DEPARTMENT OF INFORMATION TECHNOLOGY

GOVERNMENT COLLEGE OF ENGINEERING, AMRAVATI

RECURTION MINI PROJECT

IN

PYTHON

" SORTING VISUALIZER "

Prepared by: -

Mast. DEVESH DHOTE (19007055)

Mast. RITIK SINGH (19007061)

Miss. RADHIKA TIKAR (19007065)

Guided by:-

Prof. A.W.BHADE

(H.O.D.)

CERTIFICATE

This to certify that this project work is submitted by DEVESH DHOTE, RITIK SINGH, RADHIKA TIKAR having ID's 19007055, 19007061, 19007065 respectively of semester III of B.Tech in Information Technology was carried out by them under the guidance & supervision of Prof. A.W.BHADE during academic year 2020-21 for the

Course Title:-Data Structure and Algorithm Lab Course Code:-ITU324

Date: 26/12/2020

Head of Dept.	Faculty
(Prof.A.W.Bhade)	(Prof.A.W.Bhade)

ACKNOWLEDGEMENT

I wish to express my deep gratitude and sincere thanks to H.O.D, A.W.BHADE, Department Of Information Technology for his encouragement and for all the facilities that he provided for this project work .I sincerely appreciate this magnanimity by taking me into his fold for which I shall remain indebted to him .

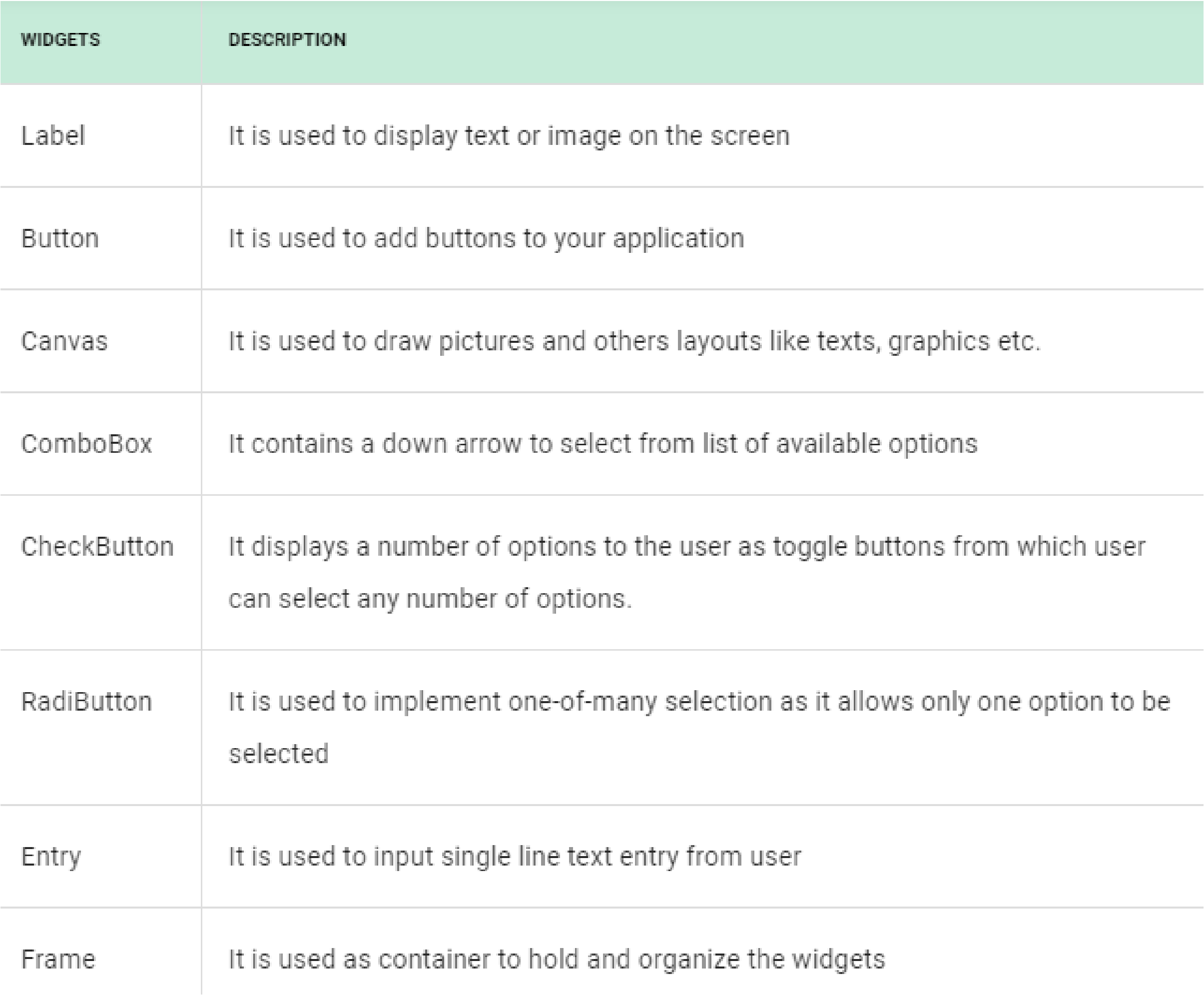
I extend my hearty thanks to Prof. A.W.BHADE, Faculty data structure and algorithm (ITU-324), who guided me to the successful completion of this project I take this opportunity to express my deep sense of gratitude for his invaluable guidance ,constant encouragement ,constructive comments ,sympathetic attitude and immense motivation ,which has sustained my efforts at all stages of this project work.

I can't forget to offer my sincere thanks to my classmates who help me to carry out this project work successfully & for their valuable advice & support, which I receive from them time to time.

