Practice Sets

Solution 1:-

```
class Solution
{
   static int majorityElement(int a[], int size)
   {
      Map<Integer,Integer> m = new HashMap<>();
      for(int i=0;i<size;i++){
         if(m.containsKey(a[i]))
            m.put(a[i],m.get(a[i])+1);
         else
            m.put(a[i],1);
      }
     Optional<java.util.Map.Entry<Integer,Integer>>
max=  m.entrySet().stream().max((e1,e2)-
>e1.getValue().compareTo(e2.getValue()));
     if(!max.isEmpty()){
       int k=max.get().getKey();
       int c=max.get().getValue();
       if(size/2<c){
            return k;
      }
     }
     return -1;
   }
}
```

Solution 2:-

```
class Solution
{
   // arr[]: Input Array
   // N : Size of the Array arr[]
   //Function to count inversions in the array.
   public static long  conquer(long arr[],int s,int e,int mid){
      long merger[]=new long[e-s+1];
      int x=0;
      int i=s;
      int j=mid+1;
long ans=0;
      while(i<=mid && j<=e){
         if(arr[i]<=arr[j]){
            merger[x]=arr[i];
            i++;
         }
         else{
            merger[x]=arr[j];
            ans+=(mid+1-i);
            j++;
         }
         x++;
      }
      while(i<=mid){
         merger[x]=arr[i];
         x++;
         i++;
      }
      while(j<=e){
         merger[x]=arr[j];
         x++;
         j++;
      }
      int d=s;
```

```
        for(int k=0;k<merger.length;k++){
            arr[d]=merger[k];
            d++;
        }
        return ans;

    }
    public static long divide(long arr[],int s,int e){
        long ans=0;
        if(s<e){

        int mid= s + (e-s)/2;
        ans+=divide(arr,s,mid);
        ans+=divide(arr,mid+1,e);
        ans+=conquer(arr,s,e,mid);
    }
    return ans;
    }
    static long inversionCount(long arr[], long N)
    {
        // Your Code Here
        int j=(int)(N-1);
        long ans=divide(arr,0,j);
        return ans;
    }
}
```