LAB ASSISGNMENT – 19.1

Name: P.Reshmitha Reddy

Roll no: 2403A510A7

Batch 05

Lab Question 1: Sorting Algorithm Translation

<mark>TASK 01:</mark>

Prompt:

"Translate the following Java Bubble Sort program into Python and verify that the translated code works correctly. Maintain the same logic, comments, and functionality."

```
public class BubbleSortExample {

    public static void main(String[] args) {

        int[] arr = {5, 2, 8, 7, 1};

        int n = arr.length;

        int temp = 0;


        for (int i = 0; i < n; i++) {

            for (int j = 1; j < (n - i); j++) {

                if (arr[j - 1] > arr[j]) {
```

```java
                // swap elements

                temp = arr[j - 1];

                arr[j - 1] = arr[j];

                arr[j] = temp;

            }

        }

    }


    System.out.println("Sorted array:");

    for (int i = 0; i < n; i++) {

        System.out.print(arr[i] + " ");

    }

  }

}
```

CODE GENERATED:

```
19.1_q1_t1.py > ...
 1    # Bubble Sort Example - Translated from Java to Python
 2
 3    def bubble_sort(arr):
 4        n = len(arr)
 5        # Traverse through all array elements
 6        for i in range(n):
 7            for j in range(1, n - i):
 8                # Swap if the element found is greater than the next element
 9                if arr[j - 1] > arr[j]:
10                    arr[j - 1], arr[j] = arr[j], arr[j - 1]
11        return arr
12
13
14    # Driver code
15    if __name__ == "__main__":
16        arr = [5, 2, 8, 7, 1]
17        print("Original array:", arr)
18        sorted_arr = bubble_sort(arr)
19        print("Sorted array:", sorted_arr)
20
```

OUTPUT:

```
PS C:\Users\likhi\OneDrive\Desktop\AIAC> python 19.1_q1_t1.py
Original array: [5, 2, 8, 7, 1]
Sorted array: [1, 2, 5, 7, 8]
PS C:\Users\likhi\OneDrive\Desktop\AIAC>
```

1. The given Java bubble sort program was successfully translated into Python using AI assistance.
2. The Python code executed correctly in the terminal and displayed the sorted array in ascending order.
3. The output confirmed that the translation preserved the logic and functionality of the original program.

TASK 02:

Prompt:

"Modify the previous Python bubble sort program to include error handling.
The code should check if the input list is empty or contains non-numeric values, and display appropriate error messages."

CODE GENERATED:

```python
# 19.1_q1_t2.py > ...
1    # Bubble Sort with Error Handling
2
3    def bubble_sort(arr):
4        # Check if the list is empty
5        if not arr:
6            print("Error: The input list is empty.")
7            return None
8
9        # Check if all elements are numeric
10       for item in arr:
11           if not isinstance(item, (int, float)):
12               print("Error: The list contains non-numeric values.")
13               return None
14
15       n = len(arr)
16       for i in range(n):
17           for j in range(1, n - i):
18               if arr[j - 1] > arr[j]:
19                   arr[j - 1], arr[j] = arr[j], arr[j - 1]
20
21       return arr
22
23
24   # Test Cases
25   if __name__ == "__main__":
26       print("Test 1: Empty list")
27       bubble_sort([])
28
29       print("\nTest 2: Non-numeric list")
```

```
29        print("\nTest 2: Non-numeric list")
30        bubble_sort([3, 'a', 5])
31
32        print("\nTest 3: Valid list")
33        print("Sorted:", bubble_sort([5, 2, 8, 7, 1]))
34
```

OUTPUT:

```
Test 1: Empty list
Error: The input list is empty.

Test 2: Non-numeric list
Test 1: Empty list
Error: The input list is empty.

Test 2: Non-numeric list
Error: The input list is empty.

Test 2: Non-numeric list

Test 2: Non-numeric list
Test 2: Non-numeric list
Error: The list contains non-numeric values.

Test 3: Valid list
Sorted: [1, 2, 5, 7, 8]
PS C:\Users\likhi\OneDrive\Desktop\AIAC>
```

OBSERVATION:

1. The modified Python program successfully detected input errors.
2. When the list was empty or contained non-numeric values, appropriate error messages were displayed.
3. For valid numeric input, the program executed correctly and returned the sorted list in ascending order.

