Project Report

On

Quicksort Algorithm Implementation In Python

Presented By:

Serial Number	Name	ID
01	S. M. Sanaul Haque	20212020010
02	Farjana Yeasmin Mohona	20212013010
03	S. M. Rokibul Islam	20212024010

Presented To:

Md. Shymon Islam

Lecturer

Department of CSE

North Western University, Khulna.

Table Of Contents

Acknowledgement

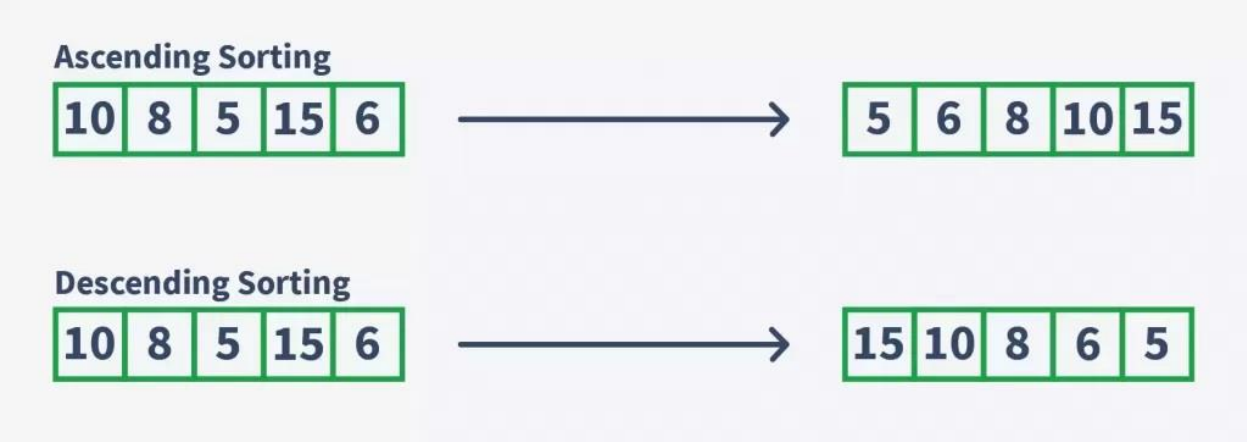
This project has been done as part of our course for the **CSE-2204**. I would like to thank my teacher, **Md. Shymon Islam** for his advice and inputs on the project. Many thanks to my group member and friends as well, which spent countless hours to listen and provide feedbacks.

Overview

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.

Consider a list with the elements – 10, 8, 5, 15, 6 in it. We have to arrange these elements in both ascending and descending order.

- How are we going to perform this arrangement? Simple answer – by using a **Sorting algorithm**. A sorting algorithm is a method of reorganizing the elements in a meaningful order. In both the above-given examples, we have implemented a sorting algorithm to get the desired results.



