**NEXT PERMUTATION**

**CODE :**

class Solution {

    public void nextPermutation(int[] nums) {

        int n = nums.length;

        int ind = -1;

        int left = 0;

        int right = n-1;

        for(int i = n-2; i >=0;i--){

            if(nums[i]<nums[i+1]){

                ind = i;

                break;

            }

        }

        if(ind == -1){

            while(left < right){

                int temp = nums[left];

                nums[left] = nums[right];

                nums[right] = temp;

                left++;

                right--;

            }

            return;

        }

        for(int i =n-1;i >=ind ;i--){

            if(nums[i]>nums[ind]){

                int temp = nums[i];

                nums[i] = nums[ind];

                nums[ind] = temp;

                break;

            }

        }

        if(ind != -1){

            left = ind+1;

            while(left < right){

                int temp = nums[left];

                nums[left] = nums[right];

                nums[right] = temp;

                left++;

                right--;

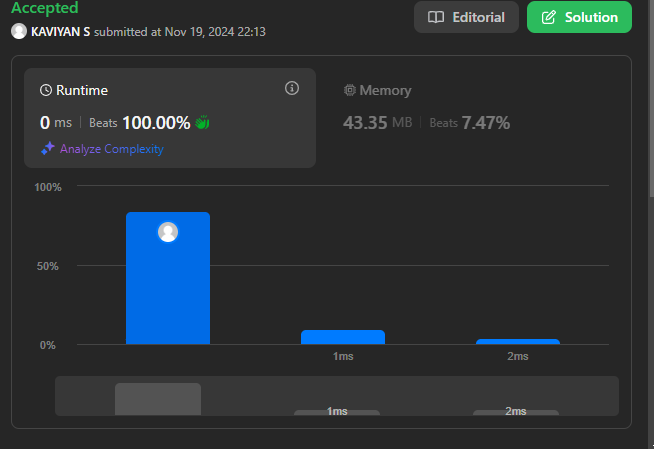
            }

        }

    }

}

**OUTPUT :**



TIME COMPLEXITY : O(n)

SPACE COMPLEXITY : O(1)

SPIRAL MATRIX

CODE :

class Solution {

    public List<Integer> spiralOrder(int[][] matrix) {

        int m = matrix.length;

        int n = matrix[0].length;

        int sr = 0, er = m-1;

        int sc = 0, ec = n-1;

        int total = m \* n;

        List<Integer> res = new ArrayList<>();

        while(total > 0){

            for(int i=sc; i<=ec && total>0; i++){

                res.add(matrix[sr][i]);

                total--;

            }

            sr++;

            for(int i=sr; i<=er && total>0; i++){

                res.add(matrix[i][ec]);

                total--;

            }

            ec--;

            for(int i=ec; i>=sc && total>0; i--){

                res.add(matrix[er][i]);

                total--;

            }

            er--;

            for(int i=er; i>=sr && total>0; i--){

                res.add(matrix[i][sc]);

                total--;

            }

            sc++;

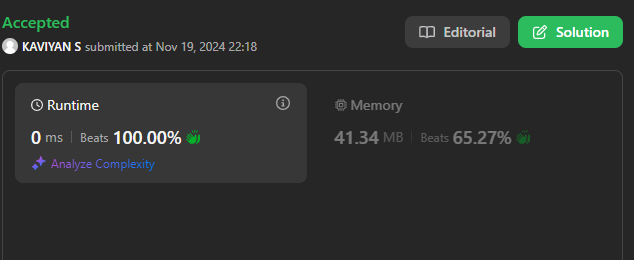
        }

        return res;

    }

}

OUTPUT :



TIME COMPLEXITY : O(m\*n)

SPACE COMPLEXITY : O(m\*n)

LONGEST SUBSTRING WITHOUT REPEATING CHARACTERS

CODE :

class Solution {

    public int lengthOfLongestSubstring(String s) {

        int ans = 0 ;

        int j = 0;

        Set<Character> hs = new HashSet<>();

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

           if(!hs.contains(s.charAt(i))){

                hs.add(s.charAt(i));

                ans = Math.max(ans,i-j+1);

           }else{

                while(hs.contains(s.charAt(i))){

                    hs.remove(s.charAt(j));

                    j++;

                }

                hs.add(s.charAt(i));

