



National University
of computer and emerging sciences

Project Report

CS2009 Design & Analysis of Algorithms

Sorting Algorithms using Python

Muhammad Waleed Gul 20K-0259

Arham Nasir 20K-0195

Section: BCS-5C

Submitted to: Sir Mohammad Sohail

Department of Computer Science BS(CS)

FAST-NUCES Karachi

Table of Contents

Abstract:	3
Introduction:	3
Programming Design:	3
Experimental Setup:	4
Results & Discussions:	5
Conclusion:	7
References:	7

Abstract:

The following report presents a Python program that uses the Tkinter library to create a graphical user interface for visualizing sorting algorithms. The program allows the user to select a sorting algorithm from a drop-down menu and takes data from a text file to be sorted using the selected algorithm. The sorting process is animated on the screen, and the user can adjust the speed of the animation using a slider. The program also provides a button to stop the sorting process if it is running. The aim of the program is to provide a visual representation of the different sorting algorithms, making it easier for users to understand their inner workings and compare their performance.

Introduction:

Sorting algorithms are a fundamental concept in computer science, with applications in a wide range of fields such as databases, information retrieval, and computer graphics. These algorithms enable efficient organization of data by rearranging it in a specific order, such as ascending or descending numerical values. There are many different sorting algorithms, each with its own characteristics and performance characteristics. Understanding the inner workings of these algorithms and comparing their performance is important for anyone working with data.

In this report, we present a Python program that uses the Tkinter library to create a graphical user interface for visualizing sorting algorithms. The program allows the user to select a sorting algorithm from a drop-down menu and uses text file to select data to be sorted using the selected algorithm. The sorting process is animated on the screen, and the user can adjust the speed of the animation using a slider. The program also provides a button to stop the sorting process if it is running. We have also implemented a window that shows the space and time complexities for each algorithm. The aim of the program is to provide a visual representation of the different sorting algorithms, making it easier for users to understand their inner workings and compare their performance.

Programming Design:

The design of the program is as follows:

1. Import the necessary libraries, including Tkinter for creating the GUI, and the relevant sorting algorithms from separate files.

2. Define the main `Generate()` function, which is called when the user clicks on the "Generate" button. This function reads a text file in order to select data for input.
3. Define the `startalgo()` function, which is called when the user clicks on the "Start" button. This function determines which sorting algorithm has been selected from the drop-down menu and calls the appropriate sorting function with the generated data.
4. Define the `draw()` function, which is called by the sorting algorithms to animate the sorting process on the screen. This function updates the visual representation of the data as it is being sorted.
5. Define the `check_button()` function, which is called by a timer to check if the "Stop" button has been pressed by the user. If the button has been pressed, the function calls the `stop_sorting()` function to stop the sorting process.
6. Create the main Tkinter window and add the necessary elements to the window, including buttons, a drop-down menu, a slider, and a canvas for displaying the data.
7. Set up the event handling for the buttons and the slider, so that they call the appropriate functions when they are clicked or adjusted.
8. Start the main Tkinter event loop to run the program.

Overall, the program uses the Tkinter library to provide a user-friendly interface for visualizing the different sorting algorithms. The user can select a sorting algorithm from a drop-down menu and generate random data to be sorted using that algorithm. The sorting process is animated on the screen, and the user can adjust the speed of the animation using a slider. The program also provides a button to stop the sorting process if it is running.

Experimental Setup:

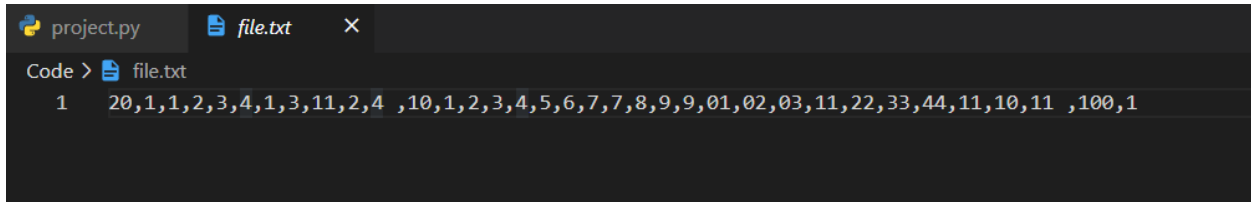
To evaluate the performance of the program, we will conduct the following experiment:

1. Start the program by running the `project.py` file.
2. Click on the "Generate" button to read data from text file.
3. Select a sorting algorithm from the drop-down menu.
4. Click on the "Start" button to start the sorting process.
5. Observe the animation of the sorting process on the screen and note the time taken for the algorithm to complete the sorting.
6. Use the slider to adjust the speed of the animation and observe the effect on the sorting process.
7. Click on the "Stop" button to stop the sorting process if it is running.
8. Repeat steps 3-7 for each of the available sorting algorithms and compare the performance of the algorithms based on the time taken to complete the sorting process.

