ALLIANCE UNIVERSITY

Private University established in Karnataka State by Act No.34 of year 2010
Recognized by the University Grants Commission (UGC), New Delhi

Project Report

Master of Computer Application
Semester – II

DATA STRUCTURES AND ALGORITHMS

FINAL COURSE PROJECT

Project title : Sorting Algorithm Visualizer and Analyzer using Tkinter

Submitted By:

1. Haidar Ali – 2411022250090
2. Akbar Husain – 2411022250092
3. Abbas Ali – 2411022250091

Faculty Signature: ______________________ **HOD Signature:** ______________________

Department of Computer Application
Alliance University
Chandapura - Anekal Main Road, Anekal
Bengaluru - 562 106

Sorting Algorithm Visualizer and Analyzer using Tkinter 🎉

Introduction :
This project is an interactive GUI application built with Python's Tkinter library, which visualizes the step-by-step execution of popular sorting algorithms. It also measures and compares their performance in real time. This celebration-worthy tool brings data structures to life!

Objective:
To develop an educational tool that demonstrates how sorting algorithms work through visualization, helping users understand sorting logic and analyze algorithm efficiency.

Scope of Project:
This project covers the visualization and performance analysis of Bubble Sort, Insertion Sort, and Selection Sort. It is designed for students and educators interested in learning or teaching sorting algorithm concepts.

Modules Used:

- `tkinter` : for GUI components
- `random` : to generate random arrays
- `time` : to measure execution time of sorting algorithms
- `matplotlib` : to generate bar charts comparing performance

System Design and Architecture:
The application consists of three main components:

1. **GUI Renderer**: Built using `tkinter` for interaction and displaying sorting bars.
2. **Sorting Engine**: Contains sorting algorithms with embedded visualization logic.
3. **Analyzer Module**: Uses `matplotlib` to compare time complexity visually.

Code and Implementation:

```
import tkinter as tk
import random
import time
import matplotlib.pyplot as plt
from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg

def draw_bars(canvas, arr, colors):
    canvas.delete("all")
    c_height = 300
    c_width = 600
    bar_width = c_width / len(arr)
    max_val = max(arr)
    for i, val in enumerate(arr):
        x0 = i * bar_width
        y0 = c_height - (val / max_val) * c_height
        x1 = (i + 1) * bar_width
        y1 = c_height
        canvas.create_rectangle(x0, y0, x1, y1, fill=colors[i], outline="black")
    canvas.update()

def bubble_sort(canvas, arr):
    n = len(arr)
    start_time = time.time()
    for i in range(n):
        for j in range(n - i - 1):
            colors = ["red" if x == j or x == j + 1 else "skyblue" for x in range(n)]
            draw_bars(canvas, arr, colors)
            if arr[j] > arr[j + 1]:
                arr[j], arr[j + 1] = arr[j + 1], arr[j]
            time.sleep(0.1)
    draw_bars(canvas, arr, ["green" for _ in range(n)])
    return time.time() - start_time

def insertion_sort(canvas, arr):
    n = len(arr)
    start_time = time.time()
    for i in range(1, n):
        key = arr[i]
        j = i - 1
        while j >= 0 and arr[j] > key:
            arr[j + 1] = arr[j]
            j -= 1
            colors = ["red" if x == j + 1 else "skyblue" for x in range(n)]
            draw_bars(canvas, arr, colors)
            time.sleep(0.1)
        arr[j + 1] = key
    draw_bars(canvas, arr, ["green" for _ in range(n)])
    return time.time() - start_time

def selection_sort(canvas, arr):
    n = len(arr)
    start_time = time.time()
    for i in range(n):
        min_idx = i
        for j in range(i + 1, n):
            if arr[j] < arr[min_idx]:
                min_idx = j
            colors = ["red" if x == j or x == min_idx else "skyblue" for x in range
(n)]
            draw_bars(canvas, arr, colors)
            time.sleep(0.1)
        arr[i], arr[min_idx] = arr[min_idx], arr[i]
    draw_bars(canvas, arr, ["green" for _ in range(n)])
    return time.time() - start_time

def start_sorting(sort_func, canvas, arr, results, analyzer_frame):
    time_taken = sort_func(canvas, arr.copy())
    results[sort_func.__name__] = time_taken
    update_analyzer(results, analyzer_frame)

def update_analyzer(results, analyzer_frame):
    for widget in analyzer_frame.winfo_children():
        widget.destroy()
    fig, ax = plt.subplots()
    ax.bar(results.keys(), results.values(), color=['blue', 'red', 'green'])
    ax.set_ylabel("Time (seconds)")
    ax.set_title("Sorting Algorithm Performance")
    canvas = FigureCanvasTkAgg(fig, master=analyzer_frame)
    canvas.get_tk_widget().pack()
    canvas.draw()

def main():
    global root
    root = tk.Tk()
    root.title("Sorting Visualizer & Analyzer 🎉")
    canvas = tk.Canvas(root, width=600, height=300, bg="white")
    canvas.pack()
    arr = [random.randint(10, 100) for _ in range(20)]
    draw_bars(canvas, arr, ["skyblue" for _ in range(len(arr))])
    results = {}
    analyzer_frame = tk.Frame(root)
    analyzer_frame.pack()
    tk.Button(root, text="Bubble Sort", command=lambda: start_sorting(bubble_
sort, canvas, arr, results, analyzer_frame)).pack(side=tk.LEFT)
    tk.Button(root, text="Insertion Sort", command=lambda: start_sorting(inserti
on_sort, canvas, arr, results, analyzer_frame)).pack(side=tk.LEFT)
    tk.Button(root, text="Selection Sort", command=lambda: start_sorting(selec
tion_sort, canvas, arr, results, analyzer_frame)).pack(side=tk.LEFT)
    tk.Button(root, text="Shuffle", command=lambda: draw_bars(canvas, rando
m.sample(arr, len(arr)), ["skyblue"] * len(arr))).pack(side=tk.LEFT)
    root.mainloop()

if __name__ == "__main__":
    main()
```

