LeetCode
March Daily Challenges

In this PDF, you will find solutions to the daily problems for the March LeetCode Challenge, each with various approaches.

Prepared by:
Adarsh Gupta

Question 1: Maximum Odd Binary Number
https://leetcode.com/problems/maximum-odd-binary-number/description/

```java
class Solution {
    public String maximumOddBinaryNumber(String s) {
        int n = s.length();
        char ch[] = new char[n];
        int countOne = 0;
        // Count the number of 1s
        for(int i=0;i<n;i++){
            if(s.charAt(i) == '1'){
                countOne++;
            }
        }

        //Putting 1 from starting because we want maximum number
        int j = 0;
        while(countOne > 1){
            ch[j] = '1';
            countOne--;
            j++;
        }
        // Filling 0s except last position
        while(j < n-1){
            ch[j] = '0';
            j++;
        }
        // Putting the last digit as 1 to be odd
        ch[n-1] = '1';
        return new String(ch);
    }
}
```

Question 2: Squares of Sorted Array
https://leetcode.com/problems/squares-of-a-sorted-array/description/

```
class Solution {
    public int[] sortedSquares(int[] nums) {
        int n = nums.length;
        int res[] = new int [n];
        int l = 0;
        int r = n-1;
        for(int i=0;i<n;i++){
            if((int)Math.pow(nums[l],2) < (int)Math.pow(nums[r],2)){
                res[n-1-i] = (int)Math.pow(nums[r],2);
                r--;
            }
            else{
                res[n-1-i] = (int)Math.pow(nums[l],2);
                l++;
            }
        }
        return res;
    }
}
```

Question 3: Remove Nth Node from End of List
https://leetcode.com/problems/remove-nth-node-from-end-of-list/description/

```
/**
 * Definition for singly-linked list.
 * public class ListNode {
 *     int val;
 *     ListNode next;
 *     ListNode() {}
```

```
 *     ListNode(int val) { this.val = val; }
 *     ListNode(int val, ListNode next) { this.val = val; this.next =
next; }
 * }
 */
class Solution {
    public ListNode removeNthFromEnd(ListNode head, int n) {
        ListNode temp1 = head;
        ListNode temp2 = head;

        int i = 0;
        while(i < n){
            temp1 = temp1.next;
            i++;
        }
        // if length of LinkedList l == n then temp1 will be null.
        if(temp1 == null){
            return head.next;
        }

        while(temp1.next != null){
            temp1 = temp1.next;
            temp2 = temp2.next;
        }
        temp2.next = temp2.next.next;

        return head;
    }
}
```

Question 4: Bag of Tokens

https://leetcode.com/problems/bag-of-tokens/description/

```
class Solution {
    public int bagOfTokensScore(int[] tokens, int power) {
        int maxScore = 0;
        Arrays.sort(tokens);
        int i = 0;
        int j = tokens.length - 1;
```

```
        int score = 0;
        while(i <= j){
            if(power >= tokens[i]){
                power -= tokens[i];
                score++;
                i++;
                maxScore = Math.max(score, maxScore);
            }else if(score >= 1){
                power += tokens[j];
                score--;
                j--;
            }else{
                return maxScore;
            }
        }
        return maxScore;
    }
}
```

Question 5: Minimum length of String After Deleting Similar Ends

https://leetcode.com/problems/minimum-length-of-string-after-deleting-similar-ends/

```
class Solution {
    public int minimumLength(String s) {
        int n = s.length();
        int i = 0;
        int j = n - 1;
        // for condition 3: i<j & For condition 4:
s.charAt(i)==s.charAt(j)
        while(i < j && s.charAt(i) == s.charAt(j)){
            char ch = s.charAt(i); // or s.charAt(j)
            // for prefix
            while(i < j && s.charAt(i) == ch){
                i++;
            }
```

```
            // for suffix
            // for s = "aa" j>=i equal to shpuld be there
            while(j >= i && s.charAt(j) == ch){
                j--;
            }
        }
        return (j - i + 1);
    }
}
```

