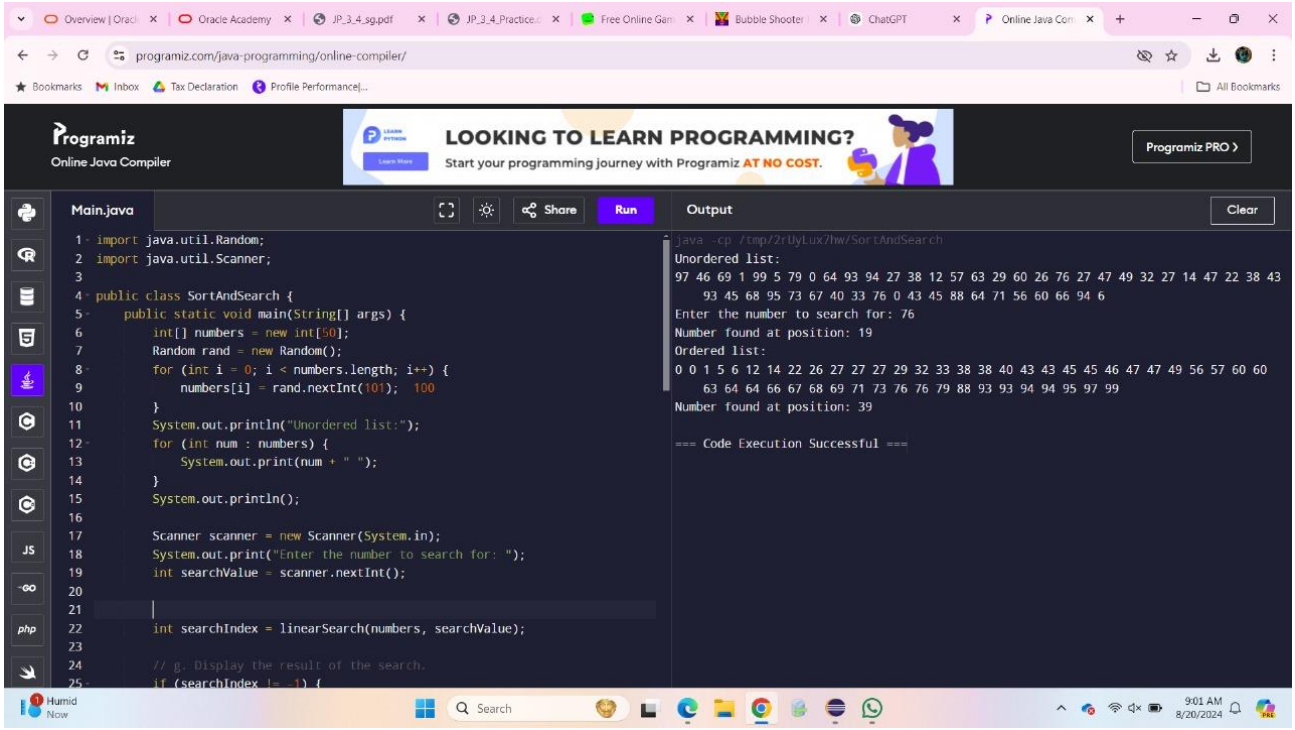
**Try It/Solve It**

**1.**

```
import java.util.Random;
import java.util.Scanner;

public class SortAndSearch {
   public static void main(String[] args) {
      // b. Create an integer array named numbers that will hold 50 values.
      int[] numbers = new int[50];

      // c. Fill the array with random integer values between 0 and 100.
      Random rand = new Random();
      for (int i = 0; i < numbers.length; i++) {
         numbers[i] = rand.nextInt(101); // Generates numbers between 0 and 100
      }

      // d. Display the contents of the array under the heading "Unordered list".
      System.out.println("Unordered list:");
      for (int num : numbers) {
         System.out.print(num + " ");
      }
      System.out.println();

      // e. Get the number to be searched for from the user.
      Scanner scanner = new Scanner(System.in);
      System.out.print("Enter the number to search for: ");
      int searchValue = scanner.nextInt();

      // f. Use a sequential/linear search to identify if the value is in the array.
      int searchIndex = linearSearch(numbers, searchValue);
```