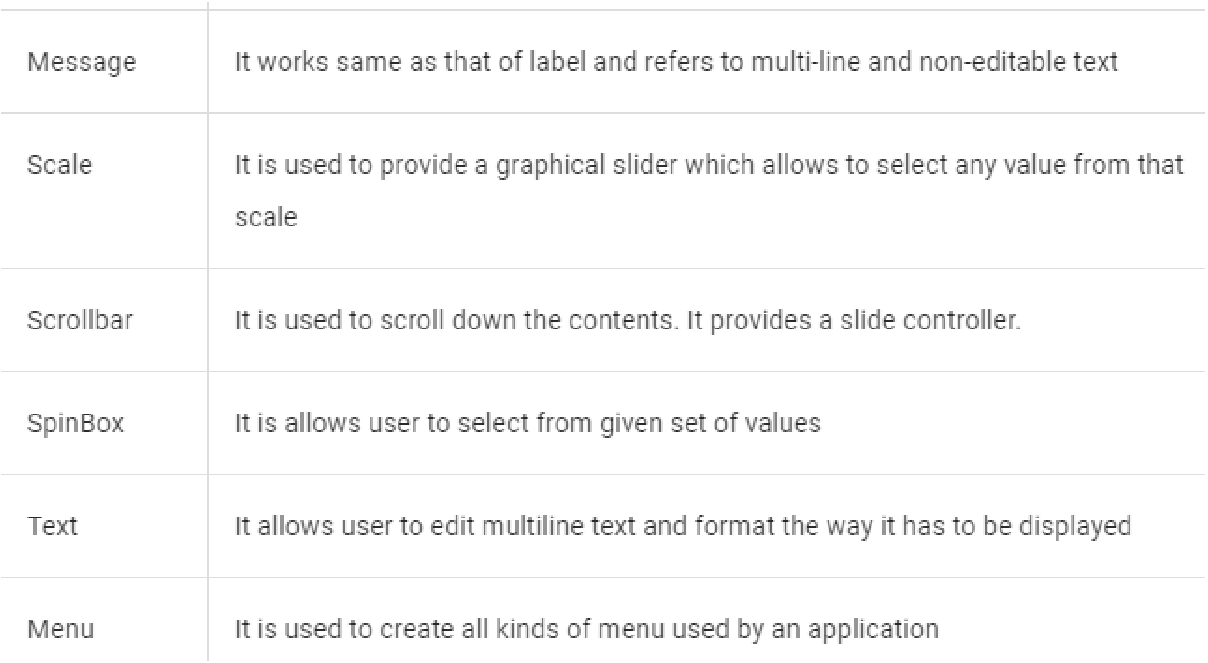
INTRODUCTION

TK-INTER : Tkinter is the most commonly used library for developing GUI (Graphical User Interface) in Python. It is a standard Python interface to the Tk GUI toolkit shipped with Python. As Tk and Tkinter are available on most of the Unix platforms as well as on the Windows system, developing GUI applications with Tkinter becomes the fastest and easiest.

Widgets in Tkinter are the elements of GUI application which provides various controls (such as Labels, Buttons, ComboBoxes, CheckBoxes, MenuBars, RadioButtons and many more) to users to interact with the application.



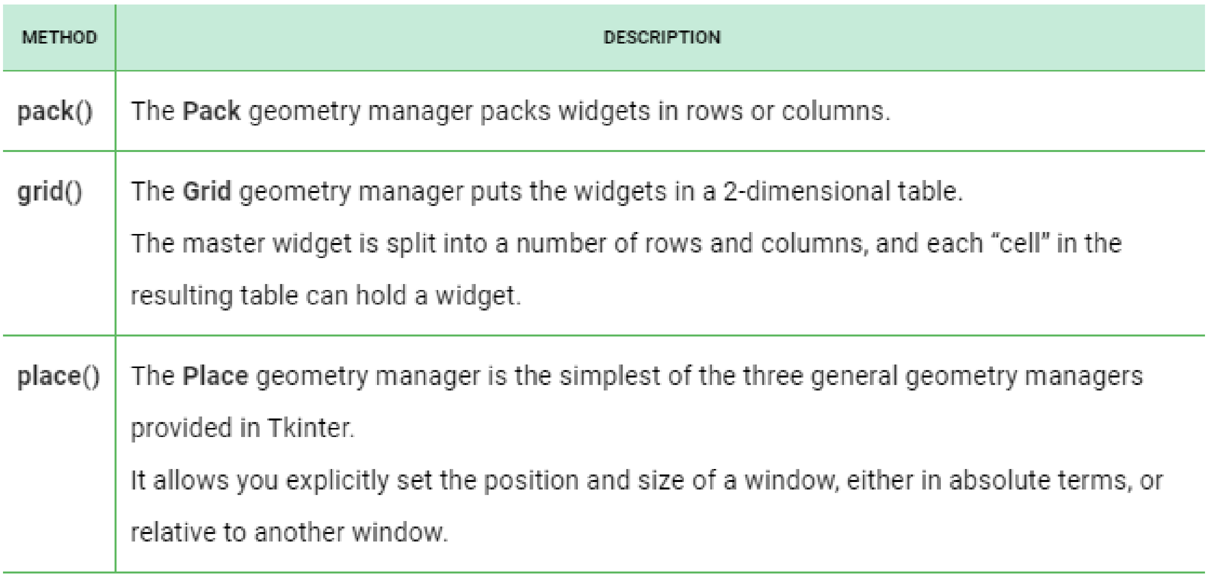
WIDGETS	DESCRIPTION
Label	It is used to display text or image on the screen
Button	It is used to add buttons to your application
Canvas	It is used to draw pictures and others layouts like texts, graphics etc.
ComboBox	It contains a down arrow to select from list of available options
CheckButton	It displays a number of options to the user as toggle buttons from which user can select any number of options.
RadiButton	It is used to implement one-of-many selection as it allows only one option to be selected
Entry	It is used to input single line text entry from user
Frame	It is used as container to hold and organize the widgets



Message	It works same as that of label and refers to multi-line and non-editable text
Scale	It is used to provide a graphical slider which allows to select any value from that scale
Scrollbar	It is used to scroll down the contents. It provides a slide controller.
SpinBox	It is allows user to select from given set of values
Text	It allows user to edit multiline text and format the way it has to be displayed
Menu	It is used to create all kinds of menu used by an application

Geometry Management

Creating a new widget doesn't mean that it will appear on the screen. To display it, we need to call a special method: either **grid**, **pack**(example above), or **place**.



METHOD	DESCRIPTION
pack()	The **Pack** geometry manager packs widgets in rows or columns.
grid()	The **Grid** geometry manager puts the widgets in a 2-dimensional table. The master widget is split into a number of rows and columns, and each "cell" in the resulting table can hold a widget.
place()	The **Place** geometry manager is the simplest of the three general geometry managers provided in Tkinter. It allows you explicitly set the position and size of a window, either in absolute terms, or relative to another window.

Merge Sort :

Like QuickSort, Merge Sort is a Divide and Conquer algorithm. It divides the input array into two halves, calls itself for the two halves, and then merges the two sorted halves. The merge() function is used for merging two halves. The merge(arr, l, m, r) is a key process that assumes that arr[l..m] and arr[m+1..r] are sorted and merges the two sorted sub-arrays into one

```
MergeSort(arr[], l,  r)
If r > l
     1. Find the middle point to divide the array into two halves:
             middle m = (l+r)/2
     2. Call mergeSort for first half:
             Call mergeSort(arr, l, m)
     3. Call mergeSort for second half:
             Call mergeSort(arr, m+1, r)
     4. Merge the two halves sorted in step 2 and 3:
             Call merge(arr, l, m, r)
```

Time Complexity: Sorting arrays on different machines. Merge Sort is a recursive algorithm and time complexity can be expressed as following recurrence relation.
$T(n) = 2T(n/2) + \theta(n)$
The above recurrence can be solved either using the Recurrence Tree method or the Master method. It falls in case II of Master Method and the solution of the recurrence is θ(nLogn). Time complexity of Merge Sort is θ(nLogn) in all 3 cases (worst, average and best) as merge sort always divides the array into two halves and takes linear time to merge two halves.
Auxiliary Space: O(n)

