**Final Assignment Report**

Programming Language 3

Searching and Sorting Program Project

Mahde Al Jamil

Student Number: 2203822

Student ID: 1.2.246.562.24.53532288716

**Worth points: 5**

# Overview:

The project consists of a menu-driven Java application that offers functionalities for searching and various sorting algorithms. It includes linear and binary searching techniques, along with sorting algorithms like Insertion Sort, Bubble Sort, Quick Sort, and Merge Sort.

# Design Solution:

There is two code file in this project. The FinalProject is the main file, which handles user interactions, allowing users to choose between search and sorting functionalities. It integrates with the SortingPerformance file when the user selects functionality 7 from the menu. The SortingPerformance file measures and compares the efficiency of sorting algorithms across various array sizes, considering the time taken and the number of comparisons made during sorting operations. This comprehensive analysis enhances the understanding of sorting algorithm performance based on various array input sizes.

## Code Structure:

1. Main Method (public static void main(String[] args))
   * Manages user interaction through a menu system.
   * Taking input from user to choose between different searching and sorting operations.
   * Invokes the processChoice method to take input and then execute the selected operation.
2. Processing User Choices (private static void processChoice(int option, Scanner scan))
   * Executes the chosen operation based on the user's selection.
   * This method calls specific methods for searching (linear and binary search) and sorting (various algorithms) based on the user's input.
3. Searching Methods (Linear Search, Binary Search)
   * Generates arrays of random integers for searching purposes.
   * Implements linear and binary search algorithms to find a specific target value within a randomly generated array.
4. Sorting Methods (Insertion Sort, Bubble Sort, Quick Sort, Merge Sort)
   * This sections implements various sorting techniques to sort the randomly generated arrays in ascending order.
   * Uses different algorithms like Insertion Sort, Bubble Sort, Quick Sort, and Merge Sort for sorting.
5. Integration with SortingPerformance Class (in separate java file):

When the user selects functionality 7, the SortingPerformance class is invoked to analyze sorting algorithm performance:

Code structure of SortingPerformance class:

1. SortingPerformance Class:
   1. Contains methods dedicated for sorting algorithm performance analysis.
   2. Measures and compares the efficiency of various sorting algorithms for different array sizes.
2. Variable for Comparison Count:
   1. public static int comCount: Tracks the number of comparisons made during sorting operations.
3. analyzeSortPerformance() Method:
   1. Initiates the sorting analysis for different array sizes.
   2. Measures the time taken by sorting algorithms and the number of comparisons made.
4. Functional Interface – “MyFunction”
   1. Defines a functional interface to reference sorting methods with varying signatures in the sorting analysis method.
5. analyzeAlgorithm() Method:
   1. Evaluates the performance of a specific sorting algorithm for different array sizes.
   2. Measures sorting time and counts comparisons for each algorithm and array size.
6. Sorting Algorithm Methods
   1. Includes sorting algorithms (Insertion Sort, Bubble Sort, Quick Sort, Merge Sort) with logic for sorting and comparison count.
7. Auxiliary Methods:
   * generateRandomArray: Creates arrays of random integers within a specified range(-100 to 100).
   * printArray: Displays the elements present in an array.

# Functionalities and testing of the code:

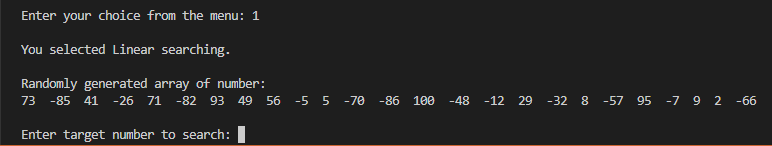
Here is a list of all the task can be executed in this project:

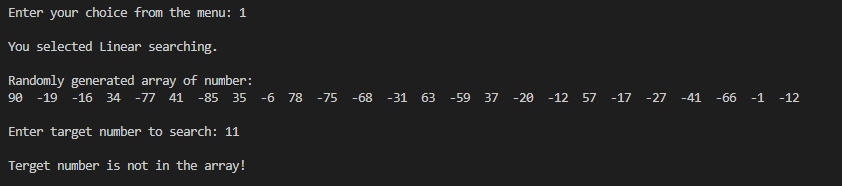
1. Main menu:
   1. First the program displays a short menu with different choices.
   2. And then ask your to enter a choice.

