**DEPARTMENT Of INFORMATION TECHNOLOGY**

**Government college of engineering, Amravati**

****

RECURTION MINI PROJECT

IN

PYTHON

‘‘ SORTING VISUALIZER ’’

Prepared by: - Guided by:-

Mast. DEVESH DHOTE (19007055) **Prof. A.W.BHADE**

Mast. RITIK SINGH (19007061) ( H.O.D. )

Miss. RADHIKA TIKAR (19007065)

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 .

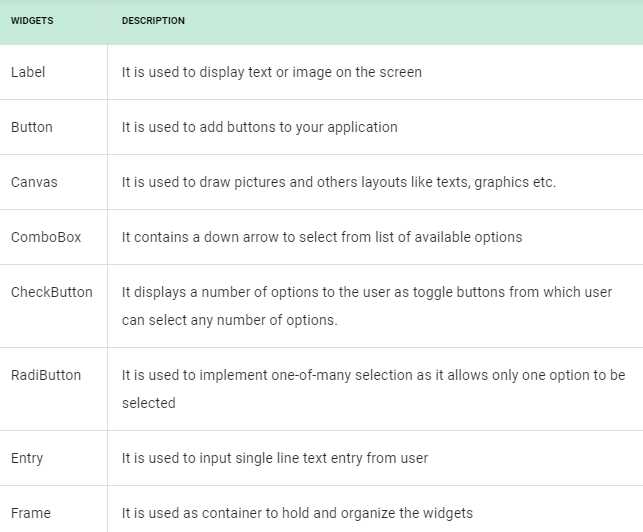
I extend my hearty thanks to Prof. A.W.BHADE, Faculty data structure and algorithm (ITU-324), who guided me to the successful completion of this project I take this opportunity to express my deep sense of gratitude for his invaluable guidance ,constant encouragement ,constructive comments ,sympathetic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

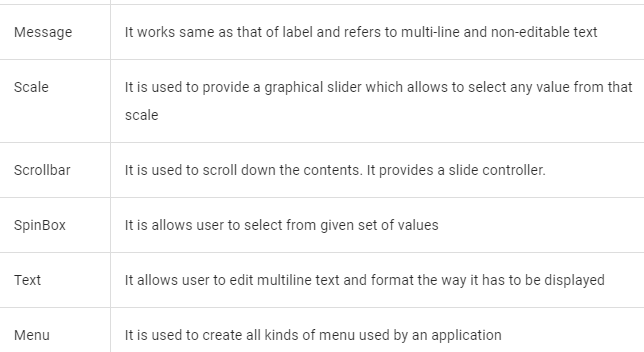
I can’t forget to offer my sincere thanks to my classmates who help me to carry out this project work successfully & for their valuable advice & support, which I receive from them time to time.

**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.

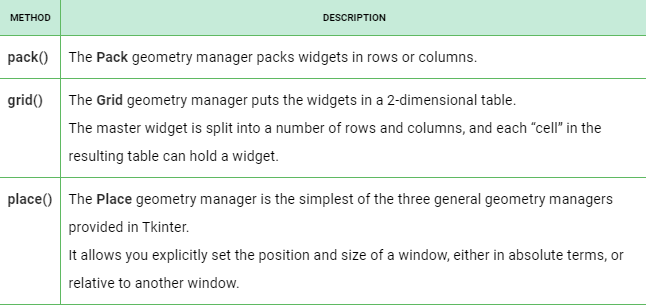
Widgets in Tkinter are the elements of GUI application which provides various controls (such as Labels, Buttons, ComboBoxes, CheckBoxes, MenuBars, RadioButtons and many more) to users to interact with the application.





**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**.

****

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) + θ(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

