Date: 21st Nov 2024

1. Valid Palindrome

class Solution {

    public boolean isPalindrome(String s) {

        String a="";

        for(int i=0;i<s.length();i++)

        {

            char k=s.charAt(i);

            if(Character.isAlphabetic(k)){

                a+=Character.toLowerCase(k);

            }

            if(Character.isDigit(k))

            {

                a+=k;

            }

        }

        int i=0;

        int j=a.length()-1;

        while(i<j)

        {

            if(a.charAt(i)!=a.charAt(j)) return false;

            i++;

            j--;

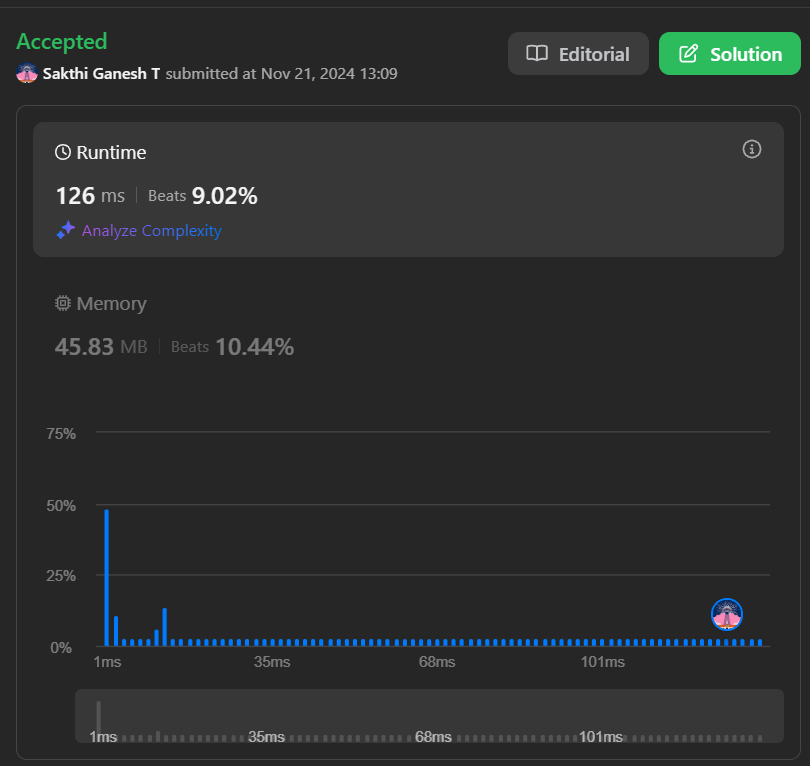
        }

        return true;

    }

}

Output:



1. Is subsequence

class Solution {

    public boolean isSubsequence(String s, String t) {

       int c=0;

       int d=0;

       while(c<s.length() && d<t.length())

       {

        if(s.charAt(c)==t.charAt(d))

        {

            c++;

        }

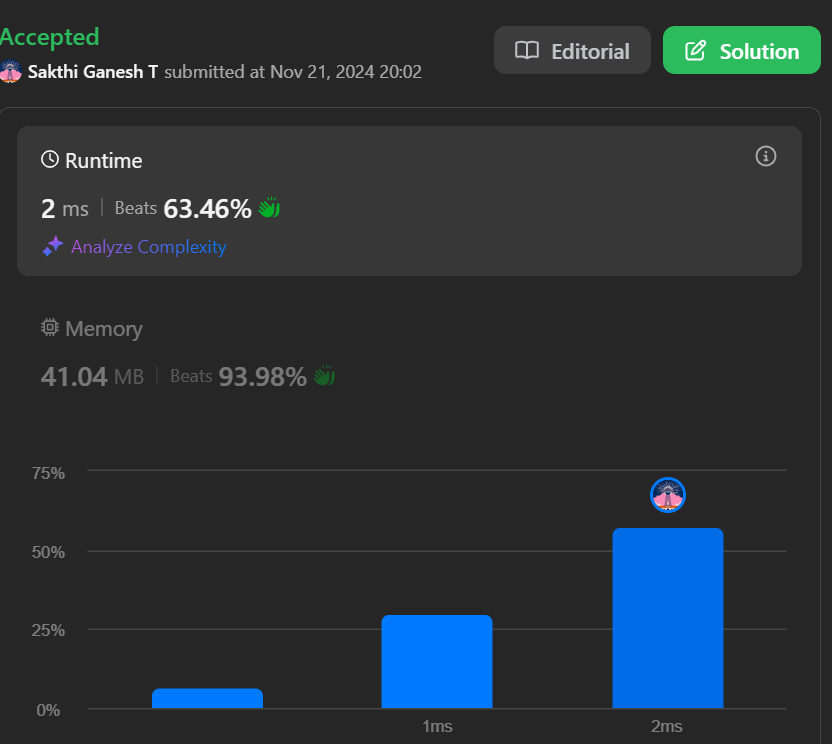
        d++;

