

ASSIGNMENT-19.1

NAME:KOLA SNEHA

ROLL NO.:2403A510F4

BATCH NO.:06

Lab Question 1: Sorting Algorithm Translation

PROMPT:

💡 "Translate the following Java bubble sort program into Python using AI-assisted coding. Extend the code to handle user input, test multiple lists, and validate whether the input list is empty or contains non-numeric values. Include detailed comments and print intermediate steps to visualize the sorting process."

CODE:

```
ai_19.1.py > bubble_sort
1 def bubble_sort(arr, show_steps=False):
2     """
3     Function to perform Bubble Sort on a given list.
4     :param arr: List of integers or floats to be sorted
5     :param show_steps: If True, prints each swap operation
6     """
7
8     # Input validation
9     if not arr:
10         print("⚠ Error: The input list is empty.")
11         return
12     if not all(isinstance(x, (int, float)) for x in arr):
13         print("⚠ Error: The list contains non-numeric values.")
14         return
15
16     n = len(arr)
17     print(f"\nStarting Bubble Sort on: {arr}")
18
19
20     for i in range(n - 1):
21         swapped = False # Optimization: stop if no swaps occur
22         print(f"\nPass {i + 1}:")
23         for j in range(n - i - 1):
24             if arr[j] > arr[j + 1]:
25                 # Swap elements
26                 arr[j], arr[j + 1] = arr[j + 1], arr[j]
27                 swapped = True
28                 if show_steps:
29                     print(f"Swapped positions {j} and {j+1}: {arr}")
30             else:
31                 if show_steps:
32                     print(f"No swap needed for {arr[j]} and {arr[j+1]}")
33
34         if not swapped:
35             print("✅ No swaps performed – list is sorted early.")
36             break
37
```

```

ai_19.1.py > bubble_sort
1  def bubble_sort(arr, show_steps=False):
37      break
38
39      print("✅ Final Sorted List:", arr)
40
41
42
43
44  def main():
45      print("=== Bubble Sort Demonstration ===")
46
47
48      test_cases = [
49          [5, 1, 4, 2, 8],
50          [12.5, 3.7, 9.1, 1.0],
51          [], # Empty list
52          [3, 'a', 2], # Non-numeric value
53          [10, 9, 8, 7, 6, 5] # Descending order
54      ]
55
56      for idx, case in enumerate(test_cases, start=1):
57          print(f"\n--- Test Case {idx} ---")
58          bubble_sort(case.copy(), show_steps=True)
59
60      try:
61          print("\nNow you can try your own list!")
62          user_input = input("Enter numbers separated by spaces: ").strip()
63          if user_input:
64              user_list = [float(x) if '.' in x else int(x) for x in user_input.split()]
65              bubble_sort(user_list, show_steps=True)
66          else:
67              print("No input provided. Skipping user test.")
68      except ValueError:
69          print("⚠ Invalid input! Please enter only numbers.")
70
71      print("\n=== Program Completed Successfully ===")
72
73
74
75  if __name__ == "__main__":
76      main()
77

```

OUTPUT:

```
--- Test Case 1 ---

Starting Bubble Sort on: [5, 1, 4, 2, 8]

Pass 1:
Swapped positions 0 and 1: [1, 5, 4, 2, 8]
Swapped positions 1 and 2: [1, 4, 5, 2, 8]
Swapped positions 2 and 3: [1, 4, 2, 5, 8]
No swap needed for 5 and 8

Pass 2:
No swap needed for 1 and 4
Swapped positions 1 and 2: [1, 2, 4, 5, 8]
No swap needed for 4 and 5

Pass 3:
No swap needed for 1 and 2
No swap needed for 2 and 4
✅ No swaps performed – list is sorted early.
✅ Final Sorted List: [1, 2, 4, 5, 8]

--- Test Case 2 ---

Starting Bubble Sort on: [12.5, 3.7, 9.1, 1.0]

Pass 1:
Swapped positions 0 and 1: [3.7, 12.5, 9.1, 1.0]
Swapped positions 1 and 2: [3.7, 9.1, 12.5, 1.0]
Swapped positions 2 and 3: [3.7, 9.1, 1.0, 12.5]

Pass 2:
No swap needed for 3.7 and 9.1
```

Pass 2:
No swap needed for 3.7 and 9.1
Swapped positions 1 and 2: [3.7, 1.0, 9.1, 12.5]

Pass 3:
Swapped positions 0 and 1: [1.0, 3.7, 9.1, 12.5]
✅ Final Sorted List: [1.0, 3.7, 9.1, 12.5]

--- Test Case 3 ---
⚠️ Error: The input list is empty.