Question 6: Linked List Cycle

https://leetcode.com/problems/linked-list-cycle/description/

```
/**
 * Definition for singly-linked list.
 * class ListNode {
 *     int val;
 *     ListNode next;
 *     ListNode(int x) {
 *         val = x;
 *         next = null;
 *     }
 * }
 */
public class Solution {
    public boolean hasCycle(ListNode head) {
        //Interviewer will not be happy with this approach
        // HashSet<ListNode> set = new HashSet<>();
        // while(head!=null){
        //     if(!set.contains(head)){
        //         set.add(head);
        //         head = head.next;
        //     }else{
        //         return true;
        //     }
        // }
        // return false;
```

```
        if(head==null) return false;

        ListNode slow = head;
        ListNode fast = head;

        while(fast.next!=null && fast.next.next!=null){
            slow = slow.next;
            fast = fast.next.next;

            if(slow == fast){
                return true;
            }
        }
        return false;
    }
}
```

Question 7: Middle of the Linked List

https://leetcode.com/problems/middle-of-the-linked-list/description/

```
/**
 * Definition for singly-linked list.
 * public class ListNode {
 *     int val;
 *     ListNode next;
 *     ListNode() {}
 *     ListNode(int val) { this.val = val; }
 *     ListNode(int val, ListNode next) { this.val = val; this.next =
next; }
 * }
 */
class Solution {
    public ListNode middleNode(ListNode head) {
        // if(head == null) return null;

        // ListNode temp = head;
```

```
        // int length = 1;
        // while(temp.next!=null){
        //     length++;
        //     temp = temp.next;
        // }

        // int len = length/2 + 1;
        // ListNode curr = head;
        // for(int i=1;i<len;i++){
        //     curr = curr.next;
        // }
        // return curr;


        ListNode slow = head;
        ListNode fast = head;

        while(fast!=null && fast.next!=null){
            slow = slow.next;
            fast = fast.next.next;
        }

        return slow;
    }
}
```

Question 8: Count Elements with Maximum Frequency

https://leetcode.com/problems/count-elements-with-maximum-frequency/description/

```
class Solution {
    public int maxFrequencyElements(int[] nums) {
        HashMap<Integer, Integer> map = new HashMap<>();
        for (int num : nums) {
            map.put(num, map.getOrDefault(num, 0) + 1);
        }

        int count = 0;
```

```
        int maxFreq = Integer.MIN_VALUE;
        for (int freq : map.values()) {
            maxFreq = Math.max(maxFreq, freq);
        }

        for (int freq : map.values()) {
            if (freq == maxFreq)
                count += maxFreq;
        }
        return count;
    }
}
```

Question 9: Minimum Common Value

https://leetcode.com/problems/minimum-common-value/description/

```
class Solution {
    public int getCommon(int[] nums1, int[] nums2) {
        //Using HashMap
        // HashMap<Integer, Integer> map = new HashMap<>();

        // for(int i=0;i<nums1.length;i++){
        //     map.put(nums1[i], map.getOrDefault(nums1[i],0)+1);
        // }

        // for(int j=0;j<nums2.length;j++){
        //     if(map.containsKey(nums2[j])){
        //         return nums2[j];
        //     }
        // }
        // return -1;

        // Using HashSet
```

```
        HashSet<Integer> set = new HashSet<>();
        for(int i=0;i<nums1.length;i++){
            set.add(nums1[i]);
        }

        for(int j=0;j<nums2.length;j++){
            if(set.contains(nums2[j])){
                return nums2[j];
            }
        }
        return -1;
    }
}
```

Question 10: Intersection of Two Arrays

https://leetcode.com/problems/intersection-of-two-arrays/description/

```
class Solution {
    public int[] intersection(int[] nums1, int[] nums2) {
        HashSet<Integer> set1 = new HashSet<>();
        for(int i=0;i<nums1.length;i++){
            set1.add(nums1[i]);
        }
        HashSet<Integer> set2 = new HashSet<>();
        for(int i=0;i<nums2.length;i++){
            if(set1.contains(nums2[i])){
                set2.add(nums2[i]);
            }
        }
        int res[] = new int[set2.size()];
        int i = 0;
        for(int num : set2){
            res[i] = num;
            i++;
        }
        return res;
    }
}
```