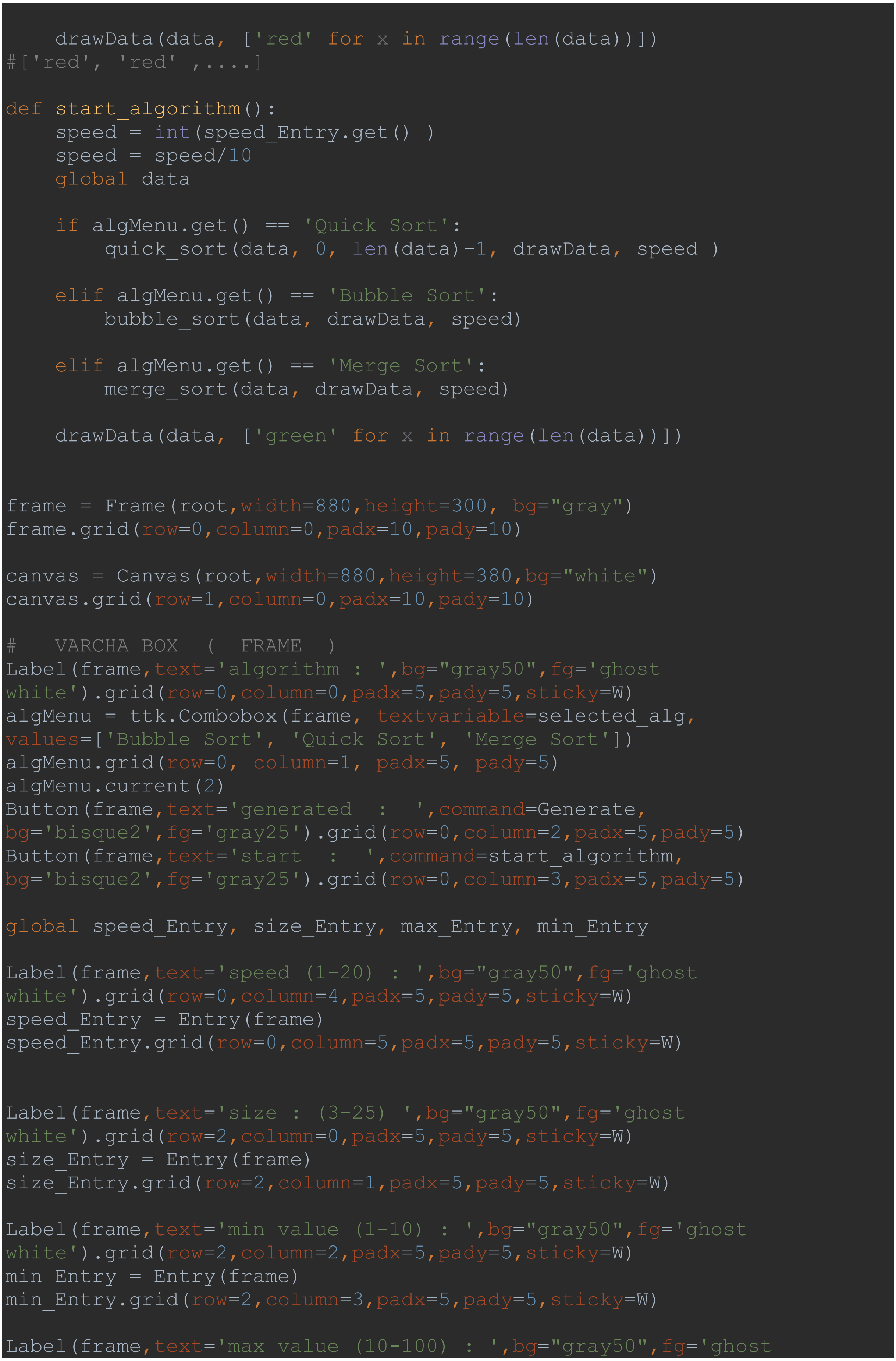
Sorting Visualizer :

Sorting is nothing but alphabetizing, categorizing, arranging or putting items in an ordered sequence. It is a key fundamental operation in the field of computer science. It is of extreme importance because it adds usefulness to data. In this papers, we have compared five important sorting algorithms (Bubble, Quick, Selection, Insertion and Merge). We have developed a program in C# and experimented with the input values 1-150, 1-300 and 1-950. The performance and efficiency of these algorithms in terms of CPU time consumption has been recorded and presented in tabular and graphical form

Source Code :

MAIN FILE

```
from tkinter import *
from tkinter import ttk
import random
from mergesort import merge_sort
from quicksort import quick_sort
from mergesort import merge_sort

root = Tk()
w = root.winfo_screenwidth()
h= root.winfo_screenheight()
C_width = int(w)
C_height = int(h)
root.geometry(f"900x500+{C_width//2-450}+{C_height//2-320}")
root.minsize(900,500)
root.maxsize(900,500)
root.title("  Sorting visualizer  ")
root.config(bg='black')

# varibles
selected_alg = StringVar()
data = []

def drawData(data, colorArray):
    canvas.delete("all")
    c_height = 380
    c_width = 600
    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 corner
        x0 = i * x_width + offset + spacing
        y0 = c_height - height * 340
        #bottom right corner
        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()

def Generate():
    global data

    minVal = int(min_Entry.get())
    maxVal = int(max_Entry.get())
    size = int(size_Entry.get())

    data = []
    for _ in range(size):
        data.append(random.randrange(minVal, maxVal+1))
```



```
    drawData(data, ['red' for x in range(len(data))])
#['red', 'red' ,....]

def start_algorithm():
    speed = int(speed_Entry.get() )
    speed = speed/10
    global data

    if algMenu.get() == 'Quick Sort':
        quick_sort(data, 0, len(data)-1, drawData, speed )

    elif algMenu.get() == 'Bubble Sort':
        bubble_sort(data, drawData, speed)

    elif algMenu.get() == 'Merge Sort':
        merge_sort(data, drawData, speed)

    drawData(data, ['green' for x in range(len(data))])


frame = Frame(root,width=880,height=300, bg="gray")
frame.grid(row=0,column=0,padx=10,pady=10)

canvas = Canvas(root,width=880,height=380,bg="white")
canvas.grid(row=1,column=0,padx=10,pady=10)

#   VARCHA BOX  (  FRAME  )
Label(frame,text='algorithm : ',bg="gray50",fg='ghost
white').grid(row=0,column=0,padx=5,pady=5,sticky=W)
algMenu = ttk.Combobox(frame, textvariable=selected_alg,
values=['Bubble Sort', 'Quick Sort', 'Merge Sort'])
algMenu.grid(row=0, column=1, padx=5, pady=5)
algMenu.current(2)
Button(frame,text='generated  :  ',command=Generate,
bg='bisque2',fg='gray25').grid(row=0,column=2,padx=5,pady=5)
Button(frame,text='start  :  ',command=start_algorithm,
bg='bisque2',fg='gray25').grid(row=0,column=3,padx=5,pady=5)

global speed_Entry, size_Entry, max_Entry, min_Entry

Label(frame,text='speed (1-20) : ',bg="gray50",fg='ghost
white').grid(row=0,column=4,padx=5,pady=5,sticky=W)
speed_Entry = Entry(frame)
speed_Entry.grid(row=0,column=5,padx=5,pady=5,sticky=W)


Label(frame,text='size : (3-25) ',bg="gray50",fg='ghost
white').grid(row=2,column=0,padx=5,pady=5,sticky=W)
size_Entry = Entry(frame)
size_Entry.grid(row=2,column=1,padx=5,pady=5,sticky=W)

Label(frame,text='min value (1-10) : ',bg="gray50",fg='ghost
white').grid(row=2,column=2,padx=5,pady=5,sticky=W)
min_Entry = Entry(frame)
min_Entry.grid(row=2,column=3,padx=5,pady=5,sticky=W)

Label(frame,text='max value (10-100) : ',bg="gray50",fg='ghost
white').grid(row=2,column=4,padx=5,pady=5,sticky=W)
max_Entry = Entry(frame)
max_Entry.grid(row=2,column=5,padx=5,pady=5,sticky=W)

root.mainloop()
```