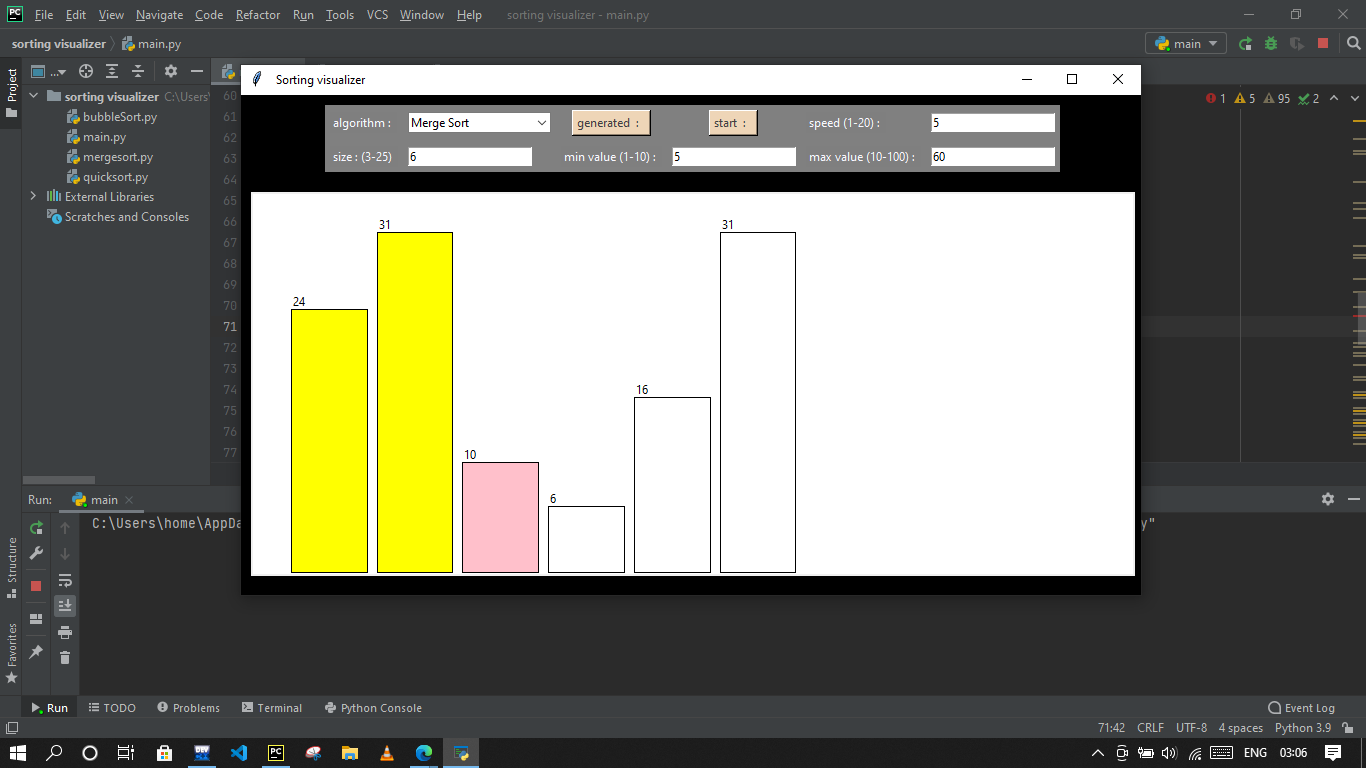
**QUICK SORT**

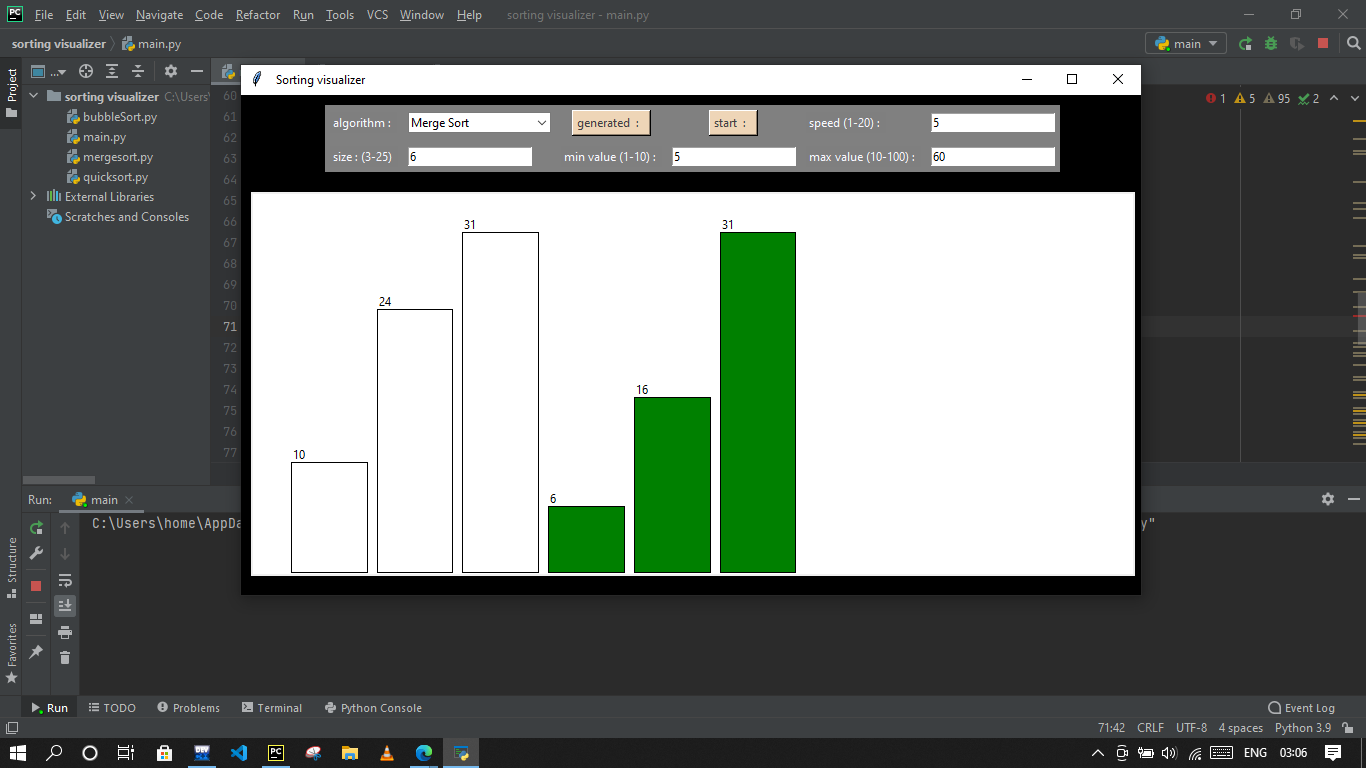
import time  
  
def partition(data, head, tail, drawData, timeTick):  
 border = head  
 pivot = data[tail]  
  
 drawData(data, getColorArray(len(data), head, tail, border, border))  
 time.sleep(timeTick)  
  
 for j in range(head, tail):  
 if data[j] < pivot:  
 drawData(data, getColorArray(len(data),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