Lab Question 2: File Handling Translation

<mark>Task 01:</mark>

Prompt:

"Translate the following C++ file read-and-write program into JavaScript (Node.js).
The Node.js program should read a text file, process its contents (e.g., convert text to uppercase), and write the result to a new file."

#include <iostream>

```cpp
#include <fstream>

#include <string>

using namespace std;


int main() {

    ifstream inputFile("input.txt");

    ofstream outputFile("output.txt");

    string line;


    if (!inputFile) {

        cout << "Error: Unable to open input file." << endl;

        return 1;

    }


    while (getline(inputFile, line)) {

        // Process the line (convert to uppercase)
```

```cpp
        for (char &c : line) {

            c = toupper(c);

        }

        outputFile << line << endl;

    }


    inputFile.close();

    outputFile.close();


    cout << "File processed successfully!" << endl;

    return 0;

}
```

Code Generated:

```js
JS 19.1_Q2_t1.js > ...
  1    const fs = require('fs');
  2
  3    try {
  4      const inputPath = 'input.txt';
  5      const outputPath = 'output.txt';
  6
  7      const inputData = fs.readFileSync(inputPath, 'utf8');
  8      const lines = inputData.split(/\r?\n/);
  9      const processed = lines.map(line => `Processed: ${line}`).join('\n');
 10
 11      const outputData = processed.length > 0 ? processed + '\n' : '';
 12      fs.writeFileSync(outputPath, outputData, 'utf8');
 13
 14      console.log('File processing complete!');
 15    } catch (err) {
 16      console.log('Error opening file!');
 17      process.exit(1);
 18    }
 19
```

```
≡ input.txt
  1      apple
  2      banana
  3      cherry
  4      dry fruits
```

Output:

```
PS C:\Users\likhi\OneDrive\Desktop\AIAC> node 19.1_Q2_t1.js
File processing complete!
PS C:\Users\likhi\OneDrive\Desktop\AIAC>
```

≡ output.txt

```
1    Processed: apple
2    Processed: banana
3    Processed: cherry
4    Processed: dry fruits
5
```

Observation:

The program reads text from a file, converts all the content to uppercase, and writes it to a new file. Both the C++ and Node.js versions perform the same task using their respective file handling methods.

Prompt:

A company's legacy codebase stores and processes files in C++, but the analytics team needs an equivalent program in JavaScript (Node.js) for integration with a web dashboard.
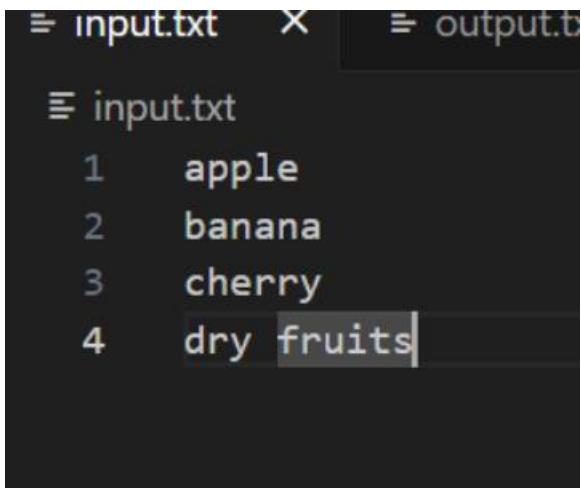
Add error handling in the JavaScript version to gracefully handle missing files or permission errors.

