Running time 0.004995107650756836

For large list size: 1000000
Only Heapsort, Mergesort, Quicksort, Quicksort3median
have been tested for large list size

system>py heapsort.py
Enter list size: 1000000
Enter highest number: 999999
Running time 28.4777896404266

system>py mergesort.py
Enter list size: 1000000
Enter highest number: 999999
Running time 9.99322915077209

system>py quicksort.py
Enter list size: 1000000
Enter highest number: 999999
Running time 7.17127728462219

system>py quicksort3median.py
Enter list size: 1000000
Enter highest number: 999999
Running time 8.67615556716918

Note: Description is provided for the all the functions in the code.
All Running time is in seconds
All sorting algorithms are in individual file as system crashed while doing comparison for large list size
System crashed when list size is beyond 10^7