Question 11: Custom Sort String

https://leetcode.com/problems/custom-sort-string/description/

```
class Solution {
    public String customSortString(String order, String s) {
        // Creating HashMap to store character and their count of String s
        HashMap<Character, Integer> map = new HashMap<>();
        for(int i=0;i<s.length();i++){
            map.put(s.charAt(i), map.getOrDefault(s.charAt(i),0)+1);
        }
        // Creating StringBuilder for storing the result
        StringBuilder res = new StringBuilder();
        for(int j=0;j<order.length();j++){
            char ch = order.charAt(j);
            if(map.containsKey(ch)){
                int count = map.get(ch);
                while(count>0){
                    // Adding ch into res until their count is 0
                    res.append(ch);
                    count--;
                }
                //Setting that character's count as 0 because already
added in res
                map.put(ch,0);
            }
        }
        //traversing those character in hashmap whose value is greater
than 0
        for(Map.Entry<Character, Integer> entry : map.entrySet()){
            int val = entry.getValue();
            while(val > 0){
                res.append(entry.getKey());
                val--;
            }
        }
        return res.toString();
    }
}
```

Question 12: Remove Zero Sum Consecutive Nodes from Linked List

https://leetcode.com/problems/remove-zero-sum-consecutive-nodes-from-linked-list/description/

```
/**
 * Definition for singly-linked list.
 * public class ListNode {
 *     int val;
 *     ListNode next;
 *     ListNode() {}
 *     ListNode(int val) { this.val = val; }
 *     ListNode(int val, ListNode next) { this.val = val; this.next =
next; }
 * }
 */
class Solution {
    public ListNode removeZeroSumSublists(ListNode head) {
        HashMap<Integer, ListNode> map = new HashMap<>();
        ListNode dummy = new ListNode(0);
        dummy.next = head;
        int prefixSum = 0;
        map.put(0, dummy);
        while(head!=null){
            prefixSum += head.val;

            if(map.containsKey(prefixSum)){
                ListNode start = map.get(prefixSum);
                ListNode temp = start;
                int sum = prefixSum;
                while(temp!=head){
                    temp = temp.next;
                    sum += temp.val;
                    if(temp!=head){
                        map.remove(sum);
                    }
                }
                start.next = head.next;
            }else{
```

```
                map.put(prefixSum, head);
            }
            head = head.next;
        }
        return dummy.next;
    }
}
```

Question 13: Find the Pivot Integer

https://leetcode.com/problems/find-the-pivot-integer/description/

```
class Solution {
    public int pivotInteger(int n) {
        //Using Binary Search
    //     int left = 1;
    //     int right = n;
    //     int fullSum = n*(n+1)/2;
    //     while(left <= right){
    //      int mid = left + (right - left)/2;
    //      int firstHalfSum = mid*(mid+1)/2;
    //      int secondHalfSum = fullSum - firstHalfSum + mid;
    //      if(firstHalfSum == secondHalfSum){
    //          return mid;
    //      }else if(firstHalfSum < secondHalfSum){
    //          left = mid + 1;
    //      }else{
    //          right = mid -1;
    //      }
    //     }
    //     return -1;

    // Another approach Using Formula (Check Note for formula)
        // for(int i=1;i<=n;i++){
        //     int sum = n*(n+1)/2;
        //     if(i*i == sum){
        //         return i;
        //     }
        // }
        // return -1;
```

```
    // Another approach using above two techniques
        int sum  = n*(n+1)/2;
        int left = 1;
        int right = n;
        while(left <= right){
            // finding mid and assuming it as pivot
            int pivot = left + (right - left)/2;
            if((pivot*pivot) == sum){
                return pivot;
            }else if((pivot*pivot) < sum){
                left = pivot + 1;
            }else{
                right = pivot - 1;
            }
        }
        return -1;
    }
}
```

Note: Let pivot be x.
1, 2, 3, 4, 5, ... x ....., n

According to question:
If x is pivot element then ,
sum[1,x] = sum[x,n]
x*(x+1)/2 = n*(n+1)/2 - x*(x+1)/2 +x
x*(x+1)/2 + x*(x+1)/2 = n*(n+1)/2 + x
x*(x+1) = n*(n+1)/2 + x
x^2 + x = n*(n+1)/2 + x
x^2 = n*(n+1)/2
Means (pivot)^2 = n*(n+1)/2
In this way , we got the formula : (pivot)^2 = n*(n+1)/2.

