SIX WEEKS SUMMER TRAINING REPORT

On

Object Oriented Programming using Python

Submitted by

Rajarshi Roy

Registration No. 12001342

Program Name: Bachelor of Technology in Computer Science

Under the Guidance of

Mr. TSV Ramana

School of Computer Science & Engineering

Lovely professional University, Phagwara, Punjab

(May-July, 2022)

DECLARATION

I hereby declare that I have completed my six weeks summer training at Object Oriented Programming using Python, Codetantra from 25th May 2022 to 10th July 2022 under the guidance of Mr. TSV Ramana. I have declare that I have worked with full dedication during these six weeks of training and my learning outcomes fulfil the requirements of training for the award of degree of Bachelor of Technology, Lovely Professional university, Phagwara, Punjab.

(Signature of student)

Name of Student: Rajarshi Roy

Registration no: 12001342

Date: 22th Sept, 2022

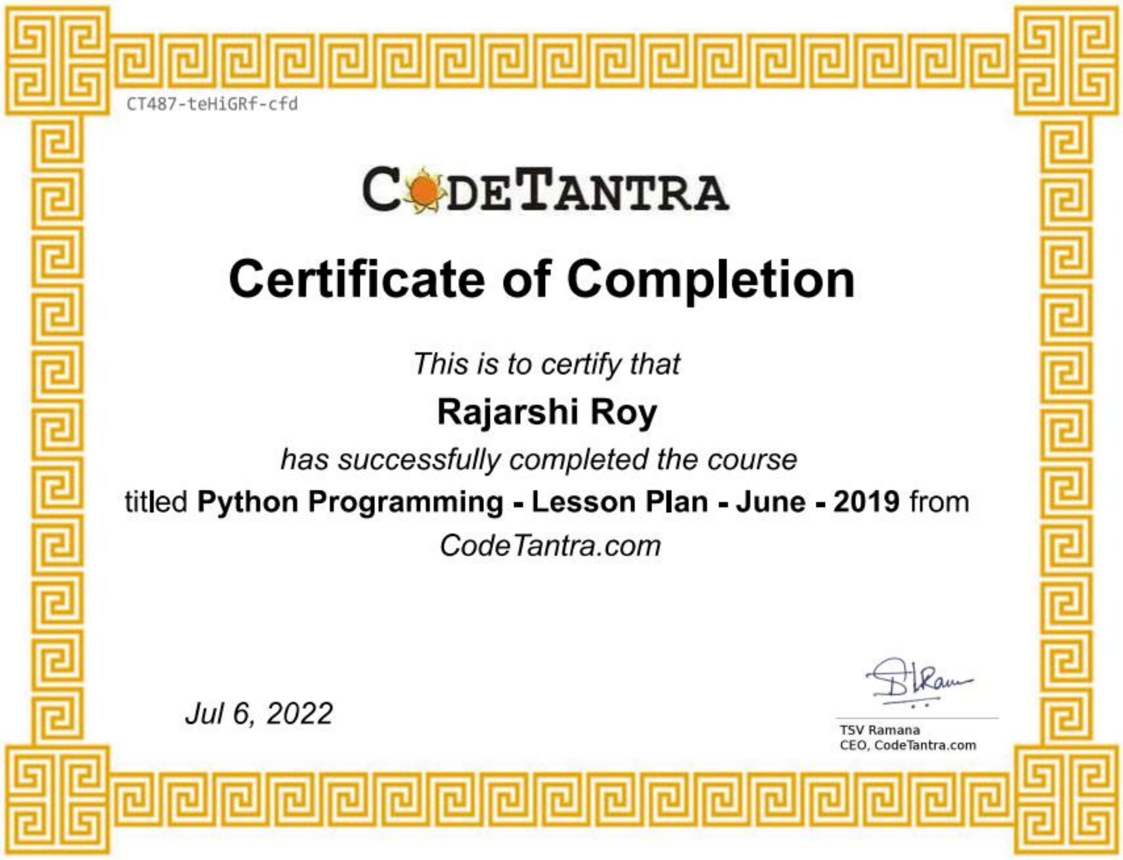
Acknowledgement

It is with sense of gratitude; I acknowledge the efforts of entire hosts of well-wishers who have in some way or other contributed in their own special ways to the success and completion of the Summer Training.

Successfully completion of any type technology requires helps from a number of people. I have also taken help from different people for the preparation of the report. Now, there is little efforts to show my deep gratitude to those helpful people.

First, I express my sense of gratitude and indebtedness to our Training mentor Mr. TSV Ramana. From the bottom of my heart, for his immense support and guidance throughout the training. Without his kind direction and proper guidance this study would have been a little success. In every of the project his Supervision and guidance shaped this training to be completed perfectly.

Training certificate from organization



CT487-teHiGRf-cfd

CODETANTRA

Certificate of Completion

This is to certify that

Rajarshi Roy

has successfully completed the course

titled **Python Programming - Lesson Plan - June - 2019** from

CodeTantra.com

Jul 6, 2022

[illegible]

TSV Ramana
CEO, CodeTantra.com

TABLE OF CONTENTS

Introduction

Python is a dynamic, interpreted (bytecode-compiled) language. There are no type declarations of variables, parameters, functions, or methods in source code. This makes the code short and flexible, and you lose the compile-time type checking of the source code. Python tracks the types of all values at runtime and flags code that does not make sense as it runs.

1. Python is an open-source language.
2. Syntax as simple as English.
3. Very large and Collaborative developer community.
4. Extensive Packages

1. UNDERSTANDING OPERATORS:
   Theory of operators: - Operators are symbolic representation of Mathematical tasks.
2. VARIABLES AND DATATYPES:
   Variables are named bounded to objects. Data types in python are int (Integer), Float, Boolean and strings.
3. CONDITIONAL STATEMENTS:
   If-else statements (Single condition)
   If- elif- else statements (Multiple Condition)
4. LOOPING CONSTRUCTS:
   For loop
5. FUNCTIONS:
   Functions are re-usable piece of code. Created for solving specific problem. Two types: Built-in functions and User- defined functions. Functions cannot be reused in python.
6. DATA STRUCTURES:
   Two types of Data structures:
   LISTS: A list is an ordered data structure with elements separated by comma and enclosed within square brackets.

DICTIONARY: A dictionary is an unordered data structure with elements separated by comma and stored as key: value pair, enclosed with curly braces {}.