--- Test Case 4 ---
⚠️ Error: The list contains non-numeric values.

--- Test Case 5 ---

Starting Bubble Sort on: [10, 9, 8, 7, 6, 5]

Pass 1:
Swapped positions 0 and 1: [9, 10, 8, 7, 6, 5]
Swapped positions 1 and 2: [9, 8, 10, 7, 6, 5]
Swapped positions 2 and 3: [9, 8, 7, 10, 6, 5]
Swapped positions 3 and 4: [9, 8, 7, 6, 10, 5]
Swapped positions 4 and 5: [9, 8, 7, 6, 5, 10]

Pass 2:
Swapped positions 0 and 1: [8, 9, 7, 6, 5, 10]
Swapped positions 1 and 2: [8, 7, 9, 6, 5, 10]
Swapped positions 2 and 3: [8, 7, 6, 9, 5, 10]
Swapped positions 3 and 4: [8, 7, 6, 5, 9, 10]

```
Pass 2:
Swapped positions 0 and 1: [8, 9, 7, 6, 5, 10]
Swapped positions 1 and 2: [8, 7, 9, 6, 5, 10]
Swapped positions 2 and 3: [8, 7, 6, 9, 5, 10]
Swapped positions 3 and 4: [8, 7, 6, 5, 9, 10]
```

```
Pass 3:
Swapped positions 0 and 1: [7, 8, 6, 5, 9, 10]
Swapped positions 1 and 2: [7, 6, 8, 5, 9, 10]
```

```
Pass 3:
Swapped positions 0 and 1: [7, 8, 6, 5, 9, 10]
Swapped positions 1 and 2: [7, 6, 8, 5, 9, 10]
Swapped positions 0 and 1: [7, 8, 6, 5, 9, 10]
Swapped positions 1 and 2: [7, 6, 8, 5, 9, 10]
Swapped positions 1 and 2: [7, 6, 8, 5, 9, 10]
Swapped positions 2 and 3: [7, 6, 5, 8, 9, 10]
```

```
Pass 4:
Swapped positions 0 and 1: [6, 7, 5, 8, 9, 10]
Swapped positions 0 and 1: [6, 7, 5, 8, 9, 10]
Swapped positions 1 and 2: [6, 5, 7, 8, 9, 10]
```

```
Pass 5:
Pass 5:
Swapped positions 0 and 1: [5, 6, 7, 8, 9, 10]
✅ Final Sorted List: [5, 6, 7, 8, 9, 10]
```

```
Now you can try your own list!
Now you can try your own list!
Now you can try your own list!
Now you can try your own list!
```

OBSERVATION:

- The program successfully translates the Java bubble sort into Python using AI.
- It includes **input validation** for:
 - Empty lists
 - Non-numeric data
- It supports **interactive input** and **step-by-step sorting visualization**.
- The code demonstrates:
 - **Error handling**
 - **AI-assisted debugging**
 - **Optimization** (early exit if sorted)
- This extended version showcases a **production-grade and testable implementation**, ideal for data science integration.

Lab Question 2: File Handling Translation

PROMPT:

Prompt:

Translate the following C++ file-handling program into JavaScript (Node.js).

The JavaScript version should read a text file, process its contents (for example, convert all text to uppercase), and write the output to a new file.

Also, add proper error handling for missing files or permission errors.