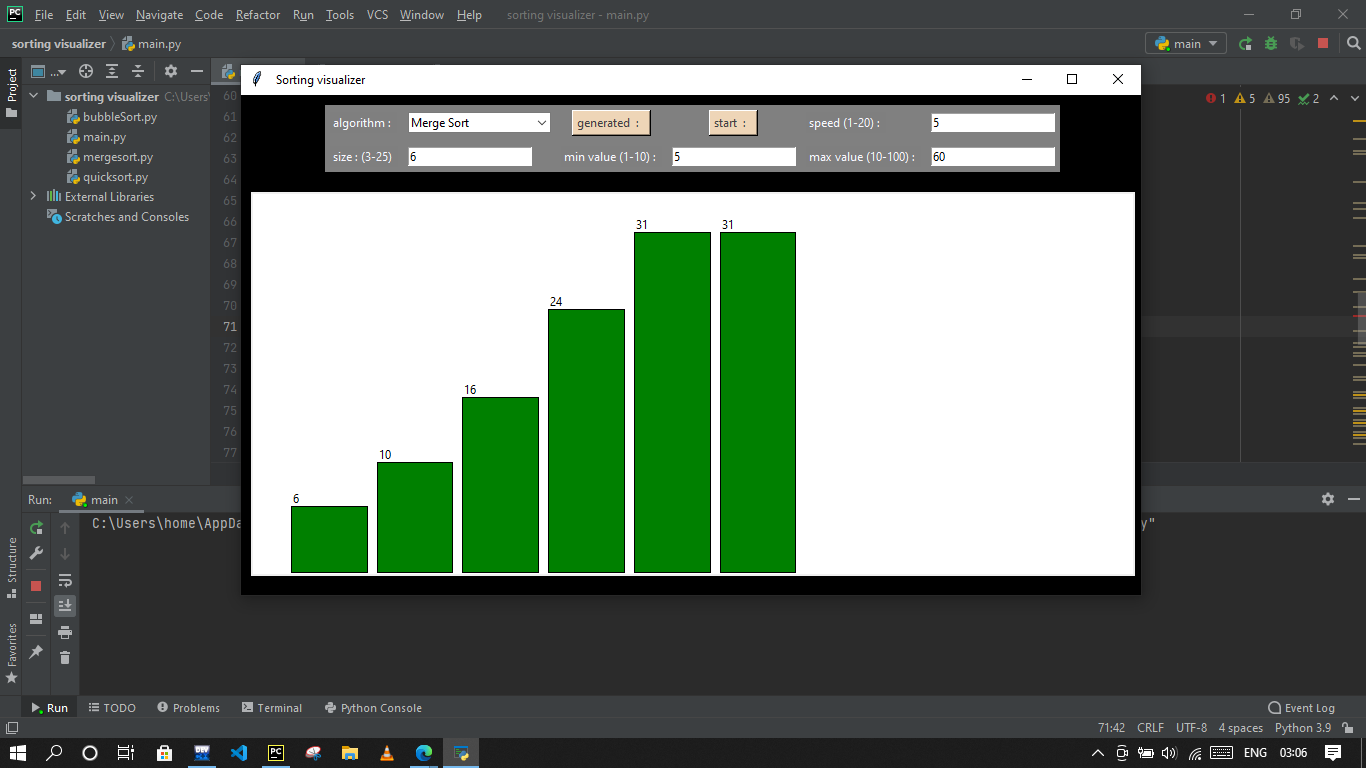
**BUBBLE SORT**

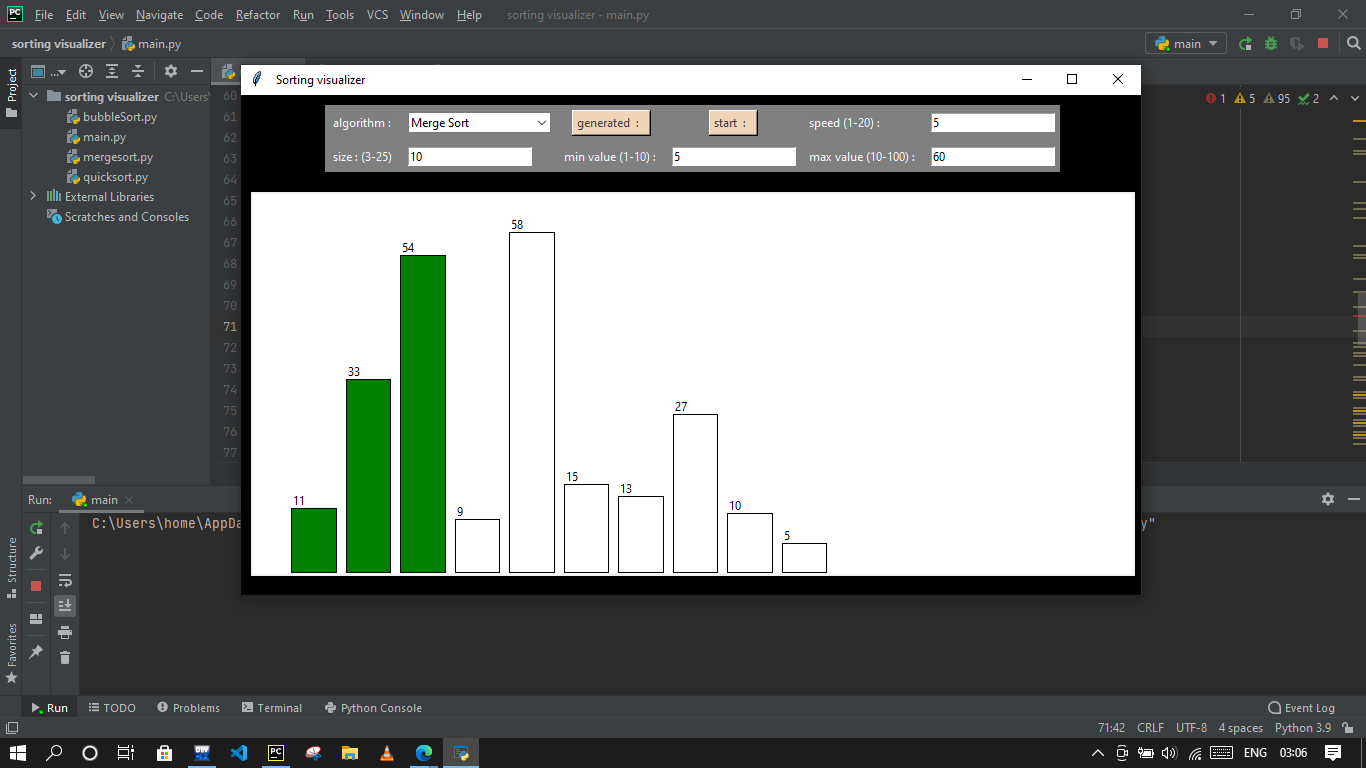
import time  
  
def bubble\_sort(data, drawData, timeTick):  
 for \_ in range(len(data)-1):  
 for j in range(len(data)-1):  
 if data[j] > data[j+1]:  
 data[j], data[j+1] = data[j+1], data[j]  
 drawData(data, ['green' if x == j or x == j+1 else 'red' for x in range(len(data))] )  
  
 time.sleep(timeTick)  
 drawData(data, ['green' for x in range(len(data))])