       }

       return c==s.length();

    }

}



1. Two sum 2 sorted array:

class Solution {

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

        int a[]=new int[2];

        int i=0;

        int j=numbers.length-1;

        while(i<j)

        {

            if((numbers[i]+numbers[j])==target)

            {

                a[0]=i+1;

                a[1]=j+1;

                break;

            }

            else if((numbers[i]+numbers[j])<target)

            {

                i++;

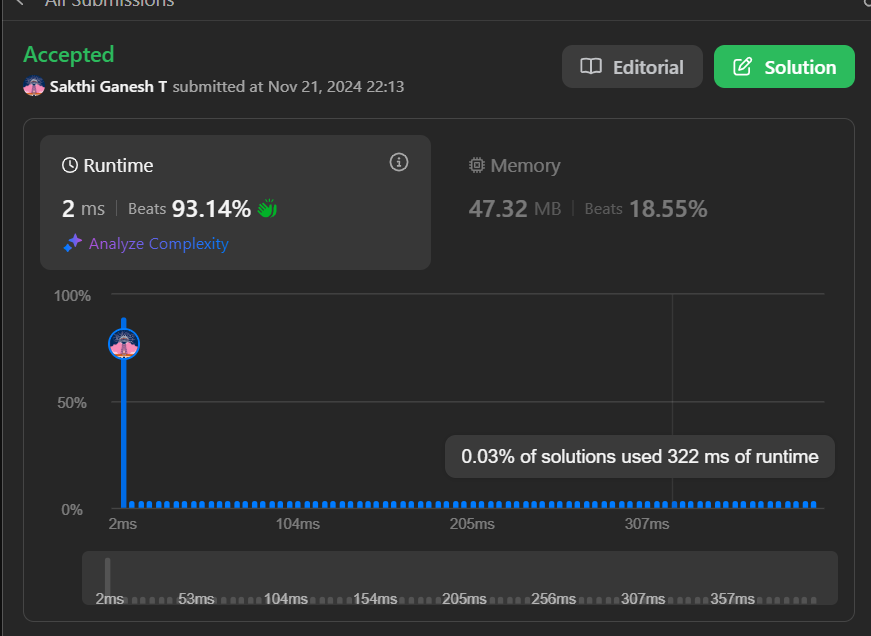
            }

            else{

                j--;}}

     return a;}}

Output:



4.Container with most water:

class Solution {

    public int maxArea(int[] height) {

        int l=0;

        int r=height.length-1;

        int max\_=0;

        while(l<r){

            int w=Math.min(height[l],height[r]);

            int b=r-l;

            int c=w\*b;

            max\_=Math.max(c,max\_);

            if(height[l]<height[r]){

                l++;

