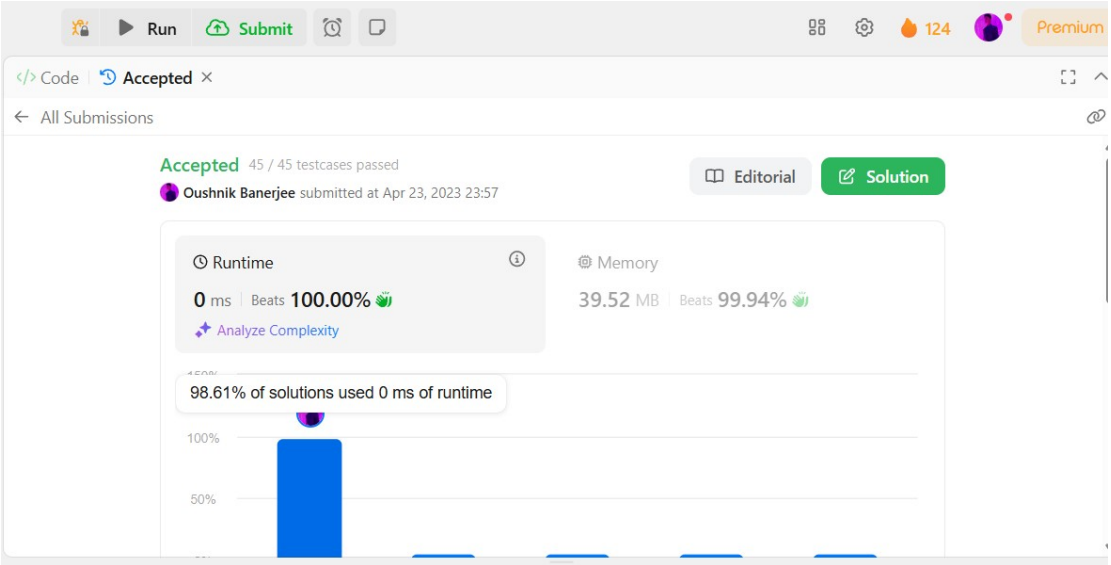
Experiment 8

Climbing Stairs

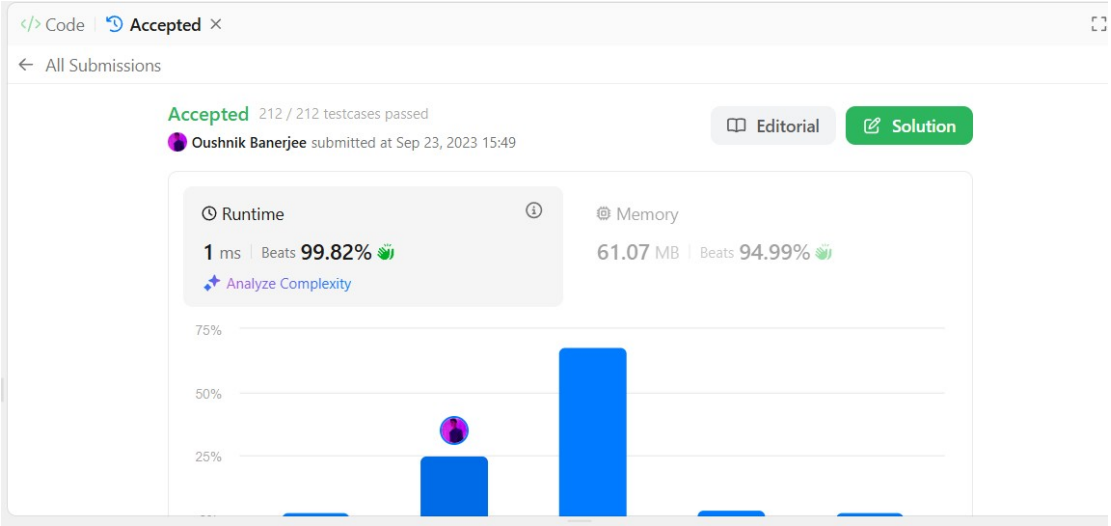
```
class Solution {
    public int climbStairs(int n) {
        int dp[]=new int[n+1];
        dp[0]=1;
        if(n==1) return 1;
        dp[1]=1;
        // if(n==2) return 2;
        for(int i=2;i<=n;i++){
            dp[i]=dp[i-1]+dp[i-2];
        }
        return dp[n];
    }
}
```



