Date-19th Nov 2024: O(N )

1. Next Permutation

class Solution {

    public void nextPermutation(int[] nums) {

        int n = nums.length;

        int idx = -1;

        for(int i = n-2 ; i>=0 ; i--) {

            if(nums[i] < nums[i+1]) {

                idx = i;

                break;

            }

        }

        if(idx == -1) {

            Arrays.sort(nums, 0, n);

            return;

        }

        for(int i=n-1 ; i>idx ; i--) {

            if(nums[i] > nums[idx]) {

                int temp = nums[i];

                nums[i] = nums[idx];

                nums[idx] = temp;

                break;

            }

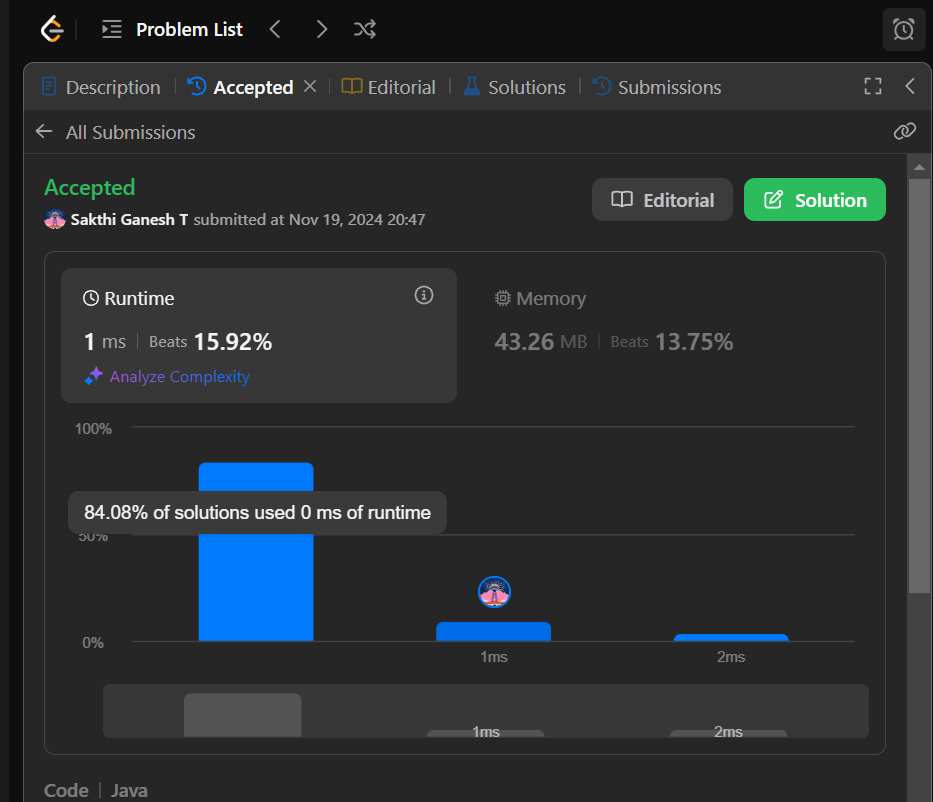
        }

        Arrays.sort(nums, idx+1, n);

    }

}

Output:



1. Spiral Matrix: O(N)

class Solution {

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

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

        int n = arr.length;

        int m = arr[0].length;

        int start\_row = 0;

        int end\_row = n - 1;

        int start\_col = 0;

        int end\_col = m - 1;

        while (start\_row <= end\_row && start\_col <= end\_col) {

            for (int j = start\_col; j <= end\_col; j++) {

                integerList.add(arr[start\_row][j]);

            }

            for (int i = start\_row + 1; i <= end\_row; i++) {

                integerList.add(arr[i][end\_col]);

            }

            if (start\_row < end\_row) {

                for (int j = end\_col - 1; j >= start\_col; j--) {

                    integerList.add(arr[end\_row][j]);

                }

            }

            if (start\_col < end\_col) {

                for (int i = end\_row - 1; i >= start\_row + 1; i--) {

                    integerList.add(arr[i][start\_col]);

                }

            }

            start\_col++;

            start\_row++;

            end\_col--;

            end\_row--;

