1)Given an array A of positive integers. Your task is to find the leaders in the array. An element of array is a leader if it is greater than or equal to all the elements to its right side. The rightmost element is always a leader.

Example 1:

Input:
n = 6
A[] = {16,17,4,3,5,2}
Output: 17 5 2
Explanation: The first leader is 17
as it is greater than all the elements
to its right. Similarly, the next
leader is 5. The right most element
is always a leader so it is also
included.
Example 2:

Input:
n = 5
A[] = {1,2,3,4,0}
Output: 4 0
Explanation: 0 is the rightmost element
and 4 is the only element which is greater
than all the elements to its right.
Your Task:
You don't need to read input or print anything. The task is to complete the function leader() which takes array A and n as input parameters and returns an array of leaders in order of their appearance.

Expected Time Complexity: O(n)
Expected Auxiliary Space: O(n)

Constraints:
1 <= n <= 107
0 <= Ai <= 107

Ans:

```java
import java.util.ArrayList;

class Solution {
   static ArrayList<Integer> leaders(int arr[], int n) {
       ArrayList<Integer> result = new ArrayList<>();

       int maxRight = arr[n - 1];

       result.add(maxRight);

       for (int i = n - 2; i >= 0; i--) {
           if (arr[i] >= maxRight) {
               maxRight = arr[i];

               result.add(0, arr[i]);
           }
       }

       return result;
   }

   public static void main(String[] args) {
       int[] arr1 = {16, 17, 4, 3, 5, 2};
       int[] arr2 = {1, 2, 3, 4, 0};

       ArrayList<Integer> result1 = leaders(arr1, arr1.length);
       ArrayList<Integer> result2 = leaders(arr2, arr2.length);

       System.out.println("Leaders in array 1: " + result1);
       System.out.println("Leaders in array 2: " + result2);
   }
}
```

2)Given a string S. The task is to print all unique permutations of the given string that may contain dulplicates in lexicographically sorted order.

Example 1:

Input: ABC
Output:
ABC ACB BAC BCA CAB CBA
Explanation:
Given string ABC has permutations in 6

forms as ABC, ACB, BAC, BCA, CAB and CBA .
Example 2:

Input: ABSG
Output:
ABGS ABSG AGBS AGSB ASBG ASGB BAGS
BASG BGAS BGSA BSAG BSGA GABS GASB
GBAS GBSA GSAB GSBA SABG SAGB SBAG
SBGA SGAB SGBA
Explanation:
Given string ABSG has 24 permutations.
Your Task:
You don't need to read input or print anything. Your task is to complete the function find_permutation() which takes the string S as input parameter and returns a vector of string in lexicographical order.

Expected Time Complexity: O(n! * n)
Expected Space Complexity: O(n! * n)

Constraints:
1 <= length of string <= 5

Ans:

```
import java.util.*;

class Solution {
   static ArrayList<String> find_permutation(String S) {
       ArrayList<String> result = new ArrayList<>();
       char[] charArray = S.toCharArray();
       Arrays.sort(charArray);
       String sortedString = new String(charArray);
       boolean[] used = new boolean[S.length()];
       StringBuilder current = new StringBuilder();
       generatePermutations(sortedString, used, current, result);
       return result;
   }

   static void generatePermutations(String str, boolean[] used, StringBuilder
current, ArrayList<String> result) {
       if (current.length() == str.length()) {
           result.add(current.toString());
```

```
            return;
        }

        for (int i = 0; i < str.length(); i++) {
            if (used[i] || (i > 0 && str.charAt(i) == str.charAt(i - 1) &&
!used[i - 1]))
                continue;
            used[i] = true;
            current.append(str.charAt(i));
            generatePermutations(str, used, current, result);
            current.deleteCharAt(current.length() - 1);
            used[i] = false;
        }
    }

    public static void main(String[] args) {
        String S1 = "ABC";
        String S2 = "ABSG";

        ArrayList<String> result1 = find_permutation(S1);
        ArrayList<String> result2 = find_permutation(S2);

        System.out.println("Unique permutations of ABC: " + result1);
        System.out.println("Unique permutations of ABSG: " + result2);
    }
}
```

3)Given a string S consisting of lowercase Latin Letters. Return the first non-repeating character in S. If there is no non-repeating character, return '$'.

Example 1:

Input:
S = hello
Output: h
Explanation: In the given string, the first character which is non-repeating is h, as it appears first and there is no other 'h' in the string.
Example 2:

Input:
S = zxvczbtxyzvy
Output: c
Explanation: In the given string, 'c' is
the character which is non-repeating.
Your Task:
You only need to complete the function nonrepeatingCharacter() that takes string S as a parameter and returns the character. If there is no non-repeating character then return '$' .

Expected Time Complexity: O(N).
Expected Auxiliary Space: O(Number of distinct characters).
Note: N = |S|

Constraints:
1 <= N <= 105

Ans:

```java
import java.util.*;

class Solution {
   static char nonrepeatingCharacter(String S) {
       Map<Character, Integer> charFreq = new LinkedHashMap<>();

       for (char c : S.toCharArray()) {
           charFreq.put(c, charFreq.getOrDefault(c, 0) + 1);
       }

       for (char c : S.toCharArray()) {
           if (charFreq.get(c) == 1) {
               return c;
           }
       }

       return '$';
   }

   public static void main(String[] args) {
       String S1 = "hello";
       String S2 = "zxvczbtxyzvy";
```

```
        System.out.println("First non-repeating character in 'hello': " +
nonrepeatingCharacter(S1));
        System.out.println("First non-repeating character in 'zxvczbtxyzvy': "
+ nonrepeatingCharacter(S2)); /
    }
}
```

4)Given an array of n distinct elements. Find the minimum number of swaps required to sort the array in strictly increasing order.

Example 1:

Input:
nums = {2, 8, 5, 4}
Output:
1
Explanation:
swap 8 with 4.
Example 2:

Input:
nums = {10, 19, 6, 3, 5}
Output:
2
Explanation:
swap 10 with 3 and swap 19 with 5.

Your Task:
You do not need to read input or print anything. Your task is to complete the function minSwaps() which takes the nums as input parameter and returns an integer denoting the minimum number of swaps required to sort the array.
If the array is already sorted, return 0.

Expected Time Complexity: O(nlogn)
Expected Auxiliary Space: O(n)

Constraints:

1 ≤ n ≤ 105

1 ≤ numsi ≤ 106

Ans:

```java
import java.util.*;

class Solution {
    static int minSwaps(int[] nums) {
        int n = nums.length;
        List<Pair<Integer, Integer>> pairs = new ArrayList<>();
        for (int i = 0; i < n; i++) {
            pairs.add(new Pair<>(nums[i], i));
        }
        pairs.sort(Comparator.comparing(Pair::getKey));

        boolean[] visited = new boolean[n];
        int swaps = 0;
        for (int i = 0; i < n; i++) {
            if (visited[i] || pairs.get(i).getValue() == i)
                continue;
            int cycleSize = 0;
            int j = i;
            while (!visited[j]) {
                visited[j] = true;
                j = pairs.get(j).getValue();
                cycleSize++;
            }
            if (cycleSize > 0) {
                swaps += cycleSize - 1;
            }
        }
        return swaps;
    }

    public static void main(String[] args) {
        int[] nums1 = {2, 8, 5, 4};
        int[] nums2 = {10, 19, 6, 3, 5};

        System.out.println("Minimum number of swaps for nums1: " +
minSwaps(nums1));
        System.out.println("Minimum number of swaps for nums2: " +
minSwaps(nums2));
    }
}
```