DSA Problems:

1. **Bubble sort:**

```
import java.util.Arrays;

public class BubbleSort {
    public static void bubbleSort(int[] arr) {
        int n = arr.length;
        for (int i = 0; i < n - 1; i++) {
            for (int j = 0; j < n - i - 1; j++) {
                if (arr[j] > arr[j + 1]) {
                    // Swap arr[j] and arr[j + 1]
                    int temp = arr[j];
                    arr[j] = arr[j + 1];
                    arr[j + 1] = temp;
                }
            }
        }
    }

    public static void main(String[] args) {
        int[] arr = {64, 34, 25, 12, 22, 11, 90};
        System.out.println("Original Array: " +
Arrays.toString(arr));
        bubbleSort(arr);
        System.out.println("Sorted Array: " +
Arrays.toString(arr));
```

```
    }
}
```

output:

```
Original Array: [64, 34, 25, 12, 22, 11, 90]
Sorted Array: [11, 12, 22, 25, 34, 64, 90]
```

2. Quick sort:

```
import java.util.Arrays;

public class QuickSort {
    public static void quickSort(int[] arr, int low, int high) {
        if (low < high) {
            int pi = partition(arr, low, high);

            // Recursively sort elements before and after partition
            quickSort(arr, low, pi - 1);
            quickSort(arr, pi + 1, high);
        }
    }

    public static int partition(int[] arr, int low, int high) {
        int pivot = arr[high];
        int i = (low - 1);
```

```
        for (int j = low; j < high; j++) {
            if (arr[j] <= pivot) {
                i++;
                // Swap arr[i] and arr[j]
                int temp = arr[i];
                arr[i] = arr[j];
                arr[j] = temp;
            }
        }

          int temp = arr[i + 1];
        arr[i + 1] = arr[high];
        arr[high] = temp;

        return i + 1;
    }

    public static void main(String[] args) {
        int[] arr = {10, 7, 8, 9, 1, 5};
        System.out.println("Original Array: " +
Arrays.toString(arr));
        quickSort(arr, 0, arr.length - 1);
        System.out.println("Sorted Array: " +
Arrays.toString(arr));
    }
```

```
}
```

output:

```
Original Array: [10, 7, 8, 9, 1, 5]
Sorted Array: [1, 5, 7, 8, 9, 10]
```

3. **K largest element:**

```
import java.util.PriorityQueue;
import java.util.Arrays;

public class KLargestElements {
    public static int[] findKLargestElements(int[]
arr, int k) {
        if (k <= 0 || arr == null || arr.length < k) {
            throw new
IllegalArgumentException("Invalid input.");
        }

        // Min-heap to store the K largest elements
        PriorityQueue<Integer> minHeap = new
PriorityQueue<>(k);

        for (int num : arr) {
            if (minHeap.size() < k) {
                minHeap.add(num);
            } else if (num > minHeap.peek()) {
```

```
            // Remove the smallest element and add
the new one
            minHeap.poll();
            minHeap.add(num);
        }
    }

    // Convert heap to an array
    int[] result = new int[k];
    int index = 0;
    for (int num : minHeap) {
        result[index++] = num;
    }

    // Optional: Sort the result array in
descending order
    Arrays.sort(result);
    for (int i = 0; i < k / 2; i++) {
        int temp = result[i];
        result[i] = result[k - i - 1];
        result[k - i - 1] = temp;
    }

    return result;
  }
```

```
    public static void main(String[] args) {
        int[] arr = {3, 2, 1, 5, 6, 4};
        int k = 3;
        int[] kLargest = findKLargestElements(arr, k);
        System.out.println("K Largest Elements: " +
Arrays.toString(kLargest));
    }
}
```

Output:

```
K Largest Elements: [6, 5, 4]
PS F:\java>
```