Data structure and algorithms.

Luong Hoang Hai

BH01797

1. Implement complex data structures and algorithms

- **1. Define the Problem**
- he task is to develop a Student Management System that allows users to:
- • Add new student records.
- • Update existing student details.
- • Delete student records.
- • Retrieve and display specific student details.
- • List all student records.

- 2. Abstract Data Type (ADT)
- 2.1 Define the ADT The ADT will include a Student class with the following attributes and operations:
- Attributes:

- • int id
- • String name
- • int age
- • String course
- • double marks
- Operations: 1. Add a new student. 2. Update student details. 3. Delete a student by ID. 4. Retrieve student details by ID.

Operations:

1. Add a new student.
2. Update student details.
3. Delete a student by ID.
4. Retrieve student details by ID.
5. Display all students.

3. Update Student Details

- Input: Student ID and new details.
- Search for the student using ID.
- If found, update details.

```
private static void updateStudent() {  1 usage
    System.out.print("Enter student name to update: ");
    String name = scanner.nextLine();

    boolean found = false;
    StudentStack tempStack = new StudentStack();
    while (!studentStack.isEmpty()) {
        Student student = studentStack.pop();
        if (student.name.equalsIgnoreCase(name)) {
            System.out.print("Enter new name: ");
            student.name = scanner.nextLine();
            System.out.print("Enter new score: ");
            if (!scanner.hasNextDouble()) {
                System.out.println("Invalid score. Please enter a numeric value.");
                scanner.nextLine(); // Clear invalid input
                return;
            }
            student.score = scanner.nextDouble();
            scanner.nextLine(); // Consume newline
            found = true;
            tempStack.push(student);
            break;
        } else {
            tempStack.push(student);
        }
    }
```

4. Implement Error Handling and Report Test Results

- **4.1. Identify Potential Errors**
- 1. Duplicate student IDs when adding.
- 2. Attempt to update/delete a non-existent ID.
- 3. Invalid input types for attributes (e.g., age as a string).
- **4.2. Define Error Handling Mechanisms**
- • Use try-catch blocks to handle exceptions.
- • Validate inputs before performing operations.

5. Sorting Algorithms

- **Bubble Sort:**
- Used for simplicity and demonstration.
- Sorts students by name or score in ascending order.
- Suitable for smaller datasets.
- **Merge Sort:**
- Demonstrates a more efficient sorting approach.
- Uses a divide-and-conquer strategy for better performance on large datasets.

6. Operations with Stack ADT

- **1. Add Student (Push):**
- o Time Complexity: O(1)O(1)O(1) (constant time operation).
- o Space Complexity: O(1)O(1)O(1) (no additional space required).
- o Critical Analysis:
- ▪ Strengths: Fast and efficient for adding students to the stack.
- ▪ Weaknesses: None, as the operation is atomic.
- **2. Delete or Update Student:**
- o Time Complexity:
- ▪ O(n)O(n)O(n) (linear traversal of the stack to find the student).
- o Space Complexity:
- ▪ O(n)O(n)O(n) (temporary stack used to maintain order during updates or deletions).
- o Critical Analysis:
- ▪ Strengths: Straightforward approach for maintaining stack consistency.
- ▪ Weaknesses: Traversal overhead increases linearly with the size of the stack.

7. Practical Example: Comparing Algorithms with Different Asymptotic Complexities

- **Example:**
- • For n = 1000:
- o Bubble Sort ($O(n^2)$): 1,000,000 comparisons
- o Merge Sort ($O(n \log n)$): 10,000 comparisons ($\log_2 1000 \approx 10$)
- **• For n = 10,000:**
- o Bubble Sort ($O(n^2)$): 100,000,000 comparisons
- o Merge Sort ($O(n \log n)$): 130,000 comparisons ($\log_2 10000 \approx 14$)