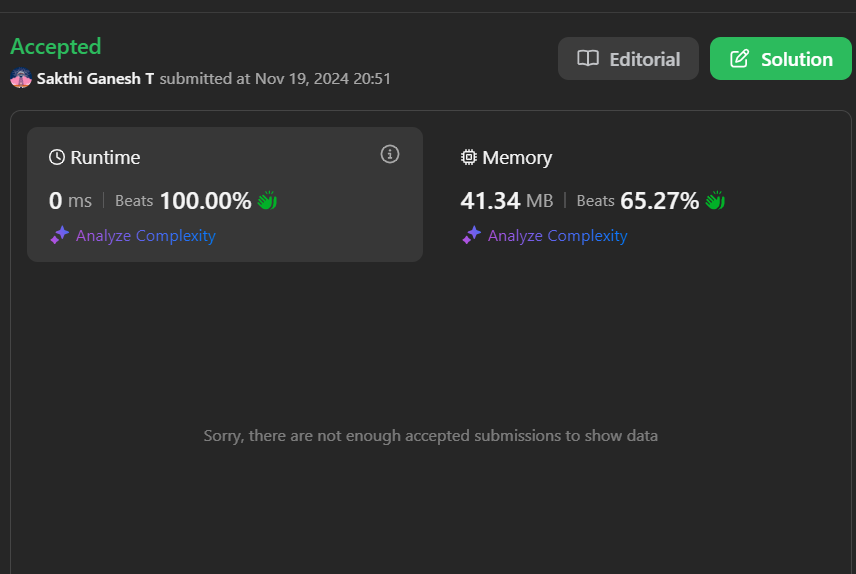
        }

        return integerList;

    }

}

Output:



3.Longest Substring without Repeating Characters: O(N)

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++;

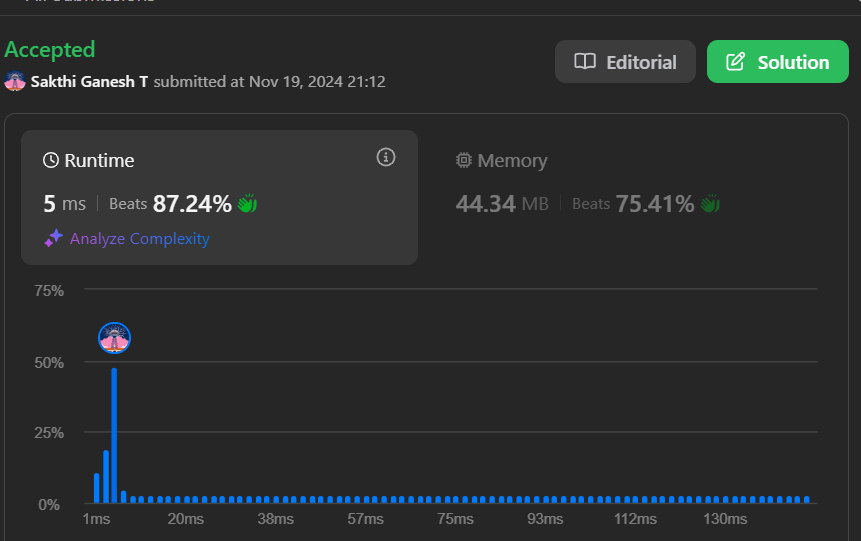
        }

        return len;

    }

}

Output:



4.Remove element from a LinkedList: O(N)

class Solution {

    public ListNode removeElements(ListNode head, int val) {

        ListNode dummy=new ListNode(-1);

        dummy.next=head;

        ListNode curr =head,prev=dummy;

        while(curr!=null)

        {

            if(curr.val==val)

            {

                prev.next=curr.next;

            }

            else{

                prev=prev.next;

            }

            curr=curr.next;

