VIT-AP UNIVERSITY

DISCRETE MATHEMATICS AND STRUCTURES

PROJECT

ALGORITHM ANALYSIS

S Vijay Balaji - 19BCE7571

ABSTRACT

This project helps in understanding the time complexity, that is to find out and analyze the time taken for each of the sorting algorithms to complete sorting a fixed or random dataset. Analysis of these algorithms help us in determining the quickest and most effective method to sort data under various circumstances and scenarios.

INTRODUCTION

Theoretical computer science includes areas of discrete mathematics relevant to computing. It draws heavily on graph theory and mathematical logic. Included within theoretical computer science is the study of algorithms and data structures. Computability studies what can be computed in principle, and has close ties to logic, while complexity studies the time, space, and other resources taken by computations.

The best method to analyze any of the sorting algorithms would be to visually look at the sorting process. Apart from visual aid, it also monitors all the different sorting times and the number of iterations.

The completed project would help people understand the different case uses of all the various sorting algorithms and also create an effective method to use them under various cases. Especially when the number of inputs is high or the entropy is high.

PROJECT EXECUTION

The entire project is written on Python. The GUI for depicting the whole sorting process is done using the help of matplotlib library. All the sorting algorithms are modified accordingly to display the changes made while sorting. Each algorithm is stored as a separate module and then imported into the main program for ease of understanding.

CODE:

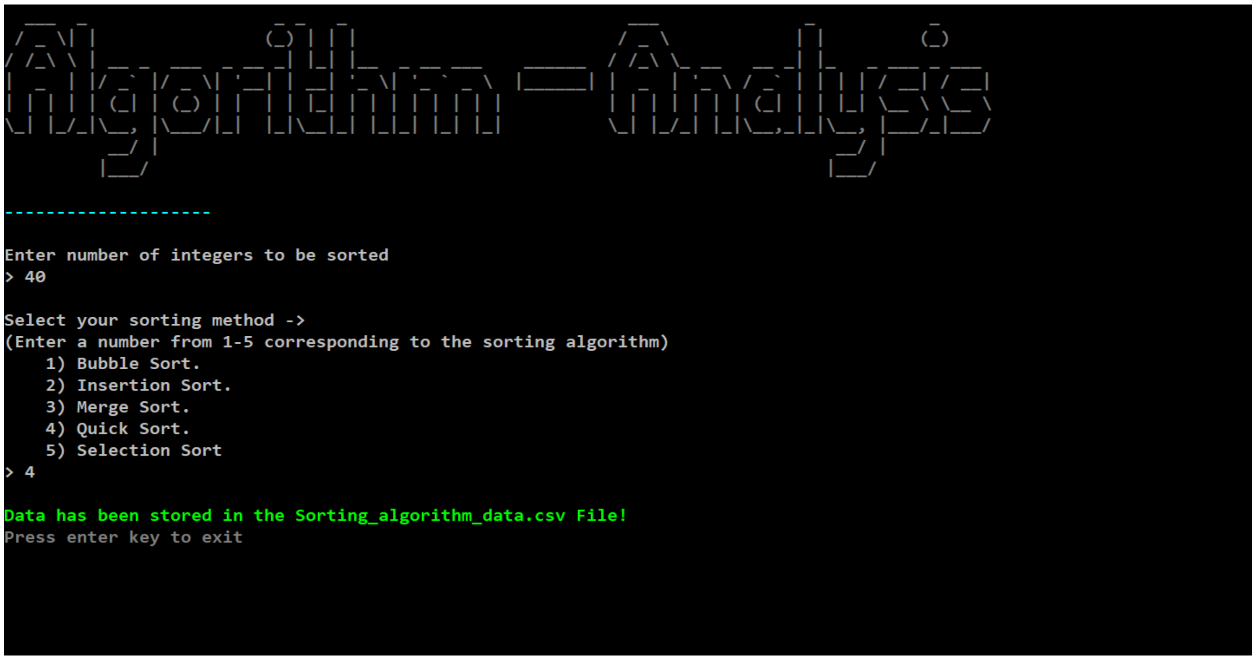
```
import random
import time
import os
import sys
import matplotlib.pyplot as plt
import matplotlib.animation as animation
from Modules.BubbleSort import BubbleSort
from Modules.InsertionSort import InsertionSort
from Modules.MergeSort import MergeSort
from Modules.QuickSort import QuickSort
from Modules.SelectionSort import SelectionSort
from Modules.Csv_writer import write
from Modules.Colours import *
if __name__ == "__main__":
    os.system('cls')
    logo = open("../assets/logo.txt","r")
    output = "".join(logo.readlines())
    grey(output)
    cyan("\n"+"-"*20)
    print()
    time.sleep(1)
    try:
        N = int(input("Enter number of integers to be sorted \n> "))
    except:
        red("ERROR : Enter only numbers!")
        grey("Press enter key to exit...")
        input()
        sys.exit(0)
    array = [x + 1 for x in range(N)]
    random.shuffle(array)
    data = array.copy()
    print()
    print("""Select your sorting method ->
(Enter a number from 1-5 corresponding to the sorting algorithm)
    1) Bubble Sort.
    2) Insertion Sort.
    3) Merge Sort.
    4) Quick Sort.
    5) Selection Sort""")
    try:
        choice = int(input("> "))
    except:
        red("ERROR : Enter numbers from 1 to 5 only!")
        grey("Press enter key to exit...")
        input()
        sys.exit(0)
    if(choice == 1):
        title = "Bubble Sort"
        generator_fn = BubbleSort(array)
    elif(choice == 2):
        title = "Insertion Sort"
        generator_fn = InsertionSort(array)
    elif(choice == 3):
        title = "Merge Sort"
        generator_fn = MergeSort(array, 0, N - 1)
    elif(choice == 4):
        title = "Quick Sort"
        generator_fn = QuickSort(array, 0, N - 1)
    elif(choice == 5):
        title = "Selection Sort"
        generator_fn = SelectionSort(array)
    else:
        print()
        red("ERROR : Incorrect choice. Select numbers from 1 to 5 only!")
        grey("Press enter key to exit...")
        input()
        sys.exit(0)
    fig, axis = plt.subplots()
    axis.set_title(title)
    rect = axis.bar(range(N), array, align="edge")
    axis.set_xlim(0, N)
    axis.set_ylim(0, int(1.05 * N))
```

```
text = axis.text(0.02, 0.95, "", transform=axis.transAxes)
count = [0]
def update(array, rect, count):
    for x, val in zip(rect, array):
        x.set_height(val)
    count[0] += 1
    text.set_text("# of operations: {}".format(count[0]))
anim = animation.FuncAnimation(fig, func=update, fargs=(rect, count),
frames=generator_fn, interval=1, repeat=False)
plt.show()
write(title,data,count[0])
```

