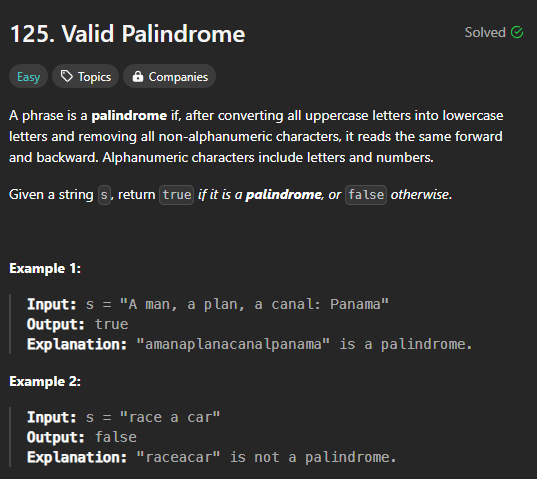
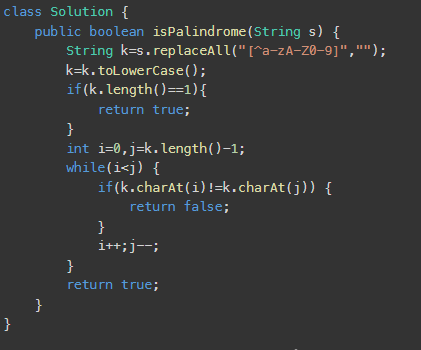
DSA Problems:

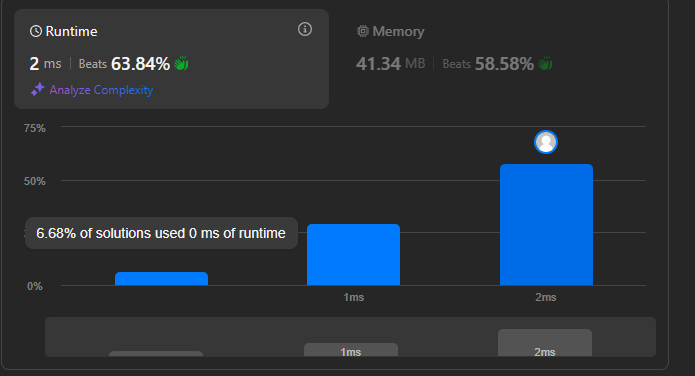
1. 

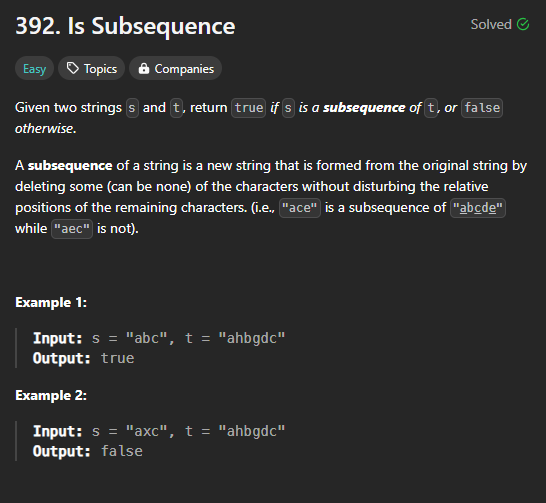
Code:



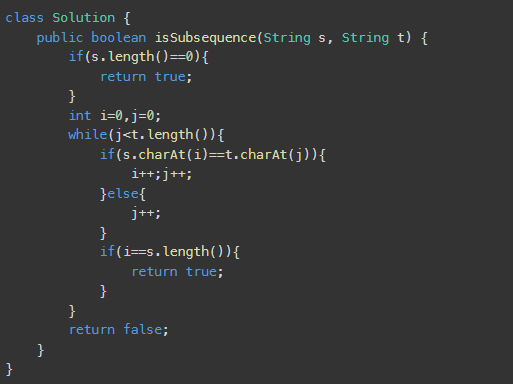
Time Complexity:

O(n)

Output: 

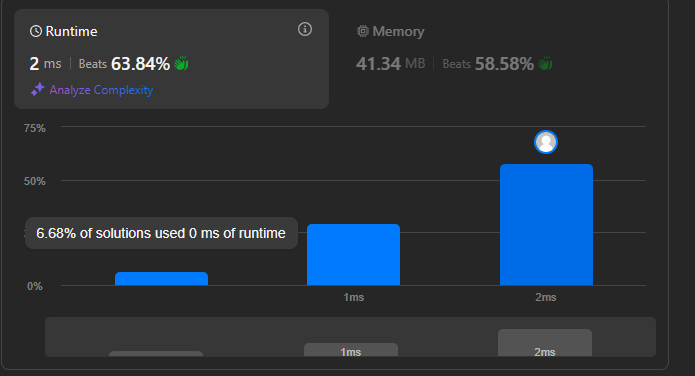
1. 

