Project Report

Project title: Sorting Visualizer

Members: Muhammad Talha Jawed, Zain Hassan, Dua Zehra

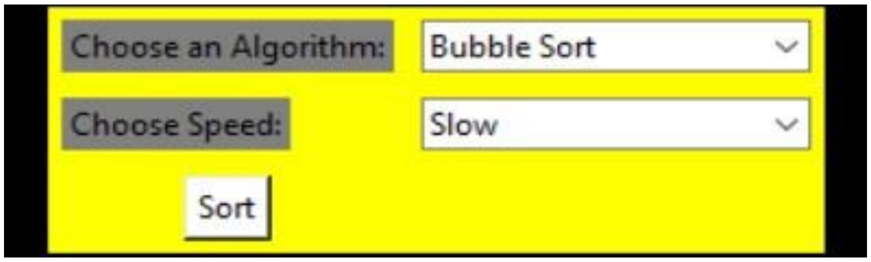
Objective: To be able to visualize different types of sorting algorithms.

Sorting is the process of arranging data into meaningful order so that you can analyze it more effectively. We have learnt sorting algorithms like Bubble sort, Merge sort etc. But it has been a problem that many people are unable to understand without the visualization of the program and so this is where our sorting visualizer can be useful.

Our program works on the following algorithms:

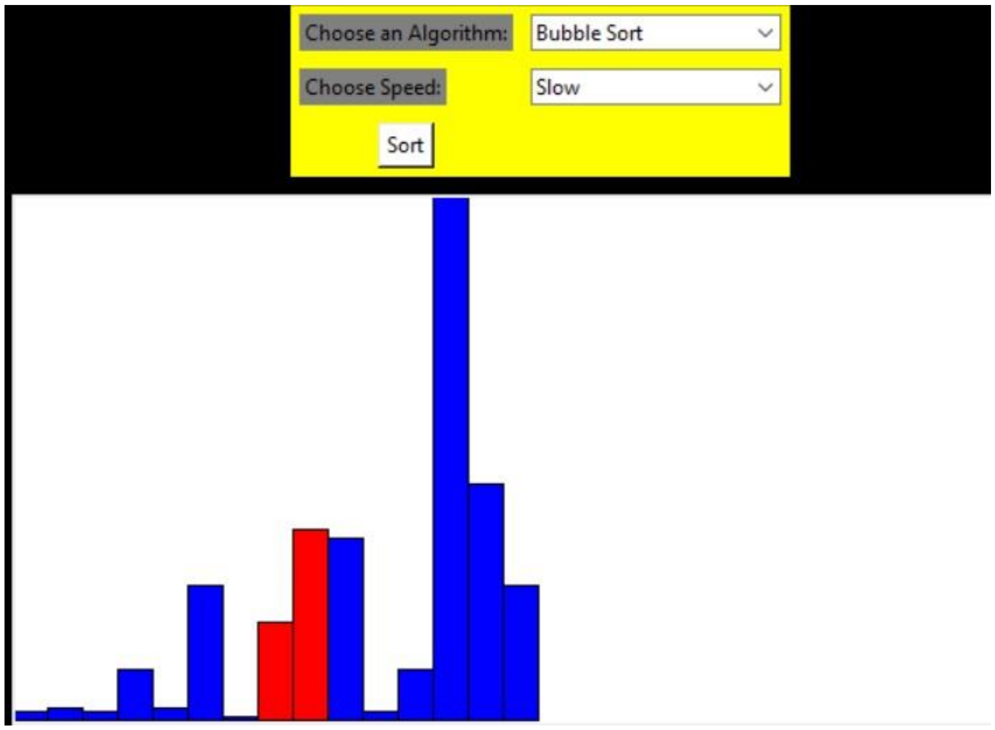
- Bubble sort
- Merge sort
- Selection sort
- Quick sort
- Bogo sort

We have constructed our program on a simple GUI that on running shows two options, one for choosing the sorting algorithm and the other for the choosing between the different speeds.

