VALID PALINDROME

CODE :

class Solution {

    public boolean isValid(String s) {

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

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

            if(c==')'){

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

                    st.pop();

                }else{

                    return false;

                }

            }else if(c=='}'){

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

                    st.pop();

                }else{

                    return false;

                }

            }else if(c==']'){

                if(!st.isEmpty() && st.peek()=='['){

                    st.pop();

                }else{

                    return false;

                }

            }else{

                st.push(c);

            }

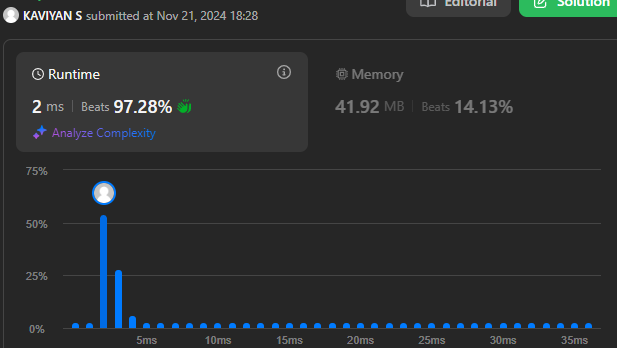
        }

        return st.size()==0 ;

    }

}

OUTPUT :



TIME COMPLEXITY : O(n)

SPACE COMPLEXITY : O(n)

IS SUBSEQUENCE

CODE :

class Solution {

    public boolean isSubsequence(String s, String t) {

        int a = 0;

        int i = 0 ;

        while(i<t.length() && a<s.length()){

            if(s.charAt(a) == t.charAt(i)){

                a++;

            }

            i++;

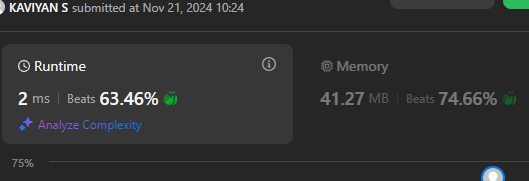
        }

        return a==s.length();

    }

}

OUTPUT :



TIME COMPLEXITY : O(n)

SPACE COMPLEXITY : O(1)

TWO SUM II - INPUT ARRAY IS SORTED

CODE :

class Solution {

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

        int[] ans = new int[2];

        int i = 0 , j = nums.length-1;

        while(i<j){

            int s = nums[i] + nums[j];

            if(s==target){

                ans[0] = i+1;

                ans[1] = j+1;

                break;

            }

            if(s<target){

                i++;

            }else{

                j--;

            }

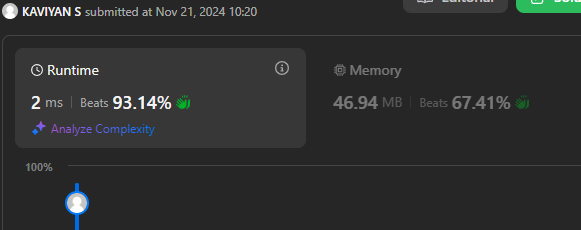
        }

        return ans;

    }

}

OUTPUT :



TIME COMPLEXITY : O(n)

SPACE COMPLEXITY : O(1)

COTAINER WITH MOST WATER

CODE :

class Solution {

    public int maxArea(int[] h) {

        int ans = 0;

        int r = h.length - 1 , l = 0;

        while(l<=r){

            int cur = Math.min(h[l],h[r]) \* (r-l);

            if(h[l]<h[r]){

                l++;

            }else{

                r--;

            }

            ans = Math.max(cur,ans);

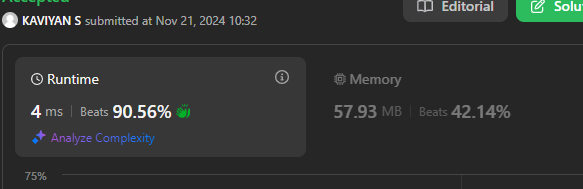
        }

        return ans;

    }

}

OUTPUT :



TIME COMPLEXITY : O(n)

SPACE COMPLEXITY : O(1)

3SUM

CODE :

class Solution {

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

        Arrays.sort(nums);

        HashSet<List<Integer>> s = new HashSet<>();

        for(int i = 0 ; i<nums.length-2 ; i++){

            int j = i+1;

            int k = nums.length - 1;

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

                continue;