Question 14: Binary Arrays With Sum

https://leetcode.com/problems/binary-subarrays-with-sum/description/

```
class Solution {
    public int numSubarraysWithSum(int[] nums, int goal) {
        // Brute Force
        // int res = 0;
        // for(int i=0;i<nums.length;i++){
        //     int sum = 0;
        //     for(int j=i;j<nums.length;j++){
        //         sum += nums[j];
        //         if(sum == goal){
        //             res++;
        //         }
        //     }
        // }
        // return res;

        // Using HashMap

        int count=0;
        Map<Integer, Integer> map = new HashMap<>();
        int sum=0;
        for(int i=0;i<nums.length;i++){
            sum+=nums[i];
            if(sum==goal)count++;
         if(map.containsKey(sum-goal)){
             count+= map.get(sum-goal);
         }
         map.put(sum,map.getOrDefault(sum,0)+1);

        }

         return count;
    }
}

//Note: Same as leetcode 560 question
```

Question 15: Product of Array Except Self
https://leetcode.com/problems/product-of-array-except-self/description/

```
class Solution {
    public int[] productExceptSelf(int[] nums) {
        // Approach 1 (Using division)
        // Count zeros and calculate product except zero
        // int countZero = 0;
        // int productExceptZero = 1;
        // int res[] = new int[nums.length]; // For storing the result
        // for(int i=0;i<nums.length;i++){
        //     if(nums[i] == 0){
        //         countZero++;
        //     }else{
        //         productExceptZero *= nums[i];
        //     }
        // }

        // for(int i=0;i<nums.length;i++){
        //     // if the current number is not zero
        //     if(nums[i] != 0){
        //         // if number of zeros is more than 0
        //         if(countZero > 0){
        //             res[i] = 0; // product except that number will be
zero
        //         }else{
        //             res[i] = productExceptZero/nums[i]; // If no of
zero is 0 then product will be productExceptZero divided by current number
        //         }
        //     }else{ // if current number is zero
        //         if(countZero > 1){ // checking if there are other zero
except current one
        //             res[i] = 0; // then product will be zero definitely
        //         }else{//if no other zero except current one
        //             res[i] = productExceptZero;// then product will be
productExceptZero
        //         }
        //     }
```

```
        // }
        // return res; // returning the result


        // Approach 2

        // first create left product array that contains the product of
all the left elements from the current one(except the current num)
        int n = nums.length;
        int left[] = new int[n];
        left[0] = 1;
        for(int i=1;i<n;i++){
            left[i] = nums[i-1]*left[i-1];
        }
        // Now create right product array that contains the product of all
the right elements from the current one(except the current num)
        int right[] = new int[n];
        right[n-1] = 1;
        for(int i=n-2;i>=0;i--){
            right[i] = nums[i+1]*right[i+1];
        }

        int res[] = new int[n];
        for(int i=0;i<n;i++){
            res[i] = left[i]*right[i];
        }
        return res;
    }
}
```

Question 16: Contiguous Array

https://leetcode.com/problems/contiguous-array/description/

```
class Solution {
    public int findMaxLength(int[] nums) {
        // currSum and index
        HashMap<Integer, Integer> map = new HashMap<>();
        int currSum = 0;
        int index = -1;
```

```
        map.put(0, -1);
        int res = 0;
        for(int i=0;i<nums.length;i++){
            currSum += nums[i]==1 ? 1: -1;
            if(map.containsKey(currSum)){
                res = Math.max(res, i-map.get(currSum));
            }else{
                map.put(currSum, i);
            }
        }
        return res;
    }
}
```

Question 17: Insert Interval

https://leetcode.com/problems/insert-interval/description/

```
class Solution {
    public int[][] insert(int[][] intervals, int[] newInterval) {
        int i = 0;
        int n = intervals.length;
        List<int[]> result = new ArrayList<>();

        while (i < intervals.length) {
            if (intervals[i][1] < newInterval[0])
                result.add(intervals[i]);
            else if (intervals[i][0] > newInterval[1]){
                break;
            } else {
                //Overlap : merge them
                newInterval[0] = Math.min(newInterval[0],
intervals[i][0]);
                newInterval[1] = Math.max(newInterval[1],
intervals[i][1]);
            }
            i++;
        }

        result.add(newInterval);
```

```
        while (i < n){
            result.add(intervals[i++]);
        }

        return result.toArray(new int[result.size()][2]);
    }
}
```

Question 18: Minimum Number of Arrows to Burst Ballons
https://leetcode.com/problems/minimum-number-of-arrows-to-burst-balloons/description/

```
class Solution {
    public int findMinArrowShots(int[][] points) {
        // Sorting based on the y-coordinate
        Arrays.sort(points,(a,b) -> Integer.compare(a[1],b[1]));
        int arrow = 1;
        int end = points[0][1];
        for(int i=1;i<points.length;i++){
            if(points[i][0] > end){
                arrow++;
                end = points[i][1];
            }
        }
        return arrow;
    }
}
```