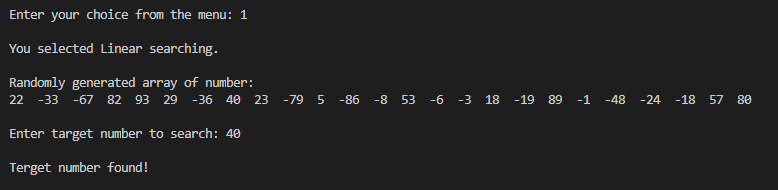
A screenshot of a computer program

Description automatically generated

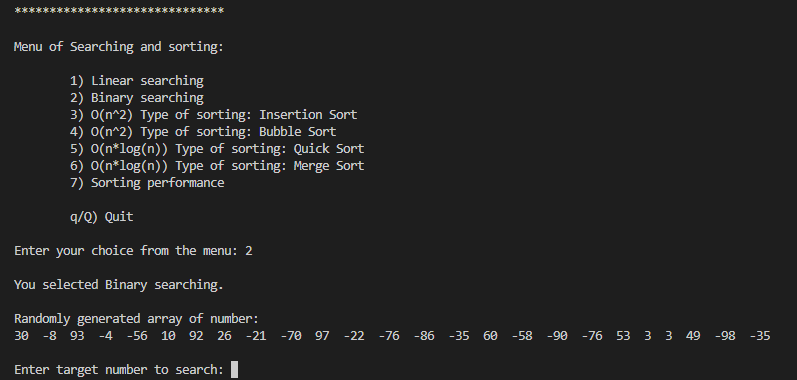
1. Linear Search:
   1. Searches for a specific number within a randomly generated array by iterating through each element sequentially.
   2. Determines if the target value exists in the array and reports its presence or absence.

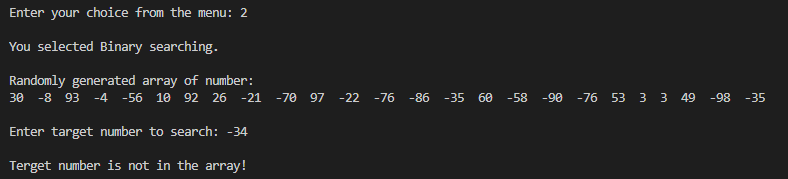


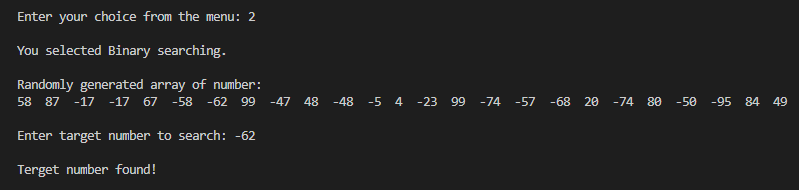




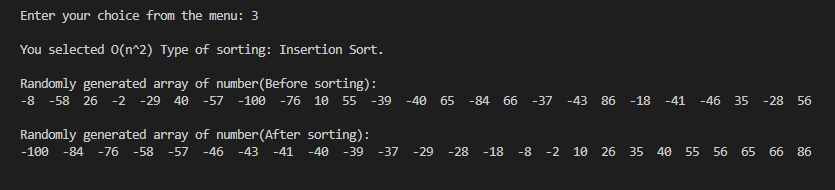
1. Binary Search:
   1. Searches for a specific number within a sorted randomly generated array using a divide-and-conquer strategy.
   2. Divides the array in half and eliminates portions that cannot contain the target value until the value is found or determined absent.



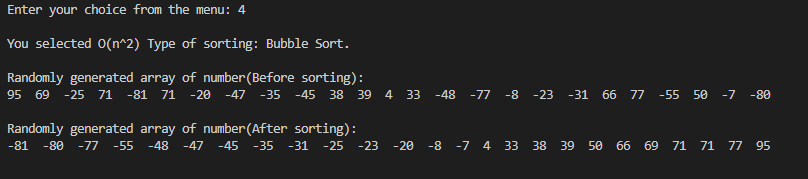




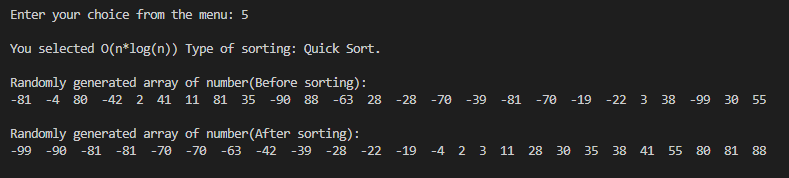
1. Insertion Sort:
   1. Sorts a randomly generated array by iteratively inserting elements into their correct positions within the sorted section of the array.
   2. Progressively builds a sorted sequence until the entire array is sorted.



