APL Experiment 1 (B) :

Class, Object and Constructor

1B]

Working with arrays and flow control statements

Problem Statement:

Design a Java program that allows the user to perform various operations on an array of integers using different sorting and searching algorithms. The program should implement linear search, binary search, bubble sort, selection sort, insertion sort, merge sort, and quick sort. Additionally, it should utilize flow control statements and a switch-case structure to offer the user different functionalities for sorting and searching operations. The program should also take input from the user to populate the array and choose the desired algorithm to apply.

Requirements:

1. **User Input:**
    - The program should prompt the user to enter the size of the array.
    - The user should then input individual elements to populate the array.
    - The user will be presented with a menu of operations to choose from (sorting or searching algorithms).
2. **Menu Options (using Switch-Case):**
    - **Option 1: Linear Search**
      Perform a linear search to find the index of a given element in the array.
    - **Option 2: Binary Search**
      Perform a binary search to find the index of a given element (ensure the array is sorted first).
    - **Option 3: Bubble Sort**
      Sort the array using the Bubble Sort algorithm.
    - **Option 4: Selection Sort**
      Sort the array using the Selection Sort algorithm.
    - **Option 5: Insertion Sort**
      Sort the array using the Insertion Sort algorithm.

- **Option 6: Merge Sort**
  Sort the array using the Merge Sort algorithm.
- **Option 7: Quick Sort**
  Sort the array using the Quick Sort algorithm.

3. **Sorting Algorithms:** Implement the following sorting algorithms:
   - **Bubble Sort:** Repeatedly step through the list, compare adjacent elements, and swap them if they are in the wrong order.
   - **Selection Sort:** Repeatedly select the minimum element from the unsorted portion of the list and place it in the correct position.
   - **Insertion Sort:** Build the sorted list one element at a time by inserting the current element into its correct position.
   - **Merge Sort:** Divide the array into two halves, sort them recursively, and then merge the sorted halves.
   - **Quick Sort:** Choose a pivot element, partition the array around the pivot, and then sort the sub-arrays.
4. **Searching Algorithms:** Implement the following search algorithms:
   - **Linear Search:** Iterate through each element of the array and return the index of the element if found.
   - **Binary Search:** Assuming the array is sorted, repeatedly divide the search interval in half to find the target element.
5. **Flow Control and Validation:**
   - Use flow control statements (if, for, while, etc.) to implement the various sorting and searching algorithms.
   - Use a switch-case structure to allow the user to choose between the different operations.
   - The program should handle invalid input gracefully, ensuring the user enters valid data for array size, element input, and algorithm choice.
6. **Sample Output:**

   Welcome to the Array Operations Program!

   Please enter the size of the array: 5

   Enter 5 integers for the array:

   Element 1: 34

   Element 2: 12

Element 3: 77

Element 4: 23

Element 5: 5

Choose an operation:

1. Linear Search
2. Binary Search
3. Bubble Sort
4. Selection Sort
5. Insertion Sort
6. Merge Sort
7. Quick Sort

Enter your choice (1-7): 3

Array before sorting: [34, 12, 77, 23, 5]

Array after Bubble Sort: [5, 12, 23, 34, 77]

Do you want to perform another operation? (y/n): y

Choose an operation:

1. Linear Search
2. Binary Search
3. Bubble Sort
4. Selection Sort
5. Insertion Sort
6. Merge Sort
7. Quick Sort

Enter your choice (1-7): 1

Enter the element to search for: 23

Linear Search result: Element 23 found at index 3

Do you want to perform another operation? (y/n): n

Algorithm Details:

- **Linear Search:** Iterate through the array and check each element for a match.
- **Binary Search:** Assume the array is sorted. Repeatedly divide the array into two halves and check the middle element.
- **Bubble Sort:** Continuously swap adjacent elements if they are in the wrong order.
- **Selection Sort:** Find the smallest element in the unsorted part of the array and swap it with the first unsorted element.
- **Insertion Sort:** Take one element at a time and insert it into the correct position in the sorted portion of the array.
- **Merge Sort:** Divide the array into two halves, recursively sort each half, and merge the sorted halves back together.
- **Quick Sort:** Pick a pivot element, partition the array, and recursively sort the subarrays.

Guidelines:

- Use appropriate methods for each algorithm and ensure the program uses modular design (separate methods for each algorithm).
- Handle invalid menu choices and array elements.
- Display the array before and after sorting operations.
- Provide meaningful messages and results for search operations.