            }

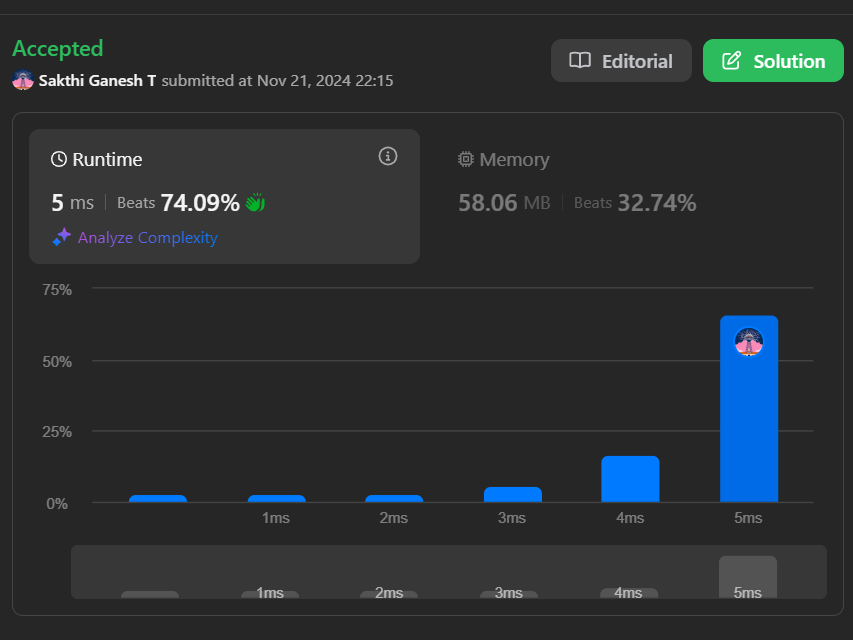
            else r--;

        }

        return max\_;

    }}

Output:



5.3Sum

class Solution {

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

        Set<List<Integer>> a=new HashSet<>();

        Arrays.sort(nums);

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

        {

            int j=i+1;

            int k=nums.length-1;

            while(j<k)

            {

                if((nums[i]+nums[j]+nums[k])==0)

                {

                    a.add(Arrays.asList(nums[i],nums[j++],nums[k--]));

                }

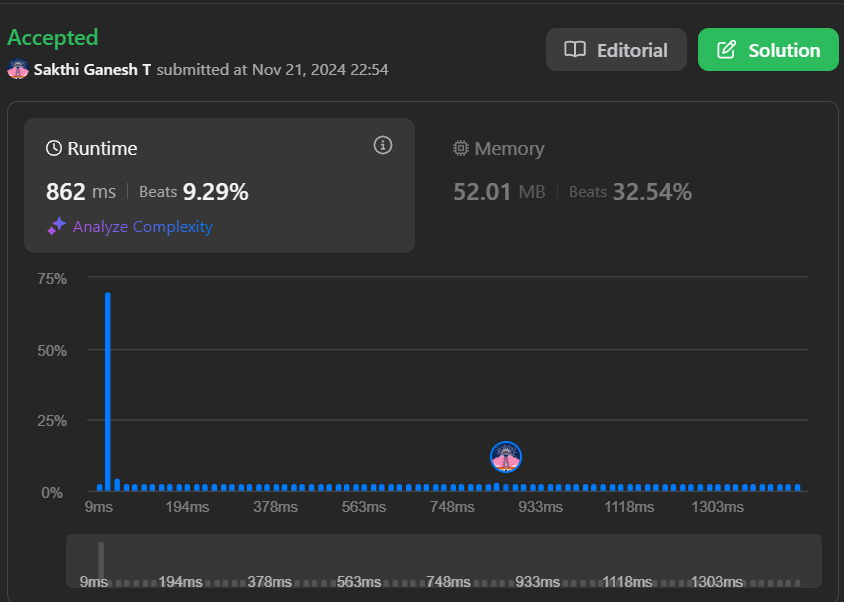
                if((nums[i]+nums[j]+nums[k])<0){  j++};

                if((nums[i]+nums[j]+nums[k])>0)

                {k--;}}  }

        return new ArrayList<>(a);   }}

Output:



6. Valid Parenthesis:

class Solution {

    public boolean isValid(String s) {

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

        for(char i:s.toCharArray()){

            if(i=='('){

                st.push(')');

            }

            else if(i=='{'){

                st.push('}');

            }

            else if(i=='['){

                st.push(']');

            }

            else if(st.isEmpty()||st.pop()!=i){

                return false;