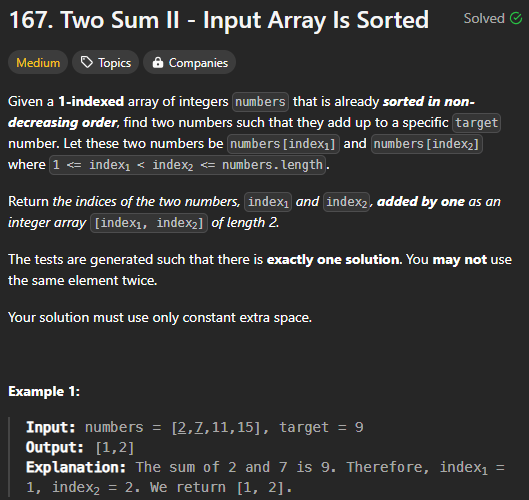
Code:



Time Complexity:O(n)

Output:

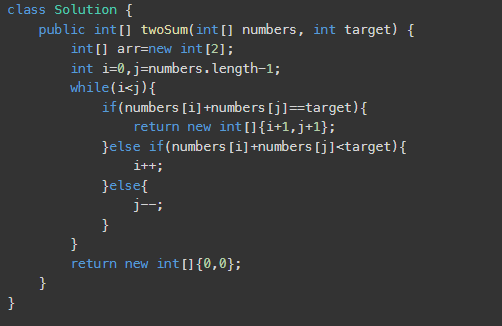


1. 

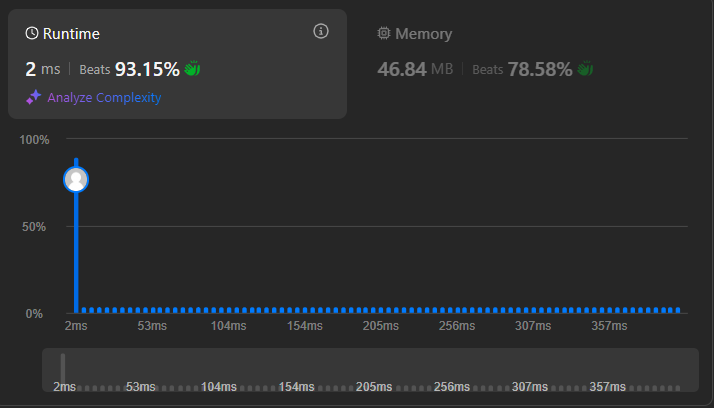
Time Complexity:

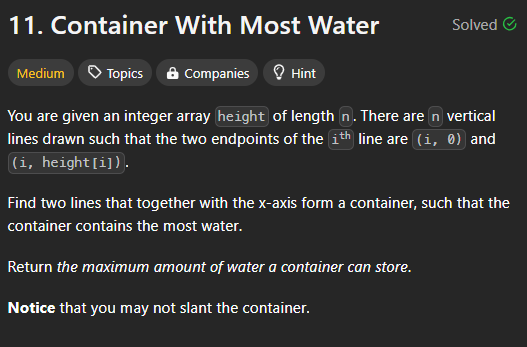
O(n)

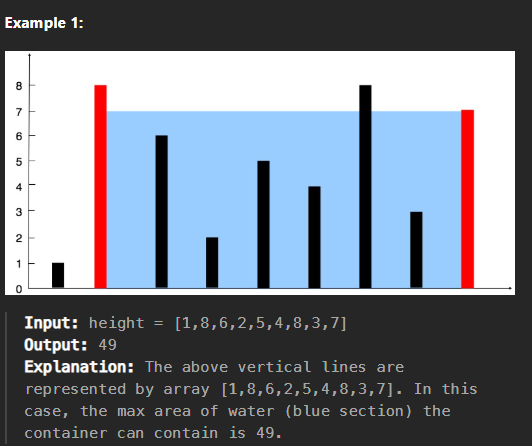
Code:



Output:



1. 



