Practice Questions

Solution 1:-

```
class Solution
{
   static int majorityElement(int a[], int size)
   {

      HashMap<Integer, Integer> map = new HashMap<>();


      int n = size/2;

      for(int i=0; i< a.length; i++){
         int num = a[i];
         if(map.containsKey(num)){
            map.put(num, map.get(num)+1);

         }else{
            map.put(num, 1);
         }
      }

      for(Map.Entry<Integer,Integer> en : map.entrySet()){
         if(en.getValue() > n){
            return en.getKey();
         }
      }
      return -1;
   }
}
```

Solution 2:-

```
class Solution
{
    // arr[]: Input Array
    // N : Size of the Array arr[]
    //Function to count inversions in the array.
    static long inversionCount(long arr[], long N)
    {
        // Your Code Here
        return mergeSort(arr,0,(int)N-1);

    }

    public static long mergeSort(long arr[], int low, int high){

        long inv = 0;
        if(low >= high)return inv;

        int mid = (low + high) / 2;

        inv += mergeSort(arr, low, mid);
        inv += mergeSort(arr, mid+1, high);
        inv += merge(arr, low, mid,high);

        return inv;

    }
```

```
private static long merge(long[] arr, int low, int mid, int high) {
    ArrayList<Long> temp = new ArrayList<>(); // temporary array
    int left = low;      // starting index of left half of arr
    int right = mid + 1;   // starting index of right half of arr

    //Modification 1: cnt variable to count the pairs:
    long cnt = 0;

    //storing elements in the temporary array in a sorted manner//

    while (left <= mid && right <= high) {
        if (arr[left] <= arr[right]) {
            temp.add(arr[left]);
            left++;
        } else {
            temp.add(arr[right]);
            cnt += (mid - left + 1); //Modification 2
            right++;
        }
    }

    // if elements on the left half are still left //

    while (left <= mid) {
        temp.add(arr[left]);
        left++;
    }
```

```
        //  if elements on the right half are still left //
        while (right <= high) {
            temp.add(arr[right]);
            right++;
        }

        // transfering all elements from temporary to arr //
        for (int i = low; i <= high; i++) {
            arr[i] = temp.get(i - low);
        }
        return cnt; // Modification 3
    }
}
```

Solution 3:-

```
class Solution{
    //Function to find the leaders in the array.
    static ArrayList<Integer> leaders(int arr[], int n){
        ArrayList<Integer> al=new ArrayList();
        int max;
        for(int i=0;i<n;i++){
            max=arr[i];
            for(int j=i+1;j<n;j++){
                if(arr[j]>max){
                    max=arr[j];
                    i=j;
                }
```

```
            }
            al.add(max);
        }
        return al;
    }
}
```

Solution 4:-

```
class Solution
{
    // arr[]: Input Array
    // N : Size of the Array arr[]
    //Function to count inversions in the array.
    static long inversionCount(long arr[], long N)
    {
        // Your Code Here
        return mergeSort(arr,0,(int)N-1);

    }

    public static long mergeSort(long arr[], int low, int high){

        long inv = 0;
        if(low >= high)return inv;

        int mid = (low + high) / 2;
```

```
    inv += mergeSort(arr, low, mid);
    inv += mergeSort(arr, mid+1, high);
    inv += merge(arr, low, mid,high);

    return inv;

}
private static long merge(long[] arr, int low, int mid, int high) {
    ArrayList<Long> temp = new ArrayList<>(); // temporary array
    int left = low;      // starting index of left half of arr
    int right = mid + 1;   // starting index of right half of arr

    //Modification 1: cnt variable to count the pairs:
    long cnt = 0;

    //storing elements in the temporary array in a sorted manner//

    while (left <= mid && right <= high) {
        if (arr[left] <= arr[right]) {
            temp.add(arr[left]);
            left++;
        } else {
            temp.add(arr[right]);
            cnt += (mid - left + 1); //Modification 2
            right++;
        }
    }
```

```
        // if elements on the left half are still left //

        while (left <= mid) {
            temp.add(arr[left]);
            left++;
        }

        //  if elements on the right half are still left //
        while (right <= high) {
            temp.add(arr[right]);
            right++;
        }

        // transfering all elements from temporary to arr //
        for (int i = low; i <= high; i++) {
            arr[i] = temp.get(i - low);
        }
        return cnt; // Modification 3
    }
}
```