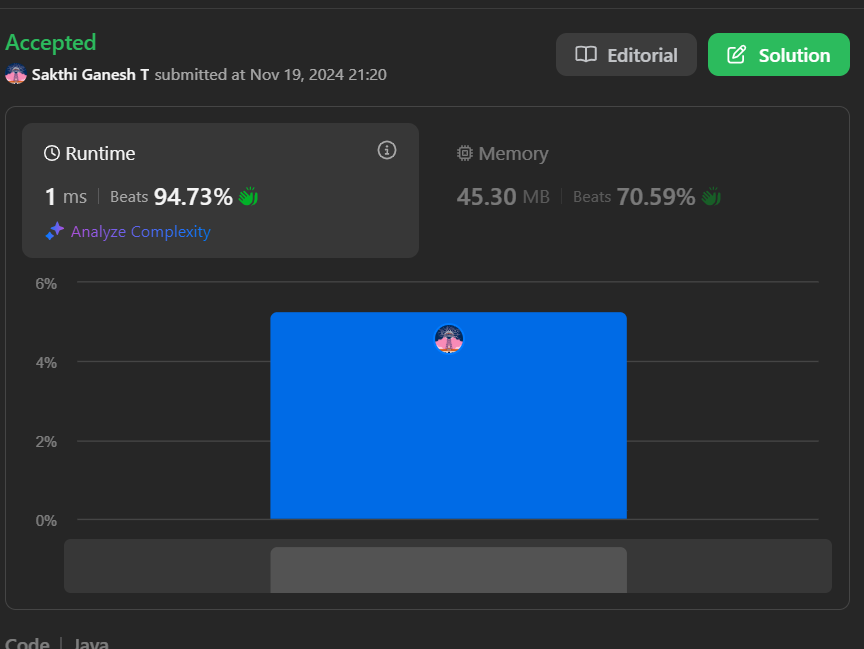
        }

        return dummy.next;

    }

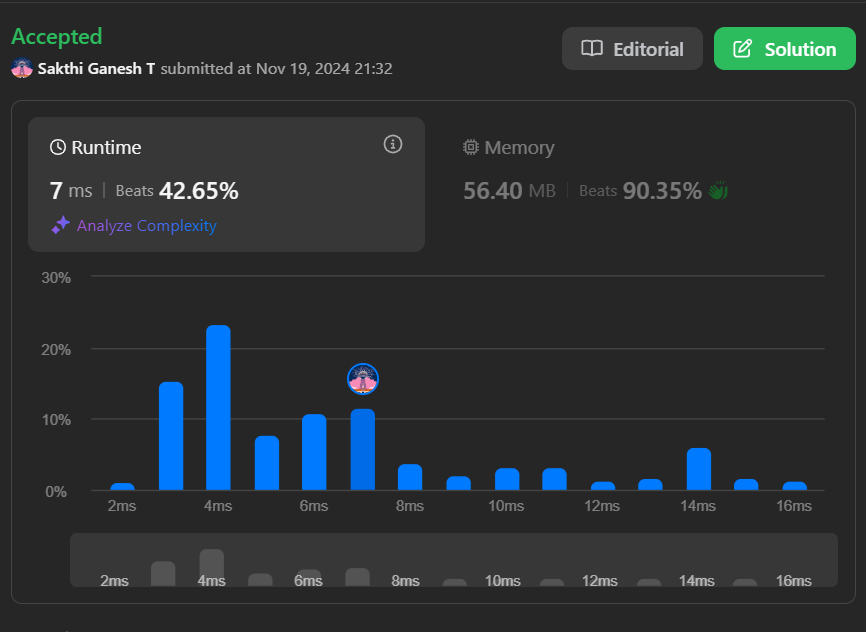
}

Output:



1. Palindrome Linked List: (O(N))
2. class Solution {
3. public boolean isPalindrome(ListNode head) {
4. ArrayList<Integer> a=new ArrayList<>();
5. while(head!=null)
6. {
7. a.add(head.val);
8. head=head.next;
9. }
10. int i=0,j=a.size()-1;
11. boolean same=true;
12. while(i<j)
13. {
14. if(a.get(i)!=a.get(j))
15. {
16. same=false;
17. break;
18. }
19. i++;
20. j--;
21. }
22. return same;
24. }
25. }

Output:



6. Minimum Path Sum O(N^2)

class Solution {

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

        int m=grid.length;

        int n=grid[0].length;

        for(int i=1;i<m;i++)

        {

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

        }

        for(int i=1;i<n;i++)

        {

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

        }

        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]);