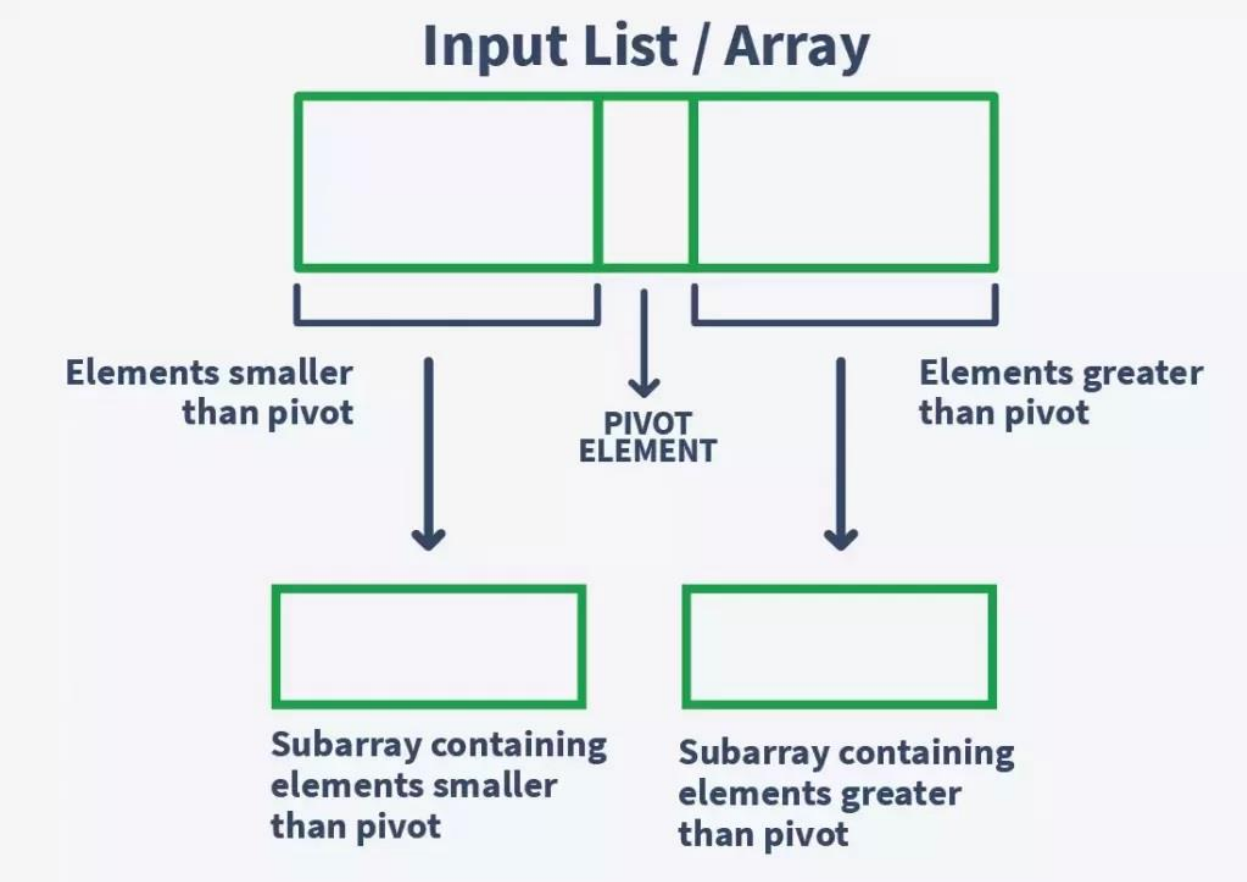
There are multiple algorithms that can be used for sorting. And, here, we will discuss Quick sort.

What is Quick Sort?

Quick sort, also known as partition-exchange sort, is an in-place sorting algorithm. It is a divide-and-conquer algorithm that works on the idea of selecting a pivot element and dividing the array into two subarrays around that pivot.

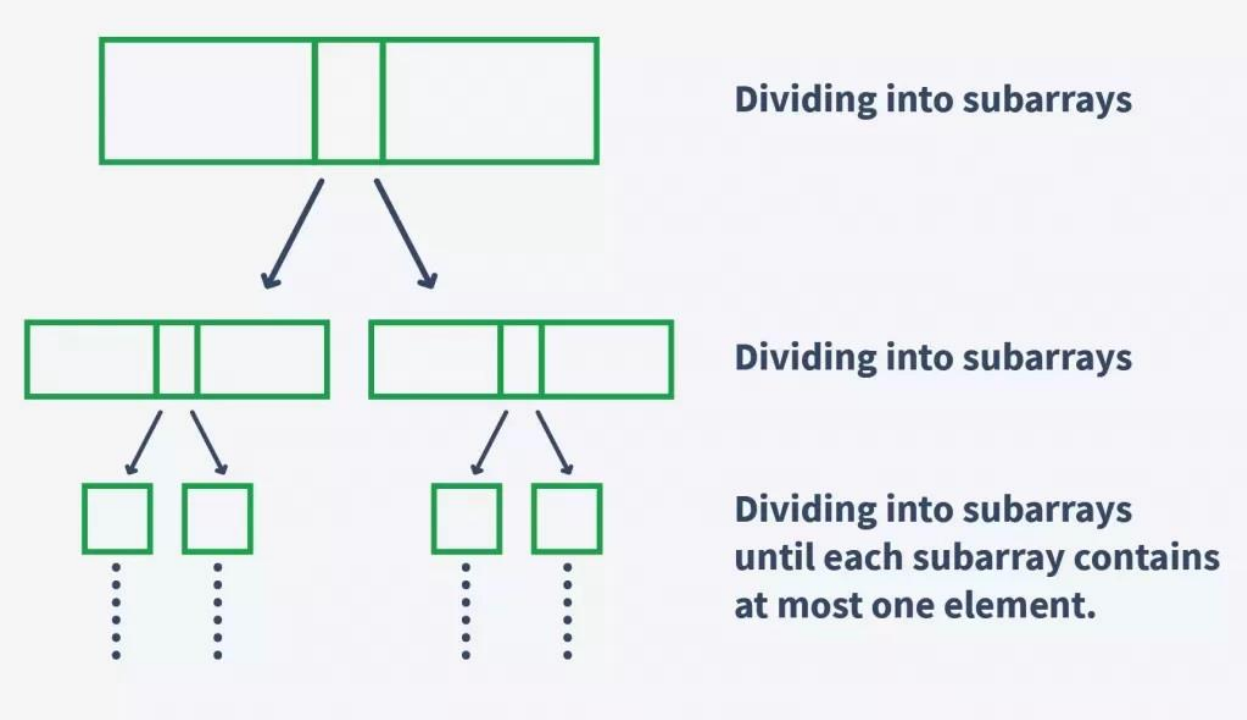
In quick sort, after selecting the pivot element, the array is split into two subarrays. One subarray contains the elements smaller than the pivot element, and the other subarray contains the elements greater than than the pivot element.

