1.

(a),(b),(c)

**import java.util.Random;**

**int[] numbers = new int[50];**

**Random rand = new Random();**

**for (int i = 0; i < numbers.length; i++) {**

**numbers[i] = rand.nextInt(101); // Random integer between 0 and 100**

**}**

**(d)**

**System.out.println("Unordered list:");**

**for (int num : numbers) {**

**System.out.print(num + " ");**

**}**

**System.out.println();**

**(e)**

**import java.util.Scanner;**

**Scanner scanner = new Scanner(System.in);**

**System.out.print("Enter the number to search for: ");**

**int searchValue = scanner.nextInt();**

**(f)**

**int position = -1;**

**for (int i = 0; i < numbers.length; i++) {**

**if (numbers[i] == searchValue) {**

**position = i;**

**break;**

**}**

**}**

**(h)**

**for (int i = 0; i < numbers.length - 1; i++) {**

**for (int j = 0; j < numbers.length - 1 - i; j++) {**

**if (numbers[j] > numbers[j + 1]) {**

**int temp = numbers[j];**

**numbers[j] = numbers[j + 1];**

**numbers[j + 1] = temp;**

**}**

**}**

**}**

**(i)**

**System.out.println("Ordered list:");**

**for (int num : numbers) {**

**System.out.print(num + " ");**

**}**

**System.out.println();**

**(j)**

**position = -1;**

**for (int i = 0; i < numbers.length; i++) {**

**if (numbers[i] == searchValue) {**

**position = i;**

**break;**

**}**

**}**

**if (position != -1) {**

**System.out.println("Number found at position: " + position);**

**} else {**

**System.out.println("Number not found");**

**}**

**2.**

**Time Complexity Table Using O Notation**

| **Algorithm** | **Worst Case** | **Average Case** | **Best Case** | **Notes** |
| --- | --- | --- | --- | --- |
| **Selection** | **O(n²)** | **O(n²)** | **O(n²)** | **Best for small arrays; simple but slow.** |
| **Bubble** | **O(n²)** | **O(n²)** | **O(n)** | **Best case when array is already sorted (O(n)); otherwise inefficient.** |
| **Merge** | **O(n log n)** | **O(n log n)** | **O(n log n)** | **Best on average, handles large data sets efficiently.** |

* **Best Average Performance: Merge sort performs the best on average due to its O(n log n) complexity, making it more efficient for larger data sets.**

**3. Difference Between Linear and Binary Search**

* **Linear Search:**
  + **Scans each element in the array sequentially until it finds the target value or reaches the end.**
  + **Time Complexity: O(n)**
  + **Usage: Works on unsorted or sorted data.**
* **Binary Search:**
  + **Divides the array in half repeatedly, discarding the half where the target cannot lie.**
  + **Time Complexity: O(log n)**
  + **Usage: Requires the data to be sorted beforehand.**

**4. Sorting Order with Mixed Data (Strings and Numbers)**

* **When sorting data that includes both strings and numbers:**
  + **Numeric Sort: Numbers are sorted by their numeric value.**
  + **String Sort: Strings are sorted lexicographically (dictionary order).**
  + **Mixed Data: Sorting order is typically determined by the data type or a custom comparator. If not explicitly defined, the sort may throw an error or produce unpredictable results.**
* **In Java, you can define custom sorting behavior by implementing a Comparator or using the compareTo method in a class that implements Comparable.**