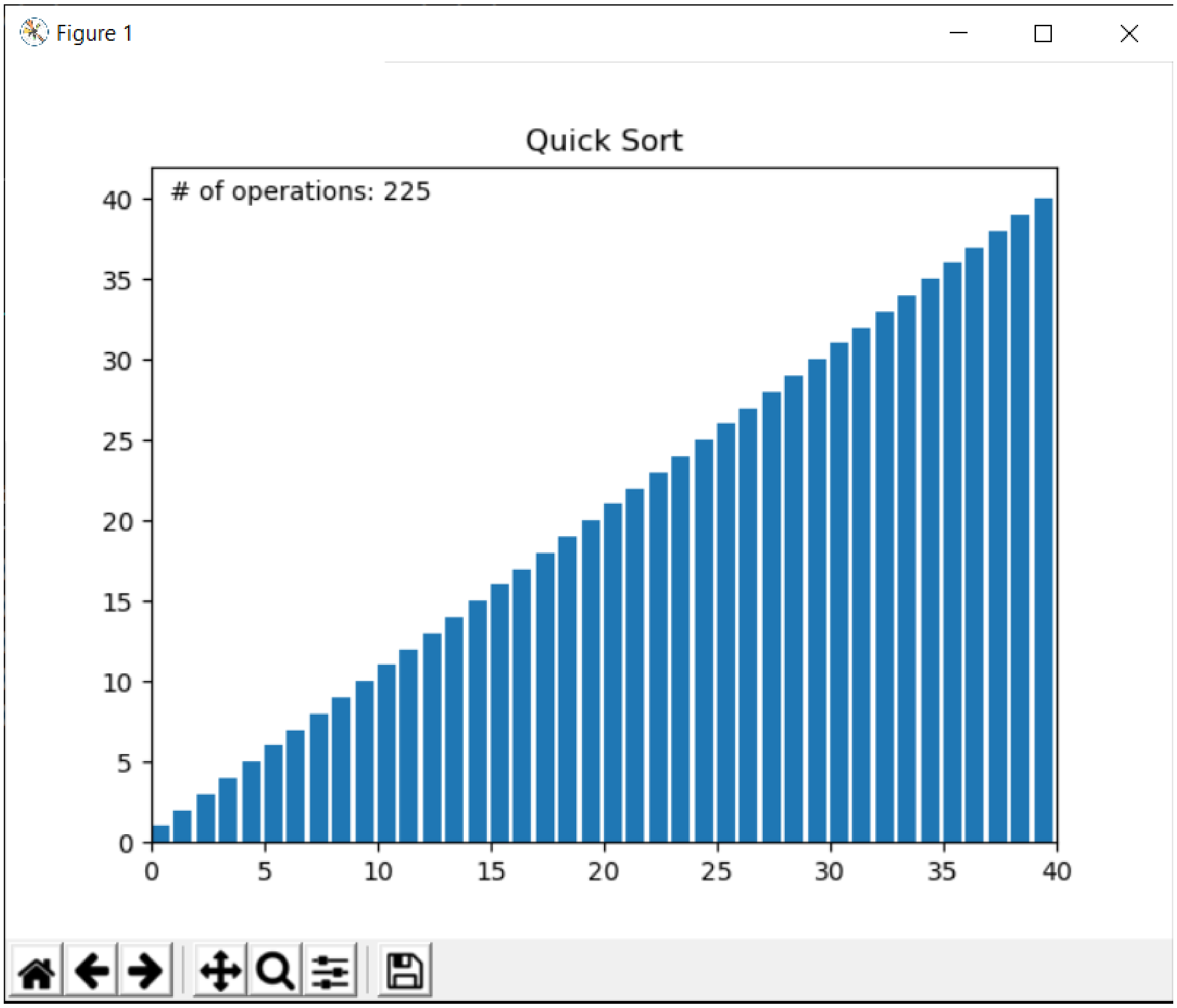
Module for the different algorithms and complete code is available on GitHub respiratory-