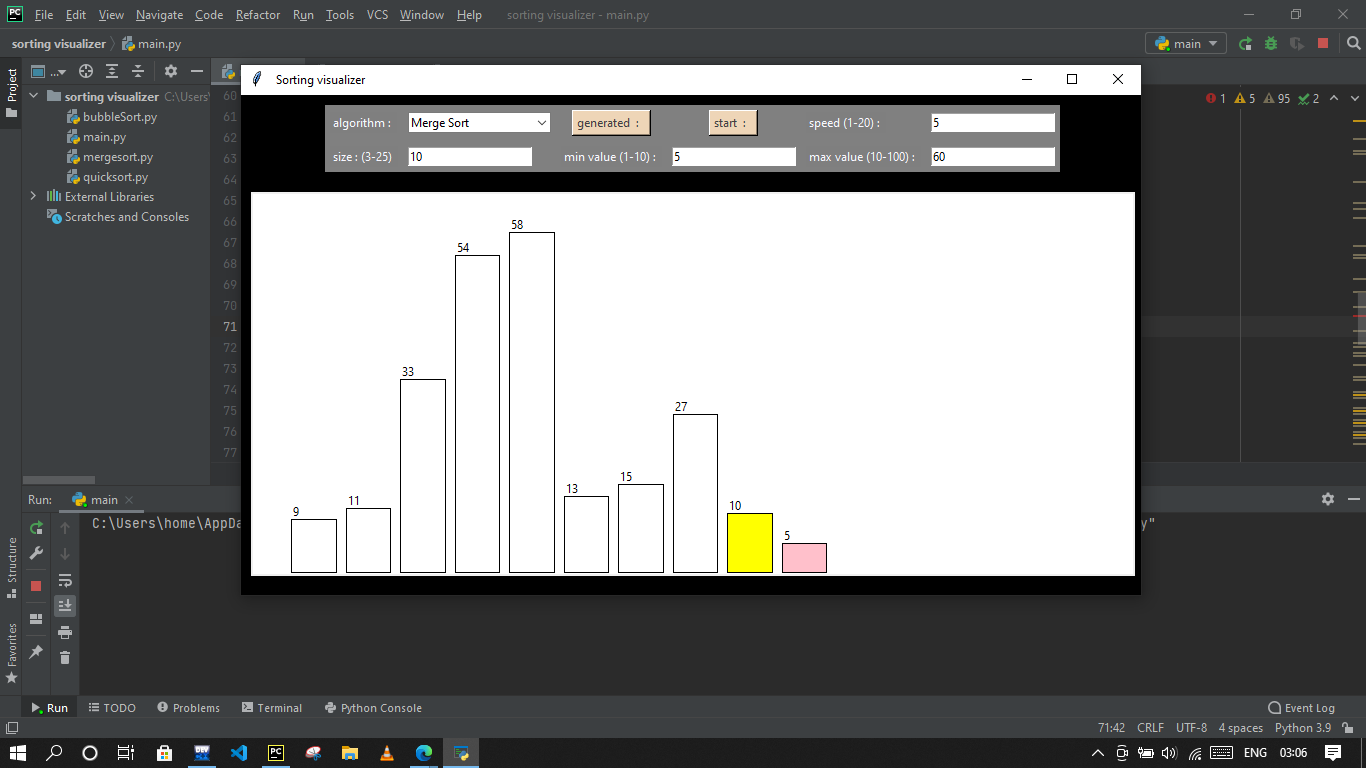
**Output :**

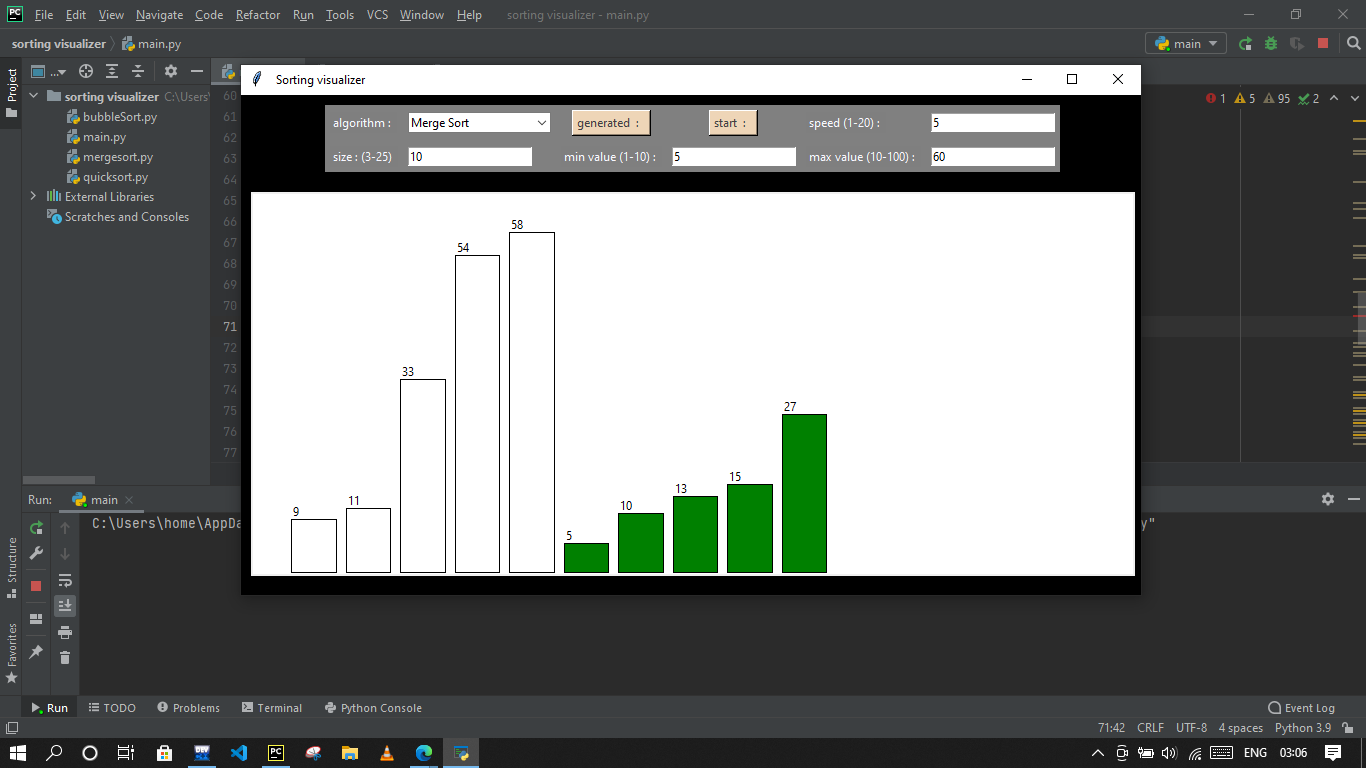