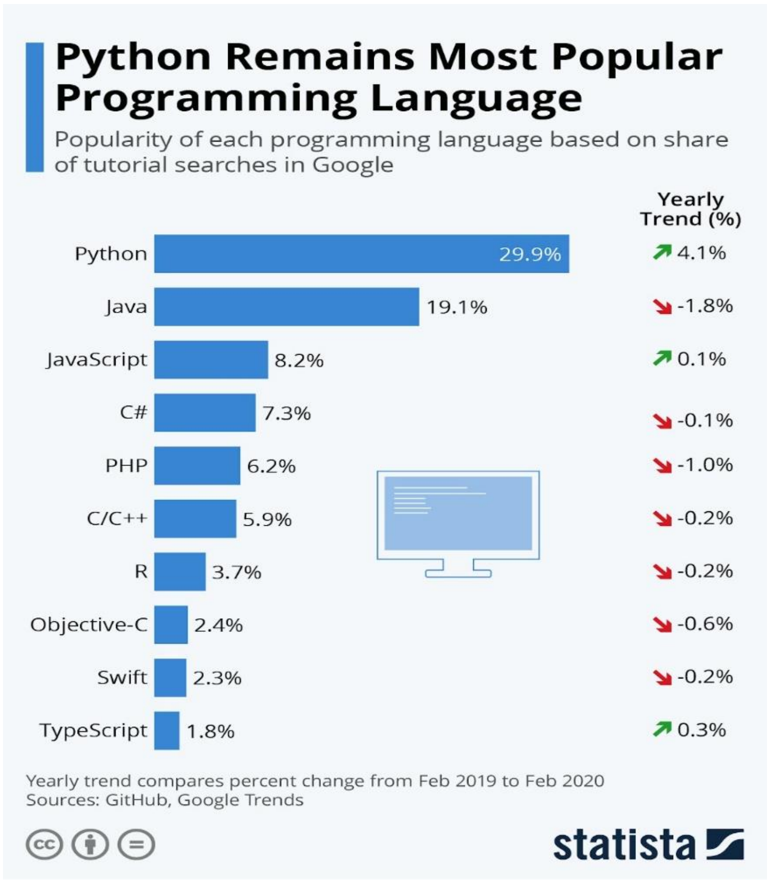
Technology Learnt

AI and Machine Learning (as well as deep learning) are constantly growing as a field - Python is a perfect programming language for that.

1. Python - Language Features
2. Comments, Identifiers and Keywords
3. Variables and Assignment
4. Expressions and Statements, Indentation
5. Data Types - Numbers, Strings
6. Data Types - Lists, Sets, Tuples
7. Data Types - Dictionaries, Data Type Conversions
8. Input and Output Statements, Basics of Python Programming
9. Arithmetic Operators, Comparison Operators
10. Assignment Operators, Bitwise Operators
11. Logical Operators, Membership Operators
12. Identity Operators, Operators Precedence
13. Practice Programs – Operators
14. Introduction to Algorithms
15. Building Blocks of Algorithms
16. Pseudo Code and Flow Charts
17. Programming Language, Algorithmic Problem Solving, Simple Strategies for Developing Algorithms
18. Problem Solving Exercise – Algorithms
19. If Statement
20. If-else Statement
21. If-elif-else Statement
22. While Loop
23. For Loop
24. Break Statement
25. Continue Statement
26. Pass Statement
27. Practice Programs - Control Statements

Reason for choosing this technology

The python language is one of the most accessible programming languages available because it has simplified syntax and not complicated, which gives more emphasis on natural language. Due to its ease of learning and usage, python codes can be easily written and executed much faster than other programming languages.



It’s An Open-source programming language. The Python Software Foundation is an organization that supports and administers this popular programming language. Python programming language can be used freely by developers. This scenario is the prime reason it is so effective and widely used by professionals. They can freely modify the code, distribute open-source Python databases with other developers of the Python community, and make lots of changes if needed.

Even while developing python-based mobile software and web development, developers are assured of very few errors and bugs in the program, along with better security than any other programming languages.

Powerful and robust programming language. Python is a programming language that can be used to create various applications for the web and mobile. Python open-source library comes with a framework used for image processing, generating interfaces on different operating systems, and running calculations. Additionally, this helps minimize the time and effort put in place by the developers.

IoT Integration. The right choice for data analysis in IoT systems is Python. The language is straightforward, and it’s easy to deploy. There is a large Python community of people who will help with the creation and connecting of libraries whenever you run into trouble or need more information. This makes the language ideal for complex applications that focus on data, an aspect commonly found in IoT systems.

Python can be used for numerous things; some of its use cases are:

- **AI**
  Python is a stable language. It is often used for Artificial Intelligence (AI) apps because of its strength in handling big computations.
- **Web Scraping**
  Python is used in web scraping applications for extracting information on web pages. It allows you to access links that would be unavailable for normal users and programs.
- **Web App**
  Platforms like Django and Flask, Python is also a comprehensive framework for developing highly scalable and secure web apps.
- **GUI Apps**
  Python is useful for creating user-friendly graphic interfaces along with robust backend architecture.

- **Data Science**
  Python libraries like NumPy, Pandas and Matplotlib are ideal Python data analysis libraries that can be used to visualize data in both an insightful and beautiful manner.
- **Enterprise Apps**
  Python is often hailed for its use in database applications. Data-heavy ERP software would greatly benefit from this language which can be ideal for creating and handling large databases of information.

Profile of the Problem

The main motive of this project is to help who's trying to understand the basic idea of common algorithms.

The main window has two dropdown menus to select the Algorithm Type and Name which the user wants to visualize. By default, the Algorithm Name menu is set to None, but if an Algorithm Type is selected it will get automatically updated with the different algorithms of that type. The next button will take you to a new window depending on the algorithm selected. For exit, it will show a warning message to check if you really want to exit or not. Below a gif is shown demonstrating the main window.

The Sorting Algorithm Visualizer window again has the option to change the sorting algorithm according to user needs (i.e., no need to go to the main window to change). Option to generate and shuffle the array is provided. **Two options are given to choose the type of visualization user wants to see. One using bar graph and other is colour bars.** The sort button starts the sorting operation. Slider for changing the size of the array and the speed of the visualization is provided. There's a label for showing number of comparisons done after each operation. Also, an algorithm info section is given which will give necessary time complexity details of the sorting algorithm. Button for going back to the main menu is also given. Once the sorting is done the entire array (bars) is painted green.