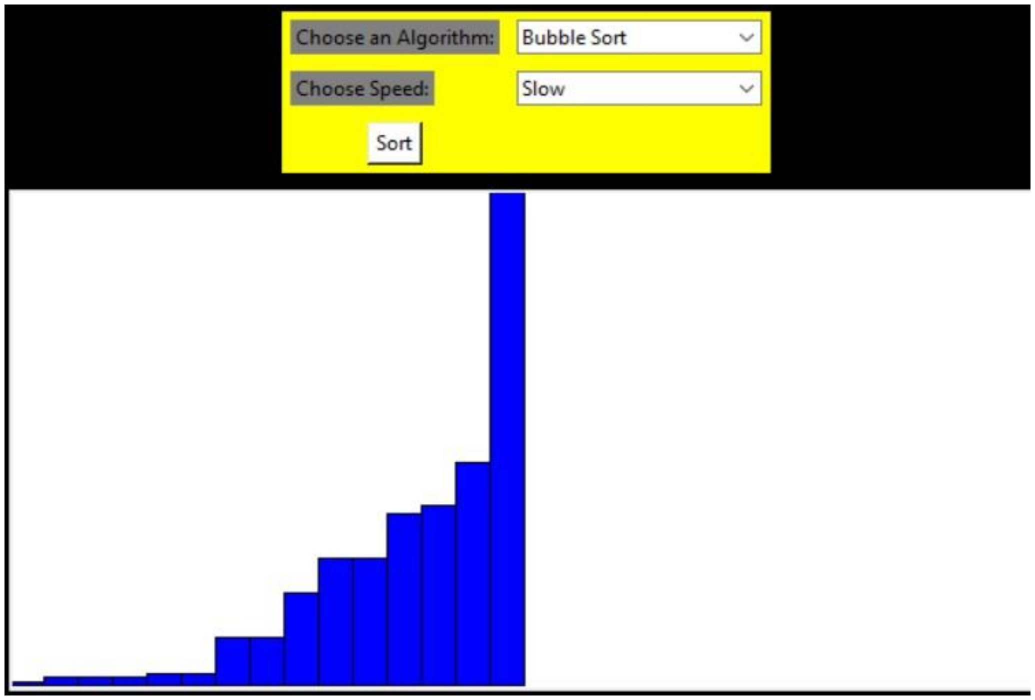


Once the code starts running, we are able to see an unsorted array get sorted by the selected algorithm at a particular speed. The rectangular bars represent the elements in the list.

Here are some snippets from the execution of the code,



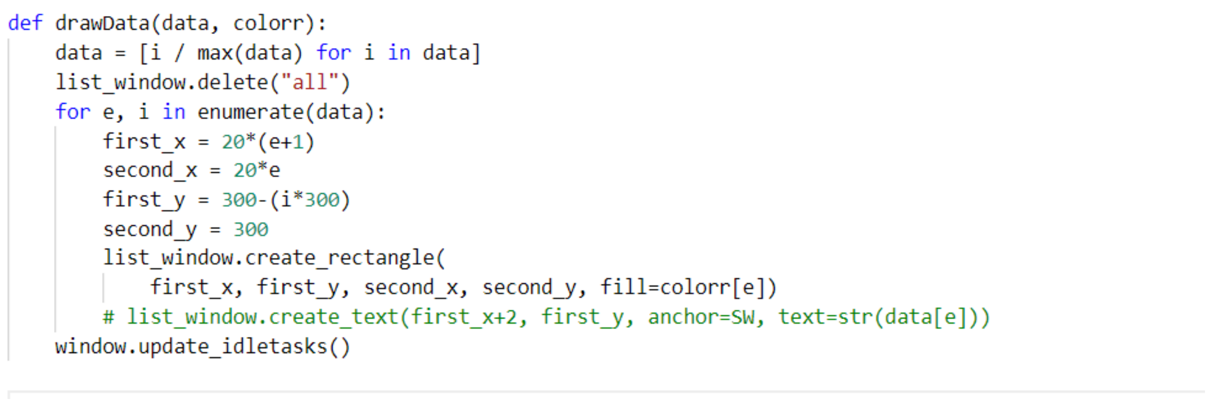
The red highlighted rectangles are the selected elements in the list on which the operation is being performed.



Here is the final output after the array has been sorted.

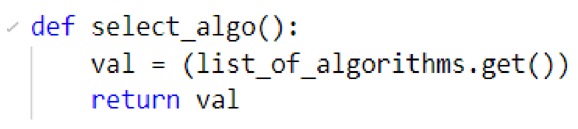
The Backend:

Made almost exclusively using ktinker GUI modules, the code contains, a few key functions. The GUI will not be documented in this report, since extensive documentation exists on ktinkers official repo.