Below given is a representation of what the input array looks like after first iteration of quick sort:



At every iteration of quick sort, the chosen pivot element is placed at its correct position that it should acquire in the sorted array. This makes sure that each chosen pivot element is at its correct sorted position.

This process of selecting the pivot element and dividing the array into subarrays is carried out recursively until all the elements in the array are sorted. In other words, each subarray is divided further until each subarray consists of only one element. All these subarrays are then combined to form a single sorted array. Since we know before partitioning of the array, the pivot is placed at its correct position, dividing the array to one single element places each element at its correct sorted position, and thus, combining all of them gives us a sorted array.



This way, with each iteration of quick sort, the problem reduces by 2 steps making quick sort a faster way to sort an array.

One interesting thing to note here is that we can choose any element as our pivot, be it the first element, last element, or any random element from the array.

To better understand the quick sort algorithm, imagine this. Owing to your good conduct, your class teacher gives you the responsibility of a monitor. One day, for a fest, your teacher calls you up and asks you to help her in arranging the students in height-wise order.

There are many ways in which you can arrange the students, you can pick every student and show them their places one-by-one, but this would take too much time if there are a lot of students (say 500), and you have to finish your job as soon as possible. So, one of the fastest ways could be asking the students to arrange themselves. **For example**, the shortest student knows that he has to stand at the front. Same for the tallest student and the students with medium height.

In this way, every student can find his/her correct position by comparing his/her height with that of students before and after him/her.

- Similarly, in quick sort, every element arranges itself at its correct position to sort the given array. Here, the pivot element is placed at its correct sorted position, and hence it is the element that we know is definitely sorted. And thus, subarrays are divided around the pivot element.

Objectives

The Quick sort visualization project is a Python program that uses the Tkinter library to create a graphical user interface (GUI) for the Quick sort algorithm. The program consists of a canvas, which is the main visual component of the project, and one widget section on the bottom of the canvas.

The canvas displays a set of boxes representing the data to be sorted. The algorithm is visualized by moving the boxes around to their correct position during the sorting process. The boxes are color-coded to show their current state during the sorting process, such as being swapped or already sorted.

The canvas also displays text annotations that describe the current state of the sorting process. The widget section on the bottom of the canvas is used to modify the algorithm and data. It contains a slider that allow the user to adjust the speed of the sorting process.

The user can also manually input data into an input section or generate random data using a generate button. The generate button also creates the boxes on the canvas to represent the generated data.

Once the data is generated, the user can start the sorting process by clicking on a start button. An edit button also appears after the generate button is clicked, which opens a new window allowing the user to add, delete, or modify the current generated data.