            }

            while(j<k){

                int sum = nums[i] + nums[j] + nums[k];

                if(sum==0){

                    List<Integer> l = new ArrayList<>(Arrays.asList(nums[i],nums[j],nums[k]));

                    s.add(l);

                }

                if(sum<0){

                    j++;

                }else{

                    k--;

                }

            }

        }

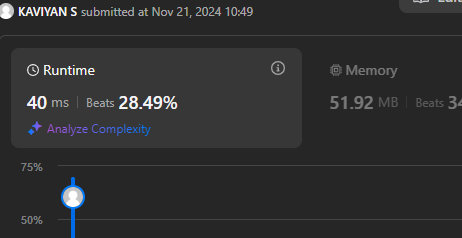
        List<List<Integer>> ans = new ArrayList<>(s);

        return ans;

    }

}

OUTPUT :



TIME COMPLEXITY : O(n)

SPACE COMPLEXITY : O(n)

MINIMUM SIZE SUBARRAY SUM

CODE :

class Solution {

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

        int ans = Integer.MAX\_VALUE;

        int s =0;

        int sum = 0;

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

            sum += nums[i];

            while(sum>=target){

                ans = Math.min(ans,i-s+1);

                sum -= nums[s];

                s++;

            }

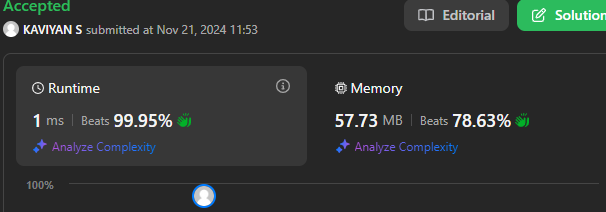
        }

        return ans>nums.length ? 0 : ans;

    }

}

OUTPUT :



TIME COMPLEXITY : O(n)

SPACE COMPLEXITY : O(1)

LONGEST SUBSTRING WITHOUT REPEATING CHARACTER

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));

           }

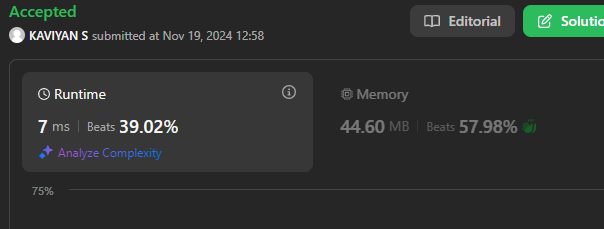
        }

        return ans;

    }

}

OUTPUT :



TIME COMPLEXITY : O(n)

SPACE COMPLEXITY : O(n)

VALID PALINDROME

CODE :

class Solution {

    public boolean isPalindrome(String s) {

        StringBuilder st = new StringBuilder();

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

            if(Character.isDigit(s.charAt(i))){

                st.append(s.charAt(i));

            }else if(Character.isLetter(s.charAt(i))){

                st.append(Character.toLowerCase(s.charAt(i)));

            }

        }

        String c = st.toString();

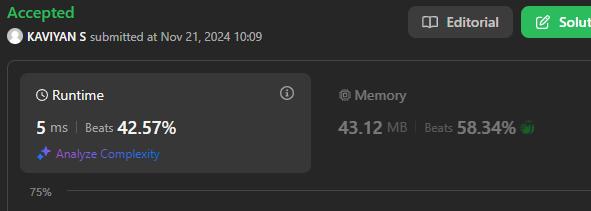
        String ans = st.reverse().toString();

        return c.equals(ans);

    }

}

OUTPUT :



TIME COMPLEXITY : O(n)

SPACE COMPLEXITY : O(n)

SIMPLIFY PATH

CODE :

class Solution {

    public String simplifyPath(String path) {

        String[] arr = path.split("/");

        System.out.println(Arrays.toString(arr));

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

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

            if(arr[i].length() == 0 || arr[i].equals(".")) continue;

            if(!s.isEmpty() && arr[i].equals("..")){

                s.pop();

                continue;

            }else if(arr[i].equals("..")){

                continue;

            }

            s.push(arr[i]);

        }

        StringBuilder ans = new StringBuilder();

        Iterator v = s.iterator();

        if(s.size()==0){

            ans.append("/");

        }

        while(v.hasNext()){

            ans.append("/" + v.next());

        }

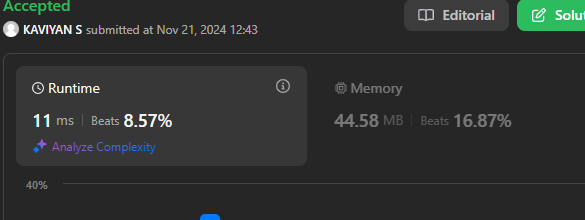
        System.out.println(s);

        return ans.toString();

