Coding practice :
(Just for revision for future)

Qno 1 : Given an array of integers `nums` and an integer `target`, return *indices of the two numbers such that they add up to `target`*.

You may assume that each input would have *exactly* one solution, and you may not use the *same* element twice.

You can return the answer in any order.

Input: nums = [2,7,11,15], target = 9

Output: [0,1]

Output: Because nums[0] + nums[1] == 9, we return [0, 1].

Solution:

```
class Solution {
    public int[] twoSum(int[] nums, int target) {
        Map<Integer,Integer> map=new HashMap<>();
        for(int i=0;i<nums.length;i++)
        {
            int complement=target-nums[i];
            if(map.containsKey(complement))
            {
                return new int[] {map.get(complement),i};
            }
            map.put(nums[i],i);
        }
        throw new IllegalArgumentException("No solution");
    }
}
```

Time complexity: O(n)
Space Complexity: O(n)

Map.get() : The get () method of Map interface in Java is used to retrieve or fetch the value mapped by a particular key mentioned in the parameter. It returns NULL when the map contains no such mapping for the key.
Map.containsKey() : The java.util.Map.containsKey () method is used to check whether a particular key is being mapped into the Map or not. It takes the key element as a parameter and returns True if that element is mapped in the map.

[Reverse Integer]

Given a signed 32-bit integer `x`, return `x` *with its digits reversed*. If reversing `x` causes the value to go outside the signed 32-bit integer range `[-2^31, 2^(31 - 1)]`, then return `0`.

Assume the environment does not allow you to store 64-bit integers (signed or unsigned).

Example 1:

```
Input: x = 123
Output: 321
```

```
Solution :
```

```
class Solution {

    public int reverse(int x)
    {

        if(x==0)
            return 0;
        long reverse=0;
        while(x!=0)
        {
            int last_digit = x%10;
            reverse = reverse*10+last_digit;
            if(reverse > Integer.MAX_VALUE || reverse < Integer.MIN_VALUE)
{
        return 0;
```

```
        }
            x=x/10;
        }


            return (int)reverse;


    }
}
```

2. Add Two Numbers

Solution :

```
class Solution {
public ListNode addTwoNumbers(ListNode l1, ListNode l2) {
    ListNode dummyHead = new ListNode(0);
    ListNode p = l1, q = l2, curr = dummyHead;
    int carry = 0;
    while (p != null || q != null) {
        int x = (p != null) ? p.val : 0;
        int y = (q != null) ? q.val : 0;
        int sum = carry + x + y;
```

```
            carry = sum / 10;
            curr.next = new ListNode(sum % 10);
            curr = curr.next;
            if (p != null) p = p.next;
            if (q != null) q = q.next;
        }
        if (carry > 0) {
            curr.next = new ListNode(carry);
        }
        return dummyHead.next;
    }
}
```

[3Sum]

Solution :

```
class Solution {
 public  List<List<Integer>> threeSum(int[] nums) {
       Set<List<Integer>> res  = new HashSet<>();
       if(nums.length==0) return new ArrayList<>(res);
       Arrays.sort(nums);
       for(int i=0; i<nums.length-2;i++){
          int j =i+1;
         int  k = nums.length-1;
          while(j<k){
             int sum =nums[j]+nums[k];
             if(sum == -nums[i]){
              res.add(Arrays.asList(nums[i],nums[j],nums[k]));
```

```
            j++; k--;
           }
           else if ( sum >-nums[i]) k--;
           else if (sum<-nums[i]) j++;
        }

      }
      return new ArrayList<>(res);

   }
}
```

[3. Longest Substring Without Repeating Characters]

```
class Solution {
   public int lengthOfLongestSubstring(String s) {
     int a=0;
      int b=0;
      int max=0;
      HashSet<Character>  hashset = new HashSet<Character>();
      int len =s.length();
      while(b<len)
      {
         if(! hashset.contains(s.charAt(b)))
         {
            hashset.add(s.charAt(b));
            max=Math.max(max,hashset.size());
            b++;
         }
         else
         {
            hashset.remove(s.charAt(a));
            a++;
         }
      }
      return max;

}
```

```
}
```

Q : Multiply Strings

Solution :

```
import java.math.BigInteger;

class Solution {

    public String multiply(String num1, String num2) {

        BigInteger m1=new BigInteger(num1);
        BigInteger m2=new BigInteger(num2);
        BigInteger res=m1.multiply(m2);

        return res.toString();

}


    }
```

Q : 771. Jewels and Stones

Solution : (without using hashmap) O(n)

```
class Solution {
   public int numJewelsInStones(String jewels, String stones) {
      int count=0;
      char[] sa=stones.toCharArray();
      for(int i=0;i<sa.length;i++)
      {
```

```
            if(jewels.indexOf(sa[i]) != -1)
            {
                count++;
            }
        }
        return count;
    }
}

Solution : using HashMap  O(N)

class Solution {
    public int numJewelsInStones(String jewels, String stones) {

        HashMap<Character,Integer> hash=new HashMap<Character,Integer>();
        int key=0;
        int res=0;
        for(int i=0;i<stones.length();i++)
        {
            if(hash.containsKey(stones.charAt(i)))
            {
            hash.put(stones.charAt(i),hash.get(stones.charAt(i))+1);
            }
            else
            {
                hash.put(stones.charAt(i),1);
            }
        }
        for(int j=0;j<jewels.length();j++)
        {
            if(hash.containsKey(jewels.charAt(j)))
            {
                res=res+hash.get(jewels.charAt(j));
            }
        }
        return res;




    }
}
```

