1. **Longest Palindrome:**

Given a string s which consists of lowercase or uppercase letters, return the length of the longest palindrome that can be built with those letters.

Letters are case sensitive, for example, "Aa" is not considered a palindrome here.

**Example 1:**

Input: s = "abccccdd"

Output: 7

Explanation: One longest palindrome that can be built is "dccaccd", whose length is 7.

**Example 2:**

Input: s = "a"

Output: 1

Explanation: The longest palindrome that can be built is "a", whose length is 1.

**Solution:**

public int longestPalindrome(String s) {

        int n=s.length();

        HashMap<Character,Integer> hm=new HashMap<>();

        for(int i=0;i<n;i++)

        {

            if(hm.containsKey(s.charAt(i)))

            hm.put(s.charAt(i),hm.get(s.charAt(i))+1);

            else

             hm.put(s.charAt(i),1);

        }

        int count=0,flag=0;

        for(HashMap.Entry<Character,Integer> m: hm.entrySet())

        {

           int k= m.getKey();

           int v=m.getValue();

           if(v%2==0) count=count+v;

           else{

               count=count+(v-v%2);

               if(flag==0){

                   count+=1;

                   flag=1;

               }

           }

        }

        return count;

    }

1. **Group Anagrams:**

Given an array of strings strs, group the anagrams together. You can return the answer in any order.

An Anagram is a word or phrase formed by rearranging the letters of a different word or phrase, typically using all the original letters exactly once.

**Example 1:**

**Input:** strs = ["eat","tea","tan","ate","nat","bat"]

**Output:** [["bat"],["nat","tan"],["ate","eat","tea"]]

**Example 2:**

**Input:** strs = [""]

**Output:** [[""]]

**Example 3:**

**Input:** strs = ["a"]

**Output:** [["a"]]

**Solution:**

public List<List<String>> groupAnagrams(String[] strs) {

        HashMap<String,List<String>> hm=new HashMap<>();

        List<List<String>> ans=new ArrayList<>();

        int n=strs.length;

        for(String s: strs)

        {

            char ch[]=s.toCharArray();

            Arrays.sort(ch);

            String sorted\_k=new String(ch);

            if(!hm.containsKey(sorted\_k))

            hm.put(sorted\_k,new ArrayList<>());

            hm.get(sorted\_k).add(s);

        }

        for(List<String> li : hm.values())

        {

            ans.add(li);

        }

        return ans;

    }

1. **Majority Element:**

Given an array nums of size n, return the majority element.

The majority element is the element that appears more than ⌊n / 2⌋ times. You may assume that the majority element always exists in the array.

**Example 1:**

**Input:** nums = [3,2,3]

**Output:** 3

**Example 2:**

**Input:** nums = [2,2,1,1,1,2,2]

**Output:** 2

**Solution:**

public int majorityElement(int[] nums) {

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

        int n=nums.length;

        for(int i=0;i<n;i++)

        {

            if(hm.containsKey(nums[i]))

            hm.put(nums[i],hm.get(nums[i])+1);

            else

            hm.put(nums[i],1);

        }

       return hm.entrySet().stream().filter(a->a.getValue()>n/2)

.findAny().map(k->k.getKey()).get();

    }