            }

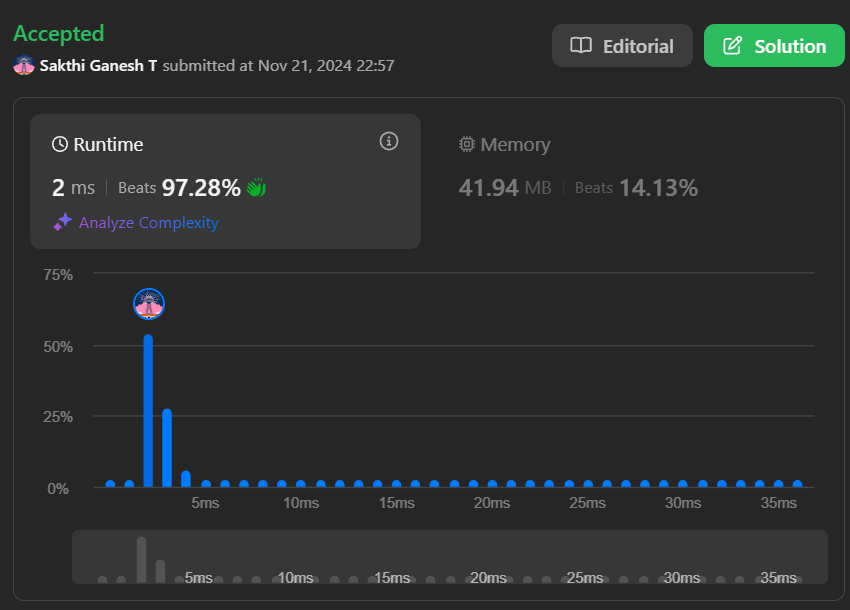
        }

        return st.isEmpty();

    }

}

Output:



7.simplify path

class Solution {

    public String simplifyPath(String path) {

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

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

        for (String dir : directories) {

            if (dir.equals(".") || dir.isEmpty()) {

                continue;

            } else if (dir.equals("..")) {

                if (!stack.isEmpty()) {

                    stack.pop();

                }

            } else {

                stack.push(dir);

            }

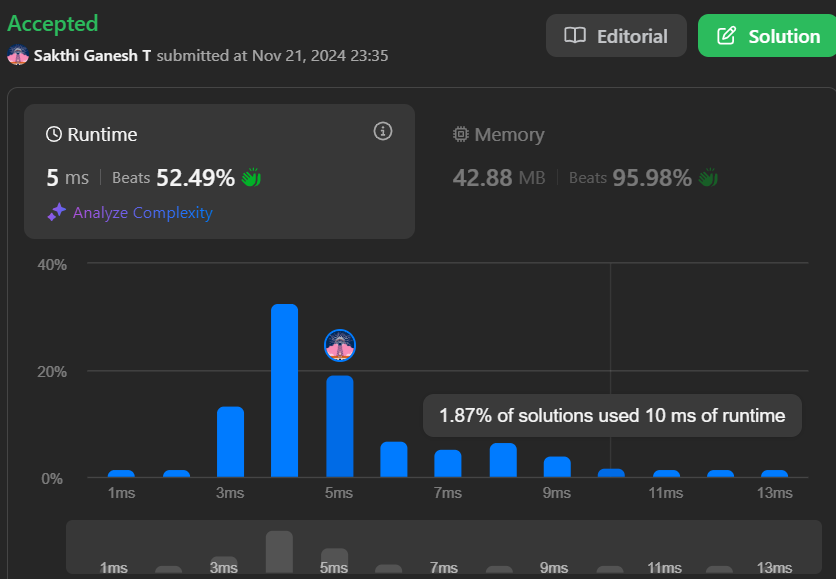
        }

        return "/" + String.join("/", stack);

    }

}

Output:



8. Search Insert Position

class Solution {

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

        int l=0,r=nums.length-1;

        while(l<=r)

        {

            int mid=(int)(l+r)/2;

            if(nums[mid]==target)

            {

                return mid;

            }

            else if(nums[mid]<target)