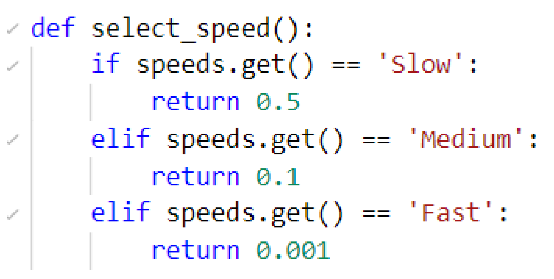


```python
def drawData(data, colorr):
    data = [i / max(data) for i in data]
    list_window.delete("all")
    for e, i in enumerate(data):
        first_x = 20*(e+1)
        second_x = 20*e
        first_y = 300-(i*300)
        second_y = 300
        list_window.create_rectangle(
            first_x, first_y, second_x, second_y, fill=colorr[e])
        # list_window.create_text(first_x+2, first_y, anchor=SW, text=str(data[e]))
    window.update_idletasks()
```

The drawData function, takes each list and makes respective rectangles out of it. This effectively shows how the list is changing with each iteration. The function is called every time a swap happens or a change is made to a list in a sorting function. The function takes two inputs, “data”, which is the list to be drawn, and “colorr” , which is a list containing the different colours that the bars may be filled with.

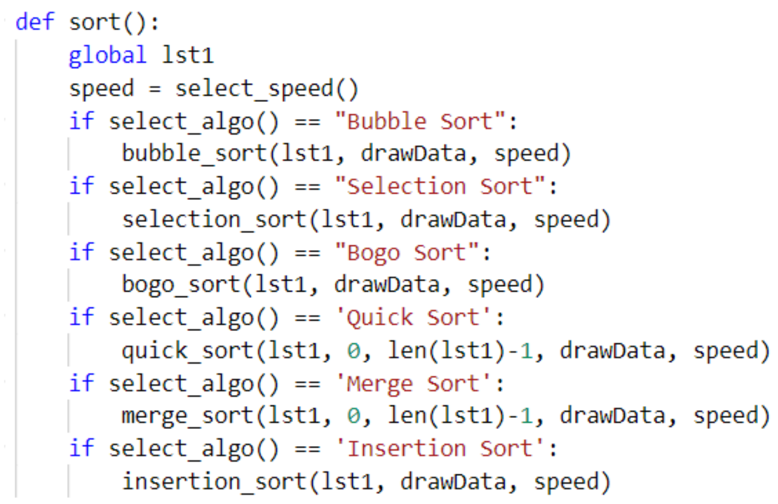


```python
def select_algo():
    val = (list_of_algorithms.get())
    return val
```



```python
def select_speed():
    if speeds.get() == 'Slow':
        return 0.5
    elif speeds.get() == 'Medium':
        return 0.1
    elif speeds.get() == 'Fast':
        return 0.001
```

The select_algo and select_speed functions are integrated in to the UI and essentially return the value selected by the tkinkter combo-box module. These are used when calling the sort functions, to specify which sorting algorithm will run and at what speed.



```python
def sort():
    global lst1
    speed = select_speed()
    if select_algo() == "Bubble Sort":
        bubble_sort(lst1, drawData, speed)
    if select_algo() == "Selection Sort":
        selection_sort(lst1, drawData, speed)
    if select_algo() == "Bogo Sort":
        bogo_sort(lst1, drawData, speed)
    if select_algo() == 'Quick Sort':
        quick_sort(lst1, 0, len(lst1)-1, drawData, speed)
    if select_algo() == 'Merge Sort':
        merge_sort(lst1, 0, len(lst1)-1, drawData, speed)
    if select_algo() == 'Insertion Sort':
        insertion_sort(lst1, drawData, speed)
```

Finally, the "sort" function, which is bound to the signal generated by the sort button in the program. Every time the button is pressed within the program, it runs a thread that calls this function. As this function is called, both "select_speed" and "select_algo" are called, and they return the value specified in the combo-boxes. Through a series of if-conditionals, the function finds the appropriate sorting algorithm to run and use the returned value of "select_speed" as a parameter.

Sorting Algorithms:
Each sorting algorithm has been modified to call the drawData function every time a change is made to a list or the list is returned. The algorithms also take a parameter known as either time_tick or speed which is effectively the time that the function waits before each iteration as the "time.sleep()" function is called before each iteration of the inner loops in a sorting algorithm.