Design and Analysis of algorithm.

Linear and binary search

Selection and bubble search

Dr/Ahmed Taha

Eng/ Doa Elabd

```python
# Linear search algorithm
def linear_search(data, target):
    steps = []  # Store steps for printing
    steps.append(f"Entered data: {data}")
    for i in range(len(data)):
        steps.append(f"Step {i+1}: Comparing {data[i]} with {target}")
        if data[i] == target:
            output = f"Target found at position {i+1}" + "\n" +
"\n".join(steps)
            return output  # Found it!
    steps.append("Target not found")
    return -1 # Not found

# Sorting algorithm (Selection Sort)
def selection_sort(data):
    steps = []  # Store steps for printing
    n = len(data)
    steps.append(f"Unsorted data: {data}")


    for i in range(n - 1):  # Iterate through the array
        # Find the index of the minimum element in the remaining unsorted array
        min_index = i
        for j in range(i + 1, n):
            steps.append(f"Step {len(steps)}: Searching for the smallest
element  from index {i} to {n - 1}")
            if data[j] < data[min_index]:
                min_index = j

        # Swap the minimum element with the current element
        if min_index != i:
            steps.append(f"Step {len(steps)}: Swapping the smallest element
{data[min_index]} with {data[i]}")
            data[i], data[min_index] = data[min_index], data[i]

    steps.append("Sorting completed")
    result = "\n".join(steps) + f"\nSorted data: {data}"
    return result



# Binary search algorithm
def binary_search(data, target):
    data = sorted(data)  # Binary search requires a sorted list
    low = 0
```