            {

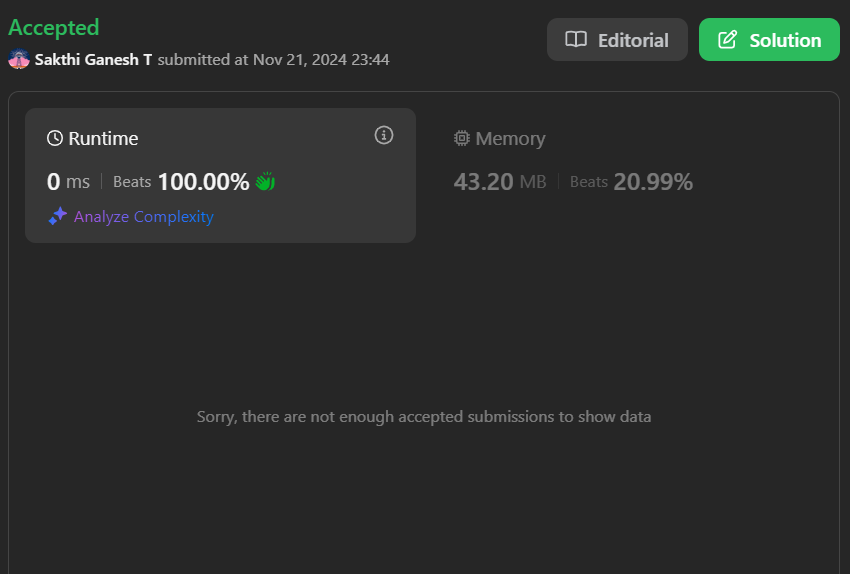
                l=mid+1;

            }

            else{

                r=mid-1;}}

    return l;}}



9. search a 2d matrix

class Solution {

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

if (matrix == null || matrix.length == 0 || matrix[0].length == 0) {

return false;

}

int rows = matrix.length, cols = matrix[0].length;

int left = 0, right = rows \* cols - 1;

while (left <= right) {

int mid = left + (right - left) / 2;

int midElement = matrix[mid / cols][mid % cols];

if (midElement == target) {

return true;

} else if (midElement < target) {

left = mid + 1;

} else {

right = mid - 1;

}

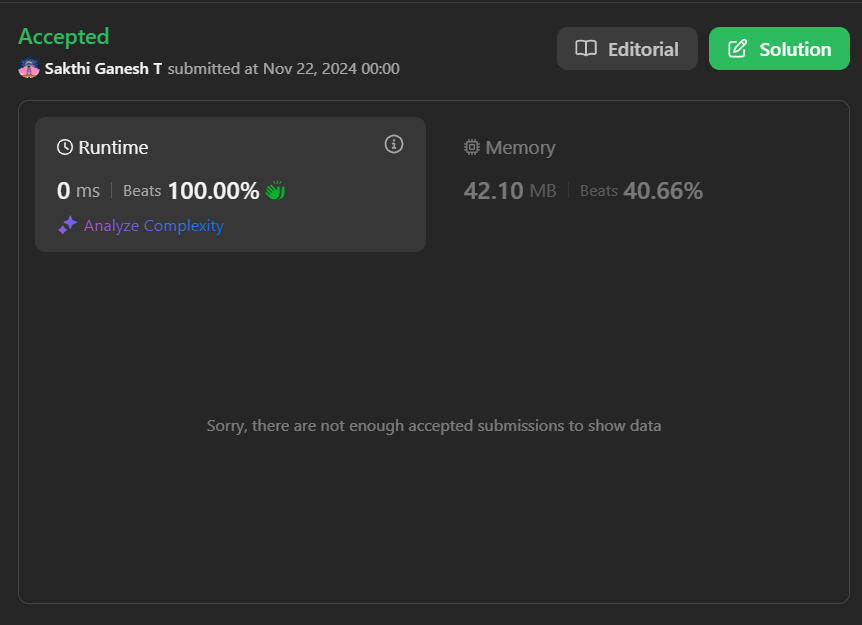
}

return false;

}

}

Output:



10. Search in a rotated Sorted array

class Solution {

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

int left = 0;

int right = nums.length - 1;

while (left <= right) {

int mid = (left + right) / 2;

if (nums[mid] == target) {

return mid;

} else if (nums[mid] >= nums[left]) {

if (nums[left] <= target && target <= nums[mid]) {

right = mid - 1;

} else {

left = mid + 1;