At the bottom of the canvas, there is a list box that displays the sorting or searching process happening in text. The list box shows the current state of the data and the actions being taken by the algorithm during the sorting or searching process. This provides an additional way for the user to follow the algorithmic process.

At the bottom of the canvas, there is a list box that displays the sorting or searching process happening in text. The list box shows the current state of the data and the actions being taken by the algorithm during the sorting or searching process. This provides an additional way for the user to follow the algorithmic process.

Overall, the Quick sort visualization project provides an interactive and intuitive way to understand and visualize the Quick sort algorithm. It allows the user to modify the data and the algorithm parameters, search for specific elements within the data, and follow the algorithmic process in real-time using visual and text-based feedback.

Description

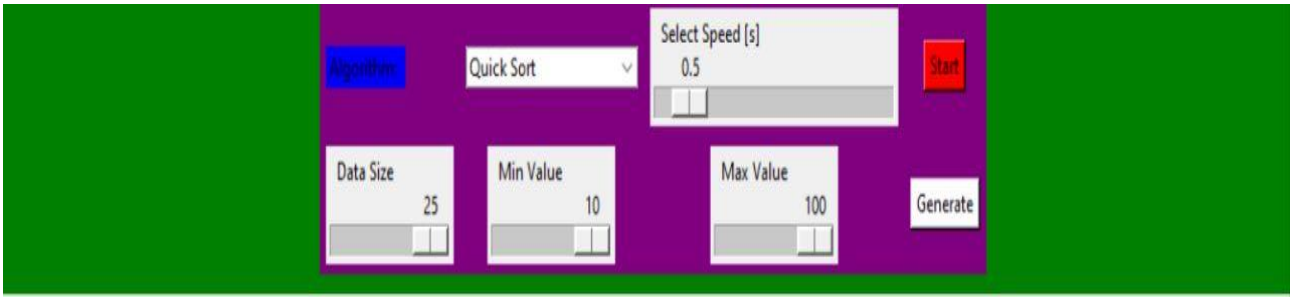
Interface Area:

This is the code of interface area of our program…

```
# User Interface Area
# Row[0]
Label(UI_frame, text="Algorithm: ", bg='blue').grid(row=0, column=0, padx=5, pady=5, sticky=W)
algMenu = ttk.Combobox(UI_frame, textvariable=selected_alg, values=['Quick Sort'])
algMenu.grid(row=0, column=1, padx=5, pady=5)
algMenu.current(0)

speedScale = Scale(UI_frame, from_=0.1, to=5.0, length=200, digits=2, resolution=0.2, orient=HORIZONTAL,
                   label="Select Speed [s]")
speedScale.grid(row=0, column=2, padx=5, pady=5)
Button(UI_frame, text="Start", command=StartAlgorithm, bg='red').grid(row=0, column=3, padx=5, pady=5)

# Row[1]
sizeEntry = Scale(UI_frame, from_=3, to=25, resolution=1, orient=HORIZONTAL, label="Data Size")
sizeEntry.grid(row=1, column=0, padx=5, pady=5)

minEntry = Scale(UI_frame, from_=0, to=10, resolution=1, orient=HORIZONTAL, label="Min Value")
minEntry.grid(row=1, column=1, padx=5, pady=5)

maxEntry = Scale(UI_frame, from_=10, to=100, resolution=1, orient=HORIZONTAL, label="Max Value")
maxEntry.grid(row=1, column=2, padx=5, pady=5)

Button(UI_frame, text="Generate", command=Generate, bg='white').grid(row=1, column=3, padx=5, pady=5)

root.mainloop()
```