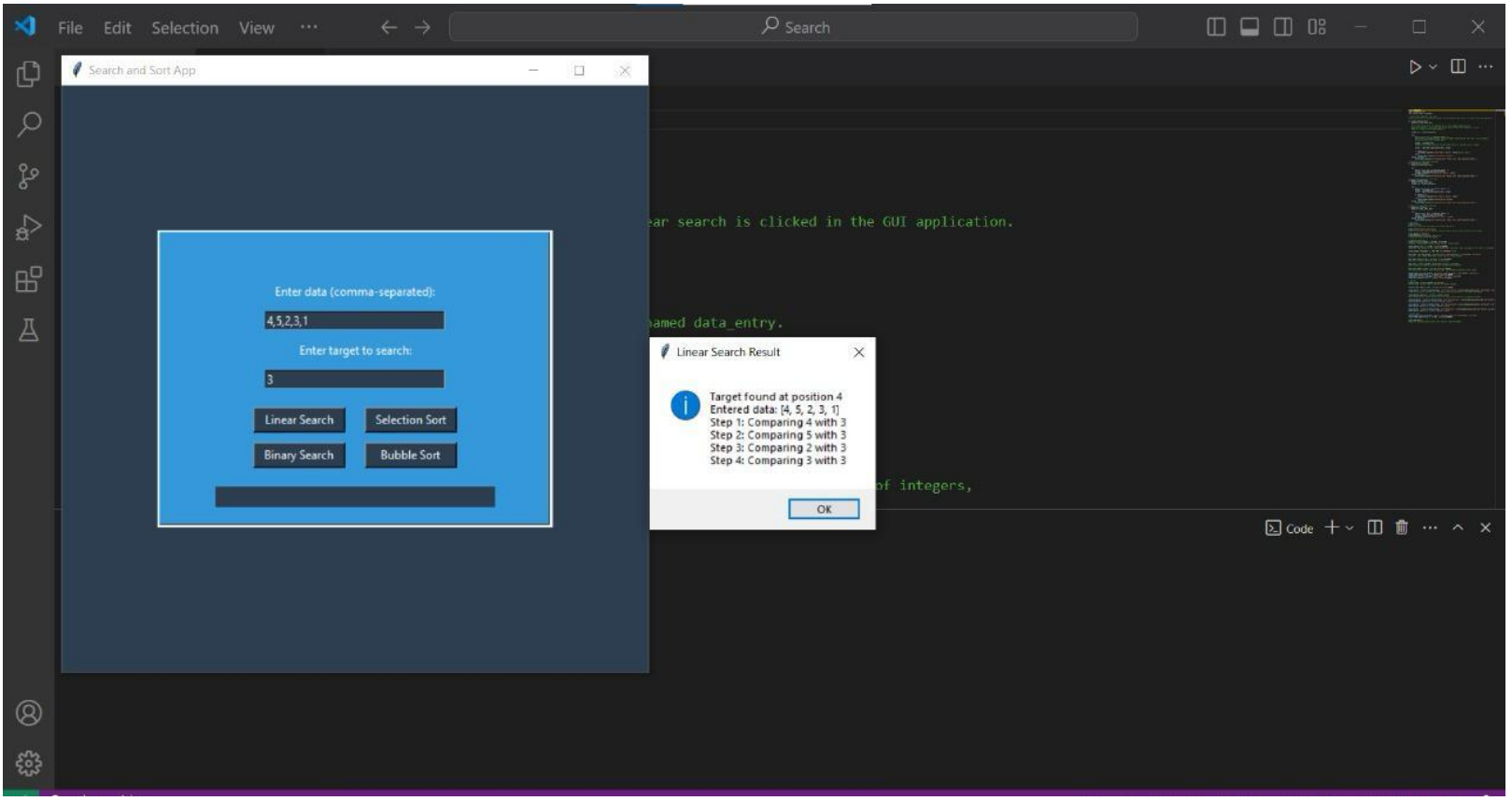
```
    high = len(data) - 1
    steps = []  # Store steps for printing
    steps.append(f'Entered data: {data}')
    while low <= high:
        mid = (low + high) // 2
        steps.append(f"Step {len(steps)}: Checking {data[mid]}")
        if target == data[mid]:
            result = "\n".join(steps) + f"\nTarget found at index {mid}"  # Found
it!
            return result
        elif target < data[mid]:
            high = mid - 1
        else:
            low = mid + 1
    result = "\n".join(steps) + "\nTarget not found"  # Target not found
    return result

# Bubble sort algorithm
def bubble_sort(data):
    steps = []  # Store steps for printing
    steps.append(f"Unsorted data: {data}")
    n = len(data)
    for i in range(1, n):
        for j in range(0, n - i):
            steps.append(f"Step {len(steps)}: Comparing {data[j]} with
{data[j+1]}")
            if data[j] > data[j + 1]:
                steps.append(f"Step {len(steps)}: Swapping {data[j]} with
{data[j+1]}")
                data[j], data[j + 1] = data[j + 1], data[j]
    steps.append("Sorting completed")

    result = "\n".join(steps) + f"\nSorted data: {data}"
    return result
```