}

} else {

if (nums[mid] <= target && target <= nums[right]) {

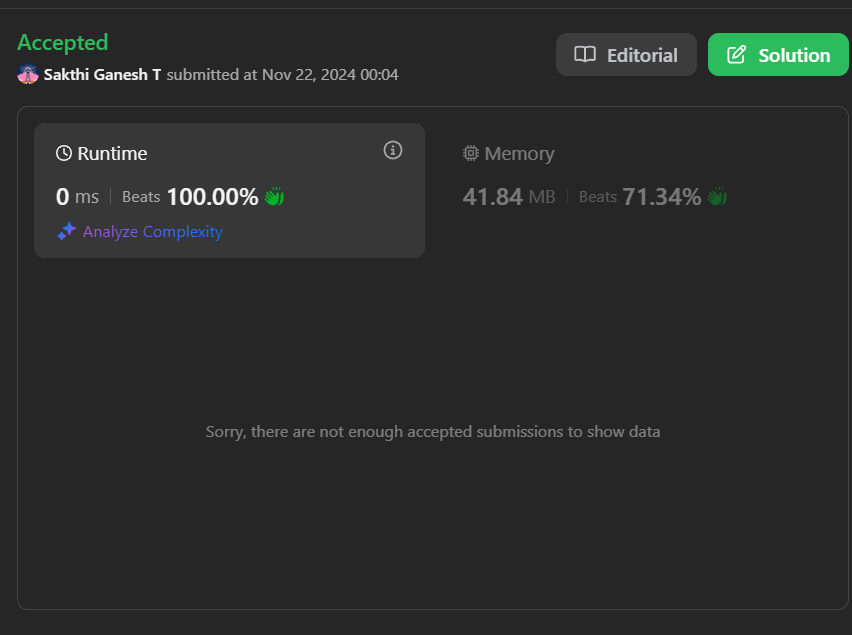
left = mid + 1;

} else {

right = mid - 1;}} }

return -1;}}

Output:



11. Find the Peak Element

class Solution {

    public int findPeakElement(int[] nums) {

        int l=0,r=nums.length-1;

        while(l<r)

        {

            int mid=(l+r)/2;

            if(nums[mid]>nums[mid+1])

            {

                r=mid;

            }

            else{

                l=mid+1;

            }

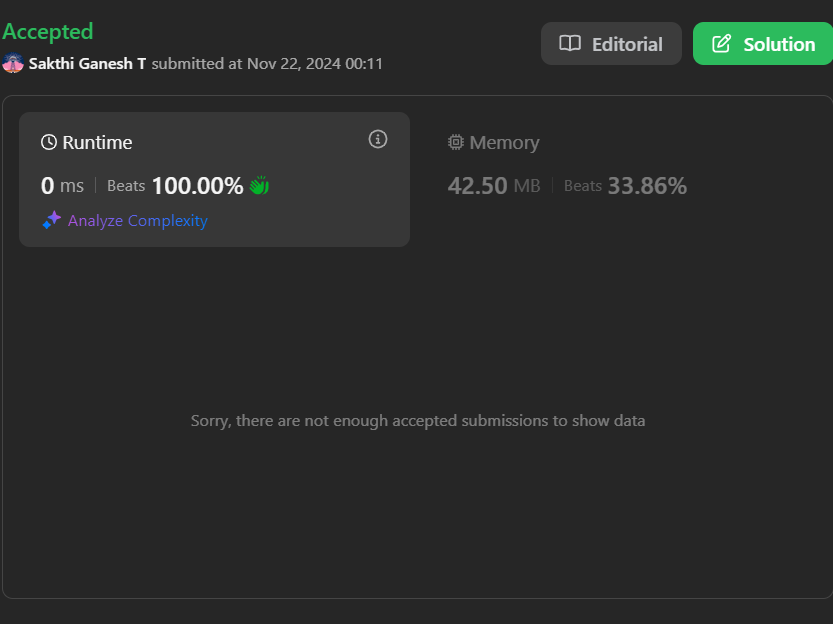
        }

        return l;

    }

}

Output:



12.longest substring without repeating characters:

class Solution {

    public int lengthOfLongestSubstring(String s) {

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

        int l=0,r=0;

        int n=s.length();

        int len=0;

        while(r<n)

        {

            if(h.containsKey(s.charAt(r)))

            {

                l=Math.max(h.get(s.charAt(r))+1,l);

            }

            h.put(s.charAt(r),r);

            len=Math.max(len,r-l+1);

            r++;

