Problem 1

Aim:

Climbing Stairs

Code:

```
class Solution {
public int climbStairs(int n) {
if (n <= 3) return n;  int prev1
= 3;  int prev2 = 2;  int cur =
0;
for (int i = 3; i < n; i++) {
cur = prev1 + prev2;  prev2
= prev1;
prev1 = cur;
}
return cur;
 }
}
```

Output:

☑ Testcase >_ Test Result

Accepted Runtime: 0 ms

• Case 1   • Case 2

Input

n =
2

Output

2
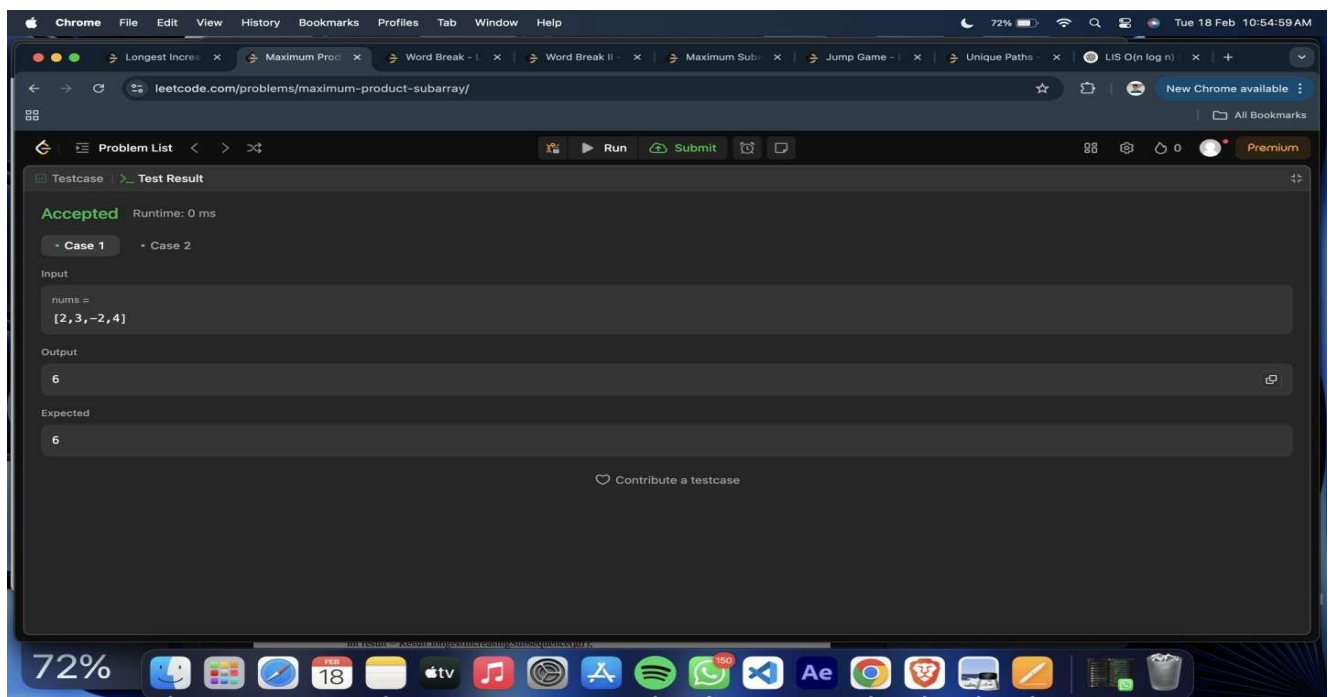
Expected

2

Problem

2 Aim: Maximum Product Subarray **Code:**

```
class Solution {
    public int maxSubArray(int[] nums) {
int maxSum = nums[0];
        int currentSum = nums[0];

        for (int i = 1; i < nums.length; i++) {
            currentSum = Math.max(nums[i], currentSum + nums[i]);
            maxSum = Math.max(maxSum, currentSum);
        }

        return maxSum;
    } }
```

Output:

Test Case 1

<div align="center">

Problem 3
Aim:

</div>

Unique Path **Code:**

```
class Solution {
    public int uniquePaths(int m, int n) {
int N = m + n - 2; // Total moves

    int K = Math.min(m - 1, n - 1);  // Choose the smaller value to reduce computations
long result = 1;  // Use long to prevent overflow

    // Compute C(N, K) using iterative multiplication
    for  (int  i  =  1;  i  <=  K;  i++) {
result = result * (N - i + 1) / i;
    }
     return (int) result;  // Convert back to int (safe since answer ≤ 2 * 10^9)
  }
}
```

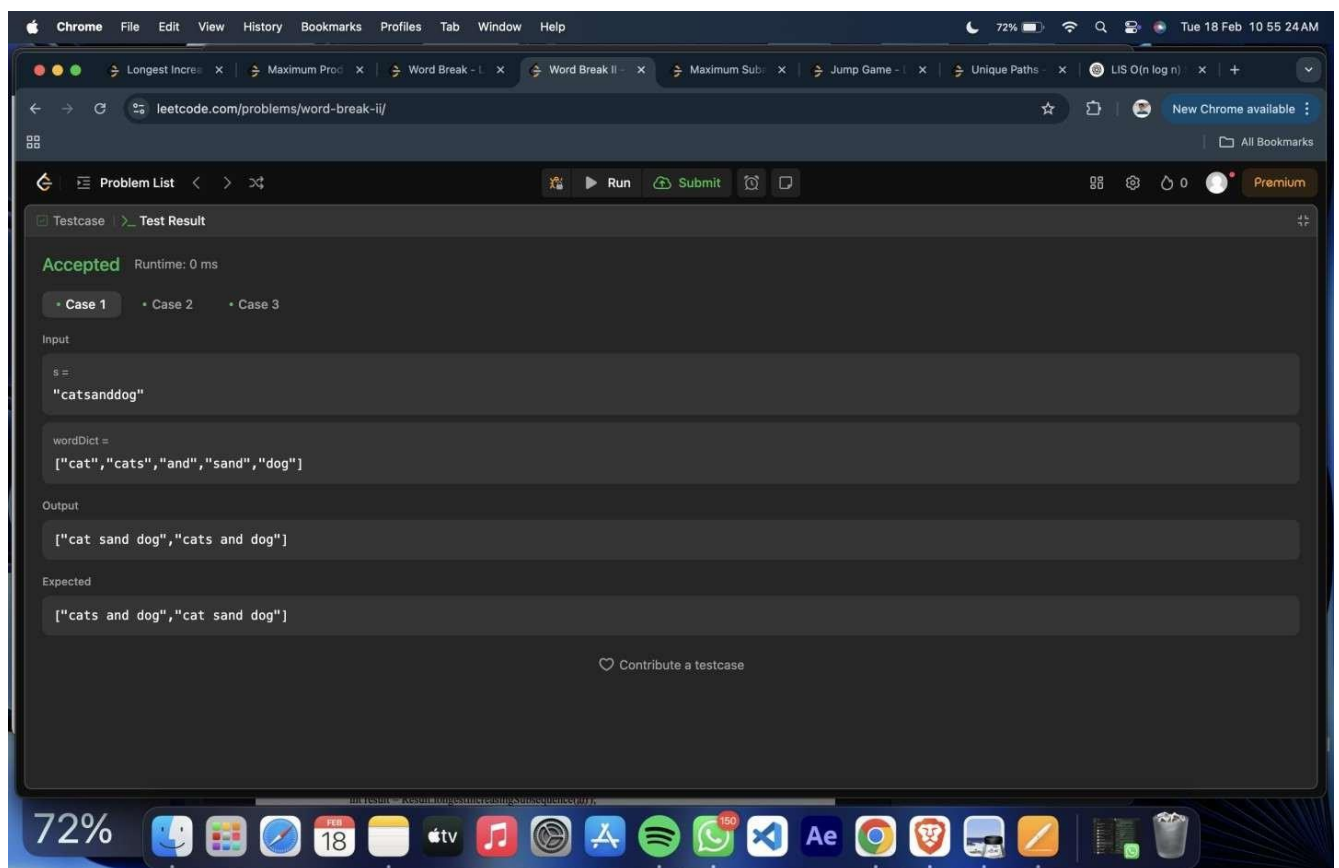**Output:**