CODE:

```
JS 19.1.2.js > processFile
1  const fs = require('fs/promises');
2  const path = require('path');
3
4  /**
5   * Reads content from an input file, converts it to uppercase, and writes it to an output file.
6   * This is a JavaScript (Node.js) translation of a common C++ file I/O task.
7   *
8   * @param {string} inputFilePath - The path to the source file.
9   * @param {string} outputFilePath - The path to the destination file.
10  * @returns {Promise<void>} A promise that resolves when the operation is complete.
11  */
12  async function processFile(inputFilePath, outputFilePath) {
13    try {
14      // --- Task 1: Read the input file ---
15      // In Node.js, we use the 'fs/promises' module for modern async file operations.
16      console.log(Reading from ${inputFilePath}...);
17      const fileContent = await fs.readFile(inputFilePath, 'utf-8');
18
19      // --- Process the data (equivalent to the C++ std::transform) ---
20      const processedContent = fileContent.toUpperCase();
21
22      // --- Task 2: Write the processed data to the output file ---
23      console.log(Writing processed data to ${outputFilePath}...);
24      await fs.writeFile(outputFilePath, processedContent);
25
26      console.log('✅ File processing completed successfully.');
```

```

34     // 'ENOENT' means 'Error, No Entry' (i.e., file or directory not found).
35     console.error(` Error: The file at '${error.path}' was not found.`);
36 } else if (error.code === 'EACCES') {
37     // 'EACCES' means 'Error, Access Denied' (i.e., permission error).
38     console.error(` Error: Permission denied. Could not read or write to '${error.path}'.`);
39 } else {
40     // For any other unexpected errors.
41     console.error(' An unexpected error occurred:', error.message);
42 }
43 }
44 }
45
46 // --- Main Execution and Demonstration ---
47 // This self-invoking async function allows us to use 'await' at the top level.
48 (async () => {
49     const inputFile = path.join(__dirname, 'input.txt');
50     const outputFile = path.join(__dirname, 'output.txt');
51
52     // --- 1. Demonstrate the successful case ---
53     console.log('--- Running Success Scenario ---');
54     // First, create a dummy input file to ensure the script can run.
55     await fs.writeFile(inputFile, 'Hello World!\nThis is a test file for the Node.js script.');
```

```

56     await processFile(inputFile, outputFile);
57     console.log('-----\n');
```

```

58
59     // --- 2. Demonstrate the "File Not Found" error ---
60     console.log('--- Running "File Not Found" Scenario ---');
61     const nonExistentFile = path.join(__dirname, 'non_existent_file.txt');
```

```

62     await processFile(nonExistentFile, outputFile);
63     console.log('-----\n');
```

```

64
65
66
67     // --- 3. Demonstrate the "Permission Denied" error (conceptual) ---
68     // This is harder to reliably test, but the logic is in place.
69     // To test this manually, you could:
70     //   - On Linux/macOS: chmod 000 input.txt to remove read permissions.
71     //   - On Windows: Change the file's security properties to deny read access.
72     console.log('--- "Permission Denied" Scenario (Conceptual) ---');
```

```

73     console.log('The code includes a check for EACCES (permission) errors.');
```

```

74     console.log('To test, manually restrict read permissions on "input.txt" and re-run.');
```

```

75     console.log('-----\n');
```

```

76
77     // --- Cleanup ---
78     // Clean up the created files.
79     try {
80         await fs.unlink(inputFile);
81         await fs.unlink(outputFile);
82     } catch (err) {
83         // Ignore cleanup errors if files were not created due to an earlier failure.
84     }
85 })();
```

OUTPUT:

```
[Running] node "c:\Users\DELL\Desktop\vs code\file_processor.js"
--- Running Success Scenario ---
Reading from c:\Users\DELL\Desktop\vs code\input.txt...
Writing processed data to c:\Users\DELL\Desktop\vs code\output.txt...
✅ File processing completed successfully.
-----

--- Running "File Not Found" Scenario ---
Reading from c:\Users\DELL\Desktop\vs code\non_existent_file.txt...
❌ An error occurred during file processing:
-----

Error: The file at 'c:\Users\DELL\Desktop\vs code\non_existent_file.txt' was not found.
--- "Permission Denied" Scenario (Conceptual) ---
The code includes a check for EACCES (permission) errors.
To test, manually restrict read permissions on "input.txt" and re-run.
-----
```

OBSERVATION:

The Node.js version successfully replicates the C++ file-handling behavior while providing better error diagnostics through exceptions. It reads the input file, processes text to uppercase, and writes results to a new file. The `fs` module in Node.js simplifies file I/O and includes robust error-handling mechanisms.

Lab Question 3: API Call Translation

PROMPT:

Prompt:

Translate the following Python code that makes an API call to fetch weather data into Java. The Java version should perform the same functionality and include error handling for invalid API keys, timeouts, and no internet connection.

CODE:

OUTPUT:

OBSERVATION:

Both the Python and Java implementations successfully make API calls to fetch weather data. The Java version includes additional structured error handling for invalid API keys, request timeouts, and connectivity issues. Using a library like Gson can improve JSON parsing in production.