Q :[5. Longest Palindromic Substring]

Solution:

```
public String longestPalindrome(String s) {

    if (s == null || s.length() < 1) return "";
    int start = 0, end = 0;

    for (int i = 0; i < s.length(); i++) {
        int len1 = expandAroundCenter(s, i, i);
        int len2 = expandAroundCenter(s, i, i + 1);  // babad
        int len = Math.max(len1, len2);
        if (len > end - start) {
            start = i - (len - 1) / 2;
            end = i + len / 2;
        }
    }

    return s.substring(start, end + 1);
}

private int expandAroundCenter(String s, int left, int right) {
    int L = left, R = right;
    while (L >= 0 && R < s.length() && s.charAt(L) == s.charAt(R)) {

        L--;
        R++;

    }
    return R - L - 1;
}
```

Q : Median of Two Sorted Arrays

Solution :

```
class Solution {
    public double findMedianSortedArrays(int[] nums1, int[] nums2) {

        int a=0;
        int b=0;
        int l1=nums1.length;
        int l2=nums2.length;
        int[] merge=new int[l1+l2];
        int k=0;
        while(a<l1 && b<l2)
        {
            if(nums1[a]<nums2[b])
            {
                merge[k]=nums1[a];
                k++;
                a++;
            }
            else
            {
                merge[k]=nums2[b];
                k++;
                b++;
            }

        }
        while(a<l1)
        {
            merge[k]=nums1[a];
            a++;
            k++;
        }
        while(b<l2)
        {
            merge[k]=nums2[b];
            b++;
```

```
            k++;
        }
        double mid=0.0;
        for(int r: merge)
        {
            System.out.println(r);
        }
        if(merge.length%2==0)
        {
            mid=(merge[(merge.length/2)-1]+merge[(merge.length/2)])/2.0;
        }
        else
        {
            mid=merge[(merge.length)/2];
        }
        return mid;


    }
}
```

Q: [Unique Email Addresses]

Solution :

```
class Solution {
    public int numUniqueEmails(String[] emails) {

        if(emails == null || emails.length == 0)
            return -1;

        Set<String> set = new HashSet<>();
        for(String email : emails){
            StringBuilder sb=new StringBuilder();
            int ln=0;
            int dn=0;
            for(char c: email.toCharArray())
            {
```

```
            if(c=='@')
            {
               ln=1;
               dn=1;
               sb.append(c);
            }
            if(ln ==0)
            {
            if(c=='.')
            {
               continue;
            }
            else if(c=='+')
            {
               ln=1;
               continue;
            }
             else
            {
                  sb.append(c);
            }

            }
            if(dn==1)
            {
               sb.append(c);
            }
         }
         set.add(sb.toString());
      }
      return set.size();
   }
}
```

Q: Reverse Linked List - LeetCode

Solution:

```
class Solution {
    public ListNode reverseList(ListNode head) {
        ListNode ln=head;
        ListNode lp=null;
        while(ln != null)
        {

            ListNode next1=ln.next;
            ln.next=lp;
            lp=ln;
            ln=next1;

        }
        return lp;

    }
}
```

Q: 368. Largest Divisible Subset

Sol: It is done through dynamic programming; (somya)

```
class Solution {
    public List<Integer> largestDivisibleSubset(int[] nums)
    {
```

```
        if(nums.length == 0) return new ArrayList<>();
        List<Integer> res=new ArrayList<>();
        int[] dp=new int[nums.length];
       Arrays.sort(nums);
        int max=1;
        dp[0]=1;
        for(int i=1;i<nums.length;i++)
        {
           dp[i]=1;
           for(int j=0;j<i;j++)
           {
              if(nums[i]%nums[j]==0)
              {
               dp[i]=Math.max(dp[i],dp[j]+1);
               max= Math.max(max,dp[i]);
              }
           }}
           int maxElement=-1;
           for(int k=nums.length-1;k>=0;k--)
           {
           if(max==dp[k])
           {
              if(maxElement==-1)
              {
                 res.add(nums[k]);
                 maxElement=nums[k];
                 max--;
              }
              else if(maxElement% nums[k]==0)
              {
                 res.add(nums[k]);
                 maxElement=nums[k];
                 max--;
              }
           }
           }
        return res;
    }
}
```