    }

}

OUTPUT :



TIME COMPLEXITY : O(n)

SPACE COMPLEXITY : O(n)

MIN STACK

CODE :

class MinStack {

    public Stack<Integer> s ;

    public Stack<Integer> min;

    public MinStack() {

        s = new Stack<>();

        min = new Stack<>();

    }

    public void push(int val) {

        s.push(val);

        if(min.isEmpty() || val<=min.peek()){

            min.push(val);

        }

    }

    public void pop() {

        int p = s.pop();

        if(p==min.peek()){

            min.pop();

        }

    }

    public int top() {

        return s.peek();

    }

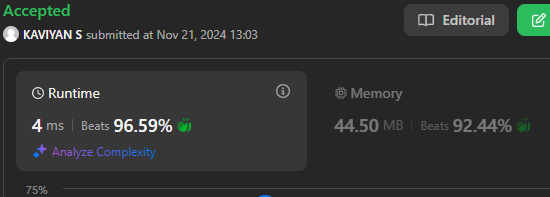
    public int getMin() {

        return min.peek();

    }

}

OUTPUT :



TIME COMPLEXITY : O(1)

SPACE COMPLEXITY : O(1)

EVALUATE REVERSE POLISH NOTATION

CODE :

class Solution {

    public int evalRPN(String[] tokens) {

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

        for(String st : tokens){

            if(Character.isDigit(st.charAt(0)) || (st.length()>1 && Character.isDigit(st.charAt(1)))){

                s.push(Integer.valueOf(st));

            }else{

                int b= s.pop();

                int a = s.pop();

                if(st.equals("+")){

                    s.push(a+b);

                }else if(st.equals("-")){

                    s.push(a-b);

                }else if(st.equals("\*")){

                    s.push(a\*b);

                }else{

                    s.push(a/b);

                }

            }

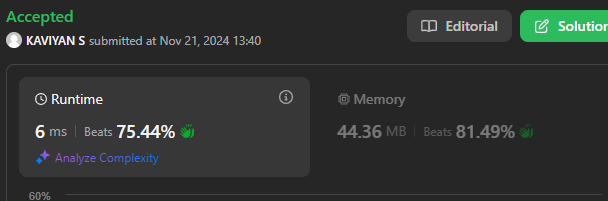
        }

        return s.pop();

    }

}

OUTPUT:



TIME COMPLEXITY : O(n)

SPACE COMPLEXITY : O(n)

SEARCH INSERTION POSITION

CODE :

class Solution {

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

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

            if(nums[i] == target) {

                return i;

            }

            if(i<nums.length-1 && nums[i]<target && nums[i+1]>target){

                return i+1;

            }

        }

        if(nums[0]>target){

            return 0;

        }else{

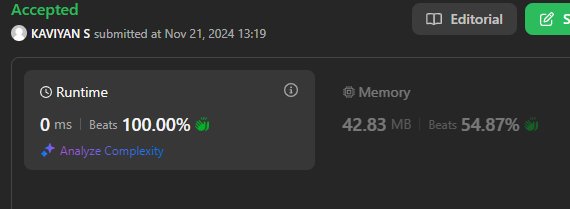
            return nums.length;

        }

    }

}

OUTPUT :



TIME COMPLEXITY : O(n)

SPACE COMPLEXITY : O(1)

SEARCH 2D MATRIX

CODE :

class Solution {

    public boolean searchMatrix(int[][] matrix, int target) {

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

            for(int j = 0 ; j<matrix[0].length ; j++){

                if(matrix[i][j]==target){

                    return true;

                }

            }

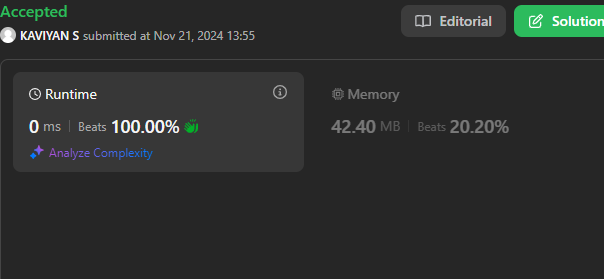
        }

        return false;

    }

}

OUTPUT :



TIME COMPLEXITY : O(n^2)