Problem Analysis

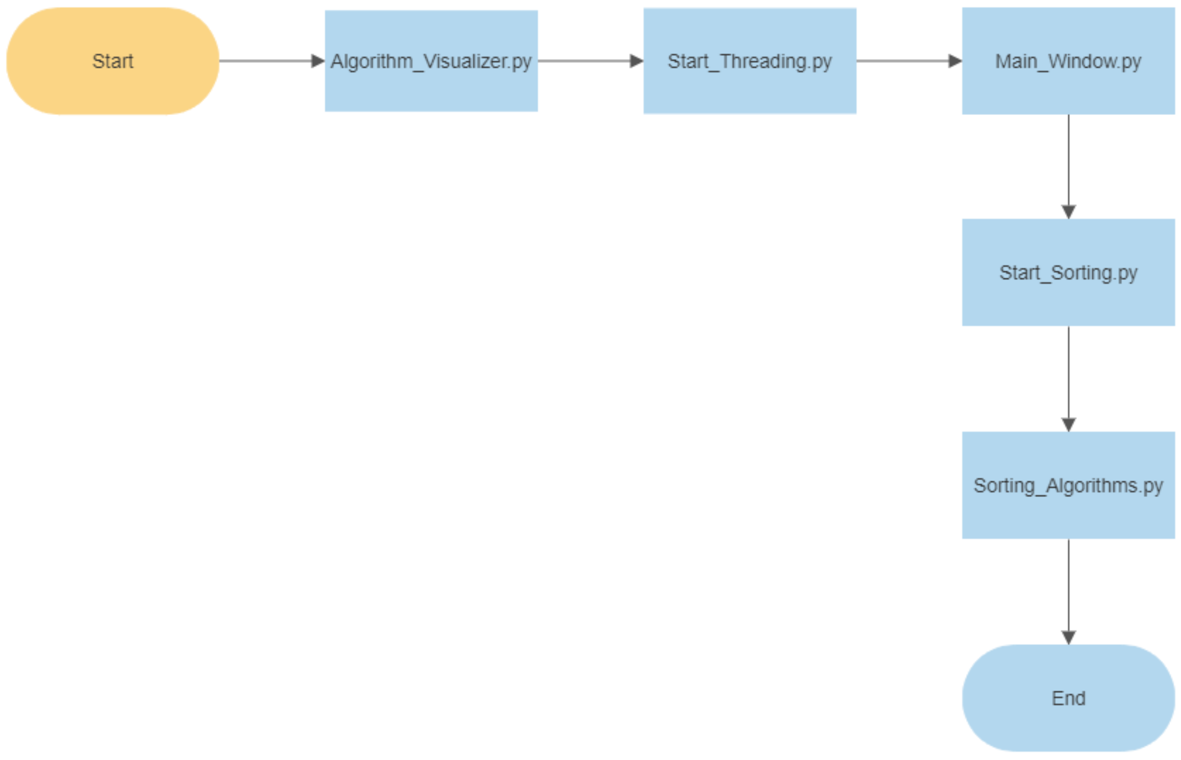
1. Product definition
    - The main motive of this project is to help who's trying to understand the basic idea of common algorithms.
2. Feasibility Analysis
    - The Sorting Algorithm Visualizer window again has the option to change the sorting algorithm according to user needs (i.e., no need to go to the main window to change). Option to generate and shuffle the array is provided. **Two options are given to choose the type of visualization user wants to see. One using bar graph and other is colour bars.** The sort button starts the sorting operation. Slider for changing the size of the array and the speed of the visualization is provided. There's a label for showing number of comparisons done after each operation. Also, an algorithm info section is given which will give necessary time complexity details of the sorting algorithm. Button for going back to the main menu is also given. Once the sorting is done the entire array (bars) is painted green.

Software Requirement Analysis

1. Python
2. Tkinter
3. IDE
4. Operating System

Design

1. Flowchart



Implementation

Project: Sorting Visualizer - Shell Sort, Radix Sort

Algorithm_Visualizer (main).py

```
import Codes.Start_Threading

Process = Codes.Start_Threading.START()
Process.start()
```

Start_Threading.py

```
import random
import time
from tkinter import *
from Codes.Main_Window import Window
from threading import *

class START(Thread):
    def run(self):
        root=Tk()
        Window(root)
        root.mainloop()
```

Main_Window.py

```
from tkinter import *
from tkinter import messagebox
import Codes.Start_Threading
from Codes.Start_Sorting import *


class Window:
    def __init__(self, root):

        # Main Window
        self.root = root
        # Warning sign for close
        self.root.protocol("WM_DELETE_WINDOW", self.Close)

        # Main Window Size and Center Aligned in the screen
        self.wx, self.wy = 500, 300
        self.wxs, self.wys = self.root.winfo_screenwidth(), 
self.root.winfo_screenheight()
        self.WINDOW_X, self.WINDOW_Y = (self.wxs / 2) - (self.wx / 2), 
(self.wys / 2) - (self.wy / 2)
        self.root.geometry('%dx%d+%d+%d' % (self.wx, self.wy, self.WINDOW_X, 
self.WINDOW_Y))
        self.root.config(bg="white")
        self.root.resizable(False, False)

        # Title And Icon
        self.root.title("Algorithm Visualizer")
        try:
            self.root.iconbitmap("Images/algorithm.ico")
        except:
            img = PhotoImage("Images/algorithm.ico")
            self.root.tk.call('wm', 'iconphoto', self.root._w, img)

        # Heading of the main window
        self.MainLabel = Label(self.root, text='Algorithm Visualizer', 
bg="white", fg="blue4", font=("calibri italic", 20, 'bold'))
        self.MainLabel.pack(pady=15)

        # Dictionary On types of Algorithms and their lists
        self.Algo = {'Sorting': ['Shell Sort', 'Radix Sort']}

        # Two dropdown menu on algorithm type and algorithm name
        self.AlgoTypeVar = StringVar()
        self.AlgoNameVar = StringVar()

        # for automatic update on the second list if something is chosen on 
the 1st list
        self.AlgoTypeVar.trace('w', self.update_options)

        # two drop down menus configurations
        self.AlgoTypeList = OptionMenu(self.root, self.AlgoTypeVar, 
*self.Algo.keys())
        self.AlgoTypeList.config(bg="pink", activebackground="hot pink", 
cursor="hand2")
        self.AlgoNameList = OptionMenu(self.root, self.AlgoNameVar, 'None')
        self.AlgoNameList.config(bg="pink", activebackground="hot pink", 
cursor="hand2")
        # label of the two dropdown menus
        self.AlgoTypeVar.set("Select Algorithm Type")
        self.AlgoNameVar.set("Select Algorithm")
        self.AlgoTypeList.pack(pady=(30, 0))
        self.AlgoNameList.pack(pady=2)

        # next button
        self.NextButton = Button(self.root, text="Next>", bg="pale green",
activebackground="lime green",
                                 command=self.Run1)
        self.NextButton.pack(pady=20)

    # for automatic update on the 2nd list if something is chosen on the 1st
list
    def update_options(self, *args):
        try:
            algo_list = self.Algo[self.AlgoTypeVar.get()]
        except:
            algo_list = ["None"]
        self.AlgoNameVar.set("Select Algorithm")
        menu = self.AlgoNameList['menu']
        menu.delete(0, 'end')
        for algo in algo_list:
            menu.add_command(label=algo, command=lambda x=algo:
self.AlgoNameVar.set(x))

    # Exit button
    def Exit(self):
        self.root.destroy()

    # Close warning
    def Close(self):
        if messagebox.askokcancel("Exit", "Do you want to exit?"):
            self.root.destroy()
            exit()

    # Secondary window back button
    def Back(self):
        self.root.destroy()
        Process = Codes.Start_Threading.START()
        Process.start()

    # For running the algorithms
    def Run2(self):
        # If Sorting is selected
        if self.AlgoTypeVar.get() == "Sorting":
            # create a new window for sorting algorithm
            sort_window = Tk()
            # send it to Start_Sort.py file
            Sorting(sort_window, self.AlgoNameVar.get())
            sort_window.mainloop()

    # For running the secondary window
    def Run1(self):

        # If nothing is selected show an error box
        if self.AlgoTypeVar.get() == "Select Algorithm Type":
            messagebox.showerror("Error!", "Please select Algorithm Type.")

        # If sorting is selected
        elif self.AlgoTypeVar.get() == "Sorting":
            # Destroy the main window
            self.root.destroy()
            # Go to run() function
            self.Run2()
```