Best Time to Buy and Sell a Stock

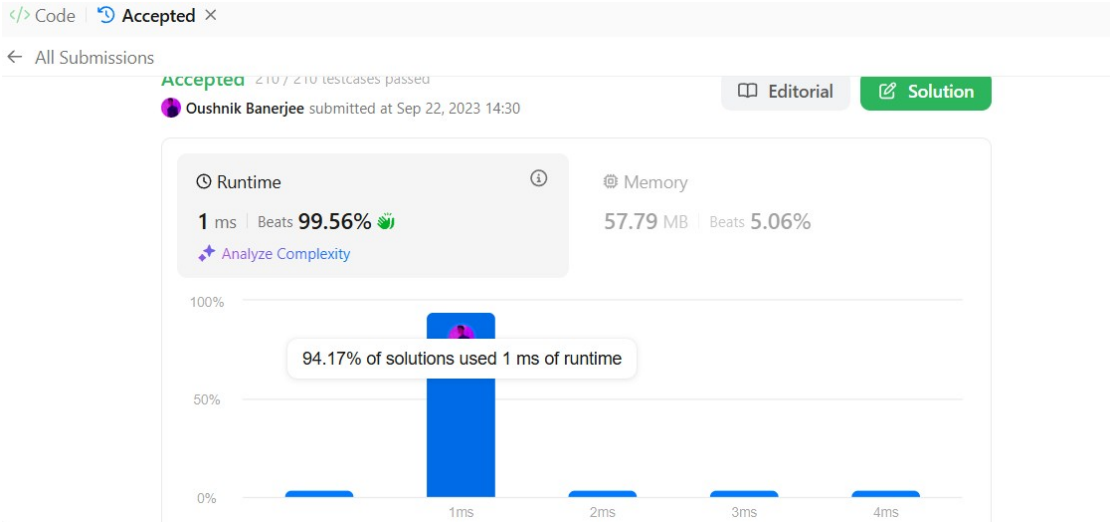
```
class Solution {
    public int maxProfit(int[] prices) {
        int min= prices[0];
        int profit=0;
        for(int i=1; i< prices.length; i++){
            if(min< prices[i] ){
                profit= Math.max(profit, prices[i]- min);
            }
            else{
                min= prices[i];
            }
        }
        return profit;
```

```
    }
}
```



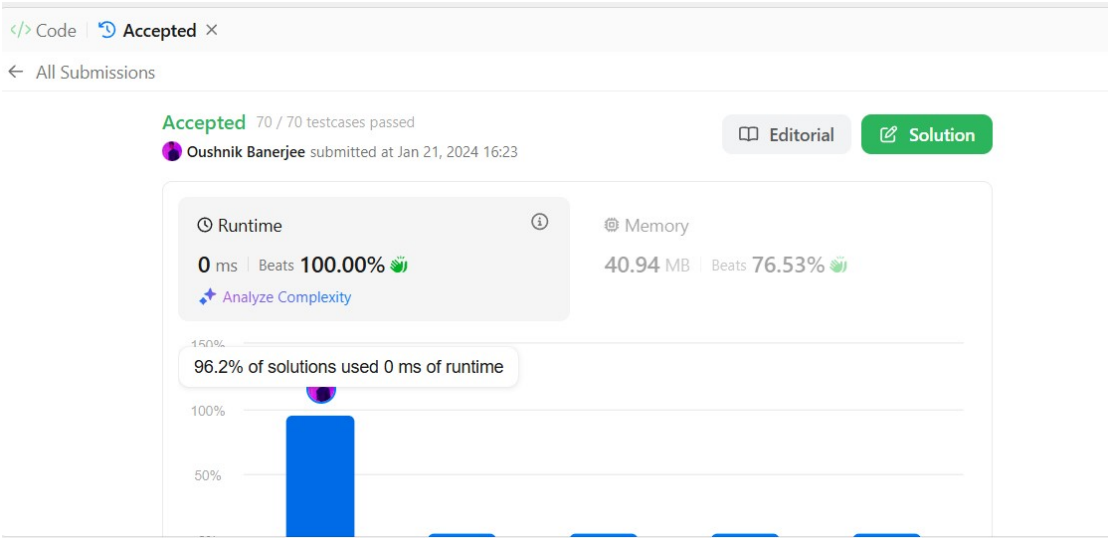
Maximum Subarray

```
class Solution {
    public int maxSubArray(int[] nums) {
        int max= Integer.MIN_VALUE;
        int sum= 0;
        for(int i=0; i< nums.length; i++){
            sum= sum+ nums[i];
            max= Math.max(max, sum);
            if(sum<0){
                sum= 0;
            }
        }
        return max;
    }
}
```



