Day 4: Manisha Assignment

Task 1: Array Sorting and Searching

a) Implement a function called BruteForceSort that sorts an array using the brute force approach. Use this function to sort an array created with InitializeArray.

1. InitializeArray Function:

- This function initializes and returns an array of integers.

- Here, it returns an array with hardcoded values: `{5, 3, 8, 6, 2, 7, 4, 1}`.

2. BruteForceSort Function:

- This function implements the Bubble Sort algorithm.

- It repeatedly steps through the list, compares adjacent elements, and swaps them if they are in the wrong order.

- The process is repeated until the array is sorted.

3. Main Method:

- Initializes the array using `InitializeArray`.

- Prints the array before sorting.

- Sorts the array using `BruteForceSort`.

- Prints the array after sorting.

Running the Program

1. Copy the code into a file named `BruteForceSortExample.java`.

2. Compile the program using the command `javac BruteForceSortExample.java`.

3. Run the compiled program using the command `java BruteForceSortExample`.

The output should display the array before and after sorting:

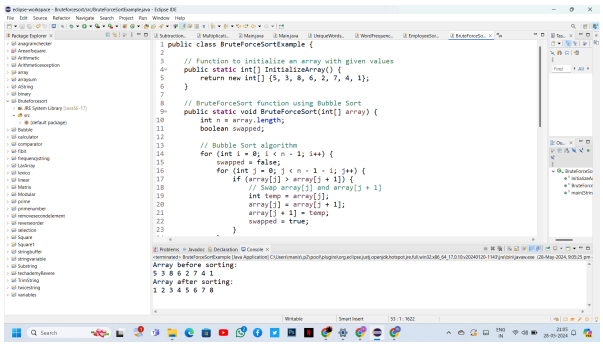
Array before sorting:

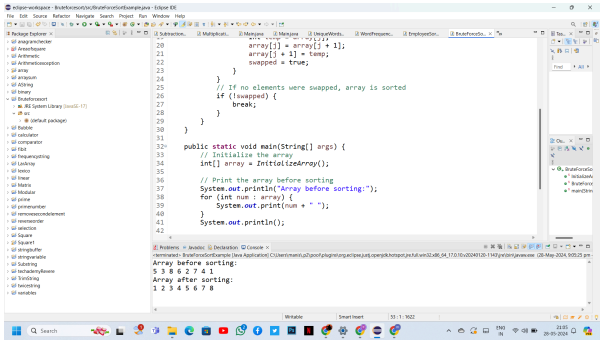
5 3 8 6 2 7 4 1

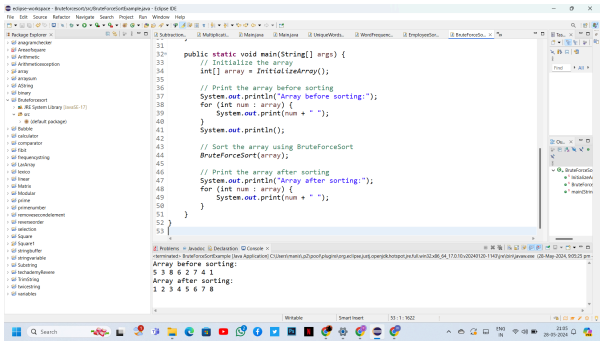
Array after sorting:

1 2 3 4 5 6 7 8

Here Output as follows:







b) Write a function named PerformLinearSearch that searches for a specific element in an array and returns the index of the element if found or -1 if not found.

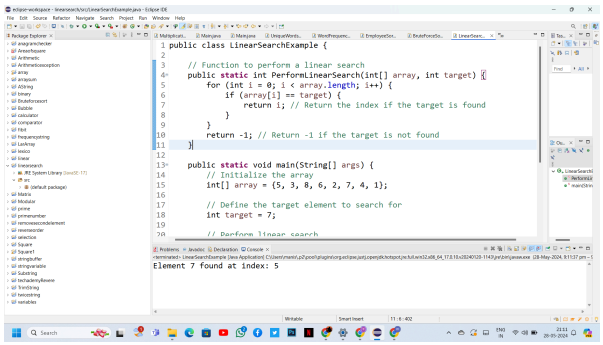
1. PerformLinearSearch Function:
   - This function takes an array of integers and a target integer to search for as parameters.
   - It iterates through the array and checks if each element is equal to the target.
   - If it finds the target, it returns the index of the target element.
   - If the loop completes without finding the target, it returns -1.