Code:

class Solution {

public int maxArea(int[] height) {

if(height.length == 2) return Math.min(height[0],height[1]);

int max = 0;

for(int i : height){

if(i > max){

max = i;

}

}

int i = 0;

int j = height.length - 1;

int [] values = new int[max + 1];

int levelIndex = 0;

while (i < j){

if(height[i] <= levelIndex){

i++;

}else if(height[j] <= levelIndex){

j--;

}else{

values[levelIndex++] = (j - i) \* levelIndex;

}

}

int answer = 0;

for(int temp : values){

if (temp > answer) answer = temp;

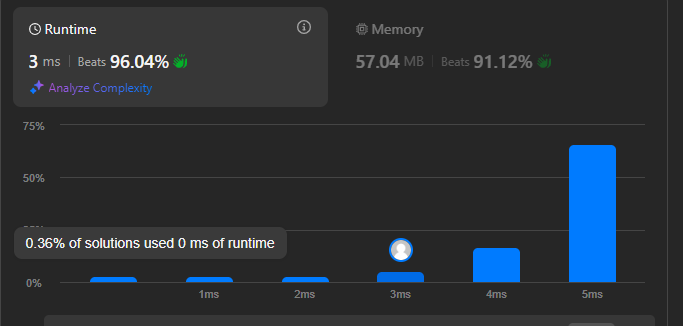
}

return answer;

}

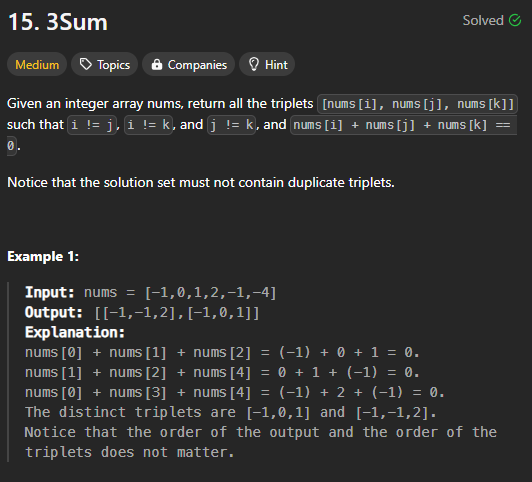
}

Output:



Time Complexity:

O(n\*m)🡺 O(n)

1. 

Code:

import java.util.\*;

class Solution {

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

Arrays.sort(arr);

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

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

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

continue;

}

int j = i + 1;

int k = arr.length - 1;

while (j < k) {

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

if (sum == 0) {

list.add(Arrays.asList(arr[i], arr[j], arr[k]));

while (j < k && arr[j] == arr[j + 1]) {

j++;

}

while (j < k && arr[k] == arr[k - 1]) {

k--;

}

j++;

k--;

} else if (sum > 0) {

k--;

} else {

j++;

}

}

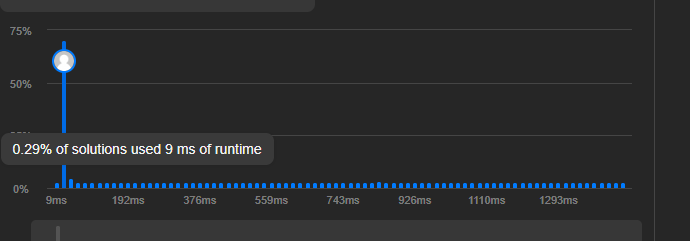
}

return list;

}

}

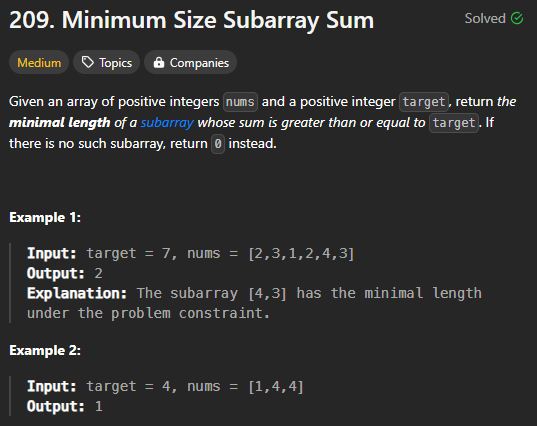
Output:



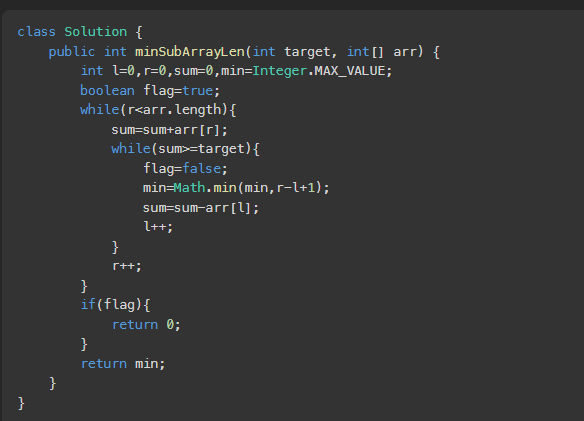
TC:

O(n2)

8.