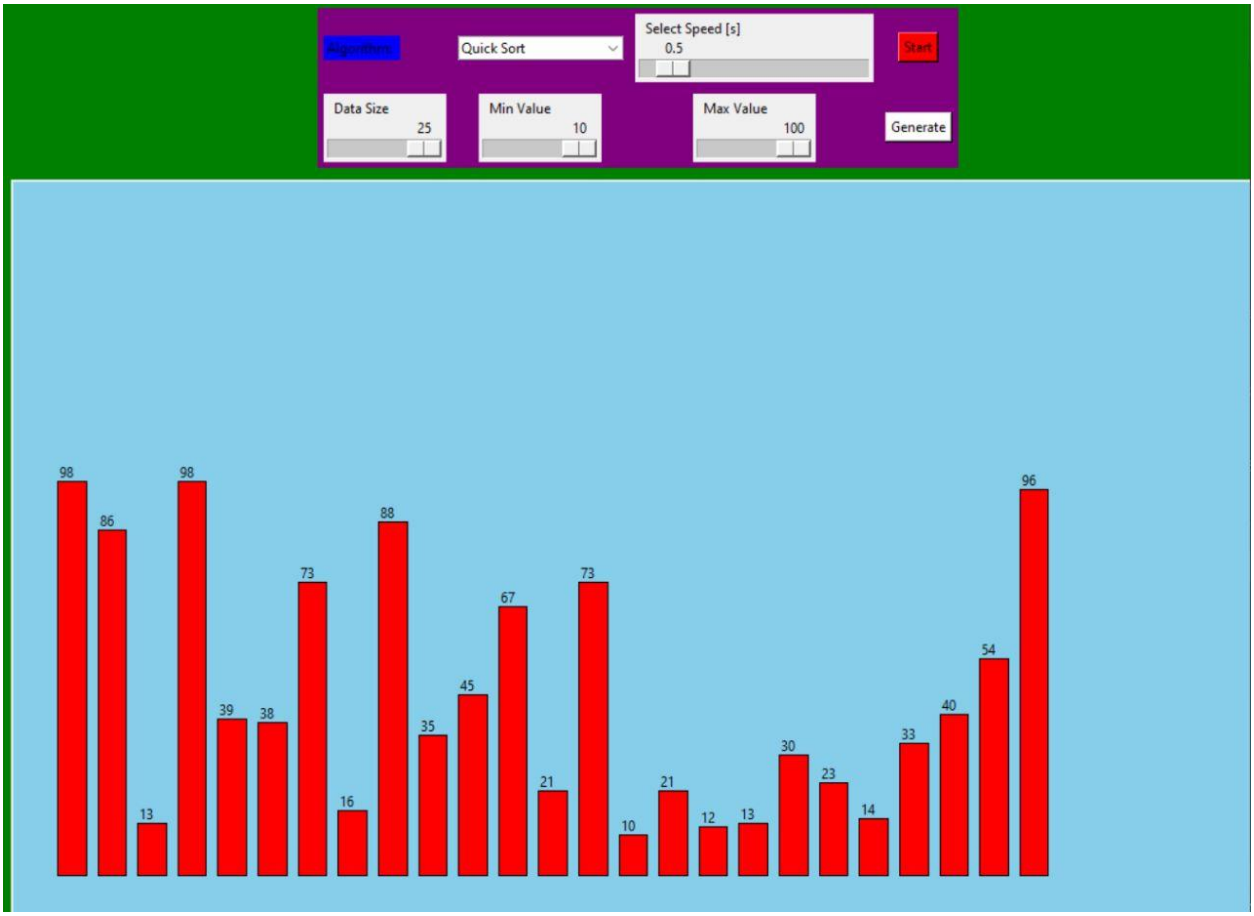
Input Section:

In the input field, we know that there are two types of user input called Random input generators & manually input integer values divided by space.



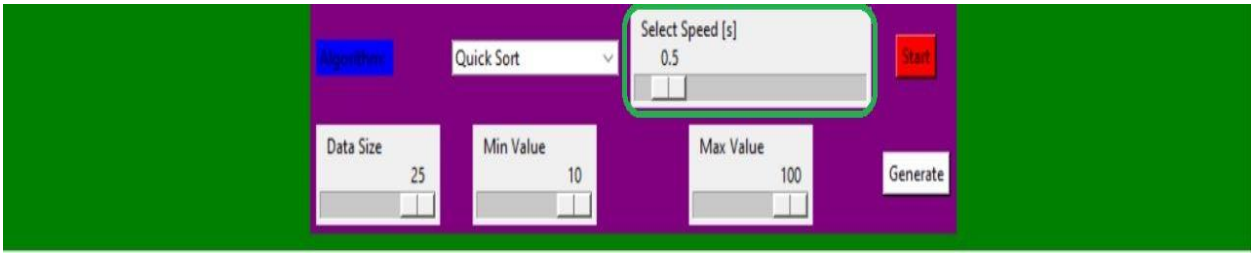
Generate Data:

After giving the Max value, Min value & Data size, we press generate button to generate the data into canvas.