Solution : (shubham)

```
class Solution {
   public List<Integer> largestDivisibleSubset(int[] nums) {
      int[] maxsub = new int[nums.length];
      int len=maxsub.length;
      StringBuilder sb=new StringBuilder();
      int max=0;
      while(len-1>=0)
      {
         maxsub[len-1]=1;
         len--;
      }
      for(int j=0;j<nums.length;j++)
      {
         for(int k=0;k<j;k++)
         {
            int e=maxsub[k];
            if(nums[j]%nums[k]==0)
            {
               if(maxsub[k]+1 > maxsub[j])
               {
                  maxsub[j]=maxsub[j]+1;
                  if(maxsub[j]>max)
                  {
                     max=maxsub[j];
                  }
               }
            }

         }
      }

      List<Integer> ls=new ArrayList<Integer>();
     int prev = -1;
      for(int i=maxsub.length-1;i>=0;i--)
      {
         if(maxsub[i]==max &&  (prev%nums[i]==0 || prev==-1))
         {
            ls.add(nums[i]);
            max -=1;
            prev = nums[i];
```

```
            }
        }

        return ls;

    }
}
```

Q: [75. Sort Colors]

Solution :

```
class Solution {
    public void sortColors(int[] nums) {
        int j=-1;
        for(int i=0; i < nums.length; i++){
            if(nums[i] == 0){
                int tmp = nums[j+1];
                nums[j+1] = nums[i];
                nums[i] = tmp;
                ++j;
            }
        }
        for(int i=j+1; i < nums.length; i++){
            if(nums[i] == 1){
                int tmp = nums[j+1];
                nums[j+1] = nums[i];
                nums[i] = tmp;
                ++j;
            }
        }
    }
}
```

Q : [Search in Rotated Sorted Array]

Solution :

```
class Solution {

    public int search(int[] nums, int target) {


        if(nums.length==0|| nums ==null)  return -1;

        int l=0;
        int r=nums.length-1;

        while(l<r)
        {

            int mid=l+(r-l)/2 ;

            if(nums[mid]>nums[r])
            {

                l=mid+1;

            }
            else
            {

                r=mid;

            }
        }

        System.out.println(nums[l]);




         int start=l;
```

```
        int left=0;
        int right=nums.length-1;

      if( target >= nums[start] && target <= nums[right] )
      {
         left=start ;
      }
      else
      {

         right=start ;

      }



      while(left<=right)
      {

         int mid=left + (right-left)/2 ;
        if(target==nums[mid])
        {
           return mid;
        }
         else if(target>nums[mid])
         {
            left=mid+1 ;
         }
         else
         {
            right=mid-1;
         }
      }


      return -1;

   }
}
```

Q : [Palindrome Number]

Solution :

```
class Solution {
    public boolean isPalindrome(int x) {
        int sx=x;
        int palin=0;
        if(x<0)
        {
            return false ;
        }
        else if(x%10 == 0 && x/10 == 0 )
        {
            return true;
        }
        else
        {
            while(x>0) //121
            {
                int d=x%10;
                palin=palin*10+d;
                x=x/10;
            }

        }
        if(palin==sx)
        {
            return true;
        }
        else
        {
            return false;
        }

    }
}
```

Q : Container With Most Water

Solution :

```
class Solution {
    public int maxArea(int[] height) {

        int l=0;
        int r=height.length-1;
        int maxval=0;
        int h=0;
        while(l<r)
        {


            if(height[l]<height[r])
            {
                h=height[l];
            }
            else
            {

                h=height[r];
            }
                int val=h * (r-l);

                if(val>maxval)
                {
                    maxval=val;
                }
            if(height[l]<height[r])
            {
                l++;
            }
            else
            {
                r--;
            }


        }
```

```
        return maxval;

    }
}
```

Q : [14. Longest Common Prefix]

Solution :

```
class Solution {
    public String longestCommonPrefix(String[] strs) {

        if(strs.length==0) return "";

        String test=strs[0];
        for(int i=1;i<strs.length;i++)
        {
            while(strs[i].indexOf(test) != 0)
            {
                test=test.substring(0,test.length()-1);
            }
        }
        return test;

    }
}
```

Q : [17. Letter Combinations of a Phone Number]

Solution :

```
class Solution {
    public List<String> letterCombinations(String digits) {
```

```
        LinkedList<String> ll=new LinkedList();

        if(digits.length()==0)
        {
           return ll;
        }

        ll.add("");

        String[] charmap= new String[]
{"0","1","abc","def","ghi","jkl","mno","pqrs","tuv","wxyz"};

        for(int d=0;d<digits.length();d++)
        {
            int digit=Character.getNumericValue(digits.charAt(d));
            String ds=charmap[digit];
           while(ll.peek().length()==d)
           {
            String pp=ll.remove();
           for(char c:ds.toCharArray())
           {
              ll.add(pp+c);
           }


        }
        }
        return ll;
    }
}
```

Q: [Number of Islands]

Solution :

```
class Solution {
    public int numIslands(char[][] grid) {
        //boolean[][] visited = new boolean[grid.length][grid[0].length];
```

```
        int count=0;
        for(int i=0;i<grid.length;i++)
        {
           for(int j=0  ;  j<grid[0].length  ;  j++)
           {
              if(Character.getNumericValue(grid[i][j]) == 1 )
              {
                  count++;
                  drawtreeforcomp(grid , i , j );

              }
           }
        }
        return count ;
    }
    public static void drawtreeforcomp(char[][] grid , int i , int j)
    {

        if(i<0 || j<0 || i == grid.length || j ==  grid[0].length  ||
Character.getNumericValue(grid[i][j]) == 0 )
        {
                  return ;
        }



        grid[i][j]='0';
        drawtreeforcomp(grid, i-1 , j );
        drawtreeforcomp(grid, i+1, j );
        drawtreeforcomp(grid, i , j-1 );
        drawtreeforcomp(grid, i , j+1 );

    }
}
```

