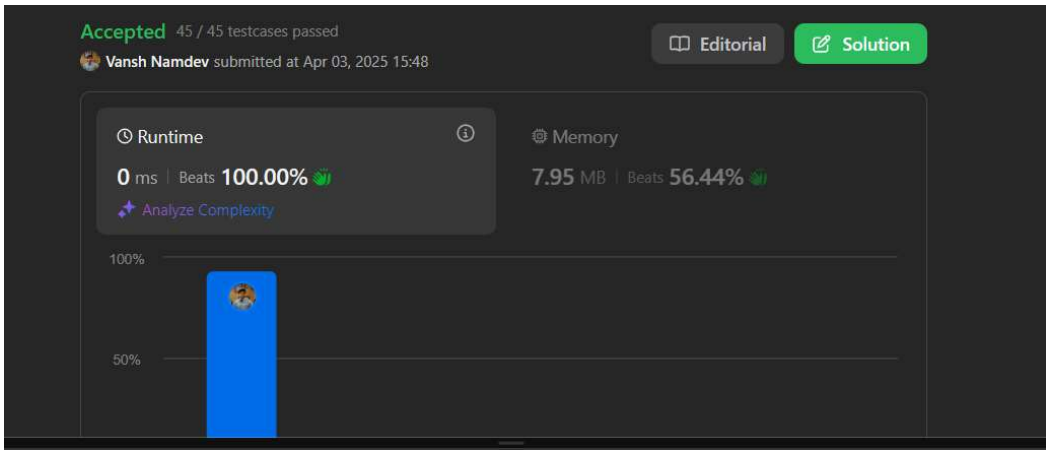
Experiment 7

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];
    }
}
```



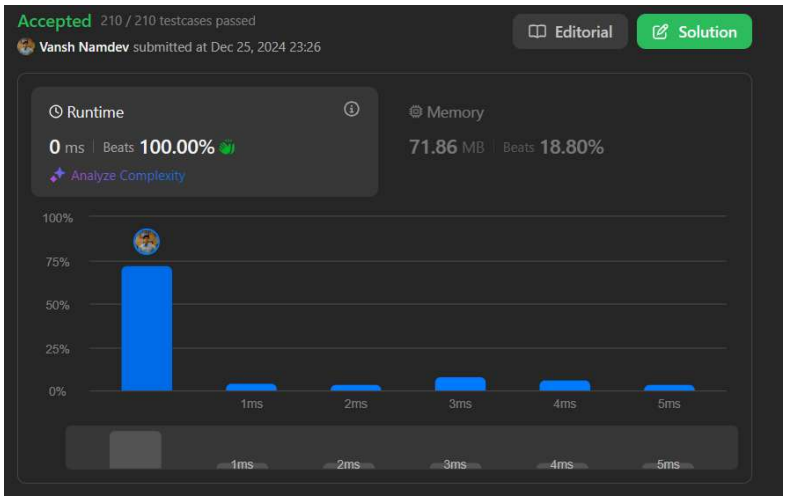
Best Time to Buy and Sell a Stock

```
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;
```

```
    }
}
```

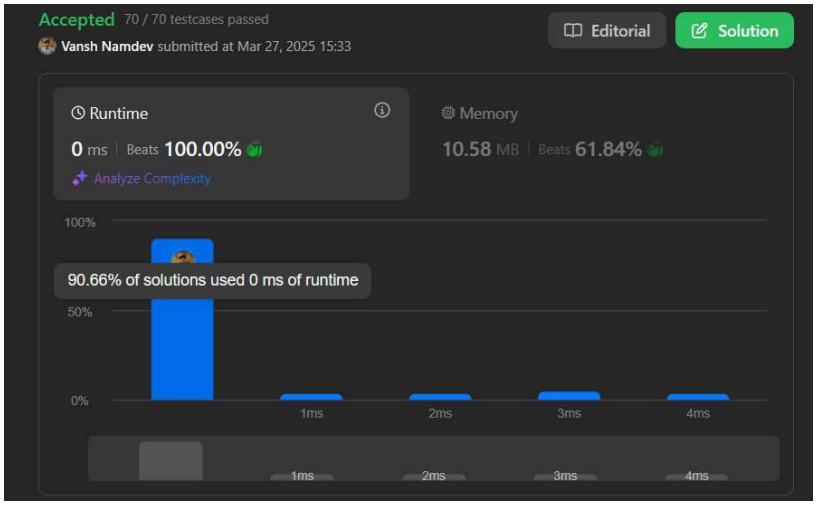
Maximum Subarray

```
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;
    }
}
```