Speed Control:

To control the speed of our algorithm we change the select speed slider. If we want the sorting in a slower manner, then we increase the slider or vice versa.



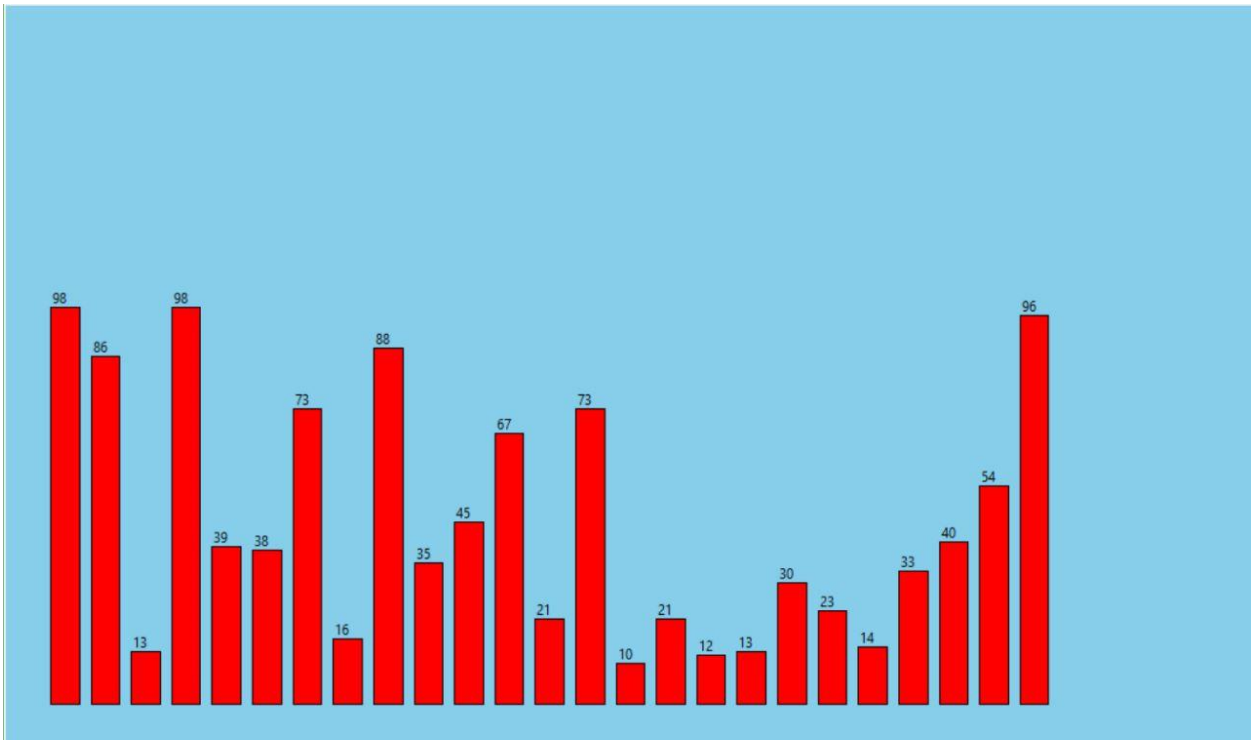
Function:

The logic is simple, it starts from the leftmost element and keep track of the index of smaller (or equal) elements. While traversing, if it finds a smaller element, it swaps the current element with array. Otherwise, it ignore the current element. Below is the code of the function of quicksort..