Q: Q. 121. Best Time to Buy and Sell Stock

Solution :

```
class Solution {
    public int maxProfit(int[] prices) {

        int maxprofit=Integer.MIN_VALUE;
        int min=prices[0];



        for(int i=0;i<prices.length;i++)
        {
            if(prices[i]<min)
            {
                min=prices[i];
            }
            if(prices[i]-min>maxprofit)
            {
                maxprofit=prices[i]-min;
            }
        }
        return maxprofit;


}
}
```

Q: [Find First and Last Position of Element in Sorted Array]

Solution :

```
class Solution {
    public int[] searchRange(int[] nums, int target) {
   int[] pos = new int[2];

   Arrays.fill(pos,-1);
   Firstpos(nums,pos,target);
   lastpos(nums,pos,target);
   return pos;

}
public void Firstpos(int[] nums,int[] pos,int target){
    int lo = 0;
    int hi = nums.length - 1;
      while(lo <= hi){
       int mid = lo + (hi - lo)/2;
       if(nums[mid] == target){
           pos[0]  = mid;
           hi = mid - 1;
       }
      else if(nums[mid]  < target){
           lo = mid + 1;
       }
       else{
           hi = mid - 1;
       }
   }

}

public void lastpos(int[] nums, int[] pos, int target){
    int lo = 0;
    int hi = nums.length - 1;
      while(lo <= hi){
       int mid = lo + (hi - lo)/2;
       if(nums[mid] == target){
           pos[1]  = mid;
           lo = mid + 1;
```

```
        }
        else if(nums[mid]  < target){
            lo = mid + 1;
        }
        else{
            hi = mid - 1;
        }
    }

}
}
```

Q : [322. Coin Change]

Solution :

```
class Solution {
    public int coinChange(int[] coins, int amount) {
        int[] dp=new int[amount+1];  // _ _ _ _ _ _ _ _ _ _ _ _
        Arrays.fill(dp, amount+1);
        dp[0]=0;
        for(int i=0;i<= amount;i++)
        {
            for(int j=0;j<coins.length;j++)
            {
                if(coins[j] <= i)
                {
                    dp[i]=Math.min(dp[i],1+dp[i-coins[j]]);
                }
            }

        }
        return dp[amount]>amount ? -1 : dp[amount];

    }
}
```

Q : [N-ary Tree Postorder Traversal]

Solution :

```
/*
// Definition for a Node.
class Node {
    public int val;
    public List<Node> children;

    public Node() {}

    public Node(int _val) {
        val = _val;
    }

    public Node(int _val, List<Node> _children) {
        val = _val;
        children = _children;
    }
};
*/

class Solution {

    public List<Integer> postorder(Node root) {

        LinkedList<Node> ll= new LinkedList<Node>();
        LinkedList<Integer> output=new LinkedList<Integer>();

        if(root == null)
        {
            return output;
        }
```

```
        ll.add(root);

        while(ll.isEmpty() == false)
        {

            Node link=ll.pollLast();
            output.addFirst(link.val);
            for(Node nn: link.children)
            {

                ll.add(nn);

            }
        }


        return output;

    }
}
```

Q : [Binary Tree Inorder Traversal]

Solution :

```
public class Solution {

    public List < Integer > inorderTraversal(TreeNode root) {
        List < Integer > res = new ArrayList < > ();
        Stack < TreeNode > stack = new Stack < > ();
        TreeNode curr = root;
        while (curr != null || !stack.isEmpty()) {
            while (curr != null) {
                stack.push(curr) ;
                curr = curr.left ;
            }
```

```
            curr = stack.pop() ;
            res.add(curr.val) ;
            curr = curr.right ;
        }
        return res ;

    }
}
```

Q: [Backspace String Compare]

Solution :

```
class Solution {
    public boolean backspaceCompare(String S, String T)
{
        return build(S).equals(build(T));
    }

    public String build(String S) {
        Stack<Character> ans = new Stack();
        for (char c: S.toCharArray()) {
            if (c != '#')
                ans.push(c);
            else if (!ans.empty())
                ans.pop();
        }
        return String.valueOf(ans);
    }
}
```

Q: 806. Number of Lines To Write String

Sol:

```
class Solution {
    public int[] numberOfLines(int[] widths, String S) {
        int lines = 1, width = 0;
        for (char c: S.toCharArray()) {
            int w = widths[c - 'a'];
            width += w;
            if (width > 100) {
                lines++;
                width = w;
            }
        }

        return new int[]{lines, width};
    }
}
```

Q: Binary Tree Tilt

Solution :

```
class Solution {
    private int totalTilt = 0;

    protected int valueSum(TreeNode node) {
        if (node == null)
            return 0;

        int leftSum = this.valueSum(node.left);
        int rightSum = this.valueSum(node.right);
        int tilt = Math.abs(leftSum - rightSum);
        this.totalTilt += tilt;
```

```
        // return the sum of values starting from this node.
        return node.val + leftSum + rightSum;
    }

    public int findTilt(TreeNode root) {
        this.totalTilt = 0;
        this.valueSum(root);
        return this.totalTilt;
    }
}
```