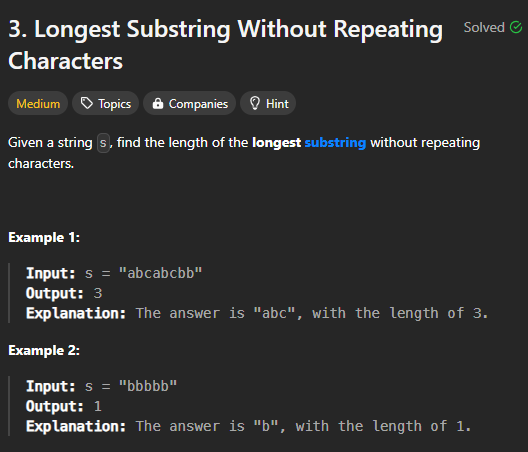


Code:

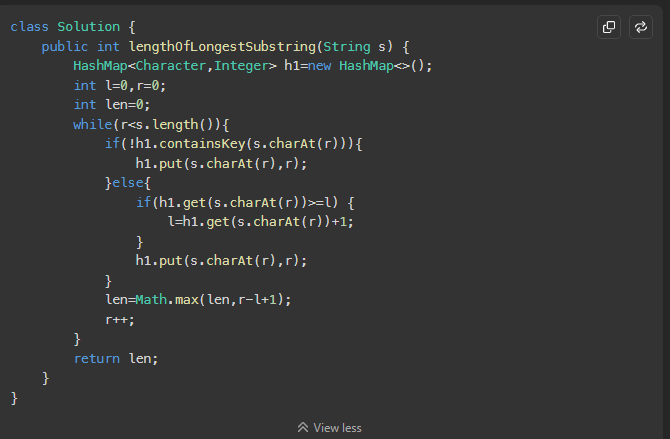


TC:O(n)

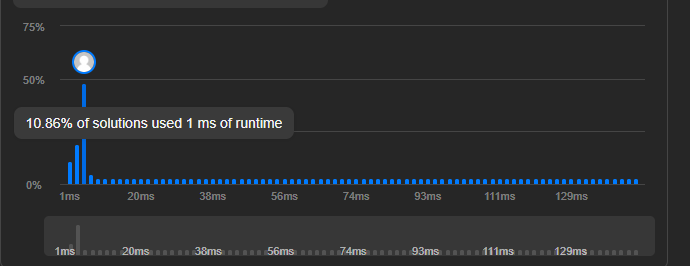
8.



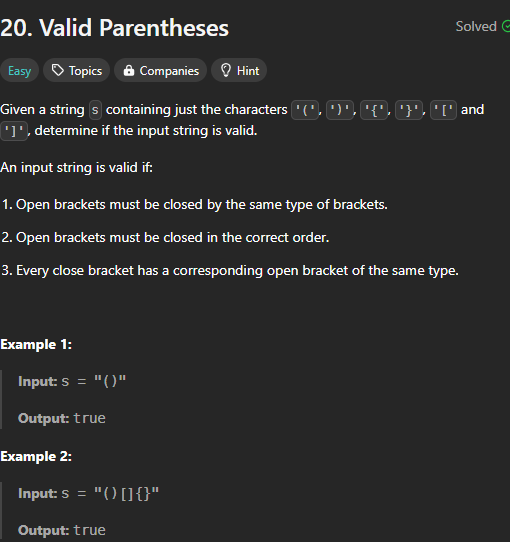
Code:



Output:



TC:O(n)

9. 