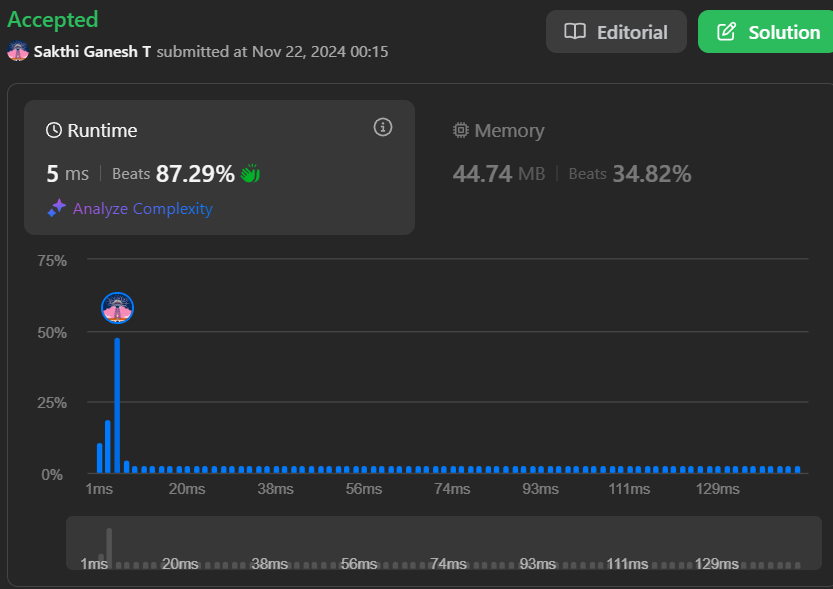
        }

        return len;

    }

}

Output:



13. find first and last position:

class Solution {

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

        int[] a = {-1, -1};

        if (nums == null || nums.length == 0) {

            return a;

        }

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

            if (nums[i] == target) {

                a[0] = i;

                break;} }

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

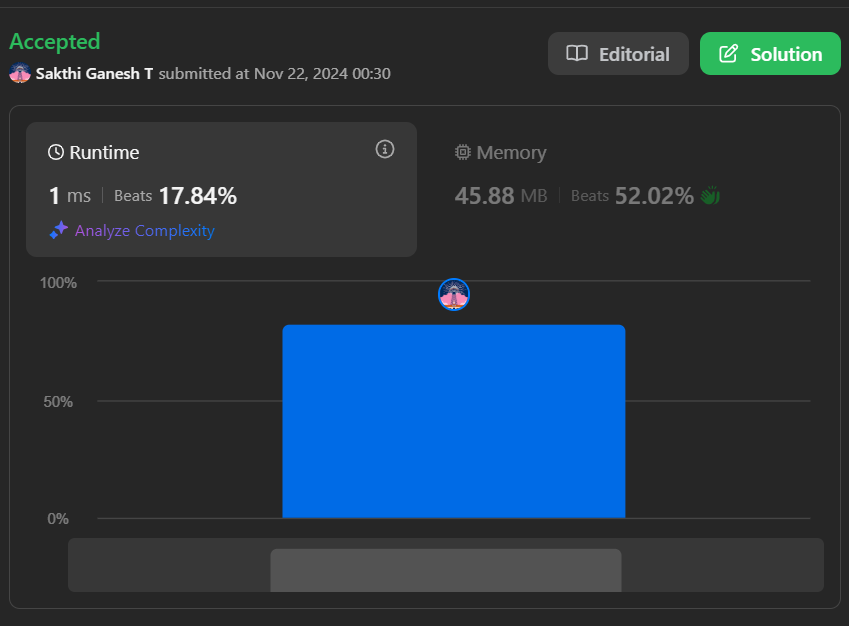
            if (nums[i] == target) {

                a[1] = i;

                break;}}

        return a;}}

Output:



**14.** Minimum size subarray with maximum sum

class Solution {

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

        int n = nums.length;

        int start = 0, sum = 0, minLength = Integer.MAX\_VALUE;

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

            sum += nums[end];

            while (sum >= target) {

                minLength = Math.min(minLength, end - start + 1);

                sum -= nums[start];

                start++;