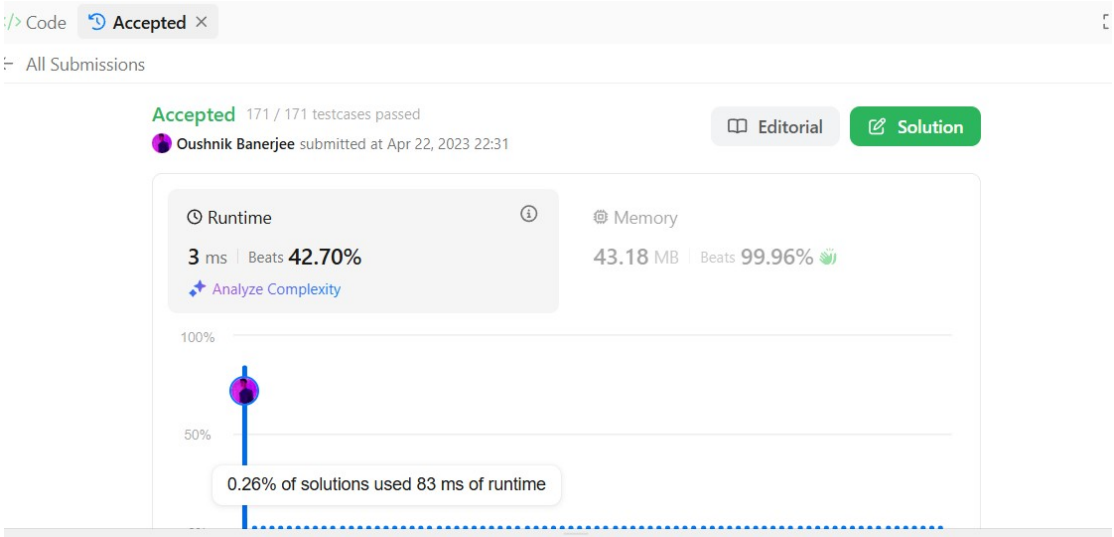
House Robber

```
class Solution {
   public int rob(int[] nums) {
      if(nums.length==0) return 0;
      int prev1=0;
      int prev2=0;
      for(int num: nums){
         int temp= prev1;
         prev1= ((prev2+num)>prev1)?(prev2+num):prev1;
         prev2= temp;
      }
      return prev1;
   }
}
```



