1. BUBBLE SORT

ALGORITHM:

1. Check if n is equal to 1:
2. If true, return; this is the base case of recursion, indicating that the array is sorted.
3. Initialize an integer variable count with the value 0 to keep track of the number of swaps in the current pass.
4. Iterate through the array from the beginning to the second-to-last element (indices 0 to n - 2):
5. Check if the current element arr[i] is greater than the next element arr[i + 1]:
6. If true, swap arr[i] and arr[i + 1].
7. Increment the count by 1 to indicate that a swap occurred.
8. After the loop, check if count is still 0:
9. If true, return; no swaps were made in this pass, indicating that the array is already sorted.
10. Recursively call bubbleSort with the same array arr and n - 1 to sort the remaining unsorted portion of the array.

```
package demo4;
import java.util.Arrays;

public class Bubble {

   static void bubbleSort(int arr[], int n) {

      if (n == 1)
         return;

      int count = 0;

      for (int i = 0; i < n - 1; i++) {
         if (arr[i] > arr[i + 1]) {
```

```
                int temp = arr[i];
                arr[i] = arr[i + 1];
                arr[i + 1] = temp;
                count = count + 1;
            }
        }

        if (count == 0)
            return;

        bubbleSort(arr, n - 1);
    }

    public static void main(String[] args) {
        int arr[] = { 64, 34, 25, 12, 22, 11, 90 };

        bubbleSort(arr, arr.length);

        System.out.println("Sorted array: ");
        System.out.println(Arrays.toString(arr));
    }
}
```

Sorted array:
[11, 12, 22, 25, 34, 64, 90]

2. SELECTION SORT

Algorithm:

1. Define a static method selectionSort that takes an integer array arr and an integer currentIndex as arguments.
2. Inside the selectionSort method:
3. Check if currentIndex is equal to the index of the last element in the array (arr.length - 1):

4. If true, return; this is the base case of recursion, indicating that the array is sorted.
5. Initialize an integer variable minIndex with the value of currentIndex, assuming the current element is the minimum.
6. Iterate over the remaining unsorted portion of the array (starting from currentIndex + 1 to the end of the array):
7. Compare each element with the element at minIndex.
8. If the current element is smaller than the element at minIndex, update minIndex with the index of the current element.
9. After the loop, swap the element at currentIndex with the element at minIndex to move the minimum element to its correct position.
10. Recursively call selectionSort with the same array arr and an incremented currentIndex (i.e., currentIndex + 1) to sort the remaining unsorted portion of the array.

```
package demo4;

public class Selection {

 static void selectionSort(int arr[], int currentIndex) {
                    if (currentIndex == arr.length - 1) {
                       return;
                    }

                    int minIndex = currentIndex;
 for (int i = currentIndex + 1; i < arr.length; i++) {
                       if (arr[i] < arr[minIndex]) {
                          minIndex = i;
                       }
                    }

                    int temp = arr[currentIndex];
                    arr[currentIndex] = arr[minIndex];
                    arr[minIndex] = temp;
```

```
        selectionSort(arr, currentIndex + 1);
    }

    public static void main(String[] args) {
int arr[] = {64, 34, 25, 12, 22, 11, 90};

        selectionSort(arr, 0);

        System.out.println("Sorted array:");
        printArray(arr);
    }

    static void printArray(int arr[]) {
        for (int num : arr) {
            System.out.print(num + " ");
        }
        System.out.println();
    }
}
```

Sorted array:
11 12 22 25 34 64 90