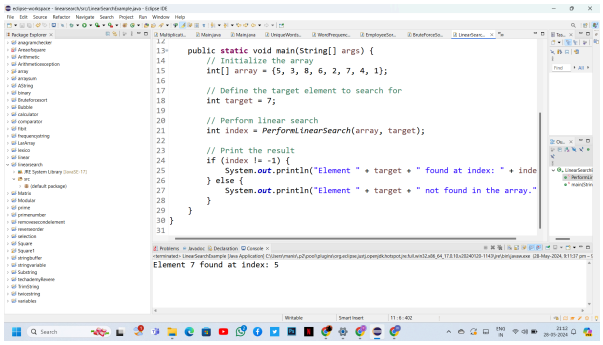
2. Main Method:
   - Initializes an array with some sample values.
   - Defines a target element to search for (in this case, `7`).
   - Calls the `PerformLinearSearch` function to search for the target element in the array.
   - Prints the result, indicating whether the element was found and at which index, or if it was not found in the array.

Running the Program
1. Copy the code into a file named `LinearSearchExample.java`.
2. Compile the program using the command `javac LinearSearchExample.java`.
3. Run the compiled program using the command `java LinearSearchExample`.

The output should display the result of the search:

```
**Element 7 found at index: 5**
```





Task 2: Two-Sum Problem

a) Given an array of integers, write a program that finds if there are two numbers that add up to a specific target. You may assume that each input would have exactly one solution, and you may not use the same element twice. Optimize the solution for time complexity.

1. HashMap:
   - `Map<Integer, Integer> map = new HashMap<>();`
   - This HashMap stores each number from the array as the key and its index as the value.

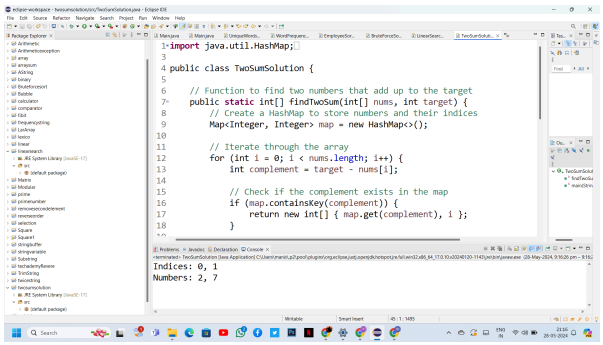
2. Iterate Through the Array:
   - `for (int i = 0; i < nums.length; i++) { ... }`
   - For each number in the array, calculate its complement by subtracting it from the target (`int complement = target - nums[i];`).

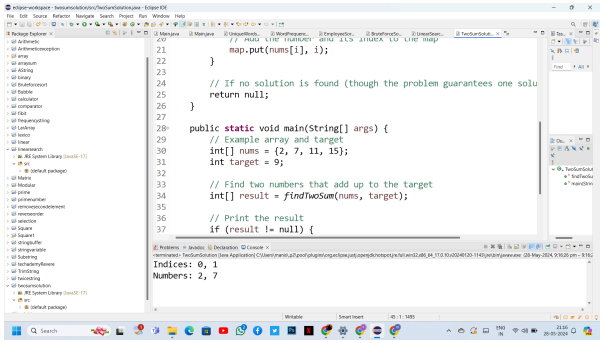
3. Check the Complement:
   - `if (map.containsKey(complement)) { ... }`
   - If the complement is already in the HashMap, we have found the two numbers that add up to the target. Return their indices.

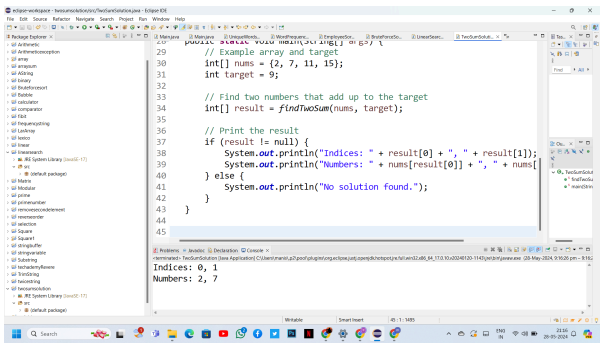
4. Add Number to HashMap:
   - `map.put(nums[i], i);`
   - If the complement is not in the HashMap, add the current number and its index to the HashMap.

5. Return Result:
   - If the solution is found, the indices of the two numbers are returned. If not, the function returns `null` (though the problem guarantees one solution, so this part is just a safeguard).

Here output as follows:







Task 3: Understanding Functions through Arrays

a) Write a recursive function named SumArray that calculates and returns the sum of elements in an array, demonstrate with example.

1. Function Definition:
- `SumArray` is a recursive function designed to calculate the sum of elements in an array.
- It takes two parameters: the array and an index to keep track of the current position in the array.