MERGE SORT

```
import time

def merge_sort(data, drawData, timeTick):
    merge_sort_alg(data, 0, len(data) - 1, drawData, timeTick)


def merge_sort_alg(data, left, right, drawData, timeTick):
    if left < right:
        middle = (left + right) // 2
        merge_sort_alg(data, left, middle, drawData, timeTick)
        merge_sort_alg(data, middle + 1, right, drawData, timeTick)
        merge(data, left, middle, right, drawData, timeTick)


def merge(data, left, middle, right, drawData, timeTick):
    drawData(data, getColorArray(len(data), left, middle, right))
    time.sleep(timeTick)

    leftPart = data[left:middle + 1]
    rightPart = data[middle + 1: right + 1]

    leftIdx = rightIdx = 0

    for dataIdx in range(left, right + 1):
        if leftIdx < len(leftPart) and rightIdx < len(rightPart):
            if leftPart[leftIdx] <= rightPart[rightIdx]:
                data[dataIdx] = leftPart[leftIdx]
                leftIdx += 1
            else:
                data[dataIdx] = rightPart[rightIdx]
                rightIdx += 1

        elif leftIdx < len(leftPart):
            data[dataIdx] = leftPart[leftIdx]
            leftIdx += 1
        else:
            data[dataIdx] = rightPart[rightIdx]
            rightIdx += 1

    drawData(data, ["green" if x >= left and x <= right else "white"
for x in range(len(data))])

    time.sleep(timeTick)

def getColorArray(leght, left, middle, right):
    colorArray = []

    for i in range(leght):
        if i >= left and i <= right:
            if i >= left and i <= middle:
                colorArray.append("yellow")
            else:
                colorArray.append("pink")
        else:
            colorArray.append("white")

    return colorArray
```

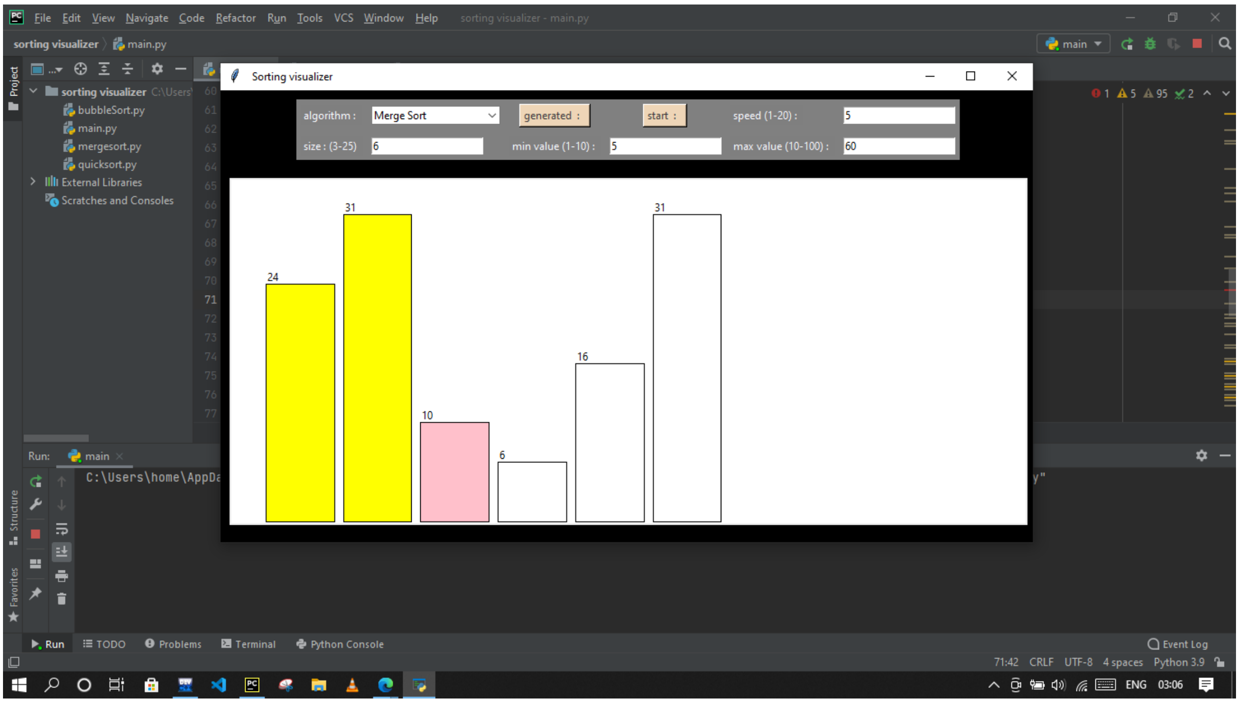
QUICK SORT

```
import time

def partition(data, head, tail, drawData, timeTick):
    border = head
    pivot = data[tail]

    drawData(data, getColorArray(len(data), head, tail, border,
border))
    time.sleep(timeTick)

    for j in range(head, tail):
        if data[j] < pivot:
            drawData(data, getColorArray(len(data), head, tail, border,
j, True))
            time.sleep(timeTick)

            data[border], data[j] = data[j], data[border]
            border += 1

        drawData(data, getColorArray(len(data), head, tail, border, j))
        time.sleep(timeTick)

    #swap pivot with border value
    drawData(data, getColorArray(len(data), head, tail, border, tail,
True))
    time.sleep(timeTick)

    data[border], data[tail] = data[tail], data[border]

    return border

def quick_sort(data, head, tail, drawData, timeTick):
    if head < tail:
        partitionIdx = partition(data, head, tail, drawData, timeTick)

        #LEFT PARTITION
        quick_sort(data, head, partitionIdx-1, drawData, timeTick)

        #RIGHT PARTITION
        quick_sort(data, partitionIdx+1, tail, drawData, timeTick)


def getColorArray(dataLen, head, tail, border, currIdx, isSwaping =
False):
    colorArray = []
    for i in range(dataLen):
        #base coloring
        if i >= head and i <= tail:
            colorArray.append('gray')
        else:
            colorArray.append('white')

        if i == tail:
            colorArray[i] = 'blue'
        elif i == border:
            colorArray[i] = 'red'
        elif i == currIdx:
            colorArray[i] = 'yellow'

        if isSwaping:
            if i == border or i == currIdx:
                colorArray[i] = 'green'

    return colorArray
```