SPACE COMPLEXITY : O(1)

FIND PEAK ELEMENT

CODE :

class Solution {

    public int findPeakElement(int[] nums) {

        if(nums.length == 1) return 0;

        int ans = 0;

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

            if(i==0 && nums[i]>nums[i+1]){

                ans = Math.max(ans,i);

            }else if(i==nums.length-1 && nums[i]>nums[i-1]){

                ans = Math.max(ans,i);

            }else if(i!=nums.length-1 && i!=0 && nums[i-1] < nums[i] && nums[i+1] < nums[i]){

                ans = Math.max(ans,i);

            }

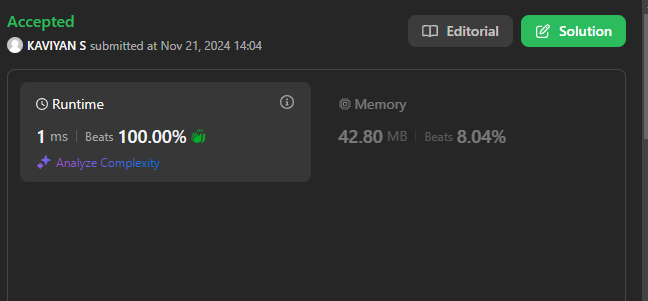
        }

        return ans;

    }

}

OUTPUT :



TIME COMPLEXITY : O(n)

SPACE COMPLEXITY : O(1);

SEARCH IN ROTATED SORTED ARRAY

CODE :

class Solution {

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

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

            if(nums[i] == target) {

                return i;

            }

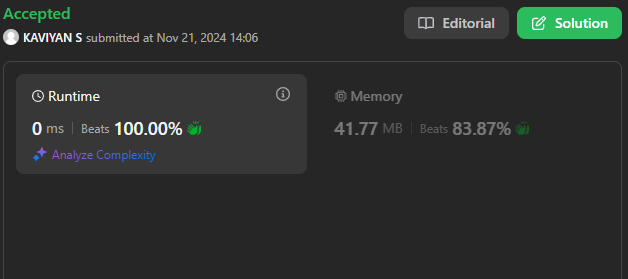
        }

        return -1;

    }

}

OUTPUT :



TIME COMPLEXITY : O(n)

SPACE COMPLEXITY : O(1)

FIND FIRST AND LAST POSITIONT OF ELEMENT IN A SORTED ARRAY   
CODE :

class Solution {

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

        int[] arr = new int[2];

        Arrays.fill(arr,-1);

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

            if(nums[i]==target){

                arr[0] = i;

                int j = i;

                while(j<nums.length && nums[j]==target){

                    j++;

                }

                arr[1] = j-1;

                break;

            }

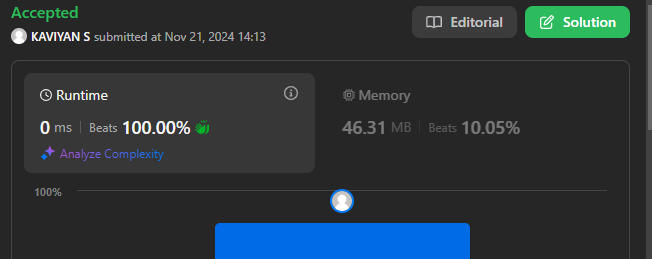
        }

        return arr;

    }

}

OUTPUT :



TIME COMPLEXITY : O(n)

SPACE COMPLEXITY : O(1)

FIND MINIMUM IN ROTATED SORTED ARRAY   
CODE :

class Solution {

    public int findMin(int[] nums) {

        for(int i = 0 ; i<nums.length -1 ; i++){

            if(nums[i]>nums[i+1]){

                return nums[i+1];

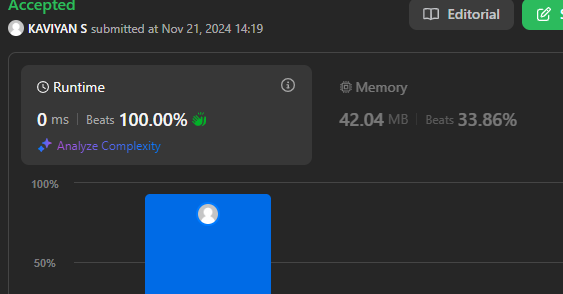
            }

        }

        return nums[0];

    }

}

OUTPUT : 

TIME COMPLEXITY : O(n)

SPACE COMPLEXITY : O(1)