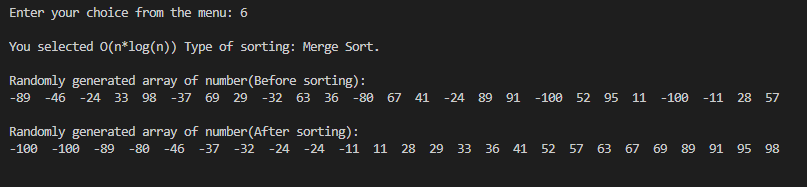
1. Bubble Sort:
   1. Sorts a randomly generated array by repeatedly swapping adjacent elements that are in the wrong order until the entire array is sorted.
   2. Iterates through the array multiple times, pushing the largest elements to their correct positions.



1. Quick Sort:
   1. Sorts a randomly generated array by selecting a pivot element, partitioning the array into sections relative to the pivot, and recursively sorting those sections.
   2. Continues partitioning and sorting until the entire array is sorted.



1. Merge Sort:
   1. Sorts a randomly generated array by recursively dividing the array into smaller arrays until they consist of single elements.
   2. Merges these smaller arrays in a sorted manner to form larger sorted arrays until the entire array is sorted.

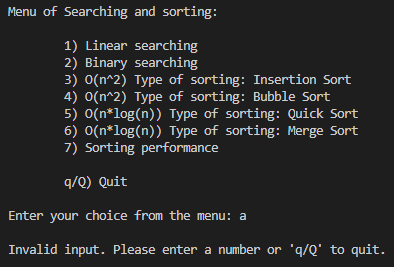
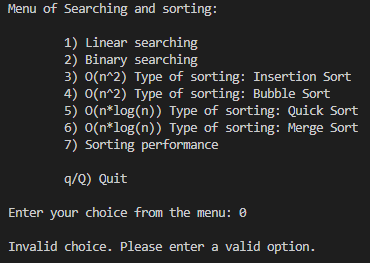


1. Sorting Performance Analysis:
   1. Offers the functionality to analyze and compare the efficiency of different sorting algorithms based on their time complexities.
   2. Measures and evaluates the performance of each sorting algorithm when dealing with varying sizes of arrays to determine their relative speeds and efficiencies.

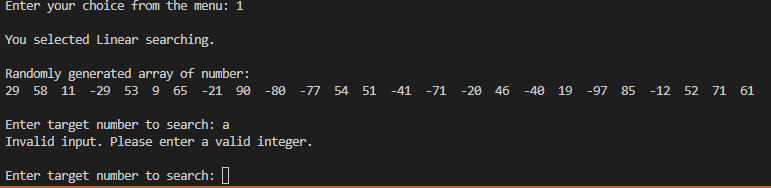
A black background with many small white and blue lines

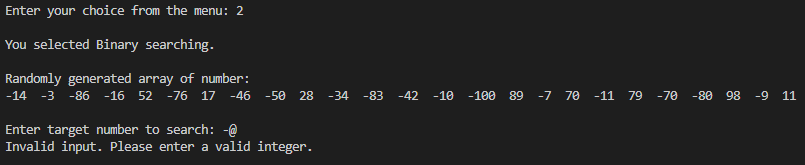
Description automatically generated with medium confidence

1. Error Handling:
   1. The code handles error really efficiently. It always checks that the user had entered the current type of value to work with to chose the correct functionalities.

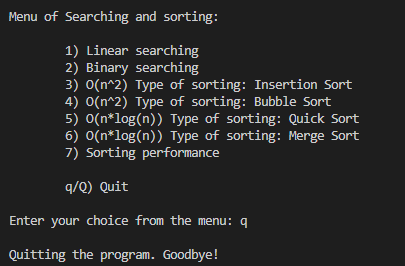
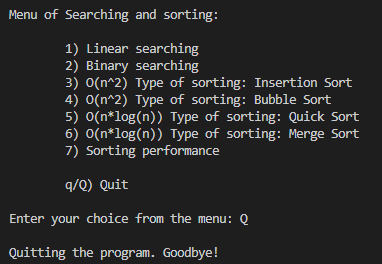
 

* 1. And also checks the input for searching, if there is any wrong type of input, it asks user to input again without crashing the program or navigating to somewhere else.