            }

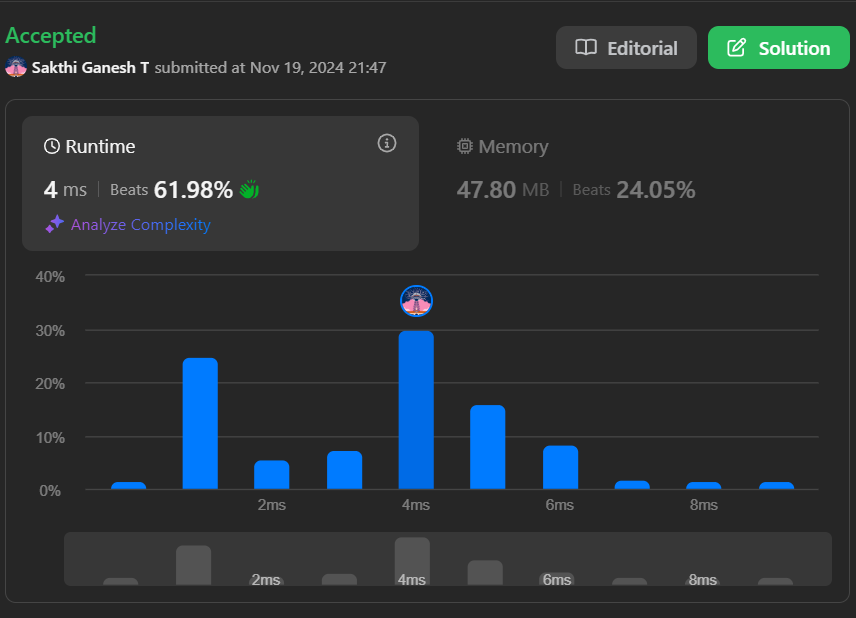
        }

        return grid[m-1][n-1];

    }

}

Output:



7.Validate Binary Search Tree: O(N^2)

class Solution {

    public boolean isValidBST(TreeNode root) {

           if (root == null) return true;

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

   TreeNode pre = null;

   while (root != null || !stack.isEmpty()) {

      while (root != null) {

         stack.push(root);

         root = root.left;

      }

      root = stack.pop();

      if(pre != null && root.val <= pre.val) return false;

      pre = root;

      root = root.right;

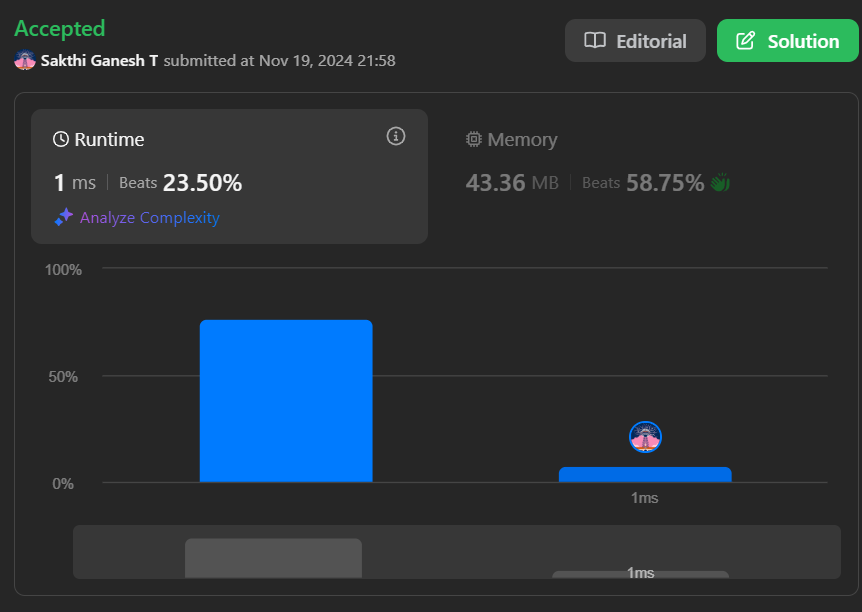
   }

   return true;

    }

}

Output:



7. Word Ladder(O(N^2))

class Solution {

    public int ladderLength(String beginWord, String endWord, List<String> wordList) {

        Set<String> myset = new HashSet<>(wordList);

        if (!myset.contains(endWord))

            return 0;

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

        q.add(beginWord);

        int depth = 0;

        while (!q.isEmpty())

        {

            depth++;

            int qsize = q.size();

            for (int i = 0; i < qsize; i++)

            {

                String t = q.poll();

                char[] curr = t.toCharArray();

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

                {

                    char orig = curr[j];

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

                    {

                        if (c == orig)

                            continue;

                        curr[j] = c;

                        String temp = new String(curr);

                        if (temp.equals(endWord))

                            return depth + 1;

                        if (myset.contains(temp))

                        {

                            q.add(temp);

                            myset.remove(temp);

                        }

                    }

                    curr[j] = orig;

                }

            }

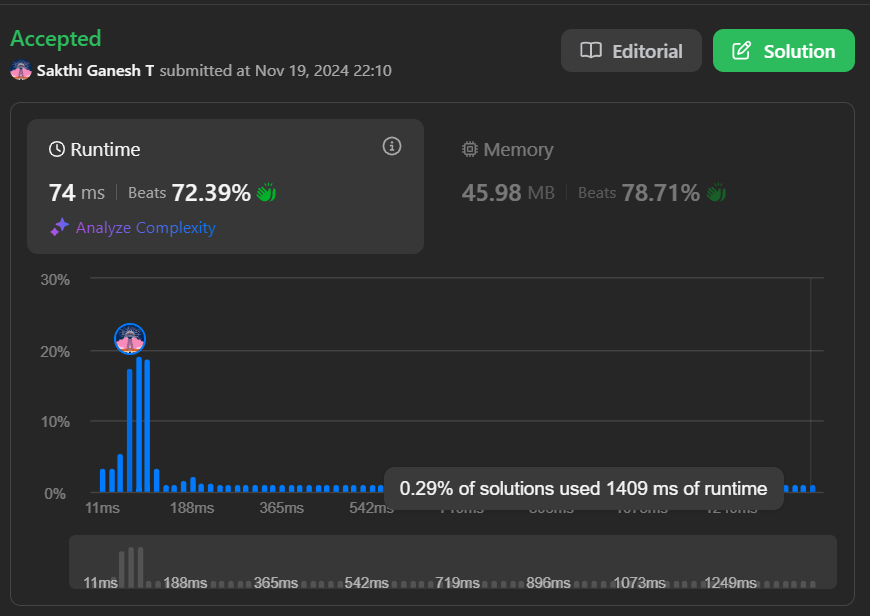
        }

        return 0;

    }

}

Output:



8. Word Ladder 2: O(N^2)

class Solution {

    Set<String> set = new HashSet();

    String beginWord, endWord;

    Map<String, Integer> dist = new HashMap();

    List<List<String>> res;

    public List<List<String>> findLadders(String beginWord, String endWord, List<String> wordList) {

        this.beginWord = beginWord;

        this.endWord = endWord;

        this.res = new ArrayList();

        for (String word : wordList) {

            set.add(word);

        }

        short\_path();

        if (dist.get(endWord) == null) return res;

        List<String> path = new ArrayList();

        path.add(endWord);

        dfs(endWord, path);

        return res;

    }

    private void short\_path() {

        Queue<String> q = new LinkedList();

        q.offer(beginWord);

        dist.put(beginWord, 0);

        while(q.size() > 0) {

            String cur = q.poll();

            if (cur.equals(endWord) ) break;

            char[] charCur = cur.toCharArray();

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

                char c = charCur[i];

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

                    charCur[i] = j;

                    String s = new String(charCur);

                    if (set.contains(s) && dist.get(s) == null) {

                        dist.put(s, dist.get(cur) + 1);

                        q.offer(s);

                    }

                }

                charCur[i] = c;

            }

        }

    }

    private void dfs(String word, List<String> path) {

        if (word.equals(beginWord)) {

            List list = new ArrayList(path);

            Collections.reverse(list);

            res.add(list);

            return;

        }

        char[] charCur = word.toCharArray();

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

            char c = charCur[i];

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

                charCur[i] = j;

                String s = new String(charCur);

                if (dist.get(s) != null && dist.get(s) + 1 == dist.get(word)) {

                    path.add(s);

                    dfs(s, path);

                    path.remove(path.size() - 1);

