Practical Assignment 2: Searching & Sorting Algorithms (15 marks)

Instructions:
This assignment has two parts designed to test your understanding of searching and sorting algorithms covered in Unit 4. You are required to submit your work in a GitHub repository with the following naming structure (folder name):

> **StudentName_studentNumber_A2**
>
> Example: `SonamYangchen_02190108_A2`

Your Python file submissions must be inside the above-mentioned repository. For each part, you must have two separate Python files, their names must be as follows:

- `StudentName_studentNumber_A2_PA.py` (for Part A)
- `StudentName_studentNumber_A2_PB.py` (for Part B)

Example:

- `SonamYangchen_02190108_A2_PA.py`
- `SonamYangchen_02190108_A2_PB.py`

Please also submit a report in .doc format for assignment grading on VLE with your complete Python code and include:

- Screenshots of the outputs for each part
- Brief explanations of your implementations

Your report assignment will be subject to Turnitin plagiarism checker, and your GitHub code submissions will be checked for functionality.

Part A: Searching Algorithms Implementation

Create a command-line program in a single Python file that implements various searching algorithms working with simple lists.

Program Flow

1. Display a menu showing available search operations (1-3)
2. Prompt the user to select an operation by entering a number from 1-3
3. Based on the selection, prompt for and collect the required input
4. Execute the selected search algorithm and display the result
5. Allow the user to continue with another search or exit

Required Functions

1. **Linear Search - Find a Student ID**

Input:

A list of student IDs (you can hardcode a list of 20 student IDs)
A target student ID to search for

Output:

Position of the student ID if found (1-indexed for user friendliness)
"Student ID not found" message if not found

Example:

```
Student IDs: [1001, 1005, 1002, 1008, 1003, 1010, 1004, 1009, ...]
Search for: 1008
Output: "Student ID 1008 found at position 4"
```

Binary Search - Find a Score

Input:

A sorted list of test scores (you can hardcode a sorted list of 20 scores)
A target score to search for

Output:

Position of the score if found
"Score not found" message if not found
Note: The list MUST be sorted for binary search to work

Example:

```
Sorted Scores: [45, 52, 58, 63, 67, 72, 75, 78, 82, 85, 88, 90, 92, 95, 98]
Search for: 78
Output: "Score 78 found at position 8"
```

Sample Program Output:

```
=== Searching Algorithms Menu ===

Select a search operation (1-3):
1. Linear Search - Find Student ID
2. Binary Search - Find Score
3. Exit program

Enter your choice: 1
Searching in the list: [1001, 1005, 1002, 1008, 1003, 1010, 1004, 1009, 1007, 1012, ...]
Enter Student ID to search: 1008
Result: Student ID 1008 found at position 4 Comparisons made: 4
Would you like to perform another search? (y/n): y

Select a search operation (1-3): 2
Sorted scores: [45, 52, 58, 63, 67, 72, 75, 78, 82, 85, 88, 90, 92, 95, 98]
Enter score to search: 78
Result: Score 78 found at position 8 Comparisons made: 3
Would you like to perform another search? (y/n): n

Thank you for using the search program!
```

Part B: Sorting Algorithms Implementation

Create a command-line program in a single Python file that implements various sorting algorithms working with simple lists.

Program Flow

1. Display a menu showing available sorting operations (1-4)
2. Prompt the user to select an operation by entering a number from 1-4
3. Based on the selection, execute the sorting algorithm
4. Display the original list, sorted list
5. Allow the user to continue with another sort or exit

Required Functions

1. **Bubble Sort - Sort Student Names**

Input:

A list of 15 student names (hardcoded)

Output:

Original unsorted list
Sorted list (alphabetically)

Example:

```
Original: ["Kado", "Bobchu",  "Zamu", "Nado", "Lemo", ...]
Sorted: ["Bobchu", "Kado", "Lemo", "Nado", "Zamu", ...]
```

2. **Insertion Sort - Sort Test Scores**

Input:

A list of 20 test scores (hardcoded, unsorted)

Output:

Original unsorted list
Sorted list (ascending order)
Display top 5 scores

Example:

```
Original: [78, 45, 92, 67, 88, 54, 73, ...]
Sorted: [45, 54, 67, 73, 78, 88, 92, ...]

Top 5 Scores:
1. 92
2. 88
3. 78
4. 73
5. 67
```

3. Quick Sort - Sort Book Prices

Input:

A list of 15 book prices (hardcoded, unsorted)

Output:

Original unsorted list

Sorted list (ascending order)

Number of recursive calls made

Example:

```
Original prices: [450, 230, 678, 125, 890, ...]
Sorted prices: [125, 230, 450, 678, 890, ...]
Recursive calls: 28
```

Sample Program Output

```
=== Sorting Algorithms Menu ===
Select a sorting operation (1-4):
1. Bubble Sort - Sort Student Names
2. Insertion Sort - Sort Test Scores
3. Quick Sort - Sort Book Prices
4. Exit program

Enter your choice: 2
Original scores: [78, 45, 92, 67, 88, 54, 73, 82, 91, 59, 76, 85, 48, 93, 71,
89, 57, 80, 69]

Performing Insertion Sort...
Sorted scores: [45, 48, 54, 57, 59, 62, 67, 69, 71, 73, 76, 78, 80, 82, 85, 8
89, 91, 92, 93]
Top 5 Scores:
1. 93
2. 92
3. 91
4. 89
5. 88

Would you like to perform another sort? (y/n): n
Thank you for using the sorting program!
```

Submission Checklist

- GitHub repository created with correct naming convention
- Part A Python file uploaded with correct name
- Part B Python file uploaded with correct name
- Report document includes code and screenshots
- All required functions implemented
- Code has helpful comments
- Program runs without error
- Screenshots show all menu options working

Submission Date:

16th November, 2025