Explanation of Code:
Each function is modularized for clarity and purpose. The sorting functions update the canvas after each swap or key comparison, providing visual feedback. The `start_sorting` function also benchmarks the performance, and `update_analyzer` dynamically renders a bar chart of the timings.

Output:

- Visualization of the sorting process with color-coded steps
- Final sorted array displayed in green
- A bar chart comparing the time taken by each algorithm

Concept Used for DSA:

- Bubble Sort (O(n^2))
- Insertion Sort (O(n^2))
- Selection Sort (O(n^2))
- Time Complexity Measurement
- GUI for visual feedback

Real-World Applications:

- Educational platforms for teaching sorting
- Algorithm analysis tools for developers
- Demonstrations in programming bootcamps and workshops

Advantages:

- Interactive and user-friendly
- Visual learning aids
- Real-time performance comparison

Limitations:

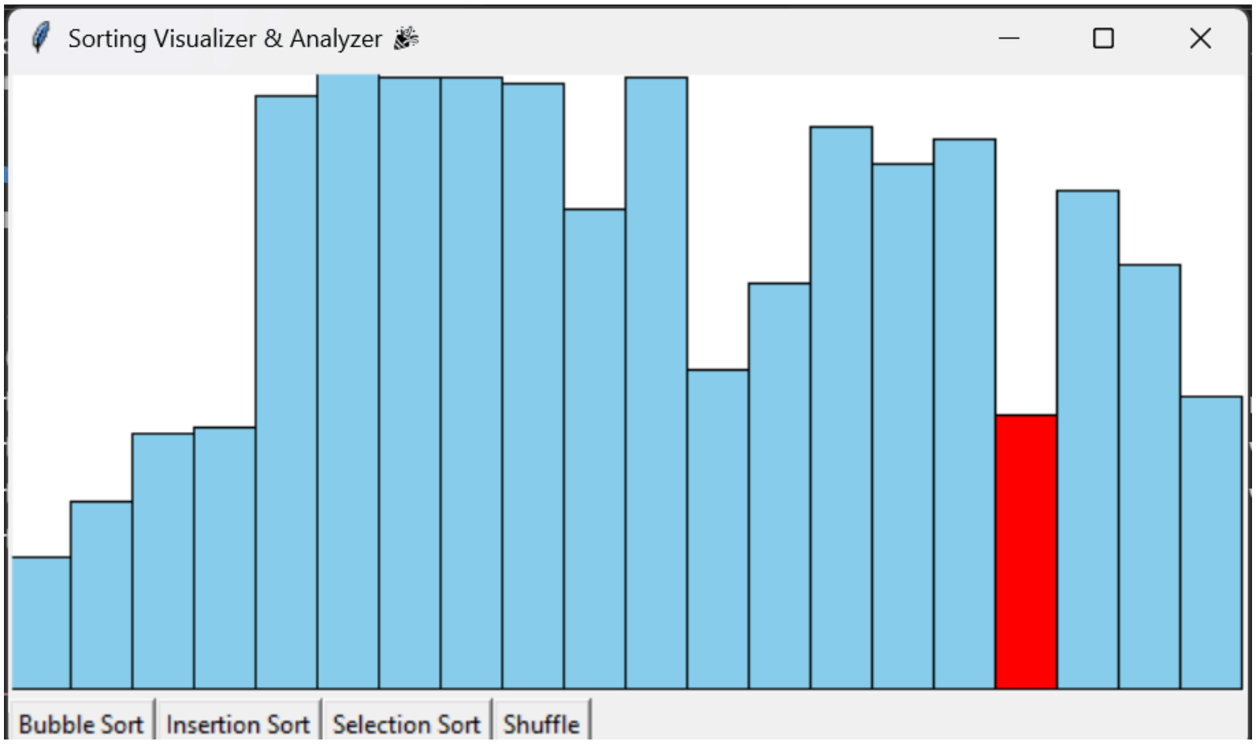
- Limited to only three sorting algorithms
- Not suitable for large data sets
- GUI may lag with higher array sizes

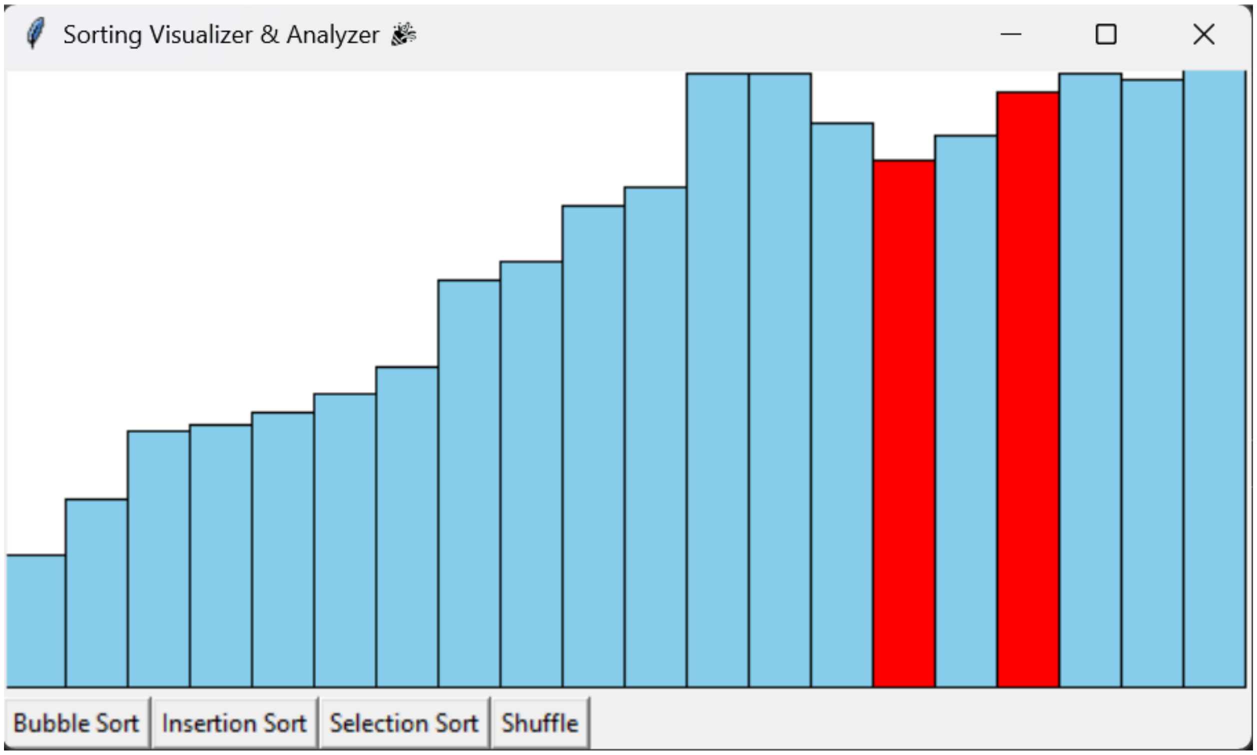
Conclusion:
The Sorting Visualizer is an effective educational tool for visualizing and comparing sorting algorithms. It helps learners grasp the inner workings of sorting logic and algorithm efficiency.

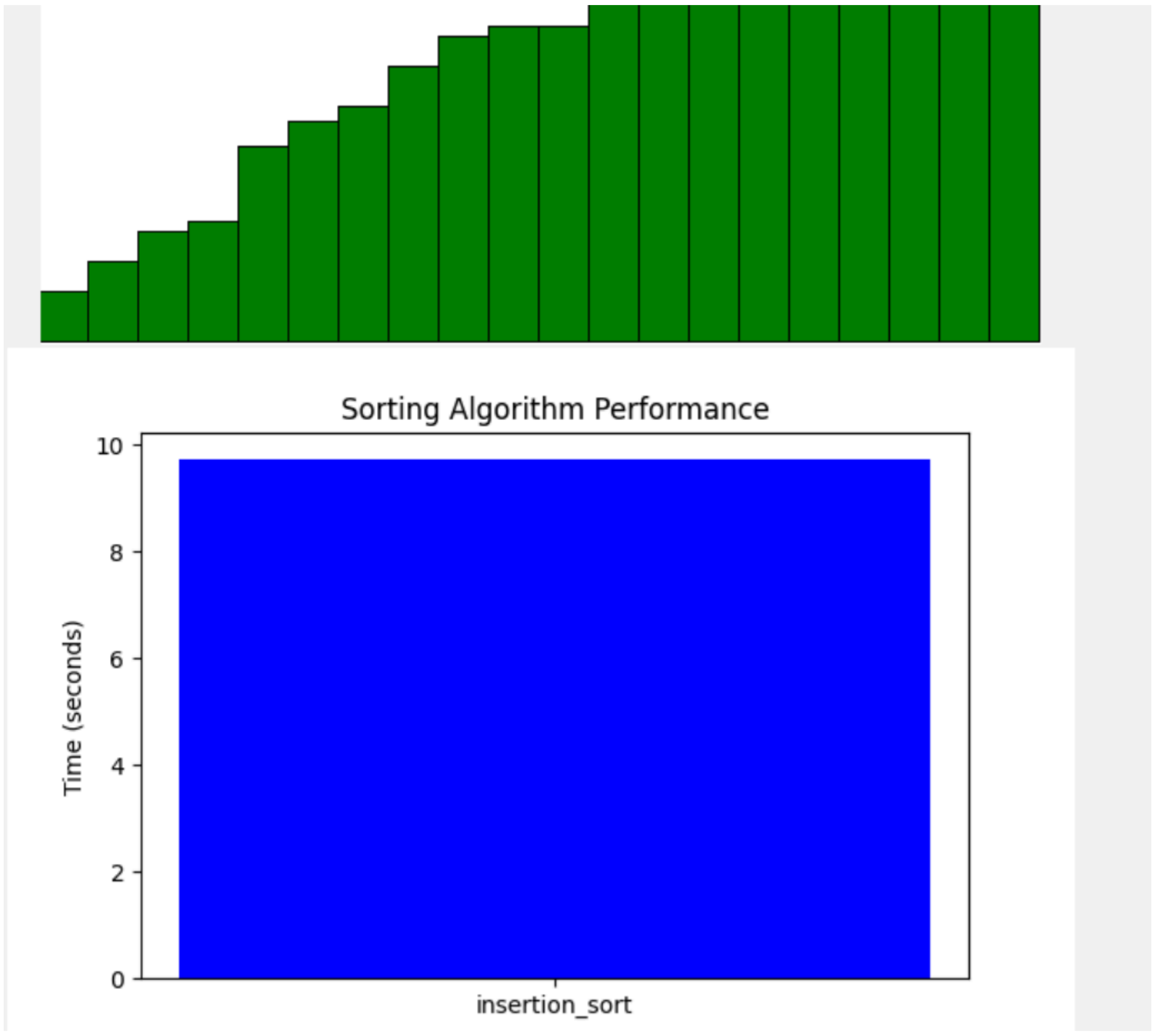
Future Scope:

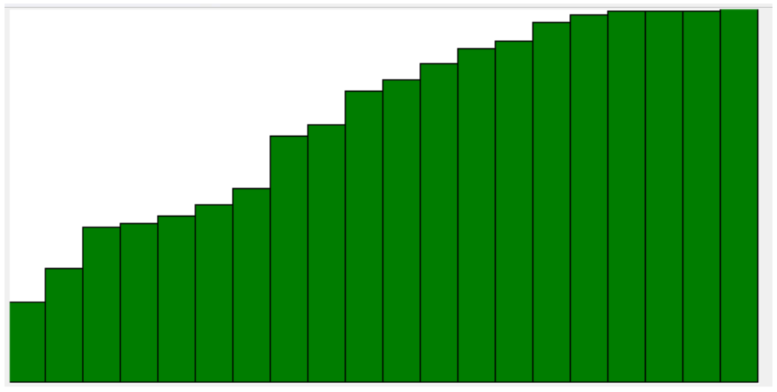
- Add more algorithms like Merge Sort, Quick Sort, and Heap Sort
- Enhance GUI with controls for speed and array size
- Export performance reports

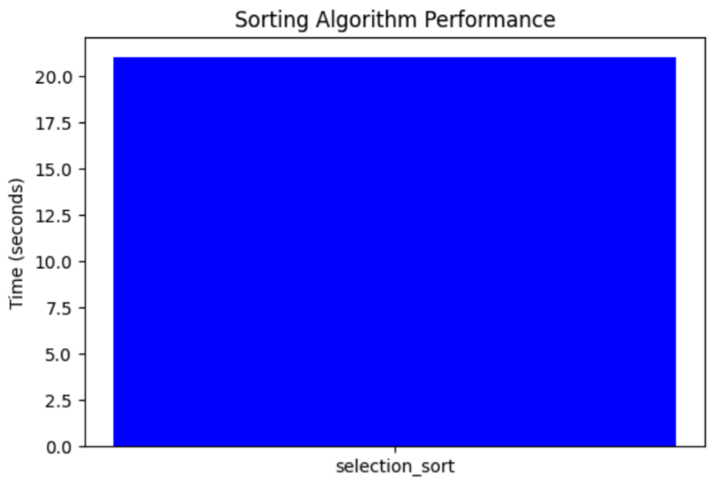
Screenshot of Output:






Sorting Algorithm Performance
Time (seconds)
0
2
4
6
8
10
insertion_sort





References:

- Python Official Documentation
- Matplotlib Guide
- GeeksforGeeks Sorting Algorithms
- Tkinter Tutorials by Real Python