Jump Game

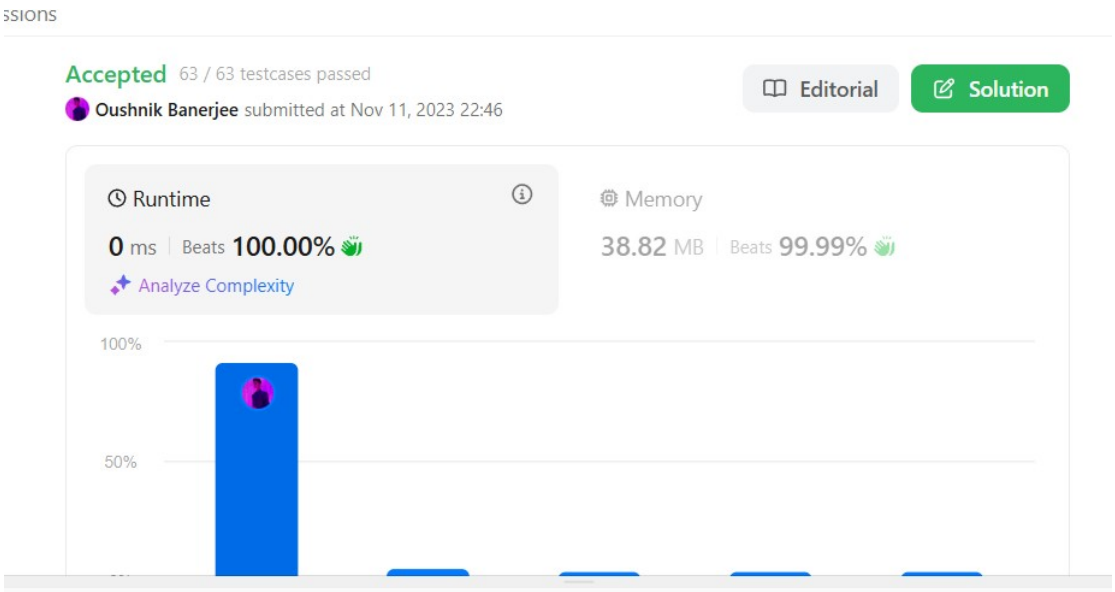
```
class Solution {
    public boolean canJump(int[] nums) {
        if(nums.length==1){
            return true;
        }
        int n= nums.length;
        int max=0;
        int curr=0;
        for(int i=0; i<n-1; i++){
            max= Math.max(max, nums[i]+ i);
            if(max<i+1){
                return false;
            }
        }
        return true;
    }
}
```



Unique Paths

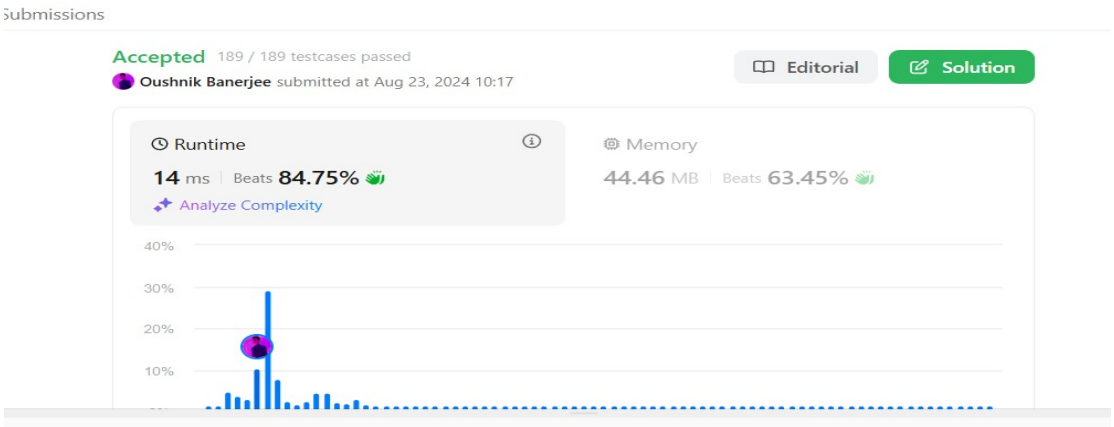
```
class Solution {
    public int uniquePaths(int m, int n) {
        int [][]dp= new int[m][n];

        for(int i=0; i<m; i++){
            for(int j=0; j<n; j++){
```

```
            if(i==0 ||j==0){
                dp[i][j]=1;
            }
            else
            dp[i][j]= dp[i-1][j]+ dp[i][j-1];
        }
    }
    return dp[m-1][n-1];
  }
}
```