Start_Sorting.py

```
from tkinter import *
from tkinter import messagebox
from random import shuffle, sample
from Codes.Sorting_Algorithms import algochooser
from colorsys import hls_to_rgb
from threading import *
from tkinter import *
import Codes.Start_Threading

# Main sorting class
class Sorting:
    def __init__(self, root, AlgoNameVar):

        # Sorting window
        self.root = root

        # warning for close/exit
        self.root.protocol("WM_DELETE_WINDOW", self.Close)

        # Selected Algorithm Name
        self.AlgoNameVar = AlgoNameVar

        # Window Size
        self.wx, self.wy = 1200, 700

        # Screen Size
        self.wxs, self.wys = self.root.winfo_screenwidth(),
self.root.winfo_screenheight()
```

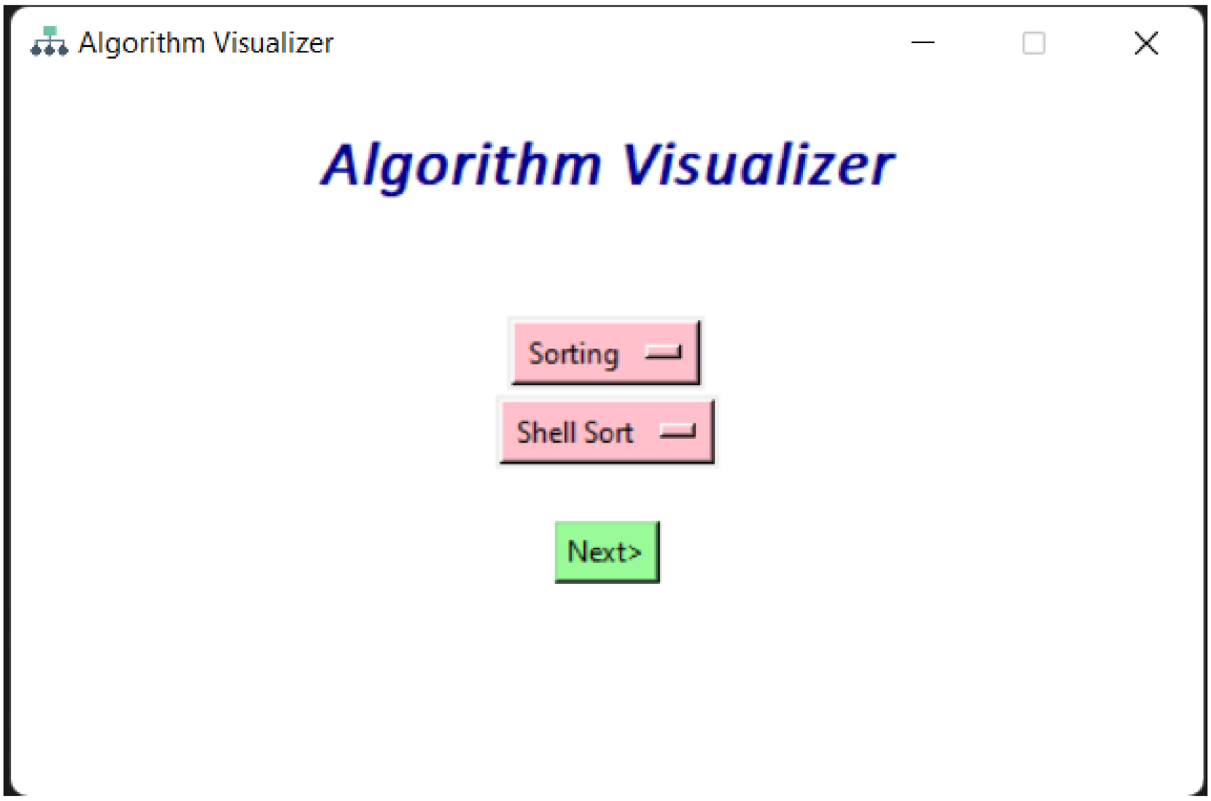
```
        # Aligning the window in the center of the screen
        self.WINDOW_X, self.WINDOW_Y = (self.wxs / 2) - (self.wx / 2),
(self.wys / 2) - (self.wy / 2)

        # Sorting canvas size
        self.CANVAS_X, self.CANVAS_Y = 950, 700

        # Left side information frame size
        self.FRAME1_X, self.FRAME1_Y = 250, 700

        # Apply changes to window
        self.root.geometry('%dx%d+%d+%d' % (self.wx, self.wy, self.WINDOW_X,
self.WINDOW_Y))
        self.root.config(bg="grey")
        self.root.wm_resizable(False, False)

        # Title And Icon
        self.root.title("Sorting Algorithm Visualizer")
        try:
            self.root.iconbitmap("Images/sorting.ico")
        except:
            img = PhotoImage("Images/sorting.ico")
            self.root.tk.call('wm', 'iconphoto', self.root._w, img)


        # Starting size of the array
        self.size_var = IntVar()
        self.size_var.set(30)


        # Starting speed of the array
        self.speed_var = IntVar()
        self.speed_var.set(20)


        # Graph type bar or color
        # 0 means bar 1 means color
        self.graph_type = IntVar()
        self.graph_type.set(0)
        self.TYPE = self.graph_type.get()


        # Starting point of the graph
        self.starting_point = 2


        # Creating frame in the left side
        self.frame1 = Frame(root, width=self.FRAME1_X, height=self.FRAME1_Y,
bg="white")
```