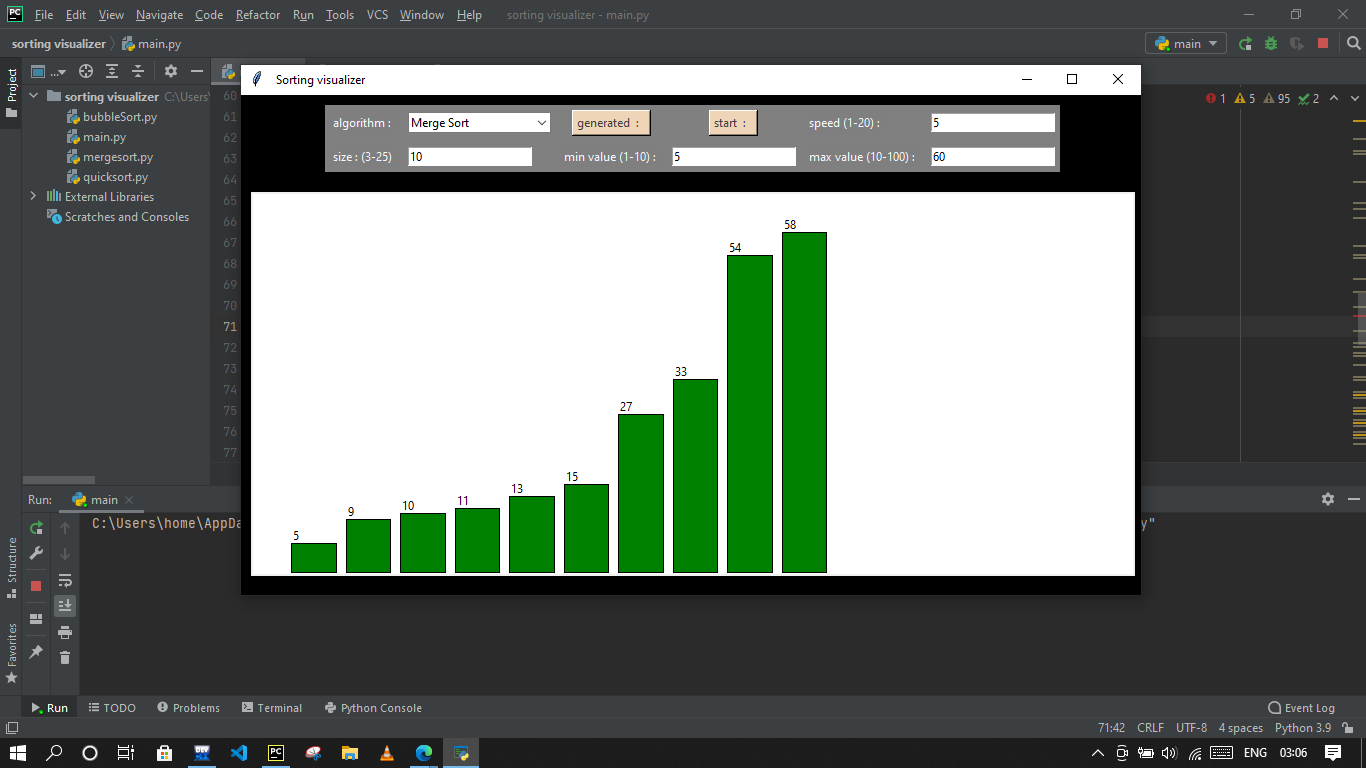


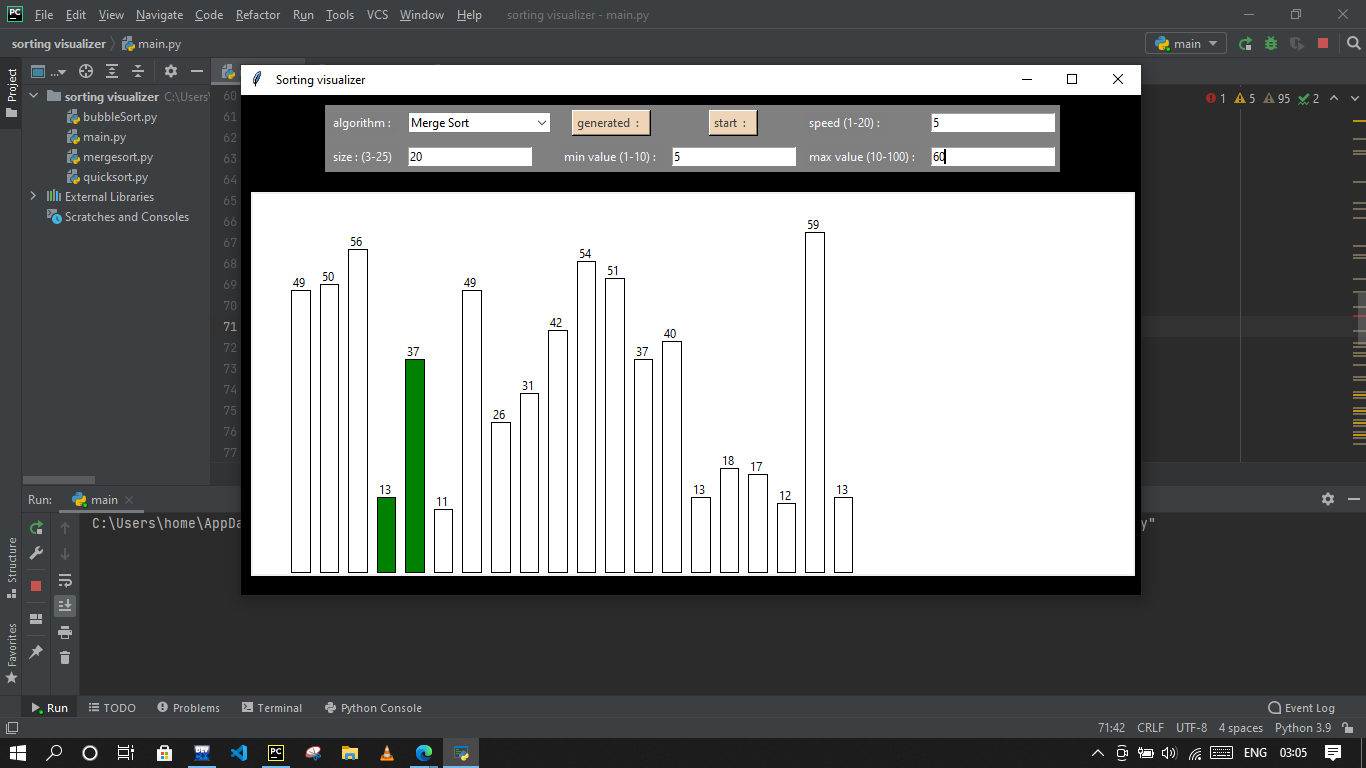