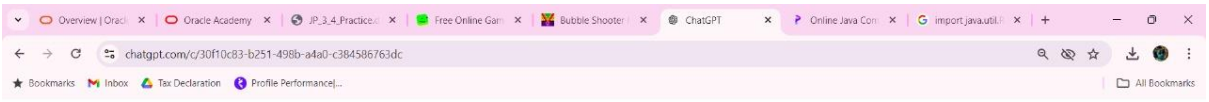
```
    // g. Display the result of the search.
    if (searchIndex != -1) {
        System.out.println("Number found at position: " + searchIndex);
    } else {
        System.out.println("Number not found.");
    }

    // h. Sort the array using a bubble sort.
    bubbleSort(numbers);

    // i. Display the contents of the array under the heading "Ordered list".
    System.out.println("Ordered list:");
    for (int num : numbers) {
        System.out.print(num + " ");
    }
    System.out.println();

    // j. Use a sequential/linear search to identify if the value is in the array.
    searchIndex = linearSearch(numbers, searchValue);

    // Display the result of the search.
    if (searchIndex != -1) {
        System.out.println("Number found at position: " + searchIndex);
    } else {
        System.out.println("Number not found.");
    }

    scanner.close();
}

// Method to perform a linear search
```

```
    public static int linearSearch(int[] array, int value) {
        for (int i = 0; i < array.length; i++) {
            if (array[i] == value) {
                return i;
            }
        }
        return -1;
    }

    // Method to perform a bubble sort
    public static void bubbleSort(int[] array) {
        int n = array.length;
        boolean swapped;
        for (int i = 0; i < n - 1; i++) {
            swapped = false;
            for (int j = 0; j < n - 1 - i; j++) {
                if (array[j] > array[j + 1]) {
                    // Swap array[j] and array[j + 1]
                    int temp = array[j];
                    array[j] = array[j + 1];
                    array[j + 1] = temp;
                    swapped = true;
                }
            }
            // If no two elements were swapped by inner loop, then the array is sorted
            if (!swapped) break;
        }
    }
}
```




**2.Complete the following table using O notation. Under the notes section describe which would on average perform the best?**

| Algorithm | Worst Case | Average Case | Best Case | Notes |
|---|---|---|---|---|
| Selection | $O(n^2)$ | $O(n^2)$ | $O(n^2)$ | Always makes $O(n^2)$ comparisons and swaps regardless of the input's order. Not stable. |
| Bubble | $O(n^2)$ | $O(n^2)$ | $O(n)$ | Simple but inefficient; performs well on nearly sorted data due to early termination |
| Merge | $O(n \log n)$ | $O(n \log n)$ | $O(n \log n)$ | Efficient for large datasets; stable and works consistently regardless of input order |




Here's the completed table with Big O notation for the Selection, Bubble, and Merge Sort algorithms:

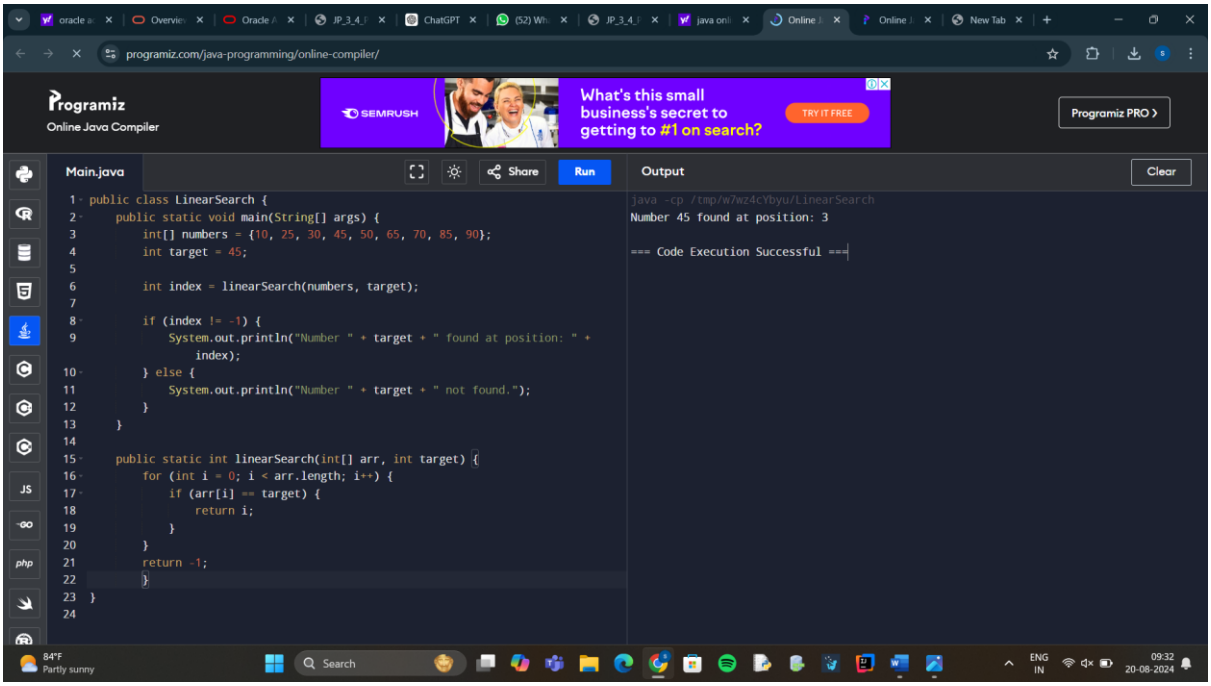
| Algorithm | Worst Case | Average Case | Best Case | Notes |
|---|---|---|---|---|
| Selection | $O(n^2)$ | $O(n^2)$ | $O(n^2)$ | Always makes $O(n^2)$ comparisons and swaps regardless of the input's order. Not stable. |
| Bubble | $O(n^2)$ | $O(n^2)$ | $O(n)$ | Simple but inefficient; performs well on nearly sorted data due to early termination. |
| Merge | $O(n \log n)$ | $O(n \log n)$ | $O(n \log n)$ | Efficient for large datasets; stable and works consistently regardless of input order. |