           }

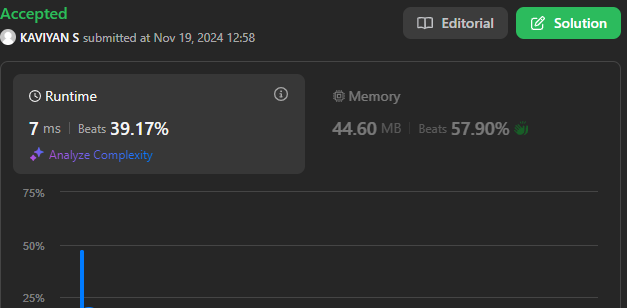
        }

        return ans;

    }

}

OUTPUT :



TIME COMPLEXITY : O(n)

SPACE COMPLEXITY : O(1)

REMOVE LINKED LIST ELEMENTS

CODE :

class Solution {

    public ListNode removeElements(ListNode head, int val) {

        ListNode temp = new ListNode(0),curr = temp;

        temp.next = head;

        while(curr.next != null){

            if(curr.next.val == val){

                curr.next = curr.next.next;

            }else{

                curr = curr.next;

            }

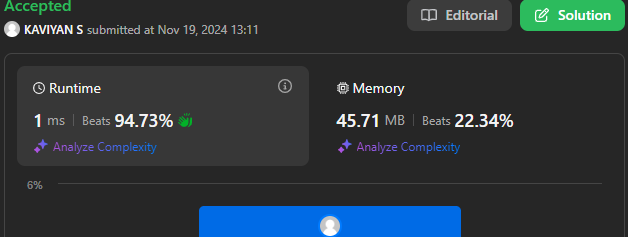
        }

        return temp.next;

    }

}

OUTPUT :



TIME COMPLEXITY : O(n)

SPACE COMPLEXITY : O(1)

PALINDROME LINKED LIST

CODE :

class Solution {

    public boolean isPalindrome(ListNode head) {

        Stack<Integer> s = new Stack<>();

        ListNode curr = head;

        while(curr != null){

            s.push(curr.val);

            curr = curr.next;

        }

        curr = head;

        while(curr != null && curr.val == s.pop()){

            curr = curr.next;

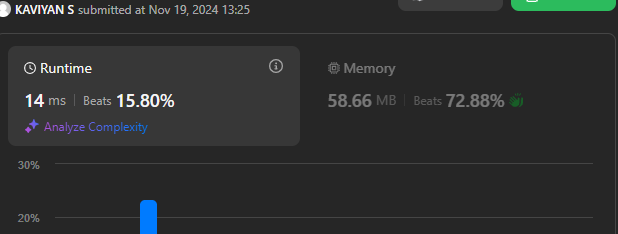
        }

        return curr == null;

    }

}

OUTPUT :



TIME COMPLEXITY : O(n)

SPACE COMLEXITY : O(n)

MINIMUM PATH SUM

CODE :

class Solution {

    public int minPathSum(int[][] grid) {

        int m = grid.length ;

        int n = grid[0].length;

        for(int i = 1 ; i<n ; i++){

            grid[0][i] += grid[0][i-1];

        }

        for(int i = 1 ; i<m ; i++){

            grid[i][0] += grid[i-1][0];

        }

        for(int i = 1 ; i<m ; i++){

            for(int j = 1 ; j<n ; j++){

                grid[i][j] += Math.min(grid[i-1][j],grid[i][j-1]);

            }

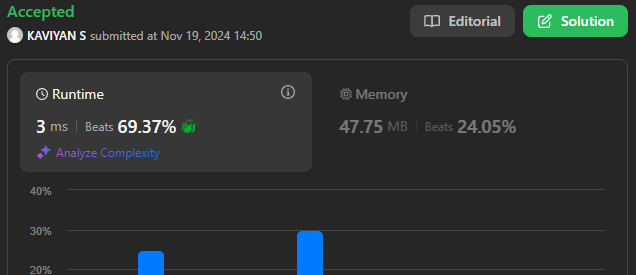
        }

        return grid[m-1][n-1];

    }

}

OUTPUT :



TIME COMPLEXITY : O(m\*n)

SPACE COMPLEXITY : O(1)

VALIDATE BINARY SEARCH TREE

CODE :

class Solution {

    public boolean isValidBST(TreeNode root) {

       if(root == null || (root.left == null && root.right== null)){

            return true;

        }

        return isvalid(root , Long.MIN\_VALUE ,Long.MAX\_VALUE);

    }

    public boolean isvalid(TreeNode root , long min , long max){

        if(root == null ){

            return true;