Q: [Leetcode #13 Roman to Integer]

Sol: (somya)

- **Java HashMap contains values based on the key.**
- **Java HashMap contains only unique keys.**
- **Java HashMap may have one null key and multiple null values.**
- **Java HashMap is non synchronized.**
- **The get() method of Map interface in Java is used to retrieve or fetch the value mapped by a particular key mentioned in the parameter.**
- **`put(K key, V value)`**
- **Put method has two arguments, key and value where key is the left argument and value is the corresponding value of the key in the map.**
- **Put method return previous value associated with the key if present, else return -1.**

```
class Solution
{
    public int romanToInt(String s) {
        Map<Character,Integer> map = new HashMap();

        map.put('I',1);
        map.put('V',5);
        map.put('X',10);
        map.put('L',50);
        map.put('C',100);
        map.put('D',500);
        map.put('M',1000);
        char[] chars = s.toCharArray();
        int sum = 0;
        for(int i=0;i<chars.length-1;i++){
            int a = map.get(chars[i]);
            int b = map.get(chars[i+1]);
            sum += a<b ? -a : a;
        }
        sum+=map.get(chars[chars.length-1]);
        return sum;
    }
}
```

Solution (shubham):

```
class Solution {
    public int romanToInt(String s) {
        HashMap<Character, Integer> hs=new HashMap<>();

        hs.put('I',1);
        hs.put('V',5);
        hs.put('X',10);
        hs.put('L',50);
        hs.put('C',100);
        hs.put('D',500);
        hs.put('M',1000);
```

```
        int sum=0;
        for(int i=0;i<s.length()-1;i++)
        {
           int val= hs.get(s.charAt(i))<hs.get(s.charAt(i+1)) ?
              -hs.get(s.charAt(i)) :
              hs.get(s.charAt(i)) ;
           sum=sum+val;

        }
        sum=sum+hs.get(s.charAt(s.length()-1));
        return sum;

    }
}
```

Q : [39. Combination Sum]

Solution :

```
class Solution {
    public List<List<Integer>> combinationSum(int[] candidates, int target) {

        Set<List<Integer>> result = new HashSet<>();
        combinationSum(candidates, target, new ArrayList<>(), result);
        return new ArrayList<>(result);

    }

public void combinationSum(int[] candidates,int target , ArrayList list ,
Set<List<Integer>> res )
   {
    if(target<0)
    {
        return ;
    }
    if(target==0)
```

```
    {
        Collections.sort(list);
        res.add(list);

    }


    for(int candidate : candidates)
    {

        ArrayList<Integer> da=new ArrayList<Integer>(list);
        da.add(candidate);

        combinationSum(candidates , target-candidate ,  da , res);

    }


   }
}
```

Q : [Permutations]

Solution :

```
class Solution {
    public List<List<Integer>> permute(int[] nums) {

        List<List<Integer>> totalpermutation = new ArrayList<>();
        List<Integer> permute = new ArrayList<>();
        recursion(totalpermutation,permute,nums);
        return totalpermutation;


    }
```

```
   private void recursion(List<List<Integer>> total, List<Integer> permute, int[]
nums)
   {


      if(permute.size()==nums.length)
      {
         total.add(new ArrayList<Integer> (permute));
         return ;
      }


      for(int i=0;i<nums.length;i++)
      {
         if(!permute.contains(nums[i]))
         {
            permute.add(nums[i]);
            recursion(total, permute, nums);
            permute.remove(permute.size()-1);
         }
         else
         {
            continue;
         }
      }


   }


}
```

Q : [Submissions - Next Permutation]

Solution :

```
class Solution {
```

```
    public void nextPermutation(int[] nums) {

        int dec=0;

        int p=nums.length-2;

        while(p>=0 && nums[p+1] <= nums[p])
        {
            p--;
        }
        if(p>=0)
        {
            int b=nums.length-1;
            while(nums[b]<=nums[p])
            {
                b--;
            }

            int dv=nums[p];
            nums[p]=nums[b];
            nums[b]=dv;
        }
        else
        {
            Arrays.sort(nums);
        }
        int pp=p+1;
        int lp=nums.length-1;
        Arrays.sort(nums,pp,lp+1);

}
}
```

Q : [Group Anagrams]

Solution :

```
class Solution {

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


        if( strs.length==0)
        {
            return new ArrayList();
        }

       Map<String, List > ans = new HashMap<String, List>();
        for(String s: strs)
        {
            char[] sa = s.toCharArray();
            Arrays.sort(sa);
            String key=String.valueOf(sa);

            if(! ans.containsKey(key))
            {
                ans.put(key,new ArrayList());
            }

                ans.get(key).add(s);




        }
        return new ArrayList(ans.values());
```

```
    }
}
```

Q : [Wildcard Matching]