Question 19: Task Scheduler
https://leetcode.com/problems/task-scheduler/description/

```
class Solution {
    public int leastInterval(char[] tasks, int n) {
        HashMap<Character,Integer> map = new HashMap<>();
        for(char t: tasks){
            map.put(t,map.getOrDefault(t,0)+1);
        }
```

```
        PriorityQueue<Integer> pq = new
PriorityQueue(map.size(),Collections.reverseOrder());
        pq.addAll(map.values());

        int result = 0;
        while(!pq.isEmpty()){
            int time = 0;
            ArrayList<Integer> list = new ArrayList<>();
            for(int i=0;i<n+1;i++){
                if(!pq.isEmpty()){
                    list.add(pq.remove()-1);
                    time++;
                }
            }
            for(int x : list)
                if(x>0){
                    pq.add(x);
                }
                result +=pq.isEmpty()?time:n+1;
        }
        return result;
    }
}
```

Question 20: Merge in Between Linked List

https://leetcode.com/problems/merge-in-between-linked-lists/description/

```
/**
 * Definition for singly-linked list.
 * public class ListNode {
 *     int val;
 *     ListNode next;
 *     ListNode() {}
 *     ListNode(int val) { this.val = val; }
 *     ListNode(int val, ListNode next) { this.val = val; this.next =
next; }
 * }
```

```
 */
class Solution {
    public ListNode mergeInBetween(ListNode list1, int a, int b, ListNode
list2) {
        // Note: count start from 0

        // Define two variable left and right
        ListNode left = null;
        ListNode right = list1;

        // Our aim is: left should be point to (a-1)th node and right
should be point to (b+1)th node
        for(int i=0;i<=b;i++){
            if(i == (a-1)){
                left = right;
            }
            right = right.next;
        }
        // Adding left.next to list2
        left.next = list2;

        // Now we will reach at the last node of list2
        ListNode temp = list2;
        while(temp.next != null){
            temp = temp.next;
        }

        //Adding last node of list2 to the right
        temp.next = right;

        return list1;
    }
}
```

Question 21: Reverse Linked List

https://leetcode.com/problems/reverse-linked-list/description/

```
/**
 * Definition for singly-linked list.
 * public class ListNode {
 *     int val;
 *     ListNode next;
 *     ListNode() {}
 *     ListNode(int val) { this.val = val; }
 *     ListNode(int val, ListNode next) { this.val = val; this.next =
next; }
 * }
 */
class Solution {
    public ListNode reverseList(ListNode head) {
        // Iterative Approach
        ListNode prev = null;
        ListNode curr = head;
        ListNode next = null;
        while(curr != null){
            next = curr.next;
            curr.next = prev;
            prev = curr;
            curr = next;
        }
        return prev;

        // Recursive Approach
        // Base Case: if head is null or only one node in the list
        // if(head == null || head.next == null) return head;

        // ListNode last = reverseList(head.next); // Future head
        // head.next.next = head;
        // head.next = null;

        // return last;
    }
}
```