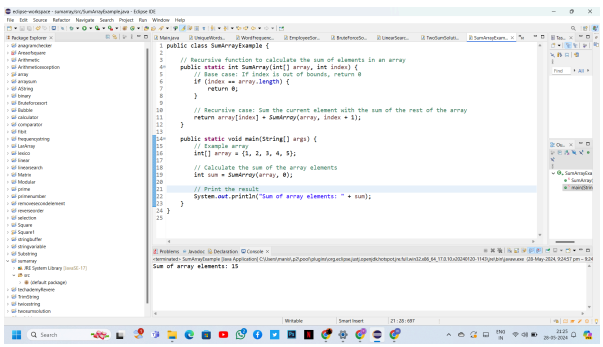
2. Base Case:
- The base case checks if the index has reached the end of the array.
- If the index equals the length of the array, it means all elements have been processed, so the function returns 0.

3. Recursive Case:
- The function adds the current element (at the given index) to the result of `SumArray` called on the next index.
- This way, the function sums the current element and recursively processes the rest of the array.

4. Main Method:
- An example array is initialized with some values.
- `SumArray` is called starting from index 0 to calculate the sum of the array.
- The sum is then printed.

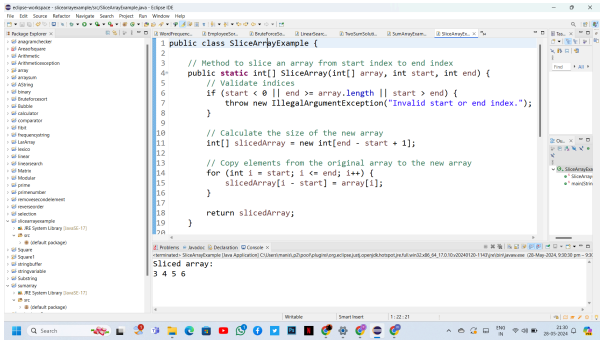
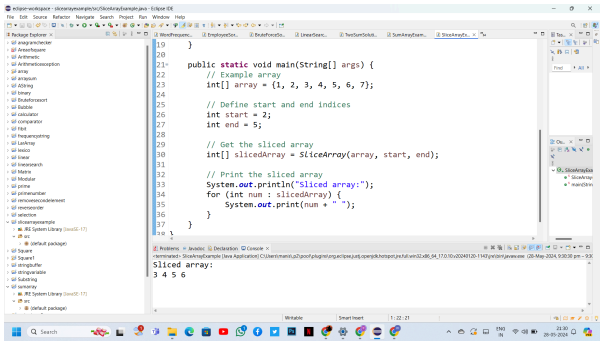
Example:
- For an array `{1, 2, 3, 4, 5}`, the `SumArray` function will:
  1. Start at index 0, adding the first element (1) to the sum of the rest of the array.
  2. Move to index 1, adding the second element (2) to the sum of the rest of the array.
  3. Continue this process until it reaches the end of the array.
  4. Finally, it returns the total sum, which is 15 for this example.

Here Output as follows 👍



Task 4: Advanced Array Operations

a) Implement a method SliceArray that takes an array, a starting index, and an end index, then returns a new array containing the elements from the start to the end index.

Explanation	Here Output as Follows:
1. SliceArray Method: **- Parameters: Takes an integer array `array`, an integer `start` index, and an integer `end` index.** **2. Main Method:** **- Initializes an example array.** **- Defines `start` and `end` indices.** **- Calls the `SliceArray` method to get the sliced array.** **- Prints the elements of the sliced array.** **Running the Program** **1. Copy the code into a file named `SliceArrayExample.java`.** **2. Compile the program using the command `javac SliceArrayExample.java`.** **3. Run the compiled program using the command `java SliceArrayExample`.** **The output should display the sliced array:** **Sliced array:** **3 4 5 6** **This demonstrates the `SliceArray` method which takes a segment of the original array based on the provided start and end indices and returns a new array containing those elements.**	 

b) Create a recursive function to find the nth element of a Fibonacci sequence and store the first n elements in an array.

1. fibonacciRecursive method: **- If `n` is 0, it returns 0.** **- If `n` is 1, it returns 1.** **- For any other value of `n`, it returns the sum of the Fibonacci numbers at positions `n-1` and `n-2`.** **2. getFibonacciSequence method:** **- It creates an array `fibSequence` of size `n`.** **- It iterates from 0 to `n-1`, calculating each Fibonacci number using the `fibonacciRecursive` method and storing it in the array.** **3. Main method:** **- It defines the number of Fibonacci numbers to generate (`n`).** **- It calls `getFibonacciSequence` to get the first `n` Fibonacci numbers.** **- It prints each number in the sequence.** **This simple Java program will output the first 10 Fibonacci numbers when run. You can change the value of `n` to generate more or fewer numbers.**	**Here Output as follows 👍**