GitHub respiratory link: https://github.com/SVijayB/Algorithm-Analysis

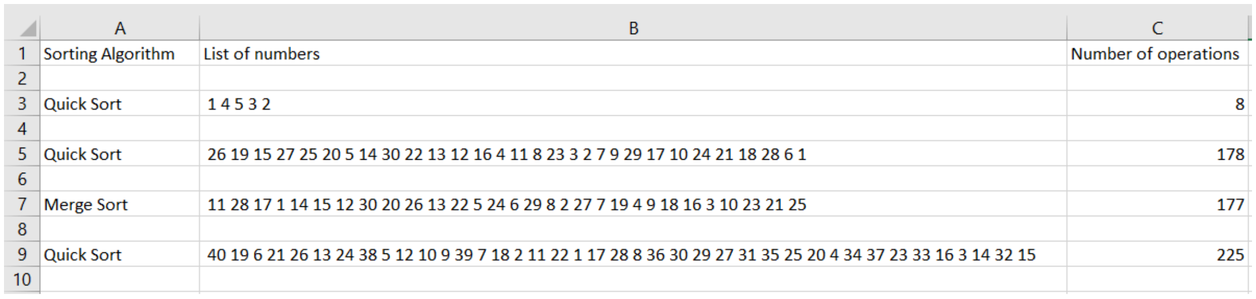
OUTPUT:



```
--------------------

Enter number of integers to be sorted
> 40

Select your sorting method ->
(Enter a number from 1-5 corresponding to the sorting algorithm)
    1) Bubble Sort.
    2) Insertion Sort.
    3) Merge Sort.
    4) Quick Sort.
    5) Selection Sort
> 4

Data has been stored in the Sorting_algorithm_data.csv File!
Press enter key to exit
```



CSV Output data:



	A	B	C
1	Sorting Algorithm	List of numbers	Number of operations
2			
3	Quick Sort	1 4 5 3 2	8
4			
5	Quick Sort	26 19 15 27 25 20 5 14 30 22 13 12 16 4 11 8 23 3 2 7 9 29 17 10 24 21 18 28 6 1	178
6			
7	Merge Sort	11 28 17 1 14 15 12 30 20 26 13 22 5 24 6 29 8 2 27 7 19 4 9 18 16 3 10 23 21 25	177
8			
9	Quick Sort	40 19 6 21 26 13 24 38 5 12 10 9 39 7 18 2 11 22 1 17 28 8 36 30 29 27 31 35 25 20 4 34 37 23 33 16 3 14 32 15	225
10			

CONCLUSION

In this report a code is implemented on five different sorting algorithms where each one was investigated and shown how it works with help of bar graph in output. As the experiment shows that some sorting algorithms are less efficient than others and that had all to do with how each sorting algorithm works.

By this experiment it was concluded that Bubble sort was the least efficient of the different sorting algorithms and the Quick Sort was the most efficient of all the sorting algorithms.

From this we can conclude that Bubble sort is best used just to explain or introduce the sorting algorithm to a new student. As soon as you want to have an efficient sorting algorithm, the Quick Sort will be the best because less time will be spent to sort the objects in the array and hence time complexity will be reduced drastically.

REFRENCES:

https://en.wikipedia.org/wiki/Discrete_mathematics

https://en.wikipedia.org/wiki/File:Sorting_quicksort_anim.gif