Code: class Solution {

public boolean isValid(String s) {

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

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

char ch=s.charAt(i);

if(ch =='(' || ch=='{' || ch=='['){

stack.push(ch);

}else{

if(stack.isEmpty()){

return false;

}

if((ch==')' && stack.peek()!='(') || (ch=='}' && stack.peek()!='{') || (ch==']' && stack.peek()!='[')){

return false;

}else{

stack.pop();

}

}

}

System.out.println(stack);

if(stack.isEmpty()){

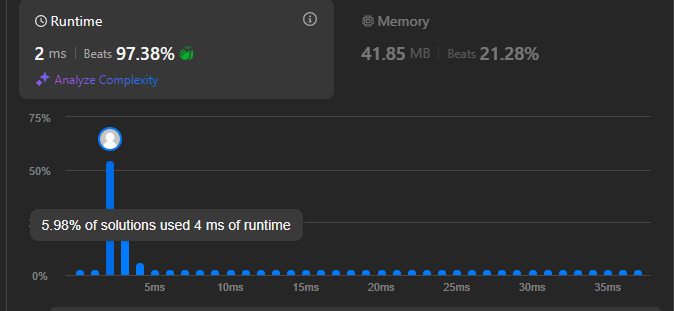
return true;

}

return false;

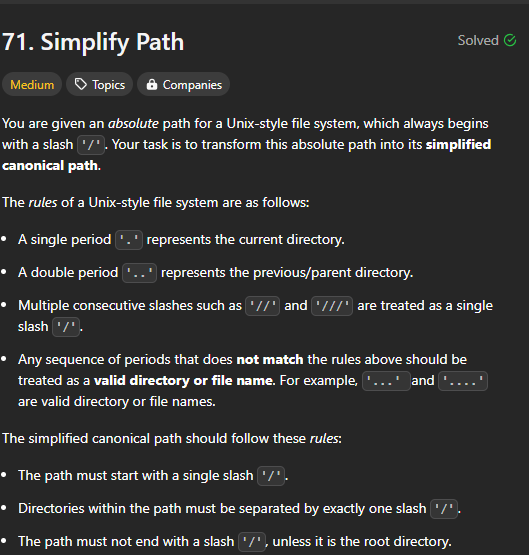
}

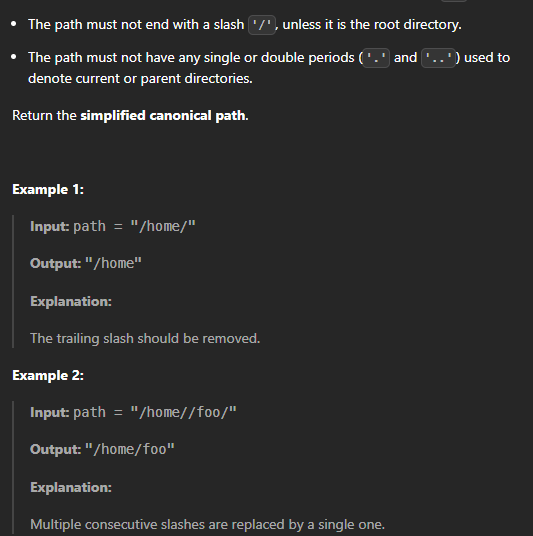
}

Output: 

TC:O(n)

10.





Code:

class Solution {

public String simplifyPath(String s) {

String k=s.replaceAll("//","/");

String str=k.replaceAll("///","/");

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

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

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

if(arr[i]!="") {

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

if(!st.isEmpty()) {

st.pop();

}else {

continue;

}

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

continue;

}else {

st.add(arr[i]);

}

}

}

StringBuilder ans = new StringBuilder();

for (String dir : st) {

ans.append("/").append(dir);

}

if (ans.length() == 0) {

ans.append("/");

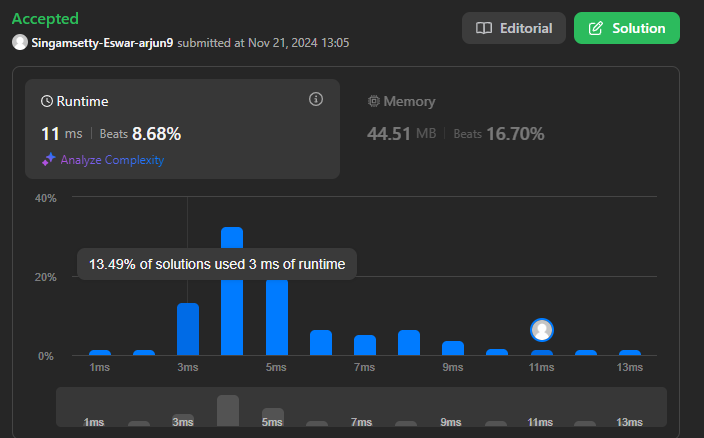
}

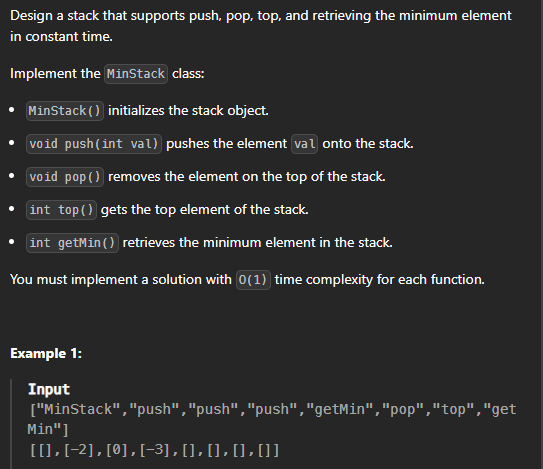
return ans.toString();