**Notes on Performance:**

**3. Describe the difference between a linear and a binary search.**

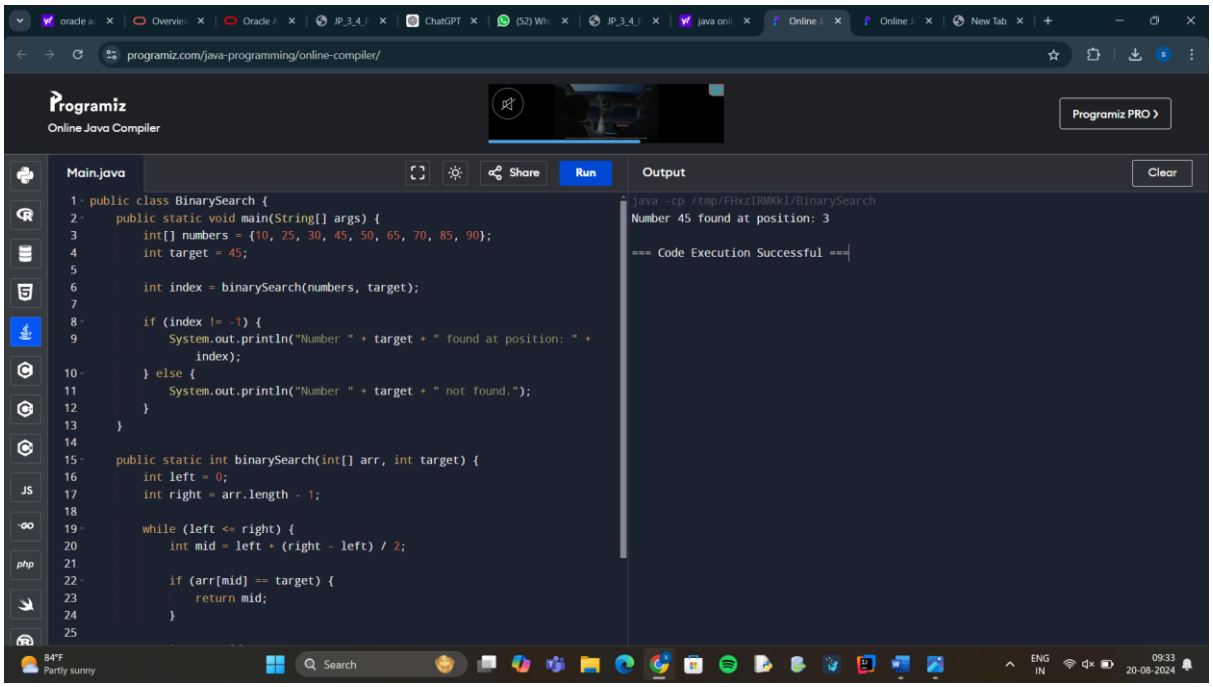
**LINEAR SEARCH:**

A linear search checks each element in a list sequentially until it finds the target or reaches the end. It works on both sorted and unsorted lists but is generally slower, with a time complexity of O(n)O(n)O(n).



BINARY SEARCH :

A binary search only works on sorted lists. It repeatedly divides the search interval in half, comparing the target to the middle element, and discards half of the list each time. This makes it faster, with a time complexity of O(logn)O(\log n)O(logn).




**4.Explain how sorting order is determined if the data contains strings and numbers.**

1. Sorting within a Single Data Type:

Strings:

Lexicographical Order: Strings are sorted based on lexicographical (dictionary) order. This means that "Apple" comes before "Banana" because "A" comes before "B". Sorting is usually case-sensitive, so "Apple" comes before "apple" if uppercase letters are sorted before lowercase.

Numbers:

Numerical Order: Numbers are sorted in numerical order. For example, 1 comes before 2, and 10 comes after 2.

2. Mixed Data Types (Strings and Numbers):

Custom Sorting Rules: If a collection contains both strings and numbers, a typical sorting method must define rules for how to compare and order different types. Here are common approaches:

Type-Based Sorting: Some sorting algorithms may separate data by type first, sorting all numbers in numerical order and all strings in lexicographical order, with numbers coming before or after strings.

Example: [3, 10, 2, "Apple", "Banana"] might be sorted as [2, 3, 10, "Apple", "Banana"] or ["Apple", "Banana", 2, 3, 10].

Coercion to Strings: Some sorting methods convert all data types to strings and then perform lexicographical sorting. This could result in the numbers being sorted based on their string representation.

Example: ["2", "10", "3", "Apple", "Banana"] would sort as ["10", "2", "3", "Apple", "Banana"], which may not be desirable for numbers.