        }

        if(root.val >=max || root.val <= min){

            return false;

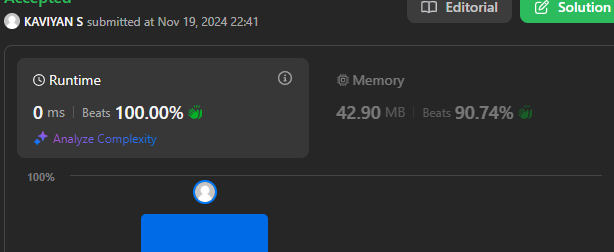
        }

        return isvalid(root.left , min , root.val) && isvalid(root.right , root.val , max);

    }

}

OUTPUT :



TIME COMPLEXITY : O(n)

SPACE COMPLEXITY : O(n)

WORD LADDER

CODE :

class Solution {

    public int ladderLength(String bw, String ew, List<String> ls) {

        HashSet<String> l = new HashSet<>(ls);

        if(!l.contains(ew)) return 0;

        HashSet<String> hs = new HashSet<>();

        Queue<String> q = new LinkedList<>();

        q.add(bw);

        hs.add(bw);

        int ans = 1;

        while(!q.isEmpty()){

            int n  = q.size();

            for(int i = 0 ; i<n ; i++){

                String s= q.poll();

                if(s.equals(ew)) return ans;

                for(int j = 0 ; j<s.length() ; j++){

                    char[] word = s.toCharArray();

                    for(char c = 'a' ; c<='z' ; c++){

                        word[j] = c;

                        String res = new String(word);

                        if(l.contains(res) && !hs.contains(res)){

                            q.add(res);

                            hs.add(res);

                        }

                    }

                }

            }

            ans++;

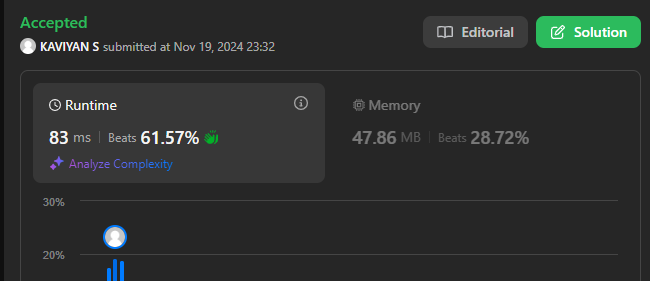
        }

        return 0;

    }

}

OUTPUT :



TIME COMPLEXITY : O(n\*L)

SPACE COMPLEXITY : O(n);

COURSE SCHEDULER

CODE :

class Solution {

    public boolean canFinish(int nc, int[][] prerequisites) {

        int c = 0;

        if (nc <= 0) {

            return true;

        }

        int[] inDegree = new int[nc];

        List<List<Integer>> graph = new ArrayList<>();

        for (int i = 0; i < nc; i++) {

            graph.add(new ArrayList<>());

        }

        for (int[] edge : prerequisites) {

            int a = edge[1];

            int b = edge[0];

            graph.get(a).add(b);

            inDegree[b]++;

        }

        Queue<Integer> sources = new LinkedList<>();

        for (int i = 0; i < nc; i++) {

            if (inDegree[i] == 0) {

                sources.offer(i);

            }

        }

        while (!sources.isEmpty()) {

            int course = sources.poll();

            c++;

            for (int b : graph.get(course)) {

                inDegree[b]--;

                if (inDegree[b] == 0) {

                    sources.offer(b);

                }

            }

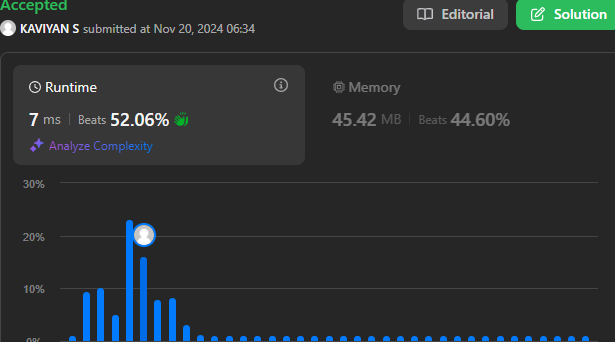
        }

        return c == nc;

    }

}

OUTPUT :



TIME COMPLEXITY : O(n)

SPACE COMPLEXITY : O(n)