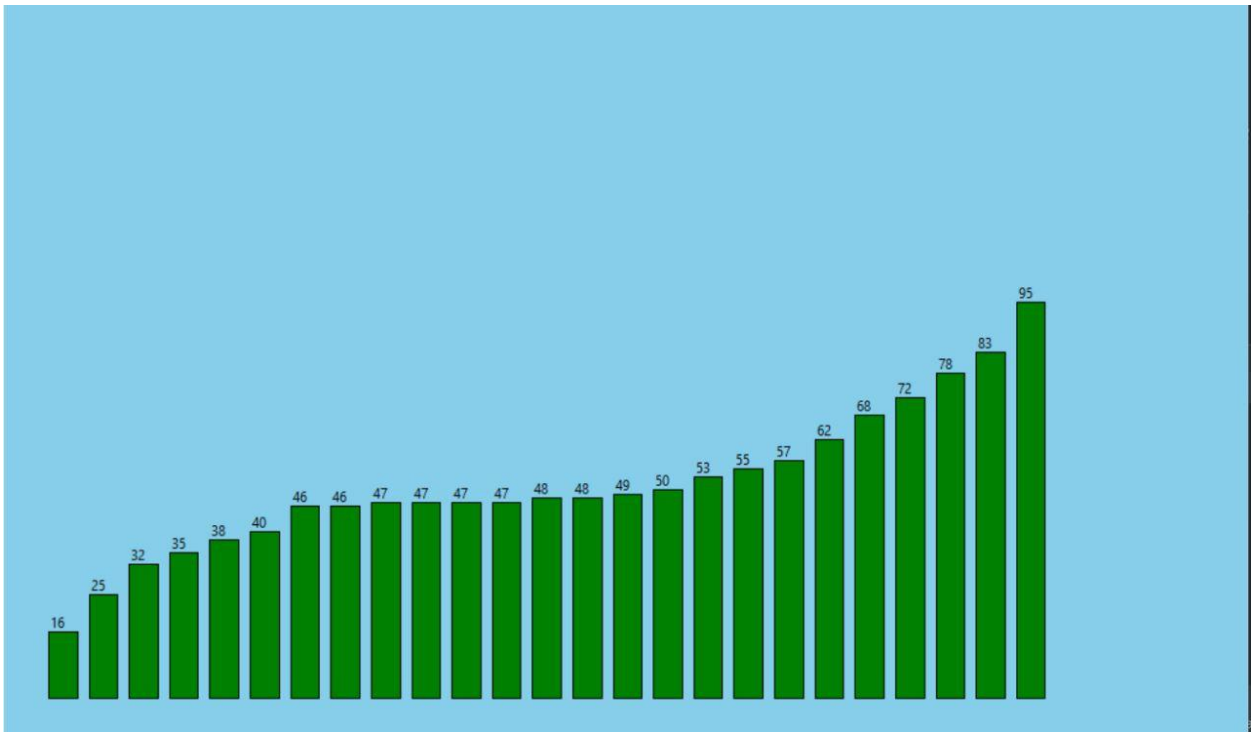
```python
# function
def drawData(data, colorArray):
    canvas.delete("all")
    c_height = 600
    c_width = 900
    x_width = c_width / (len(data) + 1)
    offset = 30
    spacing = 10
    normalizedData = [i / max(data) for i in data]
    for i, height in enumerate(normalizedData):
        # top left
        x0 = i * x_width + offset + spacing
        y0 = c_height - height * 340
        # bottom right
        x1 = (i + 1) * x_width + offset
        y1 = c_height

        canvas.create_rectangle(x0, y0, x1, y1, fill=colorArray[i])
        canvas.create_text(x0 + 2, y0, anchor=SW, text=str(data[i]))

    root.update_idletasks()
```

Sorting Data & Animation:

To start the sorting process, we press the Generate button..



After the sorting is finished



Conclusion

Our project successfully visualized the Quick Sort algorithm using Tkinter and Python, and provided users with a helpful tool for understanding how the algorithm works.

By breaking down the algorithm into smaller steps and creating visual aids, we were able to make it easier for users to follow along and gain a deeper understanding of the sorting process.

Our project also demonstrated the importance of visualizing algorithms for better understanding and provided a framework for future research and improvement.

For now, the visualizer only includes the bubble sort algorithm. But our team has worked hard to make such a framework that any sorting algorithm can be added and modified according to its need. And we will make sure to improve it further down the road as we add more complex algorithm into the mix.

By building on our findings and continuing to innovate in this area, we can help to make complex algorithms more accessible and understandable to a wider audience