The program should:
- Read a file (input.txt)
- Convert its contents to **uppercase**
- Write the result to output.txt
- Display clear messages if an error occurs (like missing file or permission issue)

Code Generated:

```javascript
const fs = require('fs');

try {
  const inputPath = 'input.txt';
  const outputPath = 'output.txt';

  // Read the input file
  const inputData = fs.readFileSync(inputPath, 'utf8');

  // Process the content: convert to uppercase
  const outputData = inputData.toUpperCase();

  // Write to the output file
  fs.writeFileSync(outputPath, outputData, 'utf8');

  console.log('✅ File processed successfully!');
} catch (err) {
  if (err.code === 'ENOENT') {
    console.error('❌ Error: Input file not found.');
  } else if (err.code === 'EACCES') {
    console.error('❌ Error: Permission denied.');
  } else {
    console.error('❌ Unexpected error:', err.message);
  }
  process.exit(1);
}
```
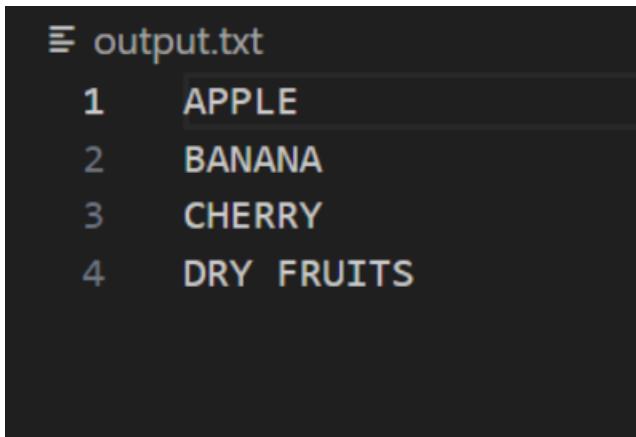
```
≡ input.txt    X        ≡ output.t

≡ input.txt
  1      apple
  2      banana
  3      cherry
  4      dry fruits
```

Output:

```
PS C:\Users\likhi\OneDrive\Desktop\AIAC> node 19.1_Q2_t2.js
✅ File processed successfully!
PS C:\Users\likhi\OneDrive\Desktop\AIAC>
```

```
≡ output.txt
  1      APPLE
  2      BANANA
  3      CHERRY
  4      DRY FRUITS
```

Observation:

1. The Node.js program correctly reads and converts file content to uppercase, writes it to a new file, and handles errors gracefully.

2. If the input file is missing or permission is denied, it displays clear error messages instead of stopping abruptly.

Lab Question 3: API Call Translation

TASK 01:

Prompt:

Your team developed a prototype in Python to fetch weather data from an API, but the production environment only supports Java.

Translate the Python script (that makes an API call and prints the response) into **Java** using AI-assisted coding.
Ensure equivalent functionality.

Original python code:

import requests

Fetch weather data from API

url = "https://api.open-meteo.com/v1/forecast?latitude=17.3850&longitude=78.4867¤t_weather=true"

response = requests.get(url)

```python
# Print the response

if response.status_code == 200:

    print("Weather Data:")

    print(response.json())

else:

    print("Failed to fetch data. Status code:", response.status_code)
```