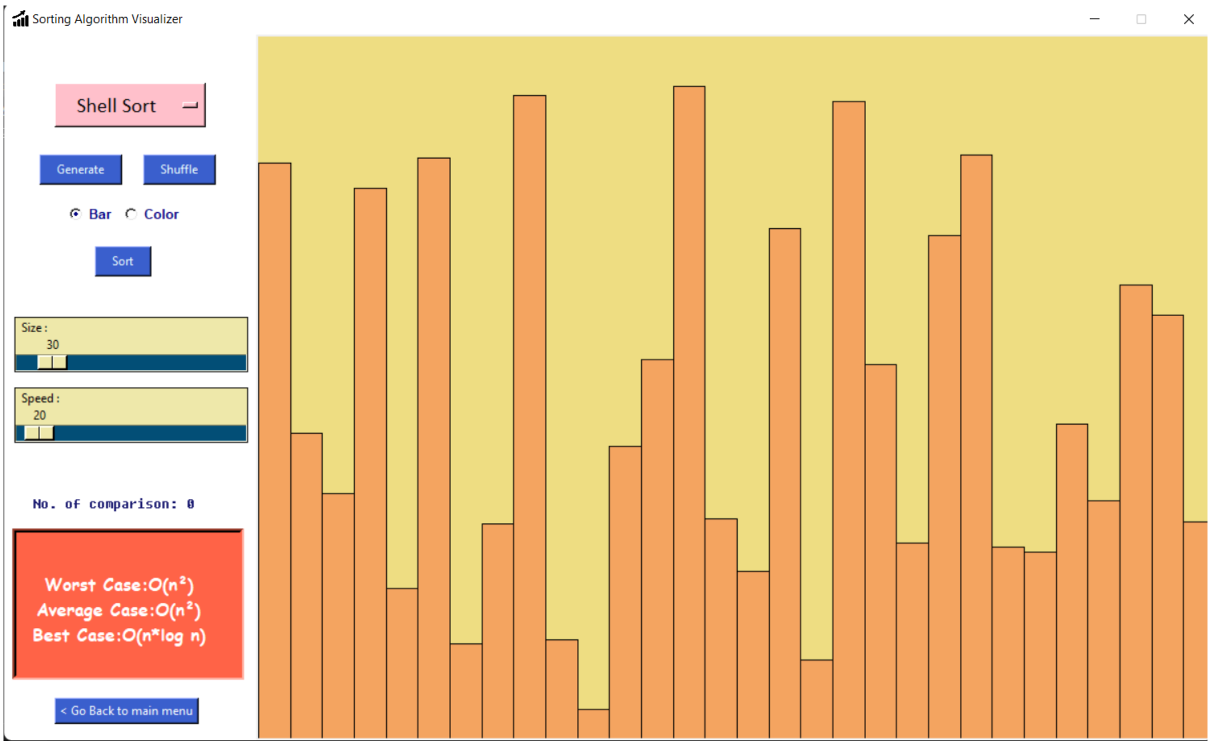
```
        self.frame1.grid_propagate(0)
        self.frame1.pack(side=LEFT)


        # Algorithm Information Table
        self.information = {'Shell Sort': "Worst Case:O(n²)\nAverage
Case:O(n²)\nBest Case:O(n*log n)",
                            'Radix Sort': "Worst Case:O(k*(n+b))\nAverage
Case:O(k*(n+b))\nBest Case:O(k*(n+b))"}


        # Algorithm Names
        self.algorithm = ['Shell Sort', 'Radix Sort']


        # Creating a drop down menu for algorithm selection
        self.algo_var = StringVar()
        # Setting it default value to what we selected previously in the main
window
        self.algo_var.set(self.AlgoNameVar)
        self.algo_menu = OptionMenu(self.frame1, self.algo_var,
*self.algorithm, command=self.case_chooser)
        self.algo_menu.config(font="calibri", bg="pink",
activebackground="sandy brown", cursor="circle")
        self.algo_menu["highlightthickness"] = 0
        self.algo_menu["padx"] = 20
        self.algo_menu["pady"] = 8
        self.algo_menu.grid_propagate(0)
        # Place for the dropdown menu
        self.algo_menu.place(rely=0.1, relx=0.5, anchor=CENTER)


        # Creating a frame for new buttons
        self.frame_btn1 = Frame(self.frame1, width=230, height=40, bg="white")
        self.frame_btn1.grid_propagate(0)
        self.frame_btn1.place(relx=0.0, rely=0.17)
        # Button for generating new array
        self.btn_new = Button(self.frame_btn1, text="Generate", padx=13,
pady=3, command=self.new_list, bg="RoyalBlue3", fg="azure", cursor="hand2")
        self.btn_new.place(relx=0.15, rely=0)
        # Button for shuffling the array
        self.btn_shuffle = Button(self.frame_btn1, text="Shuffle", padx=13,
pady=3, command=self.shuffle_list, bg="RoyalBlue3", fg="azure",
cursor="hand2")
        self.btn_shuffle.place(relx=0.60, rely=0)


        # Option for bar / color graph
        # Creating new frame for it
        self.frame_radio = Frame(self.frame1, bg="white", width=230,
height=25, relief="flat", bd=4)
        self.frame_radio.place(relx=0, rely=0.23)
        self.frame_radio.grid_propagate(0)
        # Creating the button / option
        self.bar_drawing = Radiobutton(self.frame_radio, text="Bar",
bg="white", fg="navy", variable=self.graph_type, value=0,
                                       font=("Helvetica", 10, "bold"),
command=self.draw_type, cursor="hand2")
        self.color_drawing = Radiobutton(self.frame_radio, text="Color",
bg="white", fg="navy", variable=self.graph_type,
                                         value=1, font=("Helvetica", 10,
"bold"), command=self.draw_type, cursor="hand2")
        self.bar_drawing["activebackground"] = "#83A177"
        self.color_drawing["activebackground"] = "#83A177"
        self.bar_drawing.place(relx=0.25, rely=0)
        self.color_drawing.place(relx=0.5, rely=0)


        # Creating a frame for a new button
        self.frame_btn2 = Frame(self.frame1, width=230, height=40, bg="white")
        self.frame_btn2.grid_propagate(0)
        self.frame_btn2.place(relx=0.0, rely=0.3)
        # Creating a sort button
        self.btn_sort = Button(self.frame_btn2, text="Sort", padx=13, pady=3,
command=self.sort_list, bg="RoyalBlue3", fg="azure", cursor="hand2")
        self.btn_sort.place(relx=0.39, rely=0)


        # Slider for changing size of array
        self.scale_size = Scale(self.frame1, label="Size :",
orient=HORIZONTAL, from_=10, to=200, length=230,
                                bg="pale goldenrod", troughcolor="#024e76",
variable=self.size_var, command=self.change_size,
                                relief="solid", cursor="hand2")
        self.scale_size.place(relx=0.04, rely=0.4)
        self.scale_size["highlightthickness"] = 0


        # Slider for changing speed of the operations
        self.scale_speed = Scale(self.frame1, label="Speed :",
orient=HORIZONTAL, from_=1, to=500, length=230,
                                bg="pale goldenrod", troughcolor="#024e76",
variable=self.speed_var, command=self.change_speed, relief="solid",
cursor="hand2")
        self.scale_speed.place(relx=0.04, rely=0.5)
        self.scale_speed["highlightthickness"] = 0
        # Label for showing the number of comparisons
        self.label_comparison = Label(self.frame1, text=" No. of comparisons: 
0", bg="white", fg="midnight blue", font=("Fixedsys", 12))
        self.label_comparison.place(relx=0.1, rely=0.65)


        # Frame for algorithm info
        self.frame_algo_info = Frame(self.frame1, bg="tomato", width=230, 
height=150, relief="sunken", bd=4)
        self.frame_algo_info.grid_propagate(0)
        self.frame_algo_info.place(relx=0.03, rely=0.7)
        # Label for algorithm info
        self.label_avg = Label(self.frame_algo_info, bg="tomato", fg="white", 
text=self.information[self.algo_var.get()], font=("comic sans ms", 13, 
"bold"))
        self.label_avg.pack_propagate(0)
        self.label_avg.place(relx=0.06, rely=0.25)


        # Back button to the main window
        self.BackButton = Button(self.frame1, bg="RoyalBlue3", fg="white", 
text="< Go Back to main menu", command=self.Back, cursor="hand2")
        self.BackButton.grid_propagate(0)
        self.BackButton.place(relx=0.2, rely=0.94)


        # Canvas for the graph
        self.frame2 = Frame(self.root, width=self.CANVAS_X, 
height=self.CANVAS_Y)
        self.frame2.pack(side=LEFT)
        self.canva = Canvas(self.frame2, width=self.CANVAS_X, 
height=self.CANVAS_Y, bg="light goldenrod")
        self.canva.pack()

        # creating the new array
        self.numbers = sample(range(20, self.CANVAS_Y-20), 
self.size_var.get())
        shuffle(list(set(self.numbers)))
        self.rec_width = self.CANVAS_X // self.size_var.get()
        for num in self.numbers:
            self.canva.create_rectangle(self.starting_point, self.CANVAS_Y - 
num, self.starting_point + self.rec_width, self.CANVAS_Y, fill="sandy brown")
            self.starting_point += self.rec_width

    # Function for back button to main window
    def Back(self):
        self.root.destroy()
        Process = Codes.Start_Threading.START()
        Process.start()
    # Function for exit
    def Close(self):
        if messagebox.askokcancel("Exit", "Do you want to exit?"):
            self.root.destroy()
            quit()

    # function for painting the graph
    def paint(self, colortype):
        # delete the previous graph
        self.canva.delete("all")
        # start painting from here
        self.starting_point = 2
        # width of each bar
        self.rec_width = self.CANVAS_X / self.size_var.get()
        # if bar graph is selected
        if self.TYPE == 0:
            # paint the array
            for i in range(len(self.numbers)):
                self.canva.create_rectangle(
                    self.starting_point, self.CANVAS_Y - self.numbers[i],
self.starting_point + self.rec_width, self.CANVAS_Y, fill=colortype[i])
                self.starting_point += self.rec_width
        # if color graph is selected
        else:
            # paint the array
            for i in range(len(self.numbers)):
                hls_color = hls_to_rgb(colortype[i] / 360, 0.6, 1)
                red = hls_color[0] * 255
                green = hls_color[1] * 255
                blue = hls_color[2] * 255
                self.canva.create_rectangle(self.starting_point, 0,
self.starting_point + self.rec_width, self.CANVAS_Y,
                                       outline="", fill="#%02x%02x%02x" %
(int(red), int(green), int(blue)))
                self.starting_point += self.rec_width
        # update the graph frame
        self.frame2.update()

    # function for creating new list
    def new_list(self):
        numbers = []
        self.label_comparison.configure(text="No. of comparisons: 0")
        # enter random numbers into the new array
        self.numbers = sample(range(20, self.CANVAS_Y-20),
self.size_var.get())
        # shuffle the numbers
        shuffle(list(set(numbers)))
        # if bar graph is selected
        if self.TYPE == 0:
            colortype = ["sandy brown" for x in self.numbers]
        # if color graph is selected
        else:
            colortype = [((int)(x * 360) / self.CANVAS_Y) for x in
self.numbers]
        # paint the colored array
        self.paint(colortype)

    # function for shuffling the array
    def shuffle_list(self):
        shuffle(self.numbers)
        self.label_comparison.configure(text="No. of comparison: 0")
        # if bar graph is selected
        if self.TYPE == 0:
            colortype = ["sandy brown" for x in self.numbers]
        # if color graph is selected
        else:
            colortype = [((int)(x * 360) / self.CANVAS_Y) for x in
self.numbers]
        # paint the colored array
        self.paint(colortype)

    # function for changing the size of the array
    def change_size(self, event):
        self.label_comparison.configure(text="No. of comparisons: 0")
        self.numbers = sample(range(20, self.CANVAS_Y-20),
self.size_var.get())
        shuffle(list(set(self.numbers)))
        # if bar graph is selected
        if self.TYPE == 0:
            colortype = ["sandy brown" for x in self.numbers]
        # if color graph is selected
        else:
            colortype = [((int)(x * 360) / self.CANVAS_Y) for x in
self.numbers]
        # paint the colored array
        self.paint(colortype)

    # function for changing the speed of the array
    def change_speed(self, event):
        pass

    # function for sorting the list
    def sort_list(self):
        self.label_comparison.configure(text="No. of comparisons: 0")
        startsort = Thread(target=algochooser(self.numbers, self.paint,
self.label_comparison, self.algo_var.get(), self.TYPE, self.speed_var.get()))
        startsort.start()

    # function for choosing the sorting algorithm
    def case_chooser(self, event):
        self.label_avg.pack_forget()
        self.label_avg.configure(text=self.information[self.algo_var.get()])

    # function for drawing the default random bars/colors
    def draw_type(self):
        self.TYPE = self.graph_type.get()
        if self.TYPE == 0:
            colortype = ["sandy brown" for x in self.numbers]
        else:
            colortype = [((int)(x * 360) / self.CANVAS_Y) for x in
self.numbers]
        self.paint(colortype)
```