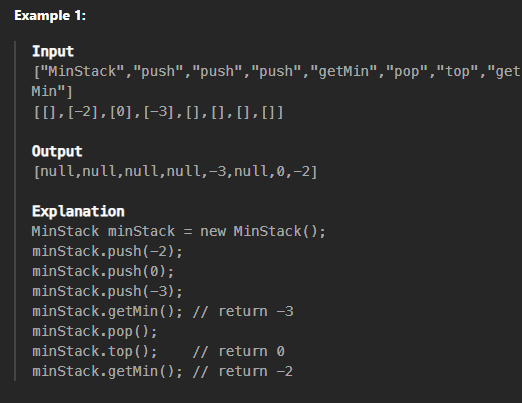
}

}

Output:



11. 



Code:

class MinStack {

LinkedList<TplusMin> stack;

private class TplusMin {

int val;

int min;

public TplusMin(int val, int min) {

this.val = val;

this.min = min;

}

}

public MinStack() {

stack = new LinkedList<>();

}

public void push(int val) {

int newMin;

if (stack.size() == 0){

newMin = val;

}

else {

int currentMin = stack.getFirst().min;

newMin = val < currentMin ? val : currentMin;

}

stack.addFirst(new TplusMin(val, newMin));

}

public void pop() {

stack.removeFirst();

}

public int top() {

return stack.peekFirst().val;

}

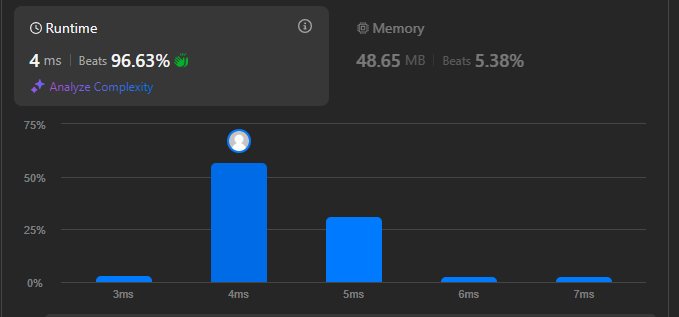
public int getMin() {

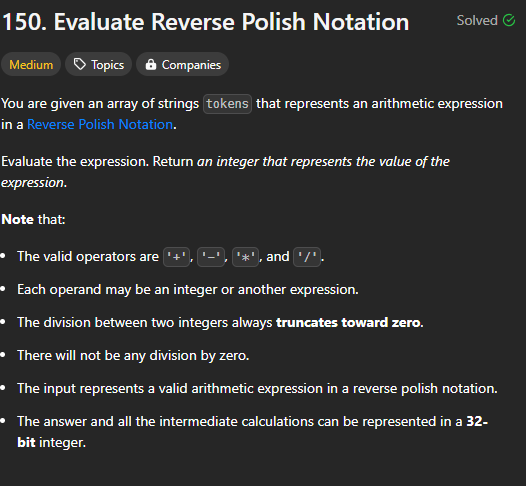
return stack.peekFirst().min;

}

}

Output:



12 

Code:

class Solution {

public int evalRPN(String[] tokens) {

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

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

if(tokens[i].equals("+")){

int n=st.pop();

int m=st.pop();

st.push(n+m);

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

int n=st.pop();

int m=st.pop();

st.push(m-n);

}else if(tokens[i].equals("\*")){

int n=st.pop();

int m=st.pop();

st.push(n\*m);

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

int n=st.pop();

int m=st.pop();

st.push(m/n);

}else{

st.push( Integer.valueOf(tokens[i]));