BUBBLE SORT

```
import time

def bubble_sort(data, drawData, timeTick):
    for _ in range(len(data)-1):
        for j in range(len(data)-1):
            if data[j] > data[j+1]:
                data[j], data[j+1] = data[j+1], data[j]
                drawData(data, ['green' if x == j or x == j+1 else
'red' for x in range(len(data))] )

                time.sleep(timeTick)
    drawData(data, ['green' for x in range(len(data))])
```

Output :



algorithm : Merge Sort

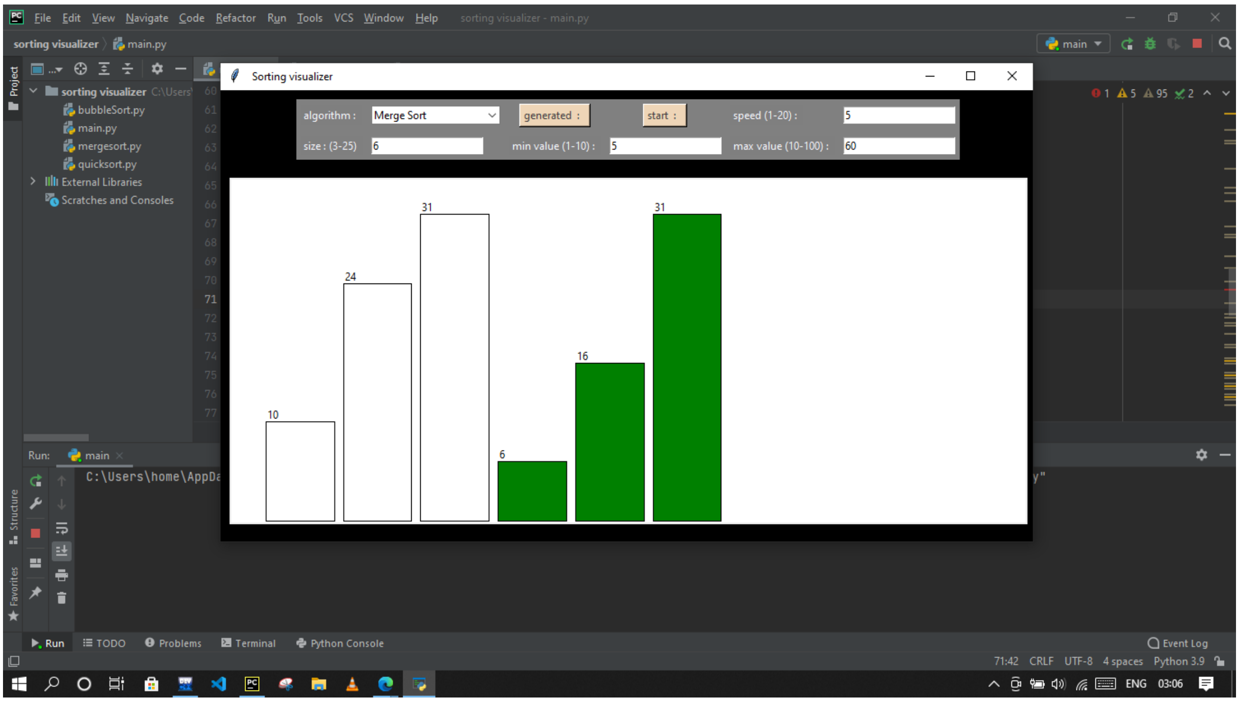
size : (3-25) 6

min value (1-10) : 5

speed (1-20) : 5

max value (10-100) : 60

```
C:\Users\home\AppDa
```



algorithm : Merge Sort

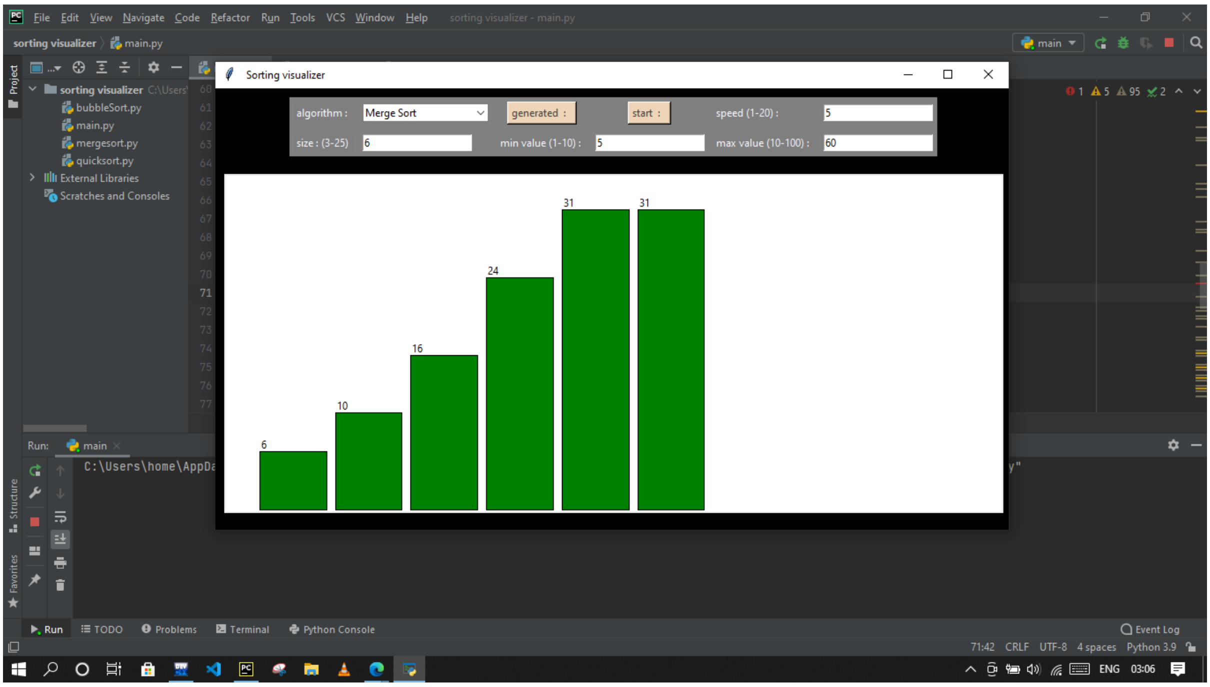
size : (3-25) 6

min value (1-10) : 5

speed (1-20) : 5

max value (10-100) : 60

```
C:\Users\home\AppDa
```



algorithm : Merge Sort

generated :

start :