Sorting_Algorithms.py

```
import time
import random

cmp = 0
TYPE = 0


def algochooser(numbers, paint, label_comparison, something, TYPE_OF_DRAW,
speed):
    global cmp, TYPE
    TYPE = TYPE_OF_DRAW

    if something == "Shell Sort":
        label_comparison.configure(text="No. of comparisons: 0")
        shellsort(numbers, paint, label_comparison, speed)
        if TYPE == 0:
            paint(["lawn green"] * len(numbers))
        cmp = 0

    elif something == "Radix Sort":
        label_comparison.configure(text="No. of comparisons: 0")
        radixsort(numbers, paint, speed)
        if TYPE == 0:
            paint(["lawn green"] * len(numbers))
        cmp = 0



def shellsort(number, paint, label_comparison, speed):
    global cmp, TYPE
    colors = []
    length = len(number)
    gap = length // 2
    while gap > 0:
        for x_sort in range(gap, length):
            j = x_sort - gap
            if TYPE == 0:
                colors = ["#cc0000" if xy == j + gap or xy ==
                                      j else "antique white" for xy in
range(len(number))]
            else:
                colors = [((int)(x * 360) / 950) for x in number]
            paint(colors)
            while j >= 0:
                if (number[j + gap] < number[j]):
                    number[j + gap], number[j] = number[j], number[j + gap]
                else:
                    break
                cmp += 1
                if TYPE == 0:
                    colors = ["#cc0000" if xy == j + gap or xy ==
                                          j else "antique white" for xy in
range(len(number))]
                else:
                    colors = [((int)(x * 360) / 950) for x in number]
                paint(colors)
                label_comparison.configure(text="No. of comparisons: " +
str(cmp))
                # ------------------------------------------------------------
--------------------------------------
                j -= gap
            time.sleep(1 / speed)
        gap //= 2


def countsort(number, exp, paint):
    global TYPE
    colors = []
    count = [0] * 10
    temp = [0] * len(number)
    for x in range(len(number)):
        count[(number[x] // exp) % 10] += 1
```