}

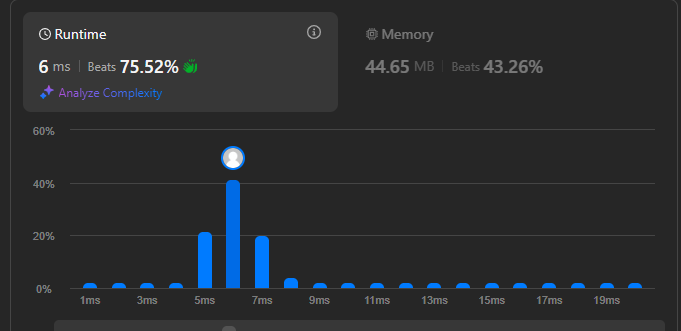
}

return st.peek();

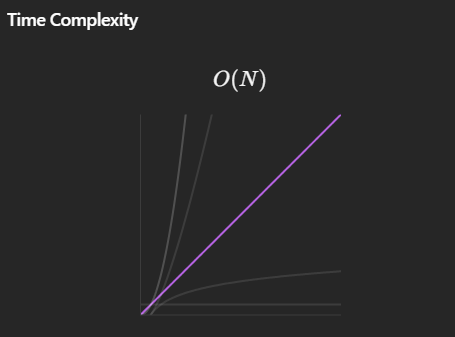
}

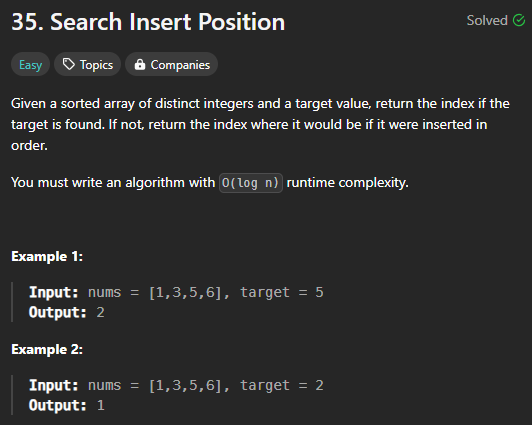
}

Output:

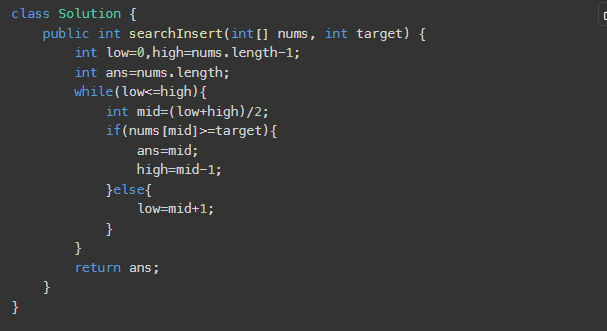


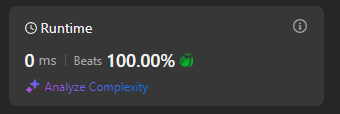
Tc:



13. 

Code:

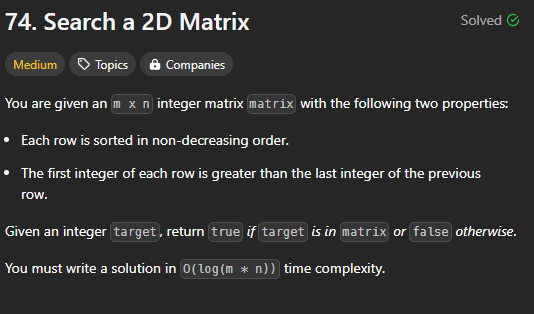
Output:



TC:

O(log n)

13.



Code:

class Solution {

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

        int n=arr.length;

        int m=arr[0].length;

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

            if(arr[i][0]<=target && target<=arr[i][m-1]) {

                int low=0;

                int high=m-1;

                while(low<=high) {

                    int mid=(low+high)/2;

                    if(arr[i][mid]==target) {

                        return true;

                    }else if(target<=arr[i][mid]) {

                        high=mid-1;

                    }else {

                        low=mid+1;

                    }

                }

            }

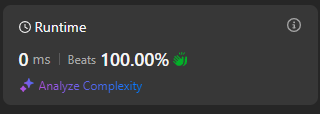
        }

        return false;

    }

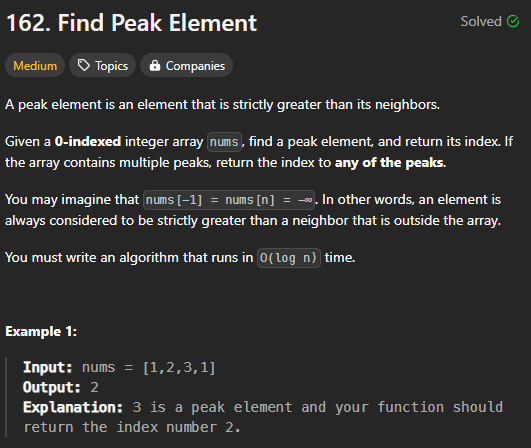
}

Output:



TC:O(log n)

14.