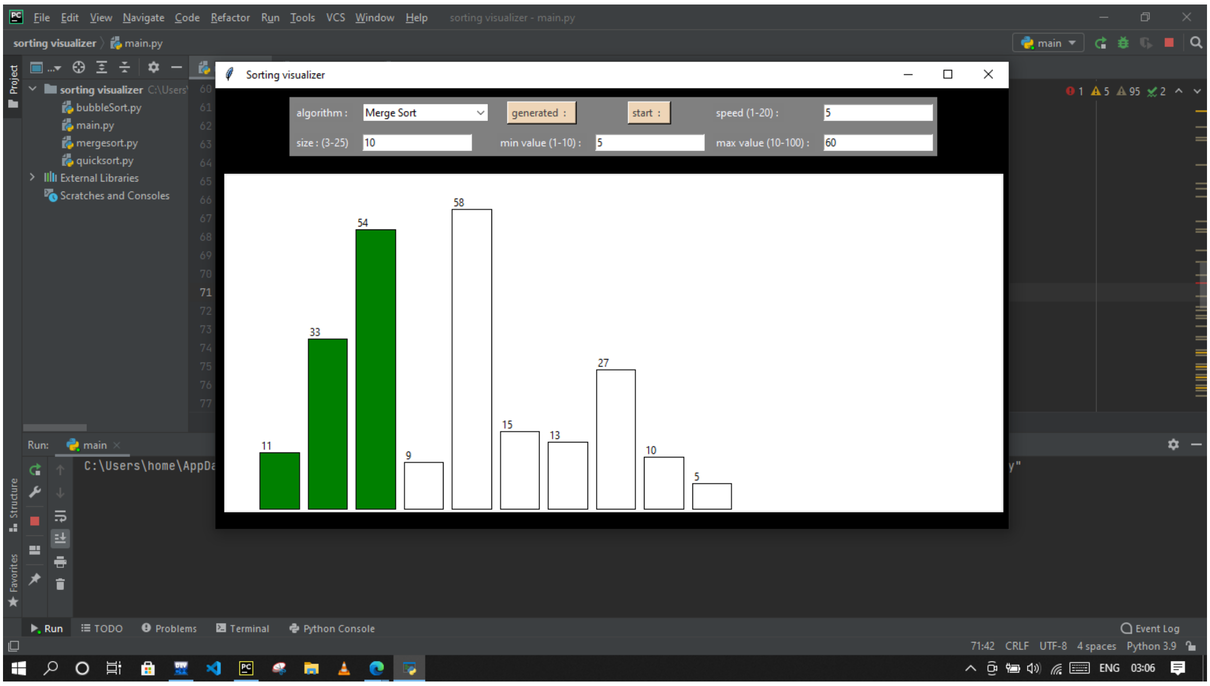
speed (1-20) : 5

size : (3-25) 6

min value (1-10) : 5

max value (10-100) : 60

```
C:\Users\home\AppDa
```



algorithm : Merge Sort

generated :

start :

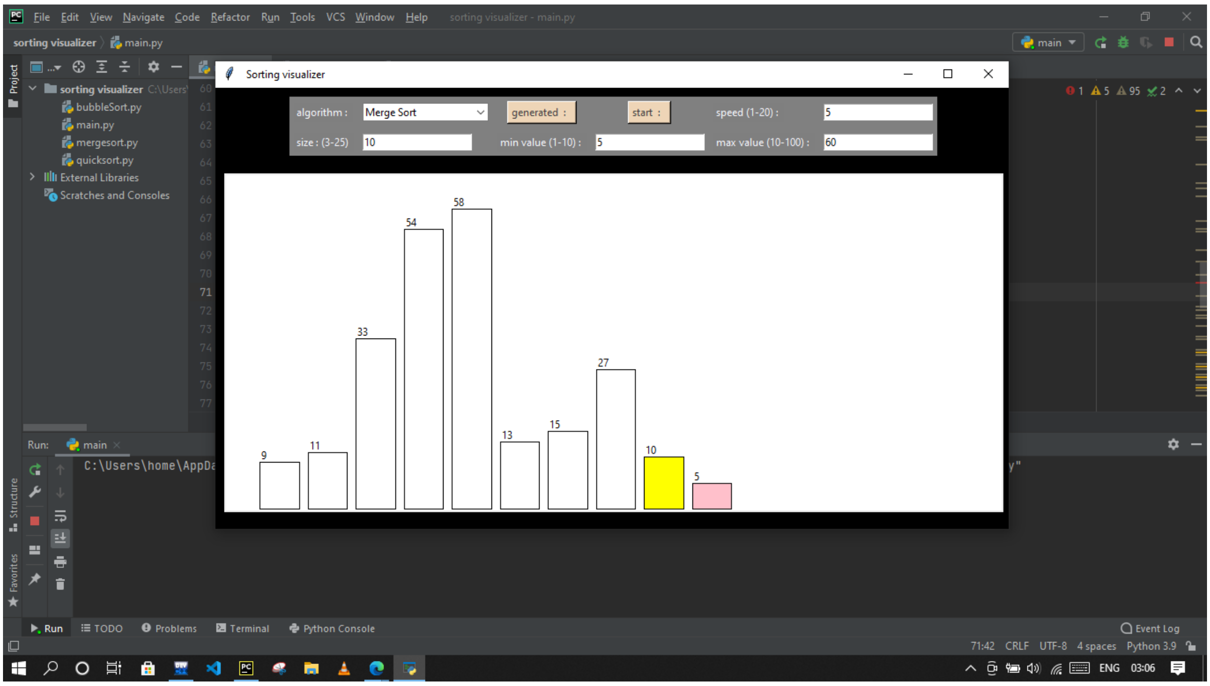
speed (1-20) : 5

size : (3-25) 10

min value (1-10) : 5

max value (10-100) : 60

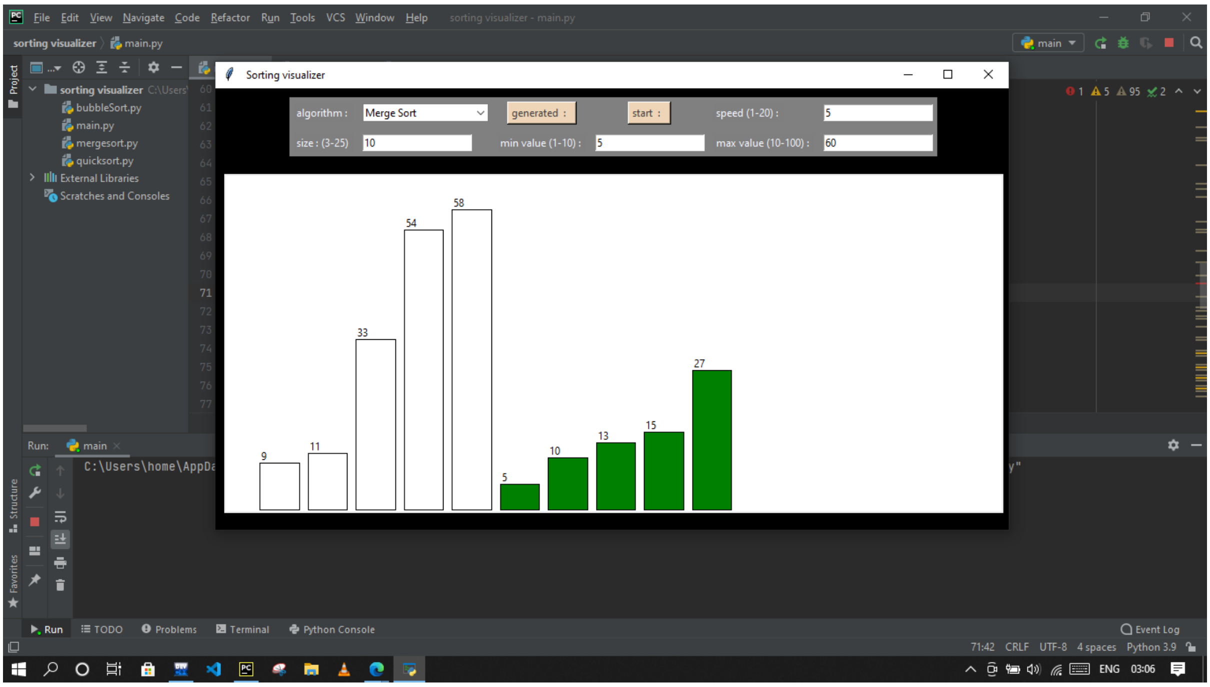
```
C:\Users\home\AppDa
```



Sorting visualizer

algorithm : Merge Sort
speed (1-20) : 5
size : (3-25) 10
min value (1-10) : 5
max value (10-100) : 60

