

ASSIGNMENT 6

70. Climbing Stairs

```
class Solution {
    public int climbStairs(int n) {
        if (n <= 2) return n;

        int[] dp = new int[n + 1];
        dp[1] = 1;
        dp[2] = 2;

        for (int i = 3; i <= n; i++) {
            dp[i] = dp[i - 1] + dp[i - 2];
        }

        return dp[n];
    }
}
```

The screenshot displays the LeetCode submission interface for the 'Climbing Stairs' problem. The top section shows the problem description and a 'Solution' button. Below this, the 'Runtime' and 'Memory' sections provide performance metrics: 0 ms runtime (100.00% beats) and 40.64 MB memory (27.51% beats). A bar chart visualizes the runtime performance across different test cases. The 'Code' section shows the Java solution, and the 'Testcase' section displays the input and output for a specific test case.

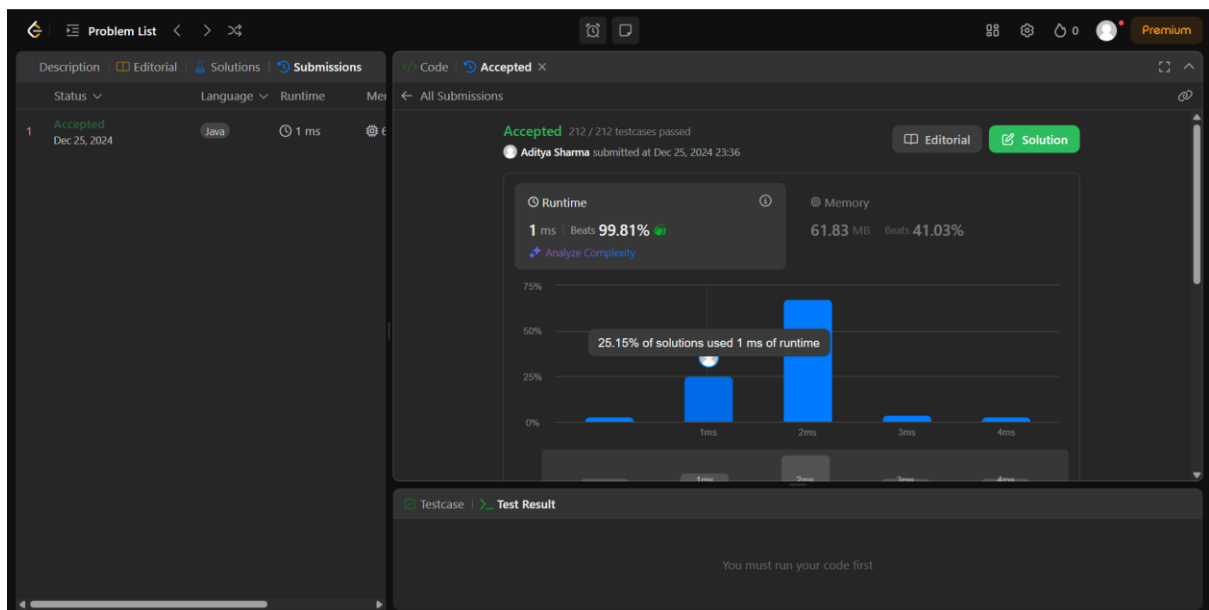
Runtime: 0 ms | Beats 100.00%
Memory: 40.64 MB | Beats 27.51%