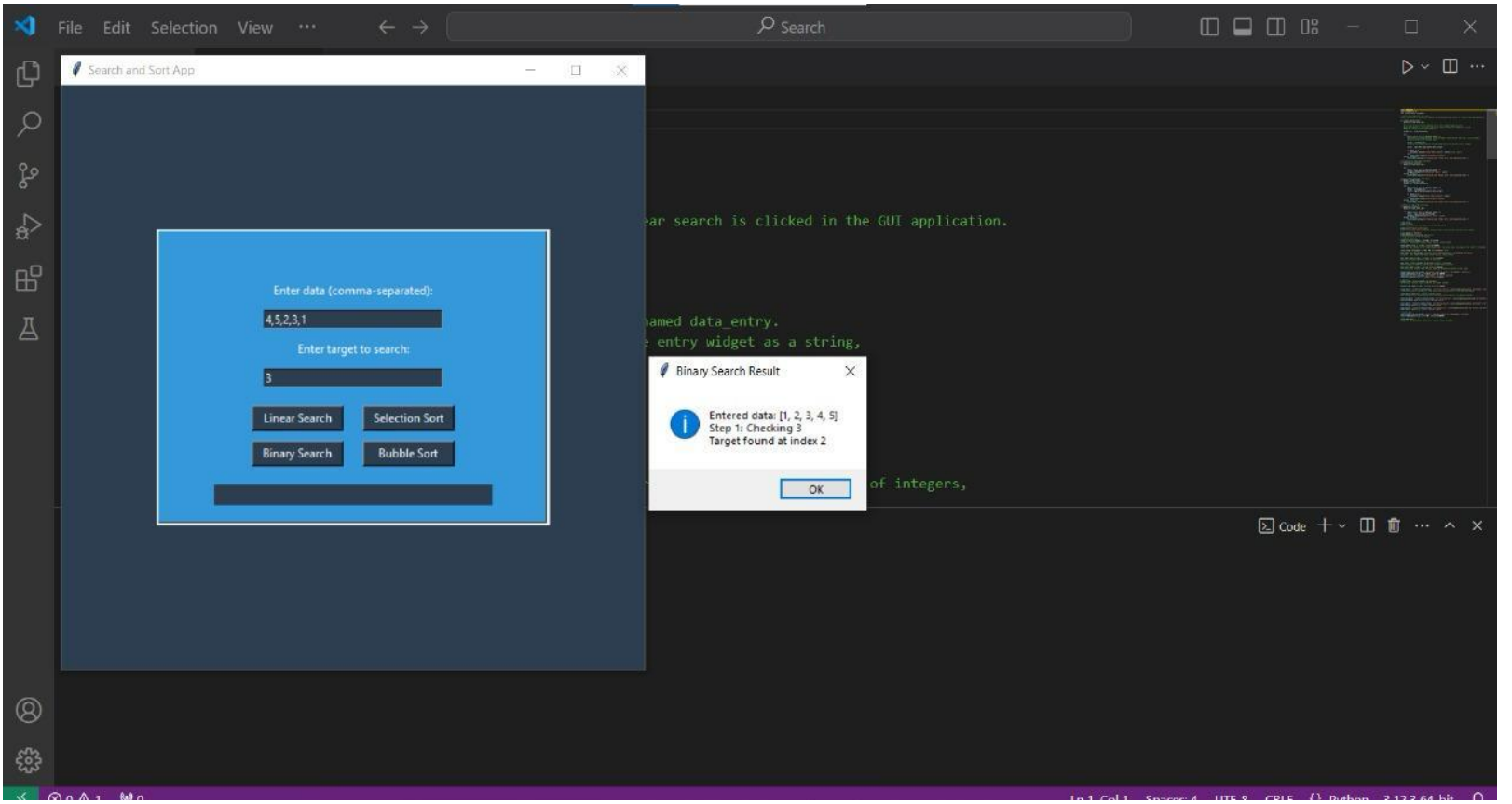
```
import algorithm
import tkinter as tk
from tkinter import messagebox

# Linear search algorithm click event
#This function will be called when a button associated with linear search is
clicked in the GUI application.

def linear_search_click():
    data_str = data_entry.get()

    #This line retrieves the text entered into an entry widget named data_entry.
    #The .get() method is used to get the current content of the entry widget as
a string,
    #and it's stored in the variable data_str.

    target_str = target_entry.get()

    try:
        data = [int(x) for x in data_str.split(',')]
        #it converts the comma-separated string of numbers entered by the user
into a list of integers,
        #which is stored in the variable data.

        target = int(target_str)
        #converts the string target_str (target value entered by the user) into
an integer

        result = algorithm.linear_search(data, target)

        if result != -1:
            messagebox.showinfo("Linear Search Result", result) # type: ignore
        else:
            result_label.config(text="Target not found.")
    except ValueError:
        result_label.config(text="Invalid input. Please enter comma-separated
numbers.")

# Selection sort algorithm click event
def selection_sort_click():
    data_str = data_entry.get()

    try:
        data = [int(x) for x in data_str.split(',')]
        result = algorithm.selection_sort(data)
```