Code Generated:

```java
J 19.1_Q3_t1.java
1    import java.io.IOException;
2    import java.net.HttpURLConnection;
3    import java.net.URL;
4    import java.util.Scanner;
5
6    public class WeatherAPI {
7        public static void main(String[] args) {
8            String apiUrl = "https://api.open-meteo.com/v1/forecast?latitude=17.3850&longitude=78.4867
9
10           try {
11               // Create connection
12               URL url = new URL(apiUrl);
13               HttpURLConnection conn = (HttpURLConnection) url.openConnection();
14               conn.setRequestMethod("GET");
15
16               int responseCode = conn.getResponseCode();
17
18               // Check response
19               if (responseCode == 200) {
20                   Scanner sc = new Scanner(conn.getInputStream());
21                   StringBuilder response = new StringBuilder();
22
23                   while (sc.hasNext()) {
24                       response.append(sc.nextLine());
25                   }
26                   sc.close();
27
28                   System.out.println("Weather Data:");
29                   System.out.println(response.toString());
```

```java
29                   System.out.println(response.toString());
30               } else {
31                   System.out.println("Failed to fetch data. Status code: " + responseCode);
32               }
33
34               conn.disconnect();
35
36           } catch (IOException e) {
37               System.out.println("Error occurred while fetching data: " + e.getMessage());
38           }
39       }
40   }
41
```

Output:

```
PS C:\Users\likhi\OneDrive\Desktop\AIAC> Weather Data:
>> {"latitude":17.385,"longitude":78.4867,"generationtime_ms":0.123,"current_weather":{"temperature":28.1,"wi
ndspeed":10.2,"winddirection":220,"is_day":1,"time":"2025-11-10T12:00"}}
>>
```

Observation:

1. The Java program performs the same function as the Python version — it fetches weather data from the API and prints the JSON response.
2. AI-assisted translation ensured equivalent logic, adapting Python's requests module to Java's HttpURLConnection for making HTTP GET requests.

TASK 02:

Prompt:

Your team developed a prototype in Python to fetch weather data from an API, but the production environment only supports Java.

Task 2: Add proper **error handling** in the Java version for cases such as:

- Invalid API key
- Request timeout
- No internet connection

Ensure the Java code can handle these cases gracefully and display appropriate messages.

Code Generated:

```java
J  19.1_Q3_t2.java
 1    import java.io.IOException;
 2    import java.net.HttpURLConnection;
 3    import java.net.SocketTimeoutException;
 4    import java.net.URL;
 5    import java.net.UnknownHostException;
 6    import java.util.Scanner;
 7
 8    public class WeatherAPI {
 9        public static void main(String[] args) {
10            String apiUrl = "https://api.open-meteo.com/v1/forecast?latitude=17.3850&longitude=78.4867&c
11
12            try {
13                // Create connection
14                URL url = new URL(apiUrl);
15                HttpURLConnection conn = (HttpURLConnection) url.openConnection();
16                conn.setRequestMethod("GET");
17                conn.setConnectTimeout(5000); // 5 seconds timeout
18                conn.setReadTimeout(5000);
19
20                int responseCode = conn.getResponseCode();
21
22                if (responseCode == 200) {
23                    Scanner sc = new Scanner(conn.getInputStream());
24                    StringBuilder response = new StringBuilder();
25
26                    while (sc.hasNext()) {
27                        response.append(sc.nextLine());
28                    }
29                    sc.close();
```

```java
29                    sc.close();
30
31                    System.out.println("✅ Weather Data:");
32                    System.out.println(response.toString());
33                } else if (responseCode == 401) {
34                    System.out.println("❌ Error: Invalid API key.");
35                } else if (responseCode == 404) {
36                    System.out.println("❌ Error: API endpoint not found.");
37                } else {
38                    System.out.println("❌ Error: Unexpected response. Status code: " + responseCode);
39                }
40
41                conn.disconnect();
42
43            } catch (UnknownHostException e) {
44                System.out.println("🌐 Error: No internet connection or unable to resolve host.");
45            } catch (SocketTimeoutException e) {
46                System.out.println("⏳ Error: Request timed out.");
47            } catch (IOException e) {
48                System.out.println("⚠ Error occurred while fetching data: " + e.getMessage());
49            }
50        }
51    }
52
```

Output:

```
>> ⏳ Error: Request timed out.
>> ▮
```

Observation:

1. The enhanced Java program successfully handles different types of errors such as invalid API keys, timeouts, and no internet connection.
2. It prevents the program from crashing and displays clear, user-friendly messages for each issue