Solution :

```
class Solution {
    public boolean isMatch(String str, String pattern) {
    boolean[][] dp = new boolean[pattern.length() + 1][str.length() + 1];
                    for(int i = dp.length - 1; i >= 0 ;i--) {

                                for(int j = dp[0].length - 1; j >= 0; j--) {

                                            if(i == dp.length - 1 && j == dp[0].length - 1) {

                                                        dp[i][j] = true;

                                            }else if(i == dp.length - 1) {

                                                        dp[i][j] = false;

                                            }else if(j == dp[0].length - 1) {

                                                        if(pattern.charAt(i) == '*') {

                                                                    dp[i][j] = dp[i + 1][j];

                                                        }
                                            }
                        else {

                                                        if(pattern.charAt(i) == '?') {

                                                                    dp[i][j] = dp[i + 1][j + 1];

                                                        }
                                    else if(pattern.charAt(i) == '*') {
```

```
                                        dp[i][j] = dp[i][j + 1] || dp[i + 1][j];

                                    }
                else if(pattern.charAt(i) == str.charAt(j)) {

                                        dp[i][j] = dp[i + 1][j + 1];

                                    }
                else {

                                        dp[i][j] = false;

                                    }
                            }
                        }
                    }

                    return (dp[0][0]);
            }
}
```

Q : [6. ZigZag Conversion]

Sol:

```
class Solution {

    public String convert(String s, int numRows) {

        int length = s.length();

        if(numRows > length || numRows <=1)
        {
            return s;
        }
```

```
        char[] zigzag =new char[length];
        int c= 0 ;


        int interval = 2*numRows - 2 ;

        for(int i=0 ; i<numRows ;i++)
        {
           int step = interval - 2*i;

           for(int j= i ; j<length ; j +=interval)
           {
              zigzag[c] = s.charAt(j);
              c++;

              if(step > 0 && step< interval && j+step < length)
              {
                 zigzag[c] = s.charAt(j+step);
                 c++;
              }
           }
        }
        return new String(zigzag);
     }
}
```

Q :[PepCoding Login Page]
Basic recursion

Solution :

```
public class Main {
   public static void main(String[] args) throws Exception {
```

```
        Scanner sc=new Scanner(System.in);
        int num=sc.nextInt();
        printIncreasing(num);
    }

    public static void printIncreasing(int n){
        if(n==0)
        {
            return;
        }
        printIncreasing(n-1);
        System.out.println(n);                    // isme print backtracking ke time pr hoga

    }

}
```

Q: [PepCoding | Print Increasing Decreasing]

Sol:

```
public class Main {

    public static void main(String[] args) throws Exception {
        Scanner sc=new Scanner(System.in);
        int num=sc.nextInt();
        pdi(num);
    }

    public static void pdi(int n){
        if(n==0)
        return;
        System.out.println(n);              // ye jate time hi print hoga
        pdi(n-1);
        System.out.println(n);              // ye backtrack ke time print hoga
    }
}
```

Q: PepCoding | Factorial

Sol:

```
public class Main {

   public static void main(String[] args) throws Exception {
      Scanner sc=new Scanner(System.in);
      int num=sc.nextInt();
      int f=factorial(num);
      System.out.println(f);
   }

   public static int factorial(int n){
      if(n==1)
      return 1;
      int fnminus1 = factorial(n-1);
      int fact=n*fnminus1;
      return fact;
   }

}
```

Q: PepCoding | Power-linear

Sol:

```
import java.io.*;
import java.util.*;
public class Main {

   public static void main(String[] args) throws Exception {
      Scanner sc=new Scanner(System.in);
```

```
        int x=sc.nextInt();
        int n=sc.nextInt();
        int f=power(x,n);
        System.out.println(f);
    }

    public static int power(int x, int n)
    {
    if(n==0)
    return 1;
    int pnminus1 = power(x,n-1);
    int powr=x*pnminus1;
    return powr;
    }

}
```

Q: [14. Longest Common Prefix]

Sol:

```
class Solution {
    public String longestCommonPrefix(String[] strs) {
      if(strs.length==0||strs==null) return "";
       for(int i=0;i<strs[0].length();i++)
       {
          char firstelementchar=strs[0].charAt(i);
          for(int j=1;j<strs.length;j++)
          {
             if(i==strs[j].length() || strs[j].charAt(i)!=firstelementchar)
                return strs[0].substring(0,i);
          }}
      return strs[0];
    }}
```

Q : Remove Duplicates from Sorted Array

Solution :

```
class Solution {

    public int removeDuplicates(int[] nums) {



        int p1=0;

        for(int p2=1 ; p2< nums.length ;p2++ )
        {
            if(nums[p1]!=nums[p2])
            {
                p1++;
                nums[p1]=nums[p2];
            }
        }

        return p1+1;

    }
}
```

Q : Valid Sudoku

Solution :

```
class Solution {
```

```
   /* Return Value: The function returns True if the element is not present in the
HashSet otherwise False if the element is already present in the HashSet.
True => defines that we can add value in hashset
False => defines that we cant add value in hashset  */

   public boolean isValidSudoku(char[][] board) {

      HashSet<String> hs=new HashSet();


      for(int i=0;i<board.length;i++)
      {
         for(int j=0;j<board[0].length;j++)
         {
            char current=board[i][j] ;
            if(current != '.')
            {

        if( ! hs.add(current + "rows" + i) ||
              ! hs.add(current + "column" + j) ||
                ! hs.add(current + "box" + i/3+ "-" + j/3 ) )
            {
               return false;
            }
         }

      }
      }
      return true;

   }
}
```