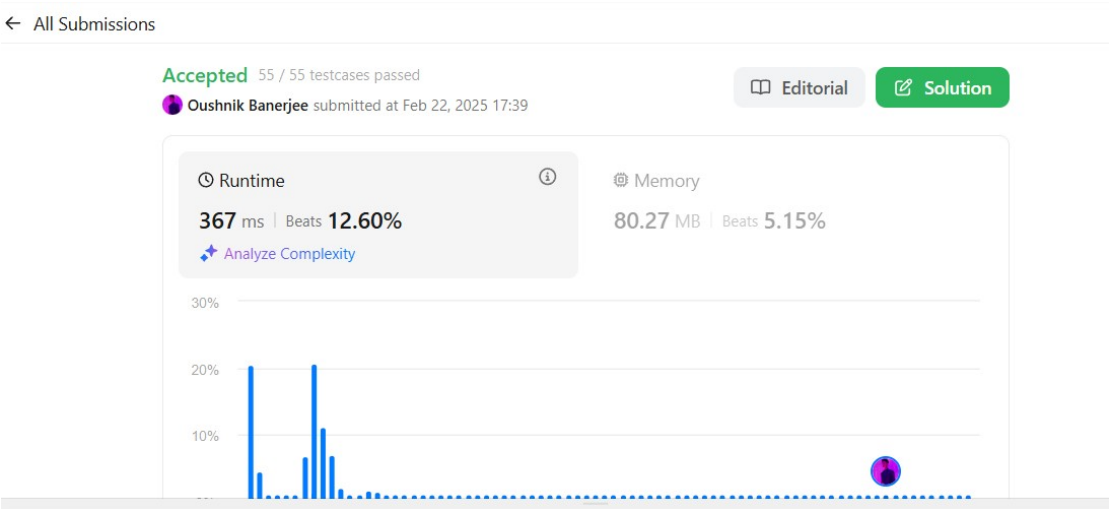
Coin Change

```
class Solution {
  public int coinChange(int[] coins, int amount) {
    int []dp= new int[amount+1];
    Arrays.fill(dp, amount+1);
    dp[0]=0;
    //int c=Integer.MAX_VALUE;
    for(int i=1;i<dp.length; i++){
        for(int coin:coins){
            if(i-coin>=0){
                dp[i]= Math.min(dp[i], dp[i-coin]+1);
            }
        }
    }
    return dp[amount]>amount?-1:dp[amount];
  }
}
```



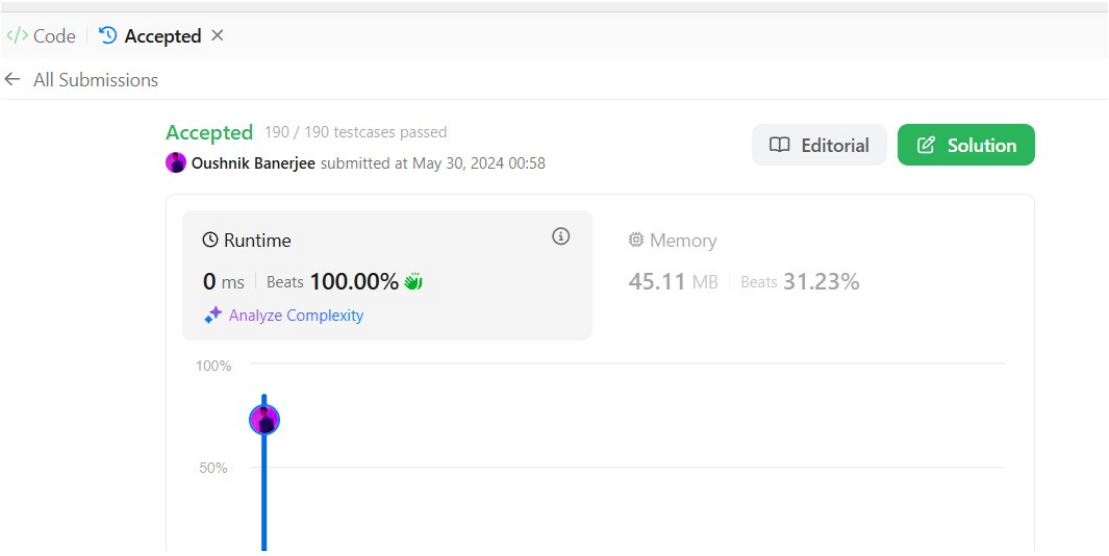
Longest Increasing Subsequence

```
class Solution {
    public int lengthOfLIS(int[] nums) {
        int [][]dp= new int[nums.length][nums.length+1];
        for(int []r: dp) Arrays.fill(r, -1);
        return find(nums, dp, 0, -1);
    }
    int find(int []nums, int [][]dp, int i, int j){
        if(i==dp.length) return 0;
        if(dp[i][j+1]!=-1) return dp[i][j+1];
        int t1= find(nums, dp, i+1, j);
        int t2=0;
        if(j==-1 || nums[i]>nums[j]){
            t2= 1+ find(nums, dp, i+1, i);
        }
        dp[i][j+1]=Math.max(t1,t2);
        return dp[i][j+1];
    }
}
```