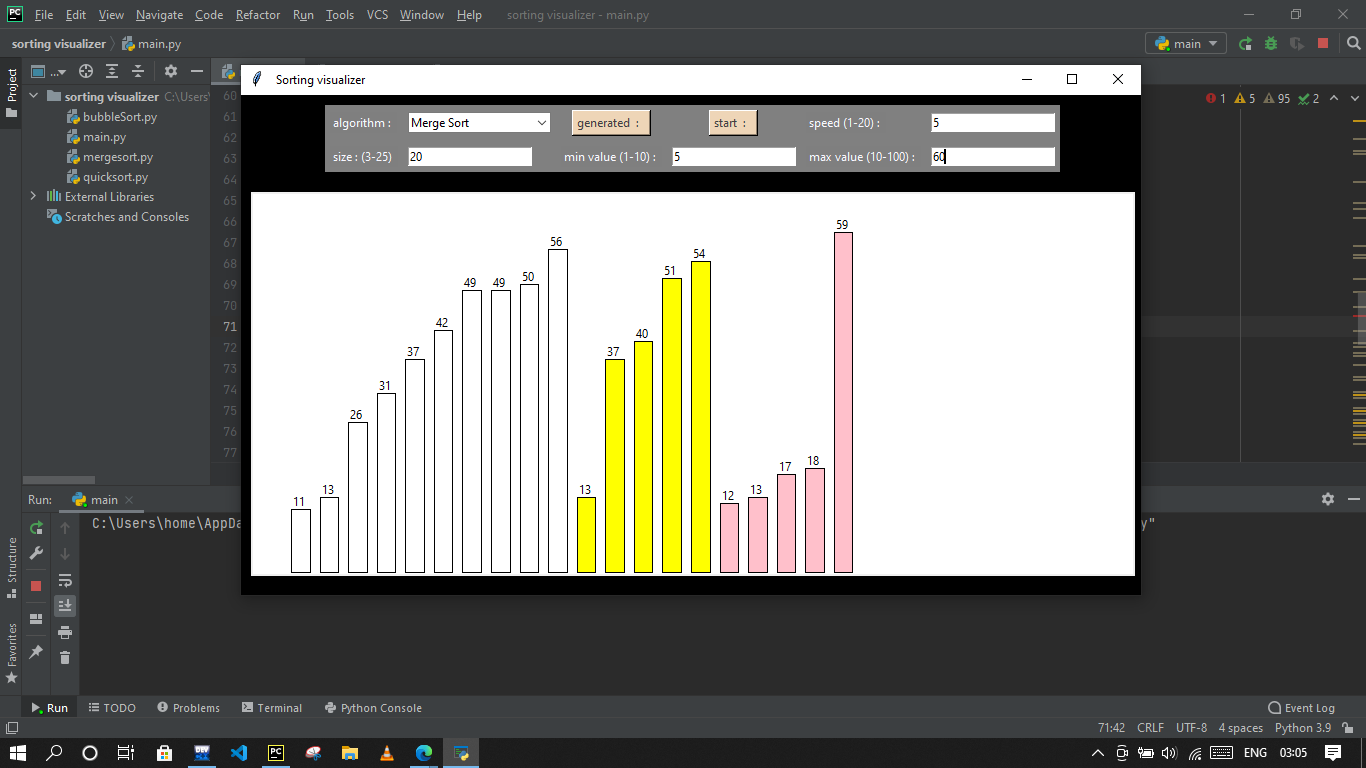


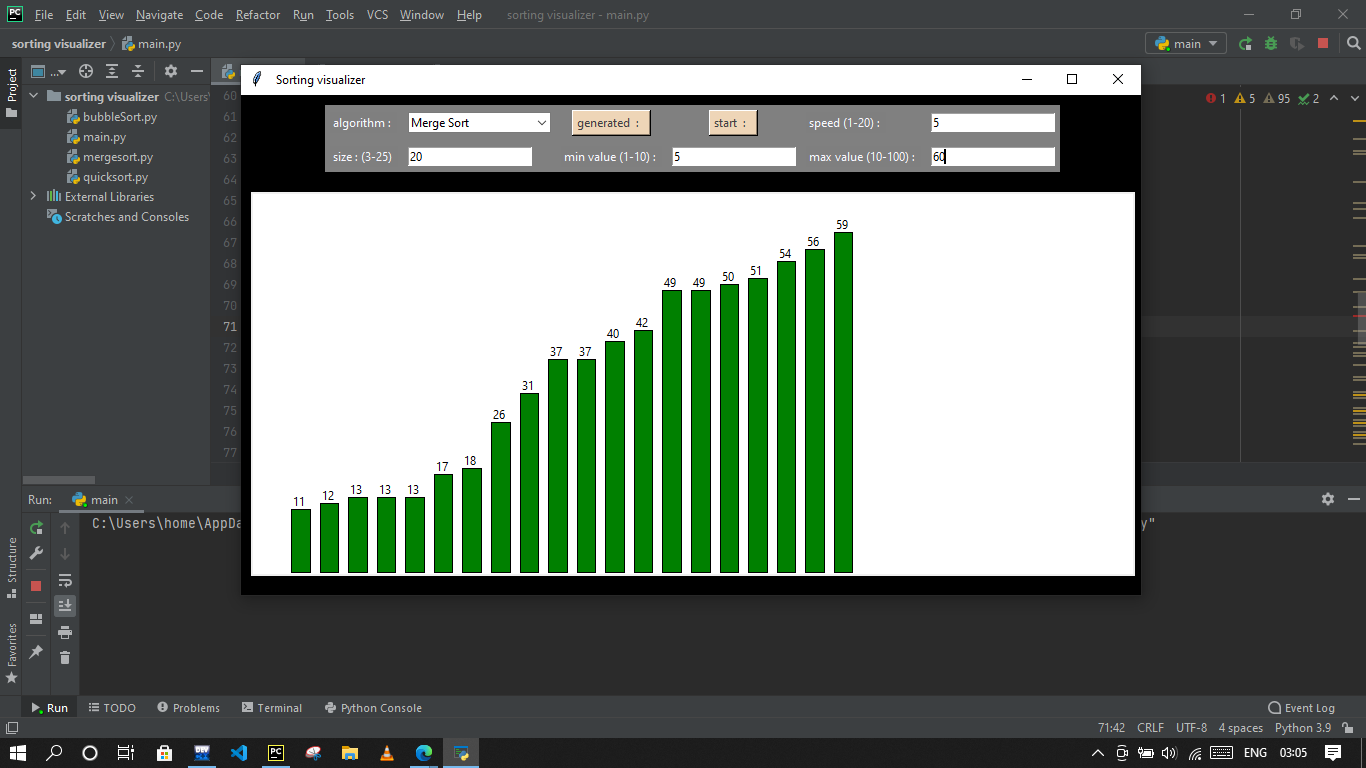












THE END!!!

**Conclusion:-**

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