Question 22: Palindrome Linked List
https://leetcode.com/problems/palindrome-linked-list/description/

```
/**
 * Definition for singly-linked list.
 * public class ListNode {
 *     int val;
 *     ListNode next;
 *     ListNode() {}
 *     ListNode(int val) { this.val = val; }
 *     ListNode(int val, ListNode next) { this.val = val; this.next =
next; }
 * }
 */
// Approach 1 (Using extra space)
// class Solution{
//     public boolean isPalindrome(ListNode head){
//         ArrayList<Integer> list = new ArrayList<>();
//         ListNode curr = head;
//         while(curr != null){
//             list.add(curr.val);
//             curr = curr.next;
//         }
//         int i = 0;
//         int j = list.size()-1;

//         while(i<=j){
//             if(list.get(i) != list.get(j)){
//                 return false;
//             }
//             i++;
//             j--;
//         }
//         return true;
//     }
// }
// Approach 2 (Reversing the 2nd half of linked list)
// class Solution {
```

```
//     public boolean isPalindrome(ListNode head) {
//         ListNode mid = middle(head); // finding middle node
//         ListNode tail = reverse(mid); // reversing second half of
linked list
//         ListNode curr = head;

//         while(tail!=null){
//             if(curr.val != tail.val){
//                 return false;
//             }
//             tail = tail.next;
//             curr = curr.next;
//         }
//         return true;
//     }
//     // Function for finding the middle node
//     public ListNode middle(ListNode head){
//         ListNode slow = head;
//         ListNode fast = head;

//         while(fast!=null && fast.next!=null){
//             slow = slow.next;
//             fast = fast.next.next;
//         }

//         // if the number of nodes are odd then definitely fast pointer
will be at the last node
//         if(fast!=null){
//             return slow.next;
//         }
//         return slow;
//     }

//     // Function for finding reverse from the mid node
//     public ListNode reverse(ListNode head){
//         ListNode prev = null;
//         ListNode curr = head;
//         ListNode next = null;

//         while(curr!=null){
```

```
//             next = curr.next;
//             curr.next = prev;
//             prev = curr;
//             curr = next;
//         }
//         return prev;
//     }
// }

// Approach 3 (Optimized reversing half of linked list)
class Solution{
    public boolean isPalindrome(ListNode head){
        ListNode slow = head;
        ListNode fast = head;
        ListNode prev = null;

        //Finding middle node (While finding middle node, we are reversing
the first half of linked list)
        while(fast!=null && fast.next!=null){
            fast = fast.next.next;
            ListNode temp = slow.next;
            slow.next = prev;
            prev = slow;
            slow = temp;
        }
        // If the number of nodes are odd
        if(fast!=null){
            slow = slow.next;
        }
        // Now we compare
        while(prev!=null && slow!=null){
            if(prev.val != slow.val){
                return false;
            }
            prev = prev.next;
            slow = slow.next;
        }
        return true;
    }
}
```

Question 23: Reorder List
https://leetcode.com/problems/reorder-list/description/

```
/**
 * Definition for singly-linked list.
 * public class ListNode {
 *     int val;
 *     ListNode next;
 *     ListNode() {}
 *     ListNode(int val) { this.val = val; }
 *     ListNode(int val, ListNode next) { this.val = val; this.next =
next; }
 * }
 */
class Solution {
    public void reorderList(ListNode head) {
        //Finding middle node
        ListNode slow = head;
        ListNode fast = head;

        while(fast!=null && fast.next!=null){
            slow = slow.next;
            fast = fast.next.next;
        }
        ListNode tail = reverse(slow);

        ListNode curr = head;
        while(tail.next!=null){
            ListNode tempCurr = curr.next;
            curr.next = tail;
            ListNode tempTail = tail.next;
            tail.next = tempCurr;

            curr = tempCurr;
            tail = tempTail;
        }
    }
    public ListNode reverse(ListNode head){
        ListNode prev = null;
```

```
        ListNode curr = head;
        ListNode next = null;
        while(curr != null){
            next = curr.next;
            curr.next = prev;
            prev = curr;
            curr = next;
        }
        return prev;
    }
}
```

Question 24: Find the Duplicate Number

https://leetcode.com/problems/find-the-duplicate-number/description/

```
/*
Approach 1: Brute Force (TLE)
Time Complexity: O(n^2)
Space COmplexity: O(1)
*/
// class Solution {
//     public int findDuplicate(int[] nums) {
//         for(int i=0;i<nums.length;i++)
//             for(int j=i+1;j<nums.length;j++)
//                 if(nums[i] == nums[j]) return nums[i];
//         return -1;
//     }
// }

/*
Approach 2: Sort the array and check adjacent elements
Time Complexity: O(n*logn)
Space Complexity: O(1)
*/
// class Solution {
//     public int findDuplicate(int[] nums) {
//         Arrays.sort(nums);
```

```
//         for(int i=0;i<nums.length-1;i++)
//             if(nums[i] == nums[i+1]) return nums[i];
//         return -1;
//     }
// }
/*
Approach 3: Using HashSet to store the visited nodes
Time Complexity: O(n)
Space Complexity: O(n)
*/
// class Solution {
//     public int findDuplicate(int[] nums) {
//         HashSet<Integer> set = new HashSet<>();
//         for(int i=0;i<nums.length;i++){
//             if(set.contains(nums[i])){
//                 return nums[i];
//             }else{
//                 set.add(nums[i]);
//             }
//         }
//         return -1;
//     }
// }

/*
Approach 4: Using Boolean Array
Time Complexity: O(n)
Space Complexity: O(n)
*/
class Solution {
    public int findDuplicate(int[] nums) {
        int n = nums.length;
        boolean[] set = new boolean[n+1];
        for(int i=0;i<n;i++) {
            if(set[nums[i]]) return nums[i];
            set[nums[i]] = true;
        }
        return -1;
    }
}
```