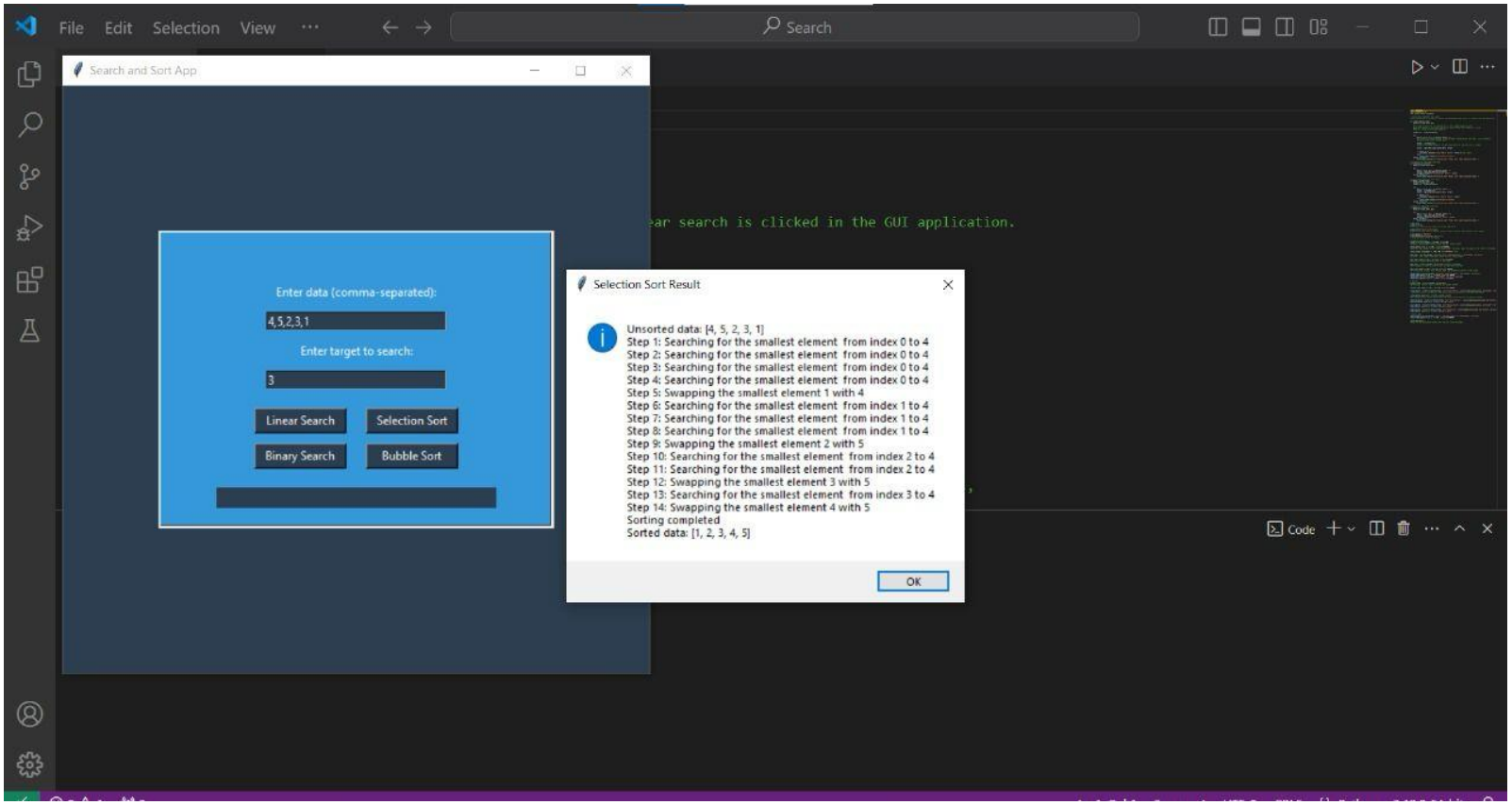
```
        messagebox.showinfo("Selection Sort Result", result)
    except ValueError:
        result_label.config(text="Invalid input. Please enter comma-separated
numbers.")

# Binary search algorithm click event
def binary_search_click():
    data_str = data_entry.get()
    target_str = target_entry.get()

    try:
        data = [int(x) for x in data_str.split(',')]
        target = int(target_str)
        result = algorithm.binary_search(data, target)

        if result != -1:
            messagebox.showinfo("Binary Search Result", result)
        else:
            result_label.config(text="Target not found")
    except ValueError:
        result_label.config(text="Invalid input. Please enter comma-separated
numbers.")

# Bubble sort algorithm click event
def bubble_sort_click():
    data_str = data_entry.get()

    try:
        data = [int(x) for x in data_str.split(',')]
        result = algorithm.bubble_sort(data)
        messagebox.showinfo("Bubble Sort Result", result)
    except ValueError:
        result_label.config(text="Invalid input. Please enter comma-separated
numbers.")

# GUI setup
window = tk.Tk()
#This line creates the main window for the GUI application

window.title("Search and Sort App")
#This line sets the title of the main window to "Search and Sort App" using the
title() method.

window.geometry("600x600")
window.configure(bg="#2c3e50") #dark blue
```