            }

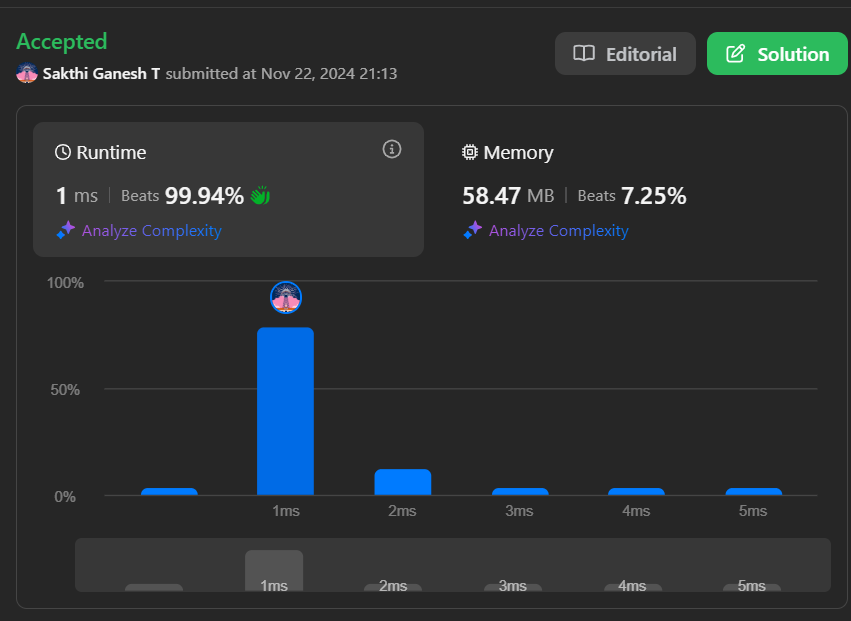
        }

        return minLength == Integer.MAX\_VALUE ? 0 : minLength;

    }

}

Output:



15. Min Stack

class MinStack {

    int min=Integer.MAX\_VALUE;

    Stack<Integer> s=new Stack<>();

    public void push(int val) {

        if(val<=min)

        {

            s.push(min);

            min=val;

        }

        s.push(val);

    }

    public void pop() {

        if(s.pop()==min)

            min=s.pop();

    }

    public int top() {

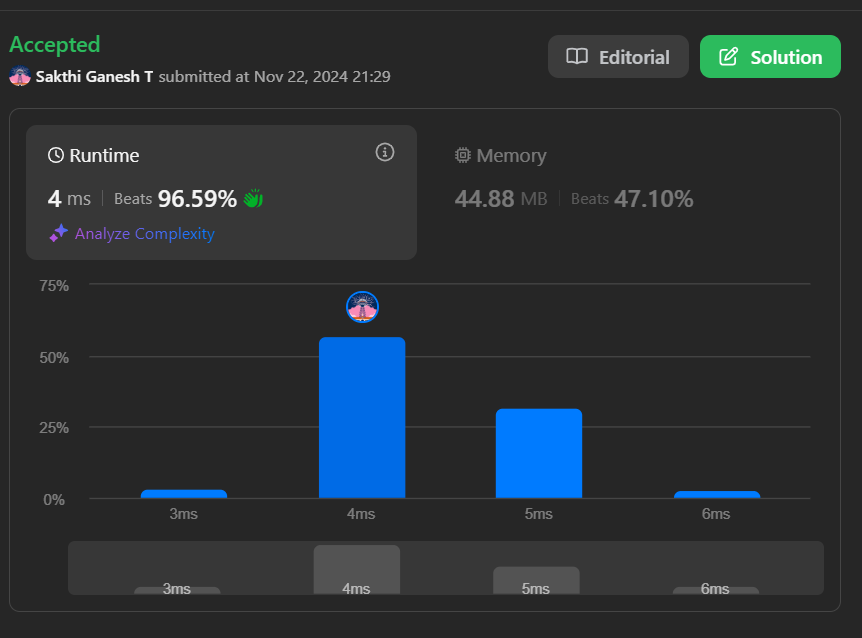
        return s.peek();}

   public int getMin() {

        return min;

       }}

Output:



16. Find minimum in rotated sorted array:

class Solution {

    public int findMin(int[] nums) {

        int l=0,r=nums.length-1;

        while(l<r)

        {

            int mid=(l+r)/2;

            if(nums[mid]<nums[r])

            {

                r=mid;

            }

            else{

                l=mid+1;

        }

    }

     return nums[l];

}

}

Output:

