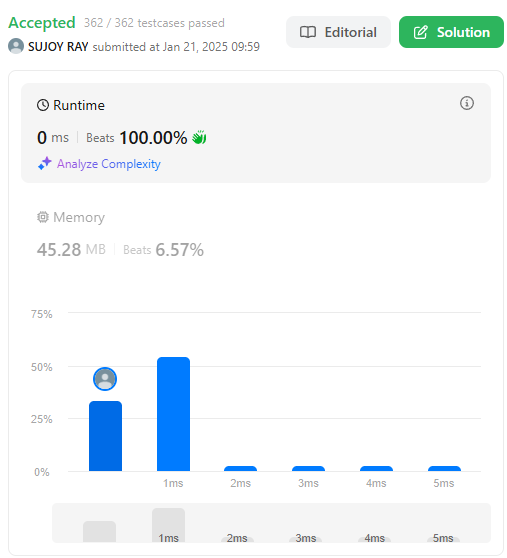
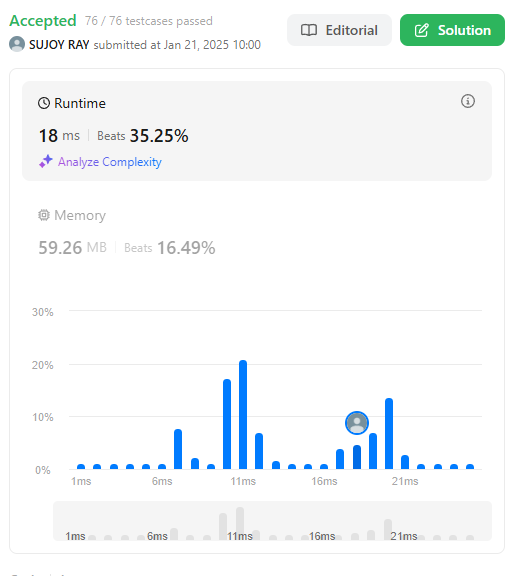
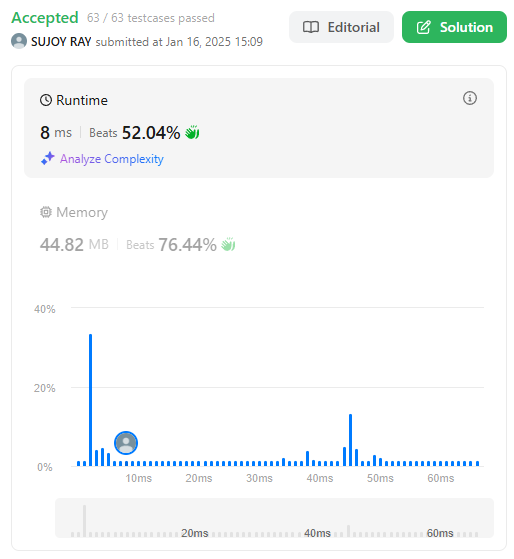
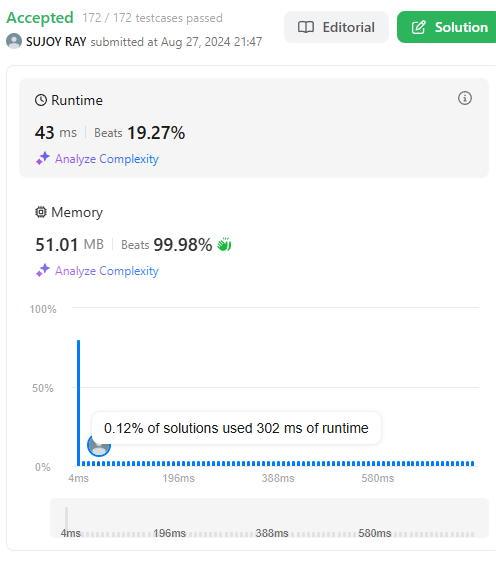
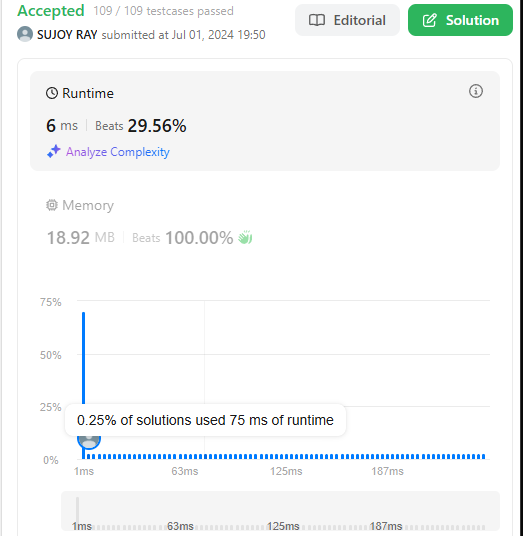
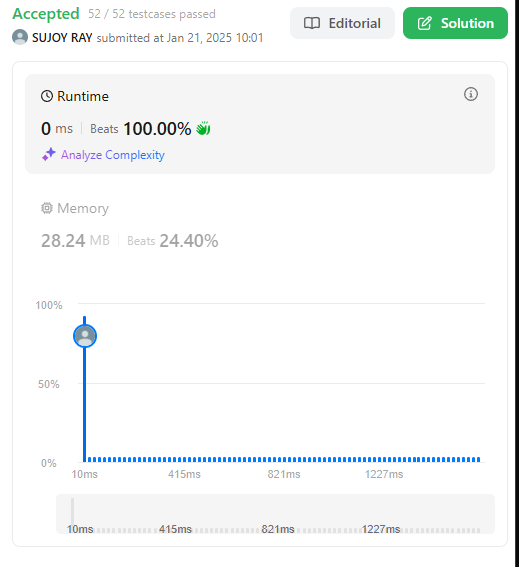
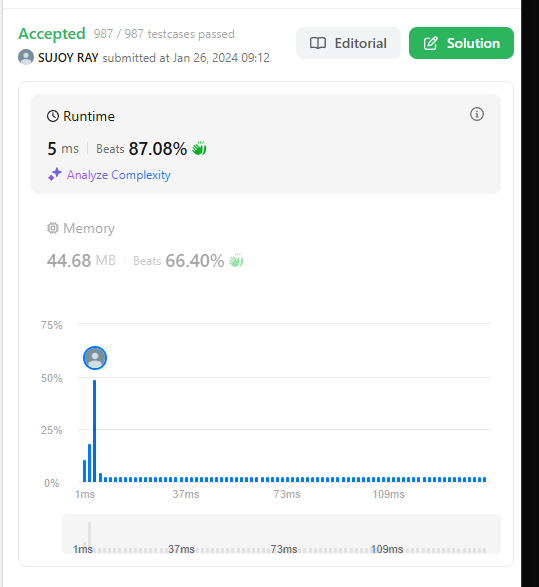
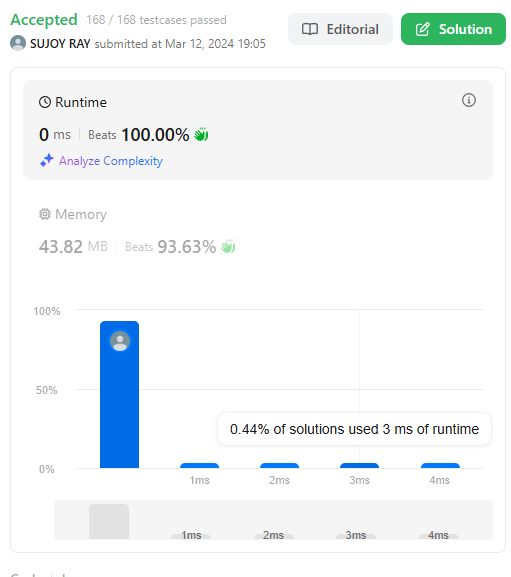
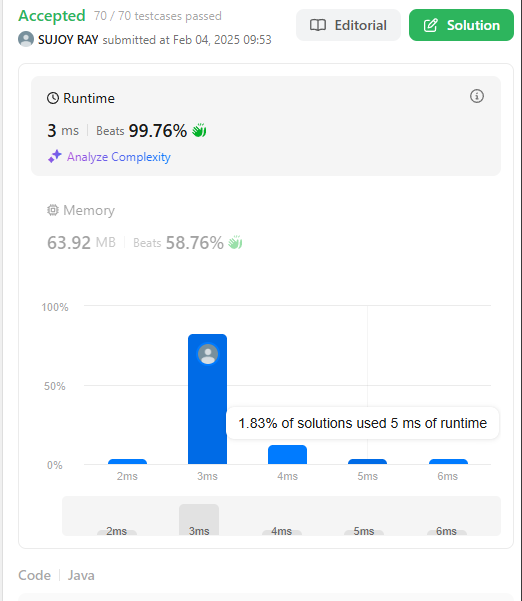
1. Remove Duplicates from Sorted Array
   * class Solution {
   * public int removeDuplicates(int[] nums) {
   * int j = 1;
   * for(int i = 1; i<nums.length; i++){
   * if(nums[i]!=nums[i-1]){
   * nums[j] = nums[i];
   * j++;
   * }
   * }
   * return j;
   * }
   * }
   * 
2. Contains Duplicate
   * class Solution {
   * public boolean containsDuplicate(int[] nums) {
   * Set<Integer> countset=new HashSet<>();
   * for(int a:nums){
   * countset.add(a);
   * }
   * if(countset.size()==nums.length){
   * return false;
   * }else{
   * return true;
   * }
   * }
   * }
   * 
3. Two Sum
   * import java.util.Arrays;
   * class Solution {
   * public int[] twoSum(int[] nums, int target) {
   * int[][] temp = new int[nums.length][2];
   * for (int i = 0; i < nums.length; i++) {
   * temp[i][0] = nums[i];
   * temp[i][1] = i;
   * }
   * Arrays.sort(temp, (a, b) -> Integer.compare(a[0], b[0]));
   * int left = 0;
   * int right = nums.length - 1;
   * while (left < right) {
   * int sum = temp[left][0] + temp[right][0];
   * if (sum == target) {
   * return new int[]{temp[left][1], temp[right][1]};
   * } else if (sum > target) {
   * right--;
   * } else {
   * left++;
   * }
   * }
   * return new int[]{};
   * }
   * }
   * 
4. Jump Game
   * class Solution {
   * public:
   * bool canJump(vector<int>& nums) {
   * int farthest=0;
   * for(int i=0;i<nums.size();i++){
   * if(i>farthest){
   * return false;
   * }
   * farthest=max(farthest,i+nums[i]);
   * }
   * return true;
   * }
   * };
   * 
5. Jump Game 2
   * class Solution {
   * public:
   * int jump(vector<int>& nums) {
   * int n=nums.size();
   * if (n<=1) return 0;
   * int jumps=0;
   * int currentEnd=0;
   * int furthest=0;
   * for (int i=0;i<n-1;++i){
   * furthest=max(furthest,i+nums[i]);
   * if (i==currentEnd){
   * ++jumps;
   * currentEnd=furthest;
   * if(currentEnd>=n-1) break;
   * }
   * }
   * return jumps;
   * }
   * };
   * 
6. Majority Element
   * class Solution {
   * public:
   * int majorityElement(vector<int>& nums) {
   * map<int,int> counter;
   * for(int a :nums){
   * counter[a]++;
   * }
   * int majority=0;
   * int maxCount=0;
   * for (auto a:counter){
   * if(a.second>maxCount){
   * maxCount = a.second;
   * majority = a.first;
   * }
   * }
   * return majority;
   * }
   * };
   * // class Solution {
   * // public:
   * //     int majorityElement(vector<int>& nums) {
   * //         int count=0;
   * //         int candidate=-1;
   * //         for(int num:nums){
   * //             if(count==0){
   * //                 candidate=num;
   * //             }
   * //             if(num==candidate){
   * //                 count++;
   * //             }else{
   * //                 count--;
   * //             }
   * //         }
   * //         return candidate;
   * //     }
   * // };
   * 
7. Longest Substring Without Repeating Characters
   * import java.util.HashMap;
   * class Solution {
   * public int lengthOfLongestSubstring(String s) {
   * HashMap<Character, Integer> charIndexMap = new HashMap<>();
   * int maxLength = 0;
   * int start = 0;
   * for (int end = 0; end < s.length(); end++) {
   * char currentChar = s.charAt(end);
   * if (charIndexMap.containsKey(currentChar) && charIndexMap.get(currentChar) >= start) {
   * start = charIndexMap.get(currentChar) + 1;
   * }
   * charIndexMap.put(currentChar, end);
   * maxLength = Math.max(maxLength, end - start + 1);
   * }
   * return maxLength;
   * }
   * }
   * 
8. Remove Duplicates from Sorted List
   * ListNode\* deleteDuplicates(ListNode\* head) {
   * ListNode\* current = head;
   * while (current && current->next) {
   * if (current->val == current->next->val) {
   * current->next = current->next->next;
   * } else {
   * current = current->next;
   * }
   * }
   * return head;
   * }
   * 
9. Reverse a Linked List
   * /\*\*
   * \* 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) {
   * ListNode node=null;
   * while(head!=null){
   * ListNode temp=head.next;
   * head.next=node;
   * node=head;
   * head=temp;
   * }
   * return node;
   * }
   * }
   * 
10. Delete Middle Node of a List
    * class Solution {
    * public ListNode deleteMiddle(ListNode head) {
    * if (head == null || head.next == null) return null;
    * ListNode slow = head, fast = head, prev = null;
    * while (fast != null && fast.next != null) {
    * fast = fast.next.next;
    * prev = slow;
    * slow = slow.next;
    * }
    * prev.next = slow.next;
    * return head;
    * }
    * }
    * 