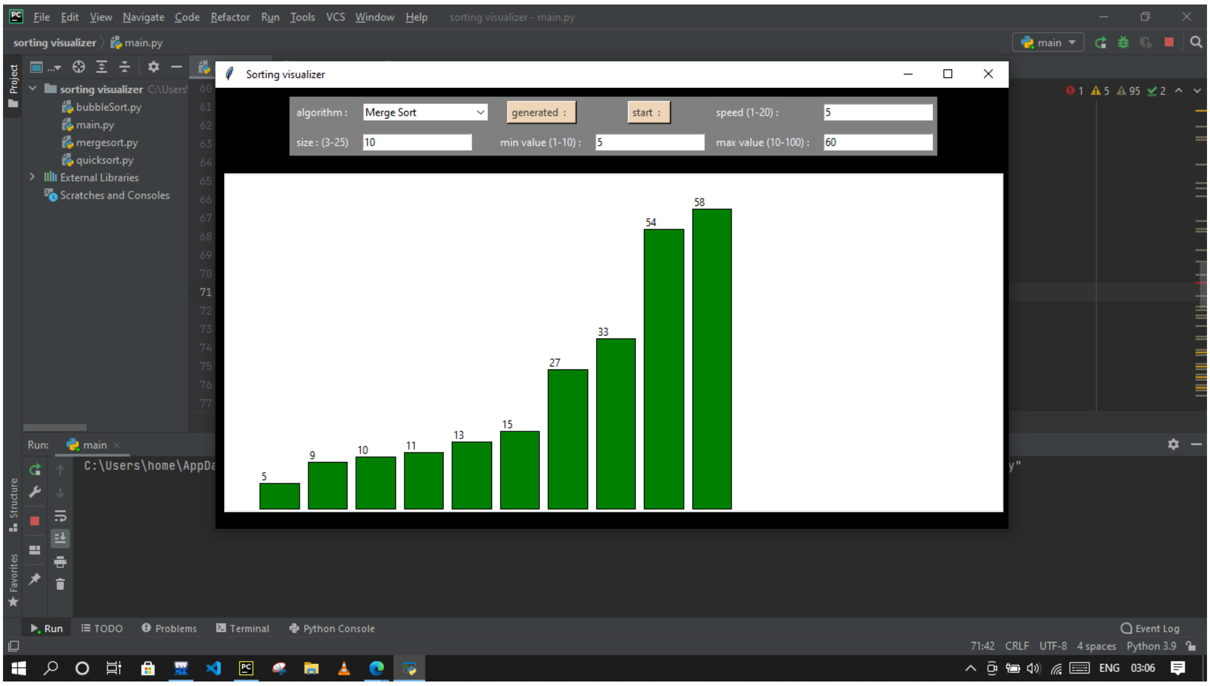
```
C:\Users\home\AppDa
```



Sorting visualizer

algorithm : Merge Sort
speed (1-20) : 5
size : (3-25) 10
min value (1-10) : 5
max value (10-100) : 60

```
C:\Users\home\AppDa
```


File Edit View Navigate Code Refactor Run Tools VCS Window Help sorting visualizer - main.py
sorting visualizer main.py
main
sorting visualizer C:\Users
bubbleSort.py
main.py
mergesort.py
quicksort.py
External Libraries
Scratches and Consoles
Sorting visualizer
generated :
start :
5
9
10
11
13
15
27
33
54
58
Run: main
Run TODO Problems Terminal Python Console Event Log
71:42 CRLF UTF-8 4 spaces Python 3.9
ENG 03:06

algorithm : Merge Sort

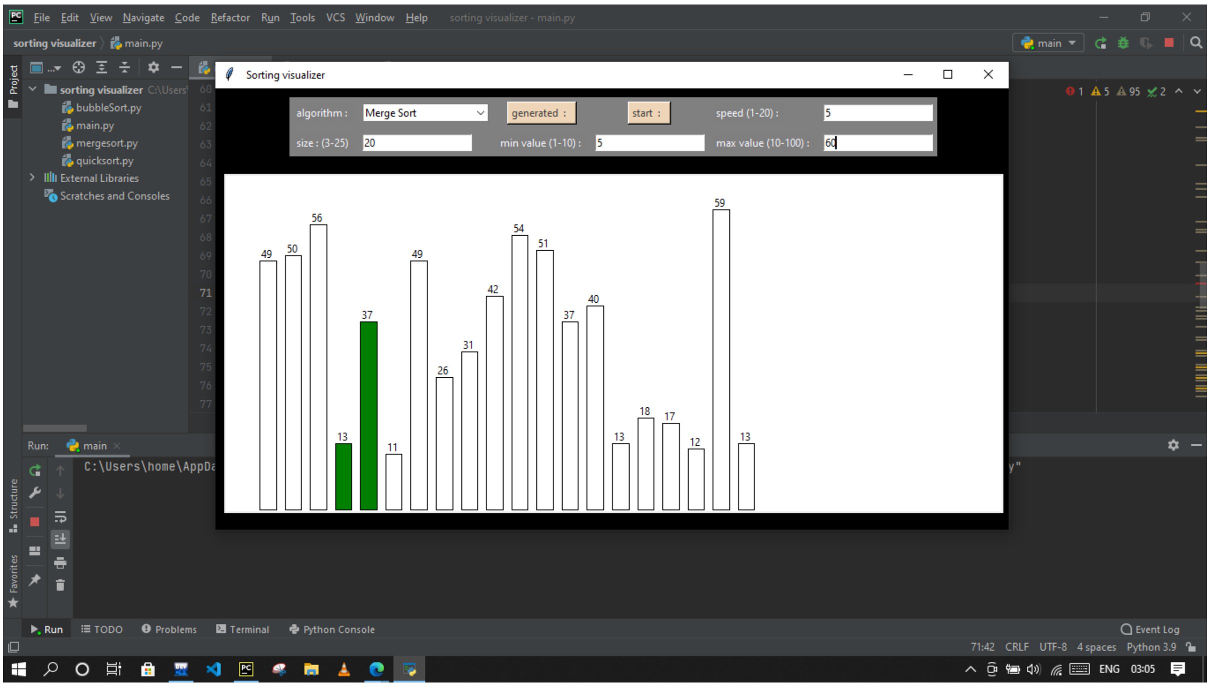
speed (1-20) : 5

size : (3-25) 10

min value (1-10) : 5

max value (10-100) : 60

```
C:\Users\home\AppDa
```


File Edit View Navigate Code Refactor Run Tools VCS Window Help sorting visualizer - main.py
sorting visualizer main.py
main
sorting visualizer C:\Users
bubbleSort.py
main.py
mergesort.py
quicksort.py
External Libraries
Scratches and Consoles
Sorting visualizer
generated :
start :
49
50
56
13
37
11
49
26
31
42
54
51
37
40
13
18
17
12
59
13
Run: main
Run TODO Problems Terminal Python Console Event Log
71:42 CRLF UTF-8 4 spaces Python 3.9
ENG 03:05