Q : [Remove Duplicates from Sorted List]

Solution :

```
/**
 * 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 deleteDuplicates(ListNode head) {

        ListNode current = head ;

        while(current != null && current.next != null)
        {

            if(current.next.val == current.val)
            {

                current.next=current.next.next ;

            }
            else
            {

                current = current.next;

            }

        }

        return head;




    }

}
```

Q : https://leetcode.com/problems/path-sum/

Solution :

Q : [Leetcode-First Bad Version]

Solution :

```
/* The isBadVersion API is defined in the parent class VersionControl.
      boolean isBadVersion(int version); */

public class Solution extends VersionControl {
    public int firstBadVersion(int n) {


        int l= 1 ;
        int r= n ;
        boolean target= false;

        while(l<r)
        {
            int m = l+(r-l)/2  ;
            target = isBadVersion(m);
            if(target == true)
            {
                r=m;
            }
            else
            {
                l=m+1;
            }


        }
        return l;

    }}
```

Q : [Longest Continuous Increasing Subsequence]

Solution 1 : (sliding window technique)

```
class Solution {

    public int findLengthOfLCIS(int[] nums) {

        int s=0;

        int result=0;

        for(int i=0;i<nums.length;i++)
        {

            if(i>0 && nums[i-1]>=nums[i])  s=i;

            result=Math.max(result,i-s+1);


        }
        return result;

    }
}
```

Solution 2 : (easy to understand)

```
class Solution {

    public int findLengthOfLCIS(int[] nums) {

        int count = 0;

        int result = 0;

        for (int i=0; i<nums.length; i++){

            if (i == 0){count++;}

            else{
```

```
            if (nums[i] > nums[i-1]){
                count++;
            }else{
                result = Math.max(result,count);
                count = 1;
            }
        }
    }

    return Math.max(result,count);
  }
}
```

Q : [Leetcode #78 Subsets]

Solution :

```
class Solution {

  public List<List<Integer>> subsets(int[] nums) {

    List<List<Integer>> subsets = new ArrayList<>();
```

```
        generatesubsets(0 , nums , new ArrayList<>() , subsets );

        return subsets;

    }


  public void generatesubsets(int index , int[] nums , List<Integer> current ,
List<List<Integer>> subsets)
  {


      subsets.add(new ArrayList<>(current));

      for(int i=index; i<nums.length ;i++)
      {
          current.add(nums[i]);

          generatesubsets(i+1 , nums, current , subsets);

          current.remove(current.size()-1) ;
```

```
    }}}
```

Q : Climbing Stairs

Solution :

```
public class Solution {
    public int climbStairs(int n) {
        if (n == 1) {
            return 1;
        }
        int[] dp = new int[n + 1];
        dp[1] = 1;
        dp[2] = 2;
        for (int i = 3; i <= n; i++) {
            dp[i] = dp[i - 1] + dp[i - 2];
        }
        return dp[n];
    }
}
```

Q: Palindrome Partitioning

Solution :

```
class Solution {
    List<List<String>> answer = new ArrayList<>();
    LinkedList<String> result = new LinkedList<>();
    public List<List<String>> partition(String s) {
        backtrack(s,0);
        return answer;
    }

public void backtrack(String ss , int start)
{
     if(start == ss.length()){
            answer.add(new ArrayList<>(result));
            return;
        }
```

```
    for(int i=start;i<ss.length();i++)
    {
        if(ispalin(ss,start ,i ))
        {
            result.add(ss.substring(start,i+1));
           backtrack(ss,i+1);
            result.removeLast();

        }
    }


}
    public boolean ispalin(String ss, int s , int e)
    {
        while(s<e)
        {
        if(ss.charAt(s) != ss.charAt(e))
        {
            return false;
        }
            s++ ; e--;
        }
        return true;
    }
}
```

Q : [Rotate Array]

Solution :

```
class Solution {
    public void rotate(int[] nums, int k) {

        int dummy[] =new int[nums.length];

        for(int i=0;i<nums.length;i++)
        {
            dummy[(i+k)%nums.length]=nums[i];
        }
        int in=0;
        for(int d:dummy)
        {
```

```
            nums[in]=d;
            in++;
        }
    }
}
```

Q : Submissions - Reverse String

Solution :

```
class Solution {

    public void reverseString(char[] s) {

        int a=0;
        int b=s.length-1;

        while(a<=b)

        {
            char d=s[a];
            s[a]=s[b];
            s[b]=d;
            a++;
            b--;



        }

    }
}
```

Q : Submissions - Is Subsequence

Solution 1 :

```
class Solution {
    public boolean isSubsequence(String s, String t) {

            if(s == null && t.isEmpty() )  return true;


        int index=-1;

        for(char c: s.toCharArray())
        {
```

```
            index=t.indexOf(c,index+1);

            if(index==-1)
            {
                return false;
            }
        }
        return true;


    }
}
```