Code:

class Solution {

public int findPeakElement(int[] arr) {

int low=1,high=arr.length-2;

if(arr.length==1) {

return 0;

}else if(arr[0]>arr[1]){

return 0;

}else if(arr[arr.length-1]>arr[arr.length-2]){

return arr.length-1;

}

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

if(arr[i]>arr[i+1] && arr[i]>arr[i-1]){

return i;

}

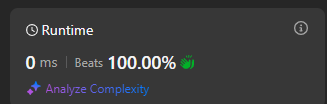
}

return 0;

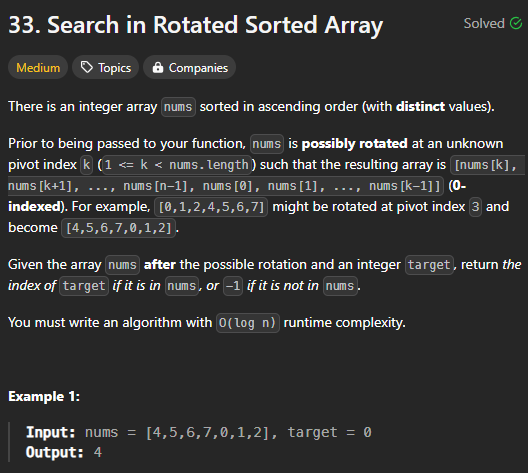
}

}

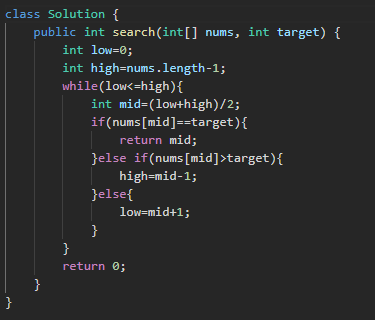
Output:



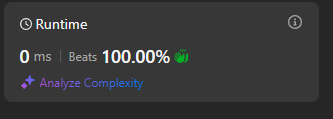
15



Code:

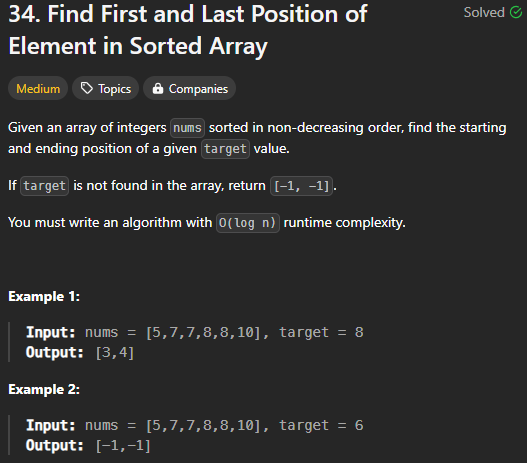


Output:



TC:O(log n)

16.



Code:

class Solution {

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

        if(nums.length==0){

            return new int[]{-1,-1};

        }

        int minIndex=-1;

        int low=0,high=nums.length-1;

        while(low<=high){

            int mid=(low+high)/2;

            if(nums[mid]>=target){

                if(nums[mid]==target){

                    minIndex=mid;

                }

                high=mid-1;

            }else{

                low=low+1;

            }

        }

        int i=0;

        if(minIndex==-1){

            return new int[] {-1,-1};

        }else{

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

                if(nums[i]!=target){

                    return new int[]{minIndex,i-1};

                }

            }

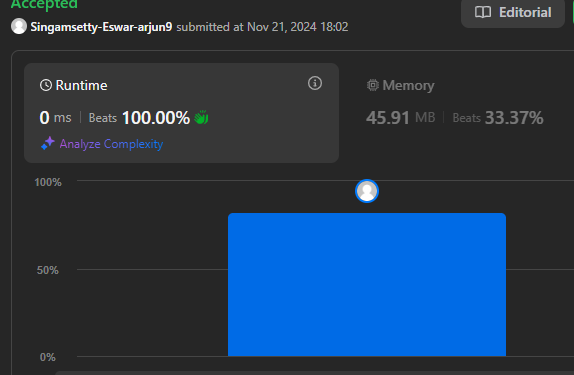
        }

        return new int[] {minIndex,i-1};

    }

}

Output:



TC:O(n)