Testcase: Runtime: 0 ms
Case 1:
Input: n = 2
Output: 2
Expected: 2

121. Best Time to Buy and Sell Stock

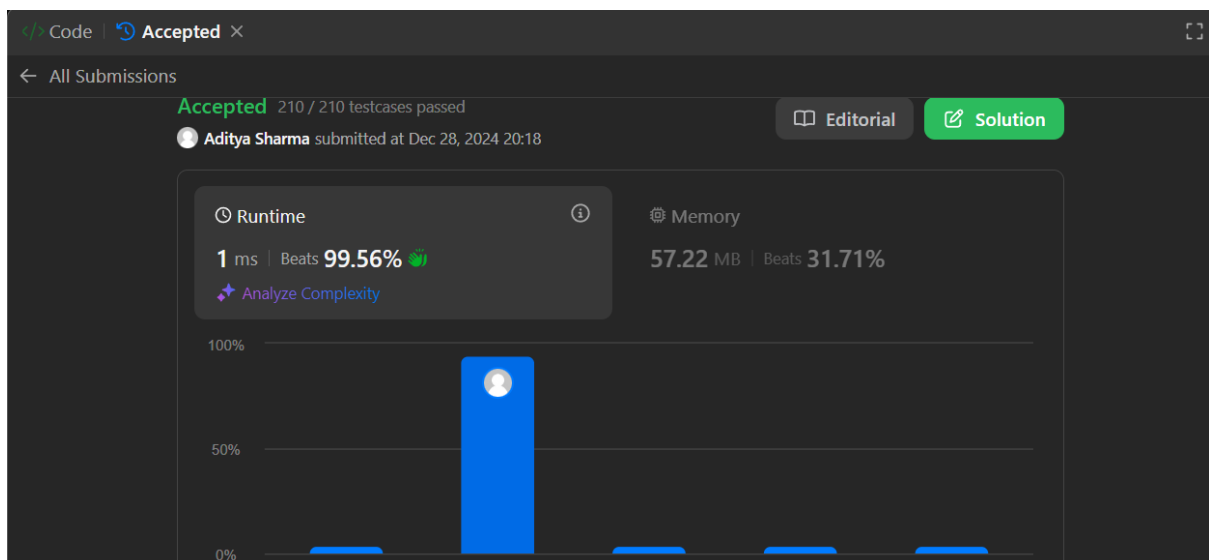
```
class Solution {
    public int maxProfit(int[] prices) {
        int lsf = Integer.MAX_VALUE;
        int op = 0;
        int pist = 0;

        for(int i = 0; i < prices.length; i++){
            if(prices[i] < lsf){
                lsf = prices[i];
            }
            pist = prices[i] - lsf;
            if(op < pist){
                op = pist;
            }
        }
        return op;
    }
}
```



53. Maximum Subarray

```
1. class Solution {
2.     public int maxSubArray(int[] nums) {
3.         int maxSum = Integer.MIN_VALUE;
4.         int currentSum = 0;
5.
6.         for (int i = 0; i < nums.length; i++) {
7.             currentSum += nums[i];
8.
9.             if (currentSum > maxSum) {
10.                 maxSum = currentSum;
11.             }
12.
13.             if (currentSum < 0) {
14.                 currentSum = 0;
15.             }
16.         }
17.
18.         return maxSum;
19.     }
20. }
```



62. Unique Paths

```
class Solution {
    public int uniquePaths(int m, int n) {
        // Create a 2D DP array filled with zeros
        int[][] dp = new int[m][n];

        // Initialize the rightmost column and bottom row to 1
        for (int i = 0; i < m; i++) {
            dp[i][n-1] = 1;
        }
        for (int j = 0; j < n; j++) {
            dp[m-1][j] = 1;
        }

        // Fill in the DP array bottom-up
        for (int i = m - 2; i >= 0; i--) {
            for (int j = n - 2; j >= 0; j--) {
                dp[i][j] = dp[i+1][j] + dp[i][j+1];
            }
        }

        // Return the result stored in the top-left corner
        return dp[0][0];
    }
}
```

The screenshot displays the LeetCode submission interface for the 'Unique Paths' problem. The left sidebar shows the problem status as 'Accepted' with 63/63 testcases passed, submitted at Apr 06, 2025 23:27. The 'Runtime' section indicates 0 ms and 100.00% beats, while the 'Memory' section shows 40.53 MB and 47.15% beats. A bar chart visualizes the runtime performance. The main area shows the Java code for the solution, which is a bottom-up dynamic programming approach. The 'Test Result' section at the bottom shows the solution is 'Accepted' with a runtime of 0 ms.

Runtime: 0 ms | Beats 100.00%
Memory: 40.53 MB | Beats 47.15%

Code:

```
9
10
11     for (int j = 0; j < n; j++) {
12         dp[m-1][j] = 1;
13     }
14
15     // Fill in the DP array bottom-up
16     for (int i = m - 2; i >= 0; i--) {
17         for (int j = n - 2; j >= 0; j--) {
18             dp[i][j] = dp[i+1][j] + dp[i][j+1];
19         }
20     }
21
22     // Return the result stored in the top-left corner
23     return dp[0][0];
24 }
```

Test Result: Accepted Runtime: 0 ms

Input: m = 3