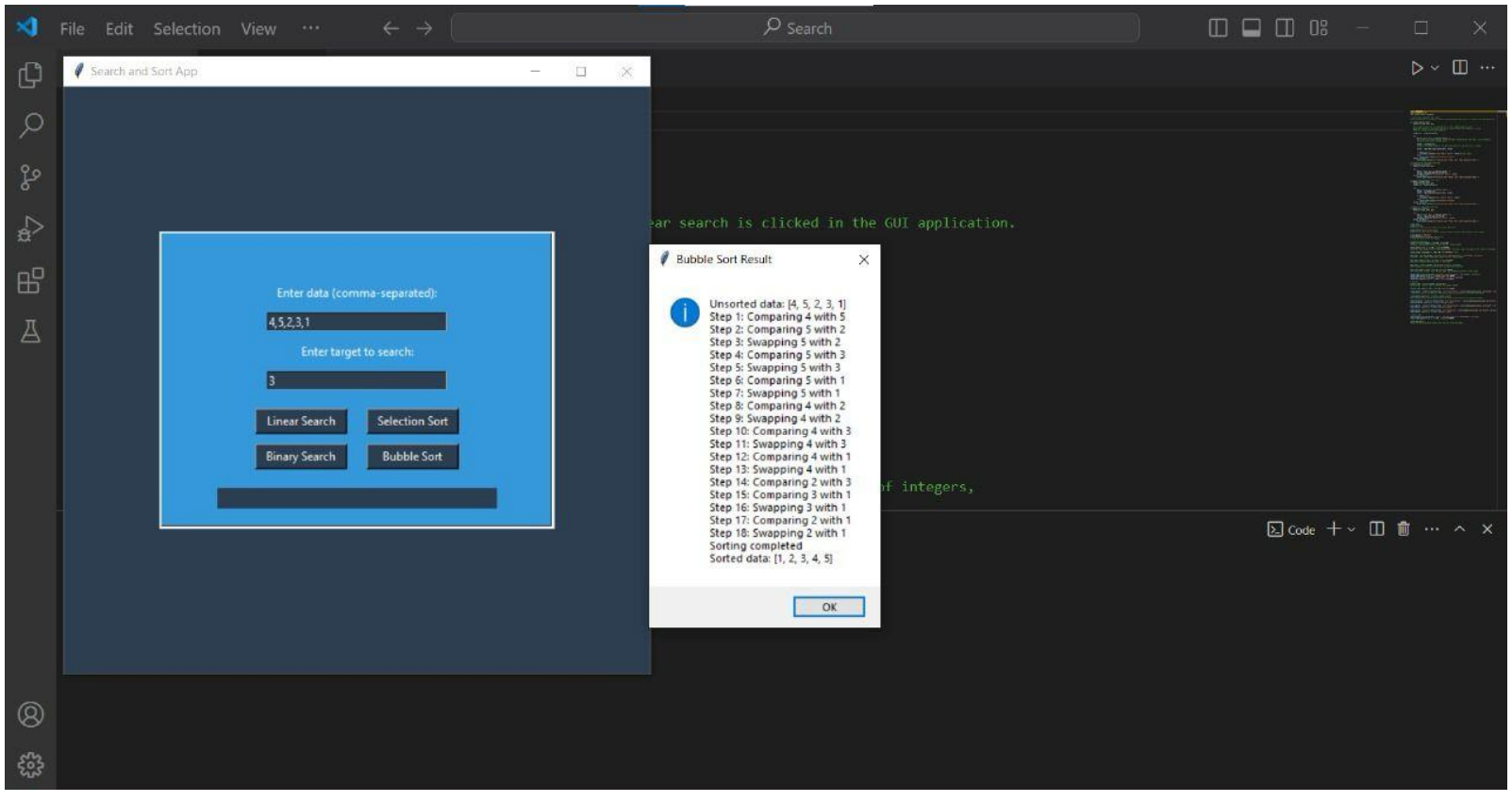
```
# background color of the main window

# Gradient background
canvas = tk.Canvas(window, width=400, height=300)
#creates a canvas widget (canvas) inside the main window (window)

canvas.place(relx=0.5, rely=0.5, anchor=tk.CENTER)
#specifies the placement of the canvas widget within the window. place the widget
in the center of the window

canvas.create_rectangle(0, 0, 400, 300, fill="#3498db") #blue

data_label = tk.Label(window, text="Enter data (comma-separated):", bg="#3498db",
fg="white")
#creates a label widget (data_label) inside the main window (window)

data_label.place(relx=0.5, rely=0.35, anchor=tk.CENTER)
#placing it slightly below the center of the window.

data_entry = tk.Entry(window, bg="#2c3e50",fg="white", width=30)
#Entry widgets are used to allow the user to input single-line text.

data_entry.place(relx=0.5, rely=0.4, anchor=tk.CENTER)
#It's positioned slightly below the data_label and centered horizontally in the
window.

target_label = tk.Label(window, text="Enter target to search:", bg="#3498db",
fg="white")
target_label.place(relx=0.5, rely=0.45, anchor=tk.CENTER)
target_entry = tk.Entry(window, bg="#2c3e50", fg="white", width=30)
target_entry.place(relx=0.5, rely=0.5, anchor=tk.CENTER)

# Buttons
buttons_frame = tk.Frame(window, bg="#3498db")
#frame widget (buttons_frame) inside the main window (window).

buttons_frame.place(relx=0.5, rely=0.6, anchor=tk.CENTER)

linear_button = tk.Button(buttons_frame, text="Linear Search",
command=linear_search_click, bg="#2c3e50", fg="white", width=12)
# when this button is clicked, the linear_search_click() function will be called.
(dark shade)

linear_button.grid(row=0, column=0, padx=10, pady=5)
```