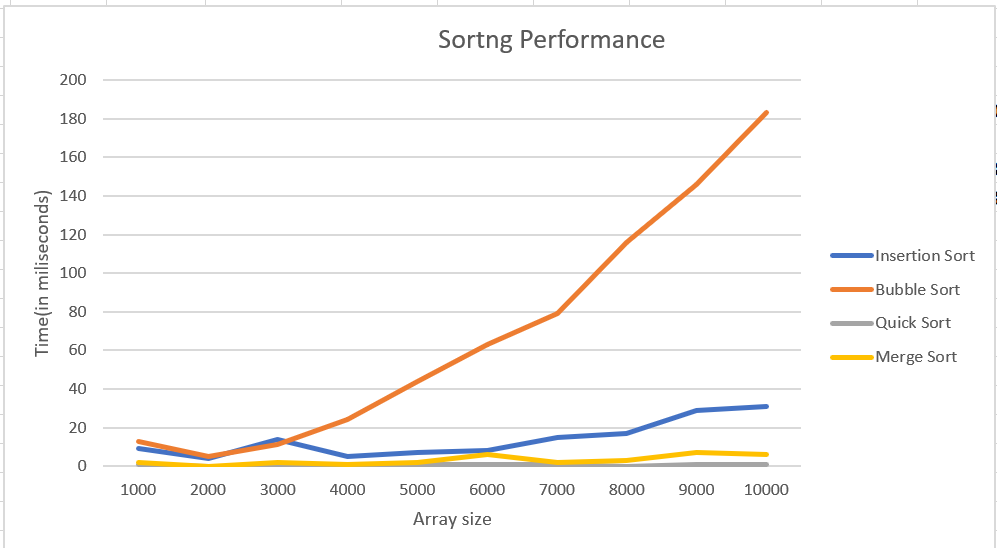


1. Quitting the program:
   1. The program exits or stops with a greeting when the user enters “q” or “Q”. It does not depend on upper or lowercase input.

# Performance Diagram:

Different algorithms performance diagram based on the 7th item execution of the program:



# Work Hour Bookkeeping:

|  |  |  |
| --- | --- | --- |
| Date | Hours | Task Description |
| 3/11/2023 | 2 | Worked on the main menu |
| 5/11/2023 | 2 | Implemented [linear Search] functionality |
| 6/11/2023 | 3.5 | Implemented [Insertion Sort] functionality |
| 9/11/2023 | 4 | Implemented [Merge Sort] functionality |
| 10/11/2023 | 2.5 | Implemented [Binary Search] functionality |
| 13/11/2023 | 5 | Worked on Additional methods and tidying the code. |
| 14/11/2023 | 4 | Writing proper comment in the code. |
| 22/11/2023 | 2 | Worked on Sorting performance functionality |
| 23/11/2023 | 4 | Worked on Sorting performance functionality |
| 24/11/2023 | 3 | Worked on Sorting performance functionality |
| 25/11/2023 | 2 | Worked on Sorting performance functionality |
| 28/11/2023 | 1 | Analyzing the code |
| 29/11/2023 | 4 | Added Bubble Sort functionality |
| 30/11/2023 | 5 | Added Merge Sort functionality |
| 02/12/2023 | 2 | Worked on Sorting performance functionality |
| 02312/2023 | 2 | Worked on Sorting performance functionality |
| 05/12/2023 | 3 | Worked on error handling functionality |
| 11/12/2023 | 7 | Worked on error project report |

# Testing Instructions for the Program

**Prerequisites:**

Ensure that Java Development Kit (JDK) is installed on the testing system.

**Steps to Test:**

1. **Compile the Code:**
   * Download the provided Java files onto the testing system.
   * Open a terminal or command prompt and navigate to the directory containing the Java files.
   * Compile the booth file.
2. **Run the Program:**
   * Execute the compiled Java program.
   * The program will display a menu of options for searching and sorting functionalities.
3. **Testing Functionalities:**
   * **Searching Functionalities:**
     + Select options 1 (Linear Search) and 2 (Binary Search).
     + Enter target numbers and observe the program's response for both search methods.
   * **Sorting Functionalities:**
     + Choose any options from 3 to 6 (Insertion Sort, Bubble Sort, Quick Sort, Merge Sort).
     + Observe the before and after arrays to ensure they are correctly sorted for each method.
   * **Sorting Performance Analysis:**
     + Choose option 7 to analyze sorting performance across different array sizes.
     + Verify that the program displays time taken (in milliseconds) and comparisons for each sorting algorithm across various array sizes.
4. **Error Handling:**
   * Input invalid characters or non-numeric values when prompted for numbers.
   * Check if the program handles these inputs gracefully, displaying appropriate error messages.
5. **Completion Verification:**
   * Confirm that the program terminates gracefully when selecting the quit option ('q' or 'Q') from the menu.
   * Ensure no unexpected errors or crashes occur during the testing process.