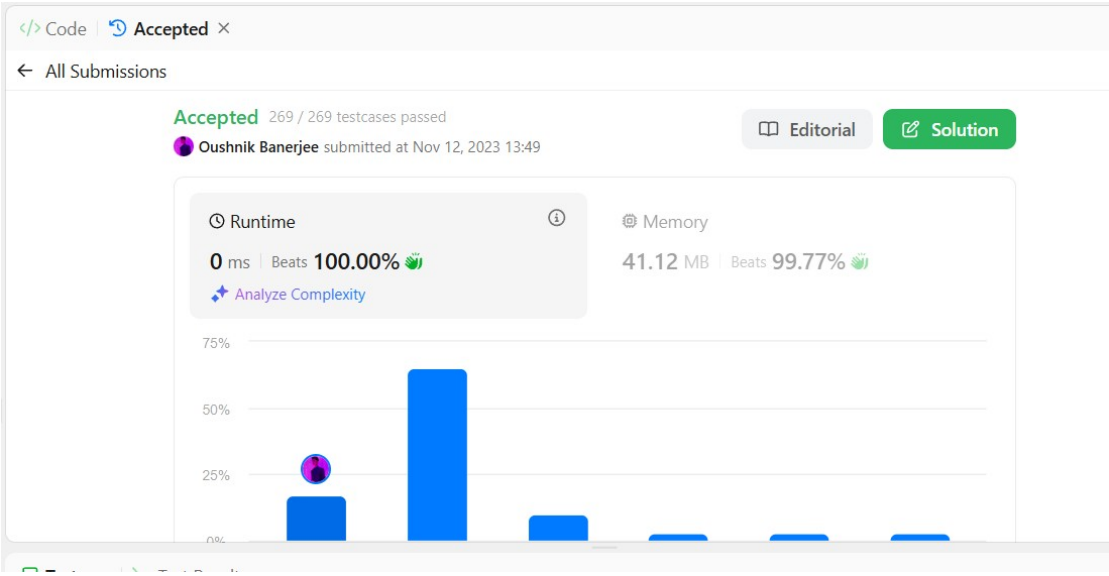
Maximum Product Subarray

```
class Solution {
   public int maxProduct(int[] nums) {
      int res=1;
      int max1=Integer.MIN_VALUE;
      for(int i=0; i<nums.length; i++){
         res=res*nums[i];
         if(max1<res){
            max1=res;
         }
         if(res==0){
            res=1;
         }
      }
      res=1;
      int max2= Integer.MIN_VALUE;
      for(int i=nums.length-1; i>=0; i--){
         res= res*nums[i];
         if(max2<res){
            max2=res;
         }
         if(res==0){
            res=1;
         }
      }
      return max1>=max2?max1:max2;
   }
}
```