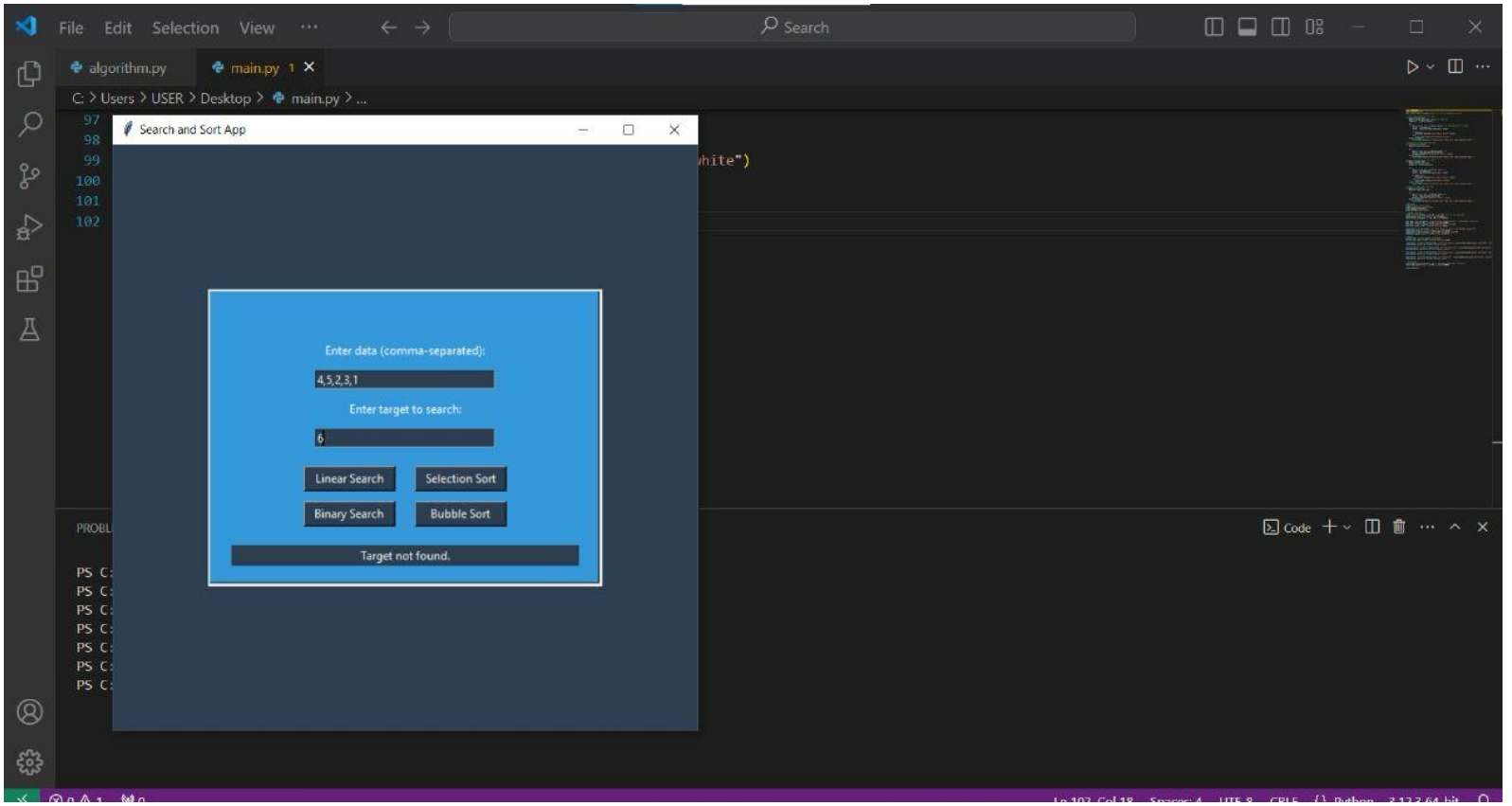
```
#the placement of the linear_button within the buttons_frame using the grid
geometry manager.

selection_button = tk.Button(buttons_frame, text="Selection Sort",
command=selection_sort_click, bg="#2c3e50", fg="white", width=12)
selection_button.grid(row=0, column=1, padx=10, pady=5)

binary_button = tk.Button(buttons_frame, text="Binary Search",
command=binary_search_click, bg="#2c3e50", fg="white", width=12)
binary_button.grid(row=1, column=0, padx=10, pady=5)

bubble_button = tk.Button(buttons_frame, text="Bubble Sort",
command=bubble_sort_click, bg="#2c3e50", fg="white", width=12)
bubble_button.grid(row=1, column=1, padx=10, pady=5)

# Result Label
result_label = tk.Label(window, text="", width=40, anchor="w", bg="#2c3e50",
fg="white")
result_label.place(relx=0.5, rely=0.7, anchor=tk.CENTER)

window.mainloop()
#keeps the GUI application running until the user closes the window.
```