Question 25: FInd All Duplicates in an Array
https://leetcode.com/problems/find-all-duplicates-in-an-array/description/

```
// Approach 1: Using HashSet
// class Solution {
//     public List<Integer> findDuplicates(int[] nums) {
//         HashSet<Integer> set = new HashSet<>();
//         ArrayList<Integer> list = new ArrayList<>();

//         for(int i=0;i<nums.length;i++){
//             if(set.contains(nums[i])){
//                 list.add(nums[i]);
//             }else{
//                 set.add(nums[i]);
//             }
//         }
//         return list;
//     }
// }

// Approach 2: Using HashMap
// class Solution{
//     public List<Integer> findDuplicates(int[] nums){
//         HashMap<Integer, Integer> map = new HashMap<>();
//         ArrayList<Integer> list = new ArrayList<>();

//         for(int i=0;i<nums.length;i++){
//             map.put(nums[i], map.getOrDefault(nums[i],0)+1);

//             if(map.get(nums[i]) == 2){
//                 list.add(nums[i]);
//             }
//         }
//         return list;
//     }
// }

// Approach 3: Using Sorting
```

```java
// class Solution{
//     public List<Integer> findDuplicates(int[] nums){
//         ArrayList<Integer> list = new ArrayList<>();
//         Arrays.sort(nums);

//         for(int i=0;i<nums.length-1;i++){
//             if(nums[i] == nums[i+1]){
//                 list.add(nums[i]);
//             }
//         }
//         return list;
//     }
// }

// Best Approach  Approach 4
/* Whenever it is mentioned that you have an array of length n and all the
elements are in the range [1,n] -->> then use numbers as an index (like
idx = num - 1); */

class Solution{
    public List<Integer> findDuplicates(int[] nums){
        ArrayList<Integer> list = new ArrayList<>();
        for(int i=0;i<nums.length;i++){
            int num = Math.abs(nums[i]);
            int idx = num - 1;

            if(nums[idx] < 0){
                list.add(num);
            }else{
                nums[idx] *= -1;
            }
        }
        return list;
    }
}
```

Question 26: First Missing Positive
https://leetcode.com/problems/first-missing-positive/description/

```
class Solution {
    public int firstMissingPositive(int[] nums) {
        int n = nums.length;
        boolean contains1 = false;

        // First we will check 1 is present in nums array
        for(int i=0;i<n;i++){
            if(nums[i] == 1){
                contains1 = true;
            }
            if(nums[i] <= 0 || nums[i] > n){ // Beacuse we are dealing
with [1,n] otherwise index will be invalid.
                nums[i] = 1;
            }
        }
        // If 1 is not present in nums array return 1 as an answer
        if(contains1 == false){
            return 1;
        }
        for(int i=0;i<n;i++){
            int num = Math.abs(nums[i]);
            int idx = num - 1;

            if(nums[idx] < 0) continue;

            nums[idx] *= -1;
        }
        for(int i=0;i<n;i++){
            if(nums[i] > 0){
                return i+1;
            }
        }
        return n+1;
    }
}
```

Question 27: Subarray Product Less Than k
https://leetcode.com/problems/subarray-product-less-than-k/description/

```
class Solution {
    public int numSubarrayProductLessThanK(int[] nums, int k) {
        //Corner case Since nums[i] = [1,1000] & k>=0
        if(k <= 1) return 0;

        // Simple Sliding Window Concept
        int n = nums.length;
        int i = 0;
        int j = 0;
        int count = 0;
        int p = 1;
        while(j < n){
            p *= nums[j];
            while(i <= j && p >= k){
                p /= nums[i];
                i++;
            }
            if(p < k){
                count += (j-i+1); // So (j-i+1) will give you the number
of subarray ending with j (note it also gives the window size)
            }
            j++;
        }
        return count;
    }
}
```