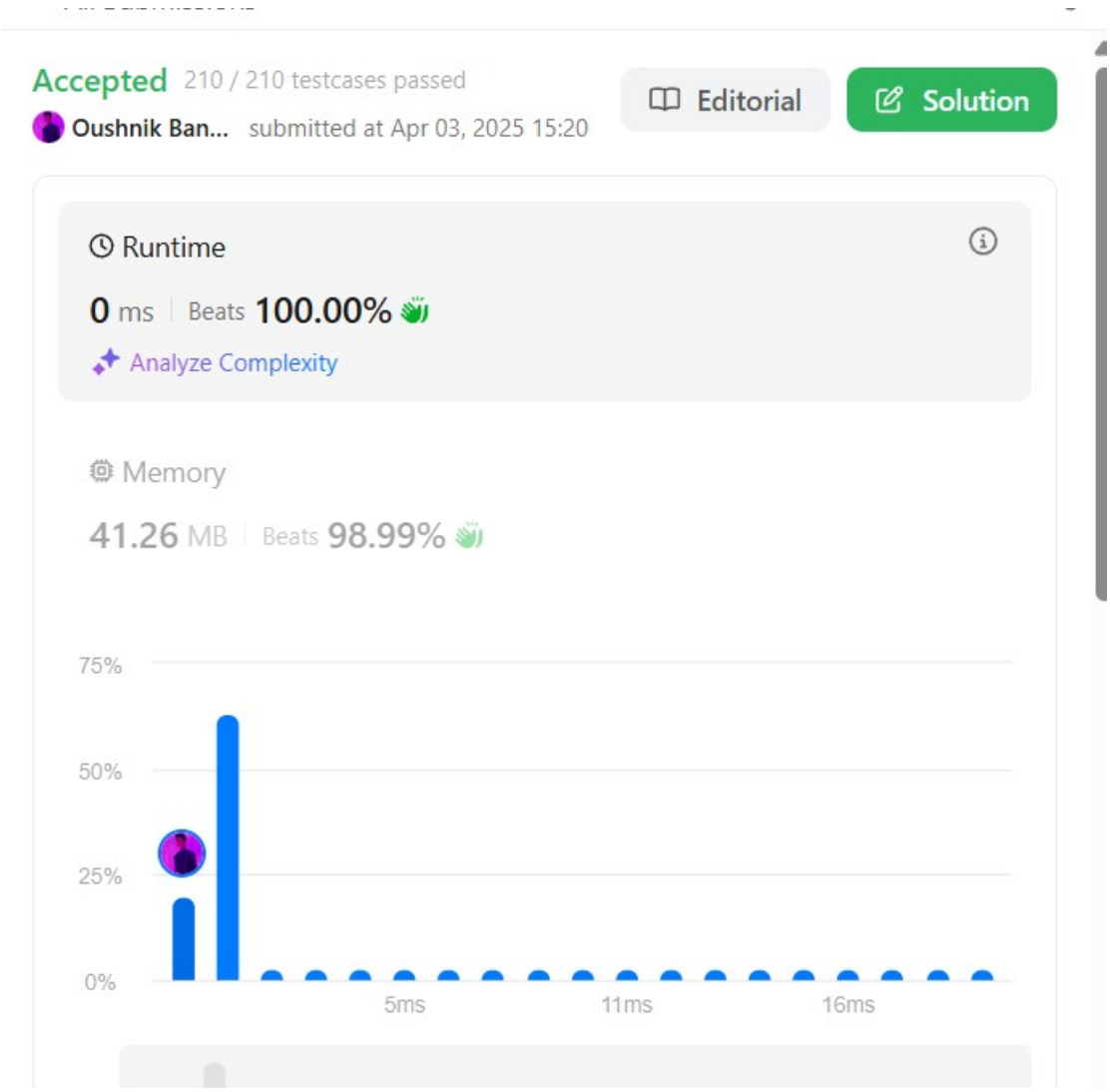
Decode Ways

```
class Solution {
   public int numDecodings(String s) {
      int dp1= 1, dp2=0;
      int n= s.length();
      for(int i=n-1; i>=0; i--){
         int dp= s.charAt(i)=='0'?0:dp1;
         if(i<n-1&&(s.charAt(i)=='1'|| s.charAt(i)=='2'&&
s.charAt(i+1)<'7')){
            dp= dp+dp2;
         }
         dp2=dp1;
         dp1= dp;
      }
      return dp1;
   }
}
```



Best Time to Buy and Sell a Stock with Cooldown

```
class Solution {
   public int maxProfit(int[] prices) {
   int sell = 0, prev_sell = 0, buy = Integer.MIN_VALUE, prev_buy;
   for (int price : prices) {
      prev_buy = buy;
      buy = Math.max(prev_sell - price, prev_buy);
      prev_sell = sell;
      sell = Math.max(prev_buy + price, prev_sell);
   }
   return sell;
}
}
```


Accepted 210 / 210 testcases passed
Oushnik Ban... submitted at Apr 03, 2025 15:20
Editorial
Solution
Runtime
0 ms | Beats 100.00%
Analyze Complexity
Memory
41.26 MB | Beats 98.99%
75%
50%
25%
0%
5ms
11ms
16ms