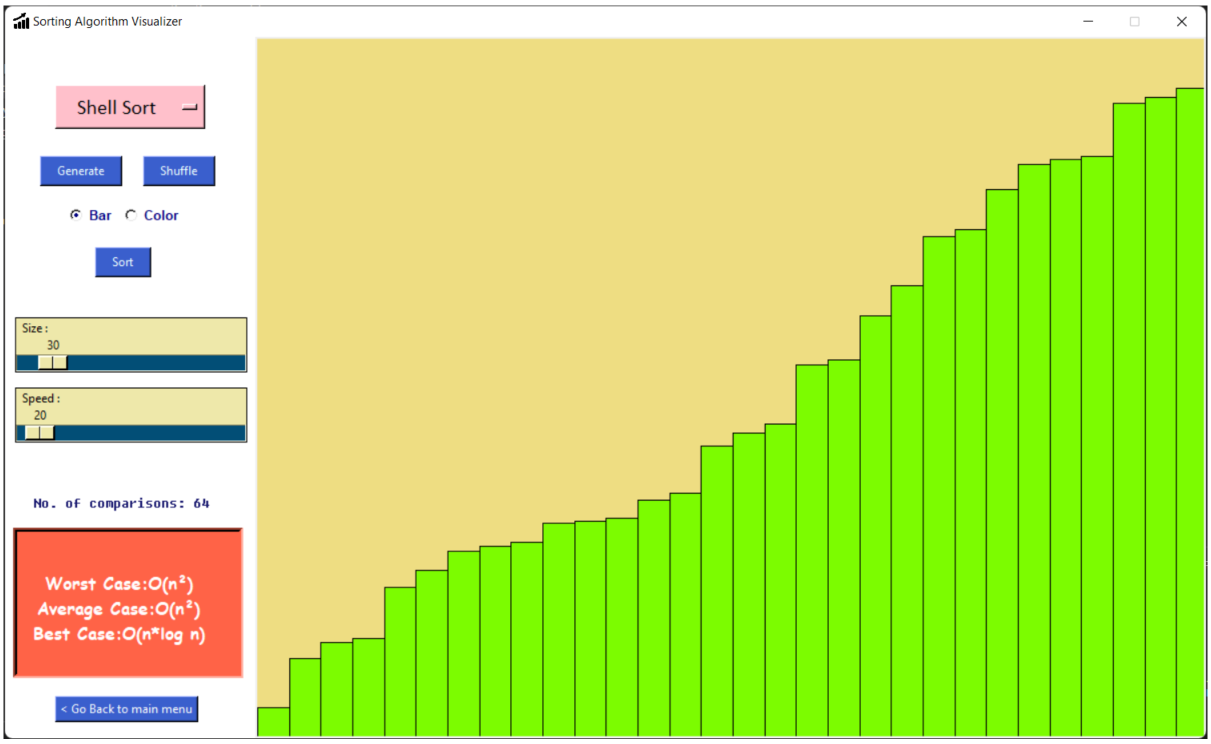
```python
    for y in range(1, len(count)):
        count[y] += count[y - 1]
    for z in range(len(number) - 1, -1, -1):
        index = count[(number[z] // exp) % 10]
        temp[index - 1] = number[z]
        count[(number[z] // exp) % 10] -= 1
    for w in range(len(temp)):
        number[w] = temp[w]
    if TYPE == 0:
        colors = ["antique white" for h in number]
    else:
        colors = [((int)(x * 360) / 950) for x in number]
    paint(colors)


def radixsort(number, paint, speed):
    global TYPE
    colors = []
    maximum = max(number)
    exp = 1
    while (maximum // exp >= 1):
        countsort(number, exp, paint)
        if TYPE == 0:
            colors = ["antique white" for h in number]
        else:
            colors = [((int)(x * 360) / 950) for x in number]
        paint(colors)
        time.sleep(1 / speed)
        exp *= 10
```