Question 28: Length of Longest Subarray With at Most K frequency
https://leetcode.com/problems/length-of-longest-subarray-with-at-most-k-frequency/description/

```
// class Solution {
//     public int maxSubarrayLength(int[] nums, int k) {

//         // Sliding Window Concept
//         HashMap<Integer, Integer> map = new HashMap<>();
//         int n = nums.length;
//         int i = 0;
//         int j = 0;
//         int res = 0;
//         while(j < n){
//             map.put(nums[j], map.getOrDefault(nums[j], 0)+1);

//             while(map.get(nums[j]) > k){
//                 map.put(nums[i], map.get(nums[i])-1);
//                 i++;
//             }

//             res = Math.max(res, (j-i+1));
//             j++;
//         }
//         return res;
//     }
// }

// Another Approach: Without using nested loop

class Solution{
    public int maxSubarrayLength(int[] nums, int k){
        HashMap<Integer, Integer> map = new HashMap<>();
        int res = 0;
        int n = nums.length;
        int i = 0;
        int j = 0;
        int culprit = 0; // culprit means guilty (Doshi)
        while(j < n){
            map.put(nums[j], map.getOrDefault(nums[j],0)+1);
            // if frequency of nums[j] immediate greater than k means
(k+1)
            if(map.get(nums[j]) == k+1){
                culprit++;
```

```
            }

            if(culprit > 0){
                map.put(nums[i], map.get(nums[i])-1);
                if(map.get(nums[i])==k){
                    culprit--;
                }
                i++;
            }
            if(culprit == 0){
                res = Math.max(res, (j-i+1));
            }
            j++;
        }
        return res;
    }
}
```

Question 29: Count Subarrays Where Max Element Appears At Least K Times

https://leetcode.com/problems/count-subarrays-where-max-element-appears-at-least-k-times/description/

```
class Solution {
    public long countSubarrays(int[] nums, int k) {
        // Finding max element of nums array
        int maxElement = Integer.MIN_VALUE;
        for(int i=0;i<nums.length;i++){
            maxElement = Math.max(maxElement, nums[i]);
        }

        // Sliding Window Concept
        int n = nums.length;
        int i = 0;
        int j = 0;
        int countMaxElement = 0;
        long res = 0;
        while(j < n){
```

```
            if(nums[j]==maxElement){
                countMaxElement++;
            }
            while(countMaxElement >= k){
                res += (n-j); // Main thing

                if(nums[i]==maxElement){
                    countMaxElement--;
                }
                i++;
            }
            j++;
        }
        return res;

    }
}
```

Question 30: Subarrays With K Different Integers

https://leetcode.com/problems/subarrays-with-k-different-integers/description/

```
class Solution {
    public int subarraysWithKDistinct(int[] nums, int k) {
       int res1 = slidingWindow(nums, k);
       int res2 = slidingWindow(nums, k-1);
       return (res1 - res2);
    }
// This function will return the total number of subarrays with <=k
different integer
    public int slidingWindow(int[] nums, int k){
        int n = nums.length;
        int i = 0;
        int j = 0;
        int count = 0;
        HashMap<Integer, Integer> map = new HashMap<>();
        while(j < n){
            map.put(nums[j], map.getOrDefault(nums[j],0)+1);
```

```
            while(map.size() > k){
                //shrink the window
                map.put(nums[i], map.get(nums[i])-1);
                if(map.get(nums[i]) == 0){
                    map.remove(nums[i]);
                }
                i++;
            }
            count += (j-i+1);
            j++;
        }
        return count;
    }
}

/*
    Note: slidingWindow(nums, k) >> First we are calculating the total
number of subarrays with <=k different integer (res1)
    And then slidingWindow(nums, k-1) >> We are calculating the total
number of subarrays with <=(k-1) different integer (res2)

    Finally we are subtracting (res1 - res2)
    In this way, we are getting the total number of subarrays with exactly
k different integers.
 */
```

Question 31: Count Subarrays With Fixed Bounds

https://leetcode.com/problems/count-subarrays-with-fixed-bounds/description/

```
class Solution {
    public long countSubarrays(int[] nums, int minK, int maxK) {
        long res = 0;
        int minIndex = -1, maxIndex = -1;
        int start = 0;
        for (int i = 0; i < nums.length; i++) {
            if (nums[i] < minK || nums[i] > maxK) {
                minIndex = maxIndex = -1;
```

```
                start = i + 1;
            }

            if (nums[i] == minK){
                minIndex = i;
            }
            if (nums[i] == maxK){
                maxIndex = i;
            }

            res = res +  Math.max(0, Math.min(minIndex, maxIndex) - start
+ 1);
        }

        return res;
    }
}
```

That’s it

Thank You