algorithm : Merge Sort

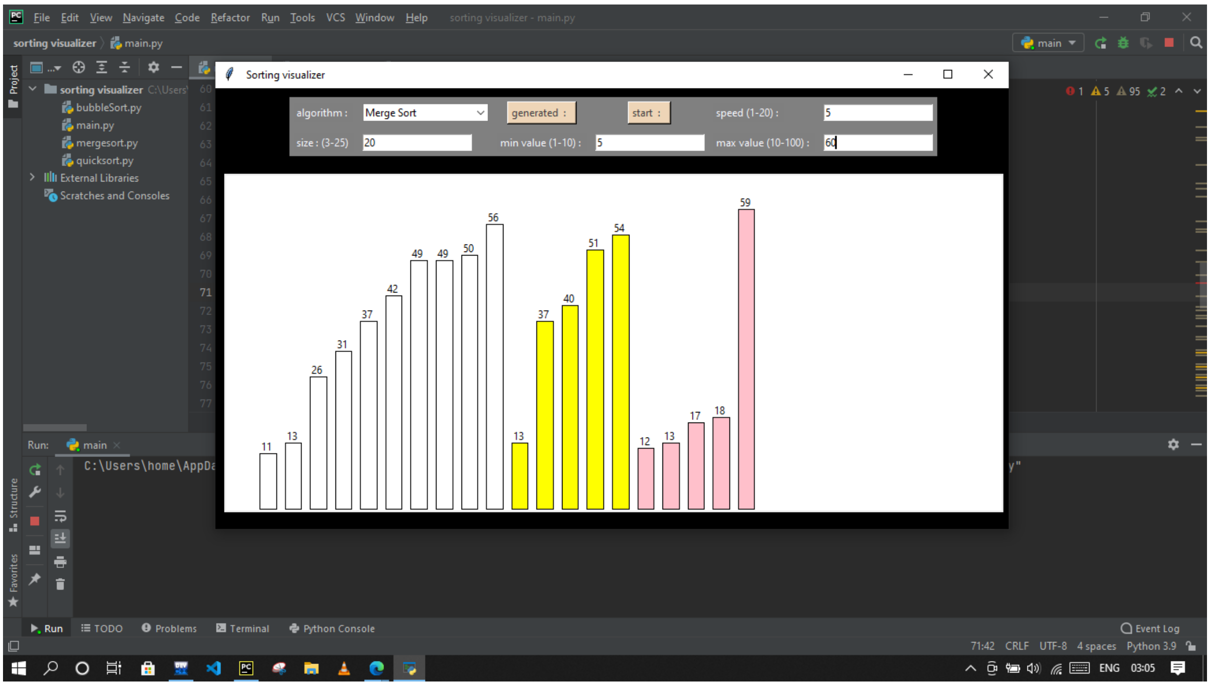
speed (1-20) : 5

size : (3-25) 20

min value (1-10) : 5

max value (10-100) : 60

```
C:\Users\home\AppDa
```



algorithm : Merge Sort

generated :

start :

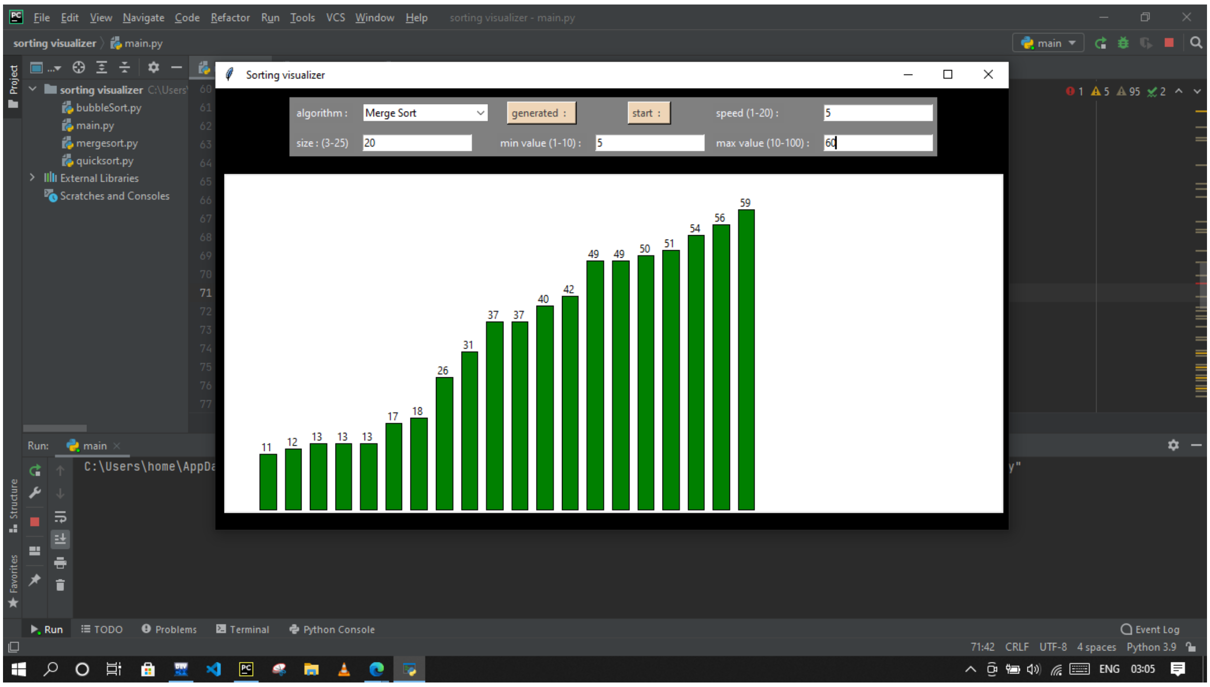
speed (1-20) : 5

size : (3-25) 20

min value (1-10) : 5

max value (10-100) : 60

```
C:\Users\home\AppDa
```



algorithm : Merge Sort

generated :

start :

speed (1-20) : 5

size : (3-25) 20

min value (1-10) : 5

max value (10-100) : 60

```
C:\Users\home\AppDa
```

THE END!!!

Conclusion:-

We make SORTING VISUALIZER successfully with the help of PYTHON language and using TK-inter module and it is very entertaining.