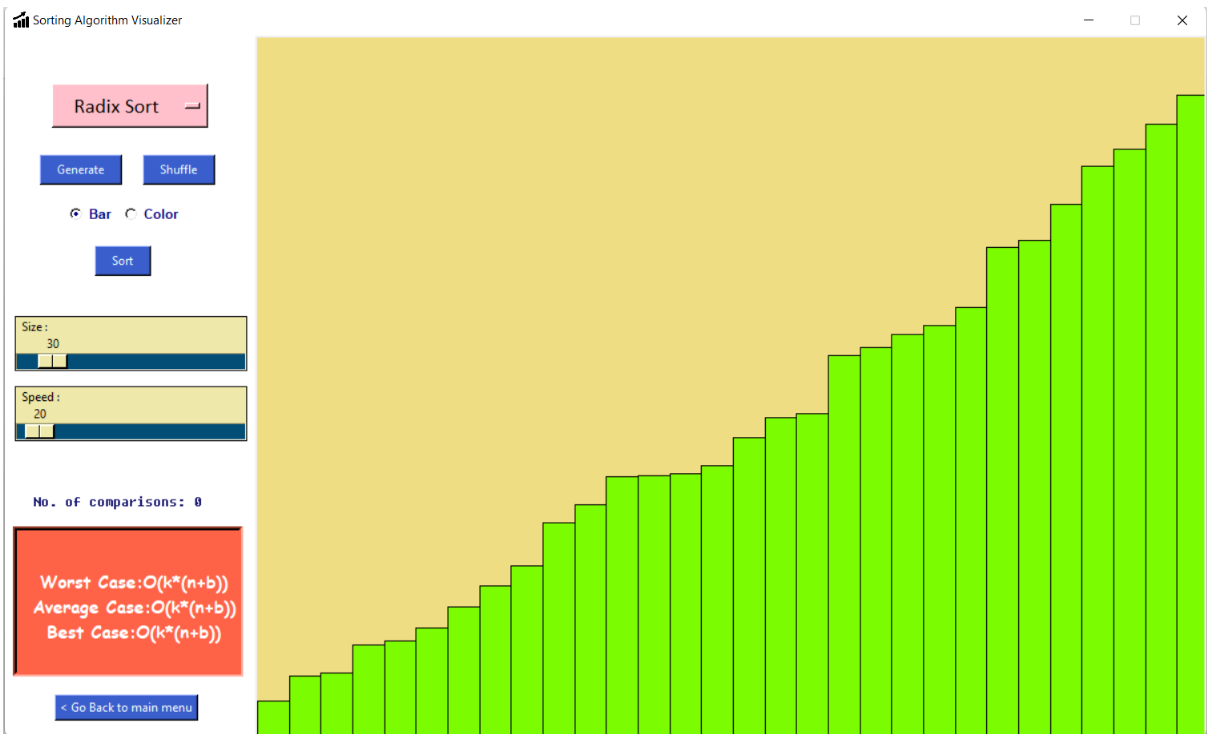
No. of comparison: 0

Worst Case:$O(n^2)$
Average Case:$O(n^2)$
Best Case:$O(n*\log n)$



No. of comparisons: 64

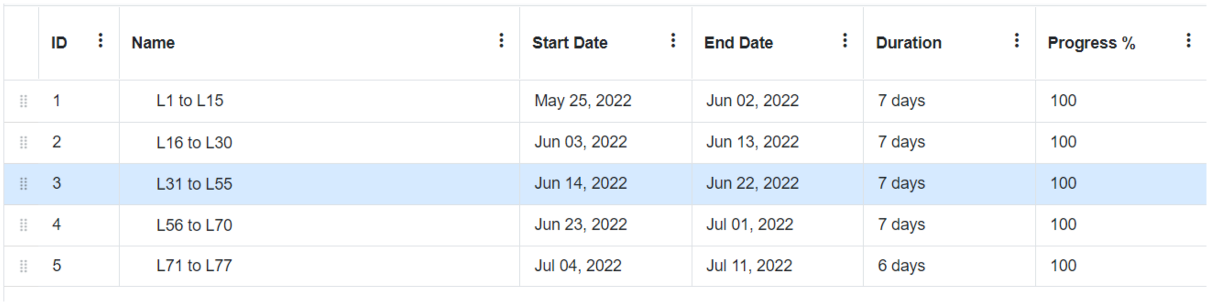
Worst Case:$O(n^2)$
Average Case:$O(n^2)$
Best Case:O(n*log n)



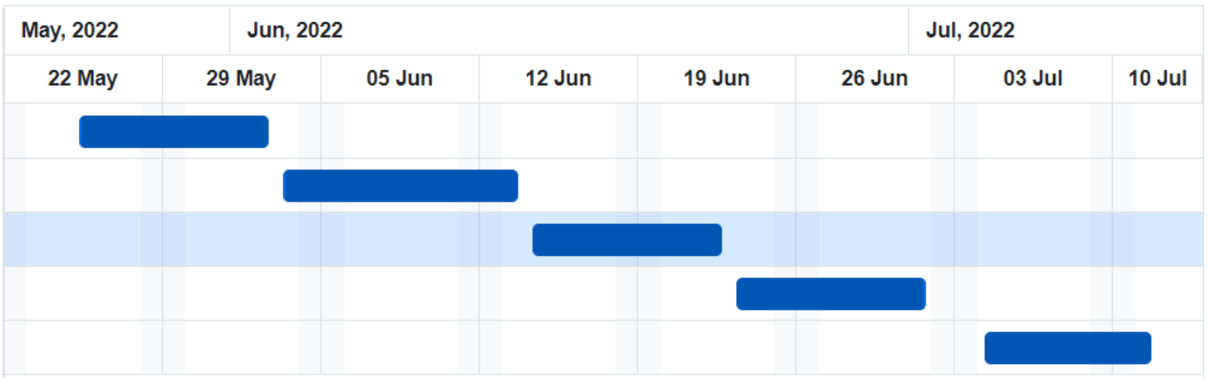
No. of comparisons: 0

Worst Case:O(k*(n+b))
Average Case:O(k*(n+b))
Best Case:O(k*(n+b))

Gantt Chart



ID	Name	Start Date	End Date	Duration	Progress %
1	L1 to L15	May 25, 2022	Jun 02, 2022	7 days	100
2	L16 to L30	Jun 03, 2022	Jun 13, 2022	7 days	100
3	L31 to L55	Jun 14, 2022	Jun 22, 2022	7 days	100
4	L56 to L70	Jun 23, 2022	Jul 01, 2022	7 days	100
5	L71 to L77	Jul 04, 2022	Jul 11, 2022	6 days	100



Bibliography

1. Google
2. YouTube
3. GitHub
4. https://stackoverflow.com/