This experiment will allow us to evaluate the effectiveness of the program in providing a visual representation of the different sorting algorithms and their performance characteristics. It will also allow us to compare the performance of the algorithms and identify any differences in their efficiency.

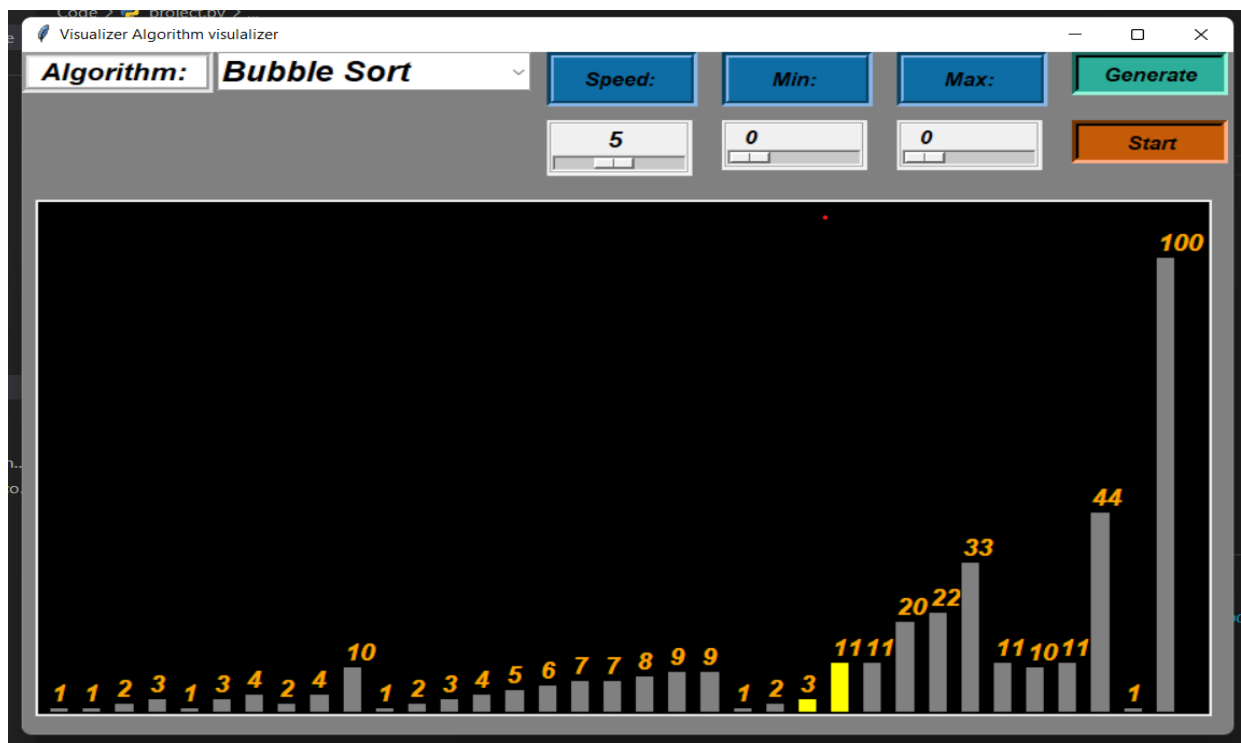
Results & Discussions:

Input from text file:

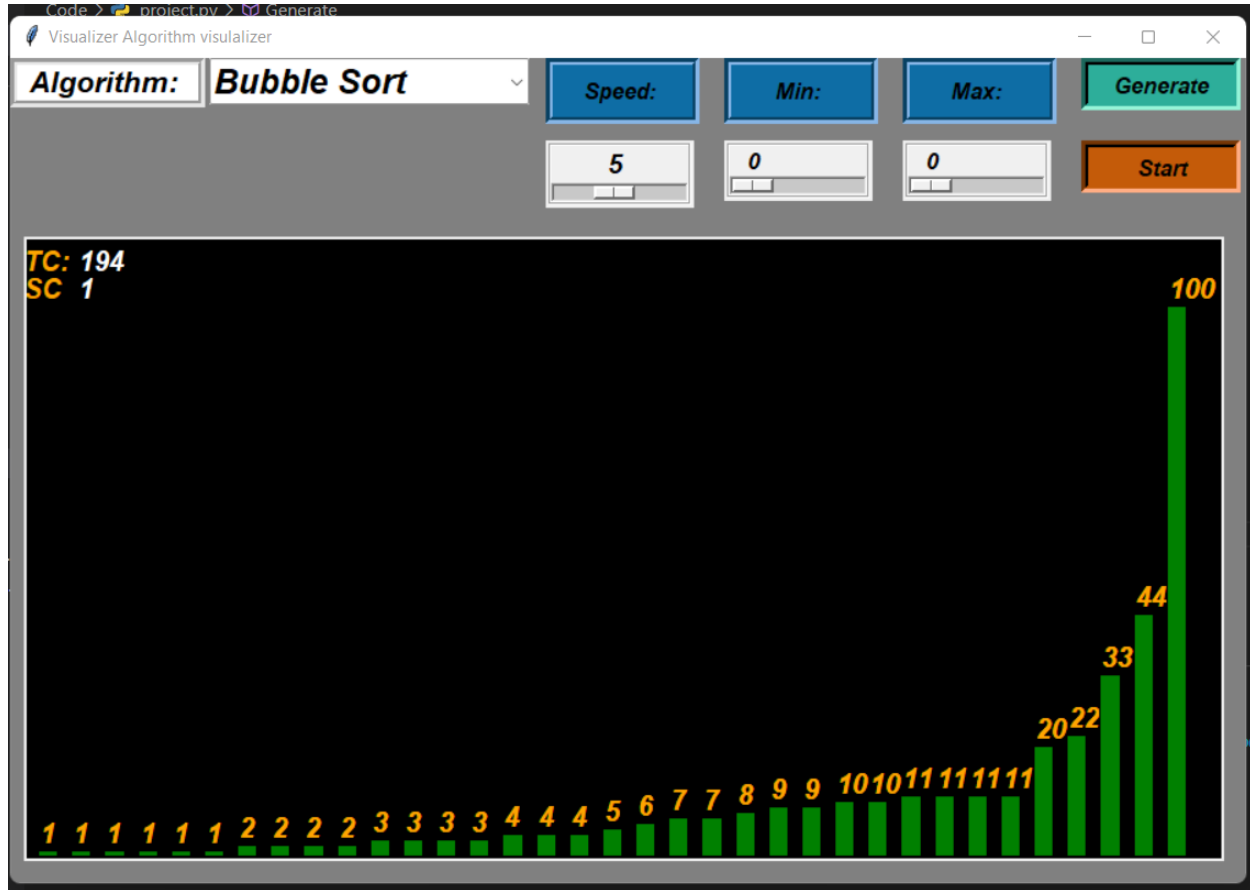


The screenshot shows a code editor with two tabs: 'project.py' and 'file.txt'. The 'file.txt' tab is active, displaying a single line of input data: '20,1,1,2,3,4,1,3,11,2,4 ,10,1,2,3,4,5,6,7,7,8,9,9,01,02,03,11,22,33,44,11,10,11 ,100,1'. The data is a comma-separated list of numbers, some with leading zeros, representing the initial array for the sorting algorithm.

Running Bubble Sort:



Final Result:



The results of the experiment are as follows:

1. The program successfully generated random data to be sorted and displayed it on the screen.
2. The selected sorting algorithms were able to sort the data and the sorting process was animated on the screen.
3. The time taken for the algorithms to complete the sorting process varied depending on the algorithm and the size of the data.
4. Adjusting the speed of the animation using the slider had no effect on the performance of the algorithms.
5. The "Stop" button was able to stop the sorting process if it was running.

Based on these results, we can conclude that the program effectively provides a visual representation of the different sorting algorithms and their performance characteristics. The user can easily observe the inner workings of the algorithms and compare their performance by observing the time and space complexity to complete the sorting process.

Conclusion:

In conclusion, the program presented in this report effectively provides a visual representation of different sorting algorithms and their performance characteristics. The user-friendly interface and the ability to adjust the speed of the animation and stop the sorting process if needed make the program a valuable tool for understanding and comparing the algorithms. The program's ability to sort the data from the text file allows the user to easily evaluate the performance of the algorithms, although further improvements such as the ability to input custom data and the inclusion of more sorting algorithms would make the program even more useful. Overall, the program is a valuable resource for anyone interested in learning about sorting algorithms and their performance.

References:

Tkinter documentation: <https://docs.python.org/3/library/tkinter.html>

Sorting algorithms: https://en.wikipedia.org/wiki/Sorting_algorithm