Solution 2 :

```
class Solution {
    public boolean isSubsequence(String s, String t) {

            if(s == null && t.isEmpty() )  return true;
            int c=0;
            for(int i=0;i<s.length();i++)
            {
                if(t.indexOf(s.charAt(i)) != -1 )
                {
                    c++;
                    t=t.substring(t.indexOf(s.charAt(i))+1 ,t.length());
                }

            }
          if(c==s.length())
              return true;
          return false;
    }
}
```

Q : [Submissions - Remove Element]

Solution :

```
class Solution {

    public int removeElement(int[] nums, int val) {



        int a=-1;
```

```
        for(int i=0;i<nums.length;i++)
        {
            if(nums[i]!=val)
            {
                a++;
                int d=nums[a];
                nums[a]=nums[i];
                nums[i]=d;
            }
        }
        return a+1;




    }
}
```

Q : [Valid Parentheses]

Solution :

```
class Solution {

    public boolean isValid(String s) {

        char[] arr=s.toCharArray();

        Stack<Character> st =new Stack<>();

        if(s.length()%2 != 0)  return false;

        for(char c:arr)
        {

            if(c=='(' || c== '{' || c== '[' )
            {

                st.push(c);

            }
            else if(c==')' && ! st.isEmpty() && st.peek()=='(' )
            {

                st.pop();
```

```
            }
            else if(c=='}' && ! st.isEmpty() && st.peek()=='{' )
            {
                st.pop();
            }
           else if(c==']' && ! st.isEmpty() && st.peek()=='[' )
            {
                st.pop();
            }
            else
            {
                st.push(c);
            }



        }
        if(st.isEmpty())
        {
            return true;
        }
        return false;







}
}
```

Q : Path Sum

Solution :

```
/**
 * Definition for a binary tree node.
 * public class TreeNode {
 *     int val;
 *     TreeNode left;
 *     TreeNode right;
 *     TreeNode() {}
 *     TreeNode(int val) { this.val = val; }
 *     TreeNode(int val, TreeNode left, TreeNode right) {
 *         this.val = val;
 *         this.left = left;
 *         this.right = right;
```

```
 *     }
 * }
 */
class Solution {


    public boolean pathsum(TreeNode root, int sum , int targetSum)
    {


        if(root==null)   return false;



        sum=sum+root.val;

        if(root.left==null && root.right==null)
        {
            if(sum==targetSum)  return true;

            return false;
        }

        boolean l =false;
        boolean r = false;

        if(root.left != null )
        {
            l = pathsum(root.left ,sum,targetSum);
        }
        if(root.right != null)
        {
           r = pathsum(root.right ,sum, targetSum);
        }

        if(l == true || r==true)
        {
            return true;
        }
        return false;






    }

    public boolean hasPathSum(TreeNode root, int targetSum) {
```

```
        boolean res= pathsum(root,0,targetSum);
         return res;

    }
}
```

Q : [Binary Tree Maximum Path Sum]

Sol :

```
/**
 * Definition for a binary tree node.
 * public class TreeNode {
 *     int val;
 *     TreeNode left;
 *     TreeNode right;
 *     TreeNode() {}
 *     TreeNode(int val) { this.val = val; }
 *     TreeNode(int val, TreeNode left, TreeNode right) {
 *         this.val = val;
 *         this.left = left;
 *         this.right = right;
 *     }
 * }
 */

class Solution {

     int sum=Integer.MIN_VALUE ;

     public int maxPathSum(TreeNode root) {


         maxpath(root) ;

         return sum;


    }

    public int maxpath(TreeNode root )
    {


         if(root==null)  return 0;
```

```
        int left=maxpath(root.left );
        int right=maxpath(root.right);

        int ret= Math.max(root.val ,Math.max(left+root.val , right+root.val) ) ;

        sum=Math.max(sum,Math.max(ret,root.val+left+right));

        return ret;




    }


}
```

Q : [56. Merge Intervals]

Solution :

```
class Solution {

    public int[][] merge(int[][] intervals) {

        if(intervals.length==1) return intervals ;

        Arrays.sort(intervals, (arr1,arr2) -> Integer.compare(arr1[0],arr2[0]) );

        int[] current_interval = intervals[0];

        List<int[] > output= new ArrayList<>();
        output.add(current_interval);

        for(int[] interval : intervals)
        {

            int current_begin = current_interval[0];
            int current_end  = current_interval[1];
```

```
            int next_begin = interval[0];
            int next_end   = interval[1];

            if(current_end >= next_begin)
            {

                current_interval[1]=Math.max(current_end,next_end);

            }
            else
            {
                current_interval=interval;
                output.add(current_interval);
            }

        }
        return output.toArray(new int[output.size()][]);

    }
}
```

Q : [Implement strStr()]

Solution :

```
class Solution {

    public int strStr(String haystack, String needle) {

        if(needle.length()==0){

            return 0;

        }

        int res=0;
        int loop=haystack.length() - needle.length();

        for(int i=0;i<=loop;i++)
        {
            String dumm=haystack.substring(i,i+needle.length());

            if(needle.equals(dumm))
            {
                res=i;
                return res;
            }
```

```
      }
      return -1;
    }
}
```

Q :

Some important question of leetcode :

[Codeforces 2.0's directory]