                }

            }

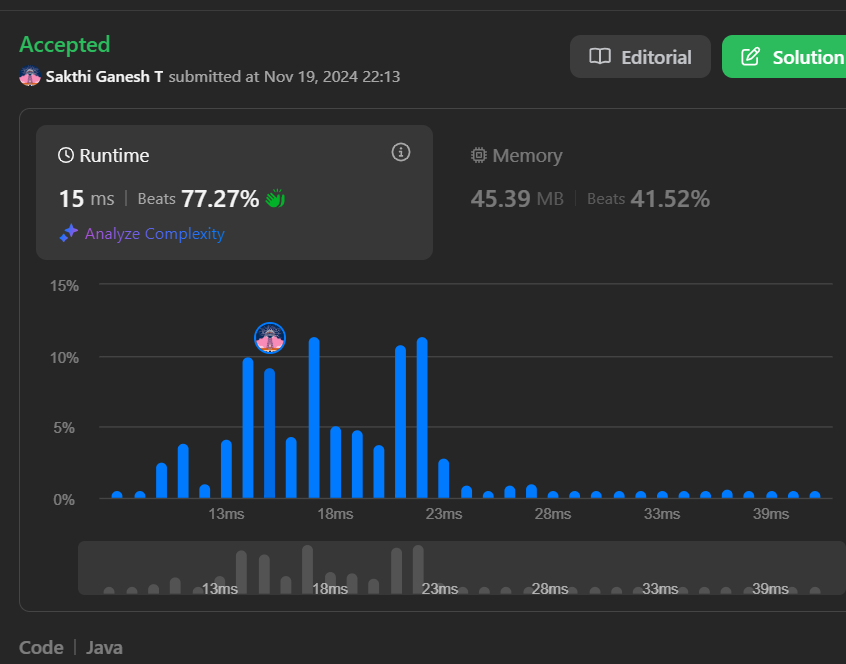
            charCur[i] = c;

        }

    }

}

Output:



9. Course Scheduler: O(N)

class Solution {

    boolean flag = true;

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

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

        Map<Integer, List<Integer>> hs = new HashMap<>();

        for (int[] pre : prerequisites) {

            int c = pre[0];

            int prereqCourse = pre[1];

            hs.putIfAbsent(c, new ArrayList<>());

            hs.get(c).add(prereqCourse);

        }

        System.out.println(hs);

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

            if (!s.contains(i)) {

                dfs(s, i, hs, new HashSet<>());

            }

        }

        return flag;

    }

    public void dfs(Set<Integer> s, int curr, Map<Integer, List<Integer>> hs, Set<Integer> s2) {

        if (s2.contains(curr)) {

            flag = false;

            return;

        }

        if (!hs.containsKey(curr)) {

            return;

        }

        s2.add(curr);

        for (int prereqCourse : hs.get(curr)) {

            if (!s.contains(prereqCourse)) {

                dfs(s, prereqCourse, hs, s2);

            }

        }

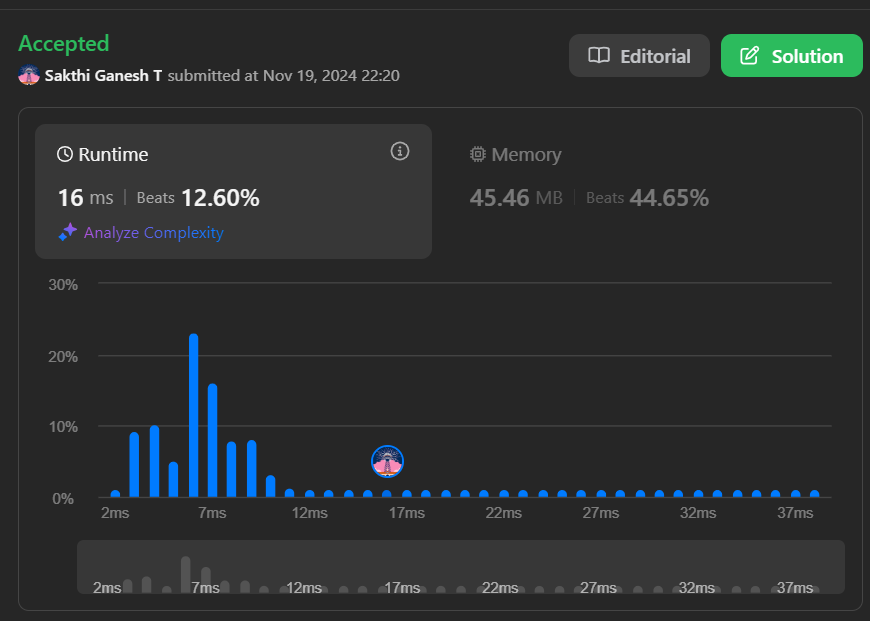
        s2.remove(curr);

        s.add(curr);

    }

}

Output:



=