File Edit Selection View ···
Search
Search and Sort App
Enter data (comma-separated):
4,5,2,3,1
Enter target to search:
3
Linear Search
Selection Sort
Binary Search
Bubble Sort
Linear Search Result
Target found at position 4
Entered data: [4, 5, 2, 3, 1]
Step 1: Comparing 4 with 3
Step 2: Comparing 5 with 3
Step 3: Comparing 2 with 3
Step 4: Comparing 3 with 3
OK
ear search is clicked in the GUI application.
amed data_entry.
of integers,
Code


File Edit Selection View ···
Search
Search and Sort App
Enter data (comma-separated):
4,5,2,3,1
Enter target to search:
3
Linear Search
Selection Sort
Binary Search
Bubble Sort
Binary Search Result
Entered data: [1, 2, 3, 4, 5]
Step 1: Checking 3
Target found at index 2
OK
ear search is clicked in the GUI application.
amed data_entry.
entry widget as a string,
of integers,
Code


File Edit Selection View ···
Search
Search and Sort App
Enter data (comma-separated):
4,5,2,3,1
Enter target to search:
3
Linear Search
Selection Sort
Binary Search
Bubble Sort
ear search is clicked in the GUI application.
Selection Sort Result
Unsorted data: [4, 5, 2, 3, 1]
Step 1: Searching for the smallest element from index 0 to 4
Step 2: Searching for the smallest element from index 0 to 4
Step 3: Searching for the smallest element from index 0 to 4
Step 4: Searching for the smallest element from index 0 to 4
Step 5: Swapping the smallest element 1 with 4
Step 6: Searching for the smallest element from index 1 to 4
Step 7: Searching for the smallest element from index 1 to 4
Step 8: Searching for the smallest element from index 1 to 4
Step 9: Swapping the smallest element 2 with 5
Step 10: Searching for the smallest element from index 2 to 4
Step 11: Searching for the smallest element from index 2 to 4
Step 12: Swapping the smallest element 3 with 5
Step 13: Searching for the smallest element from index 3 to 4
Step 14: Swapping the smallest element 4 with 5
Sorting completed
Sorted data: [1, 2, 3, 4, 5]
OK
Code


File Edit Selection View ···
Search
Search and Sort App
Enter data (comma-separated):
4,5,2,3,1
Enter target to search:
3
Linear Search
Selection Sort
Binary Search
Bubble Sort
ear search is clicked in the GUI application.
Bubble Sort Result
Unsorted data: [4, 5, 2, 3, 1]
Step 1: Comparing 4 with 5
Step 2: Comparing 5 with 2
Step 3: Swapping 5 with 2
Step 4: Comparing 5 with 3
Step 5: Swapping 5 with 3
Step 6: Comparing 5 with 1
Step 7: Swapping 5 with 1
Step 8: Comparing 4 with 2
Step 9: Swapping 4 with 2
Step 10: Comparing 4 with 3
Step 11: Swapping 4 with 3
Step 12: Comparing 4 with 1
Step 13: Swapping 4 with 1
Step 14: Comparing 2 with 3
Step 15: Comparing 3 with 1
Step 16: Swapping 3 with 1
Step 17: Comparing 2 with 1
Step 18: Swapping 2 with 1
Sorting completed
Sorted data: [1, 2, 3, 4, 5]
OK
Code


File Edit Selection View
Search
algorithm.py
main.py 1
C: > Users > USER > Desktop > main.py > ...
Search and Sort App
Enter data (comma-separated):
4,5,2,3,1
Enter target to search:
6
Linear Search
Selection Sort
Binary Search
Bubble Sort
Target not found.
Code

Team members:

Merna Hesham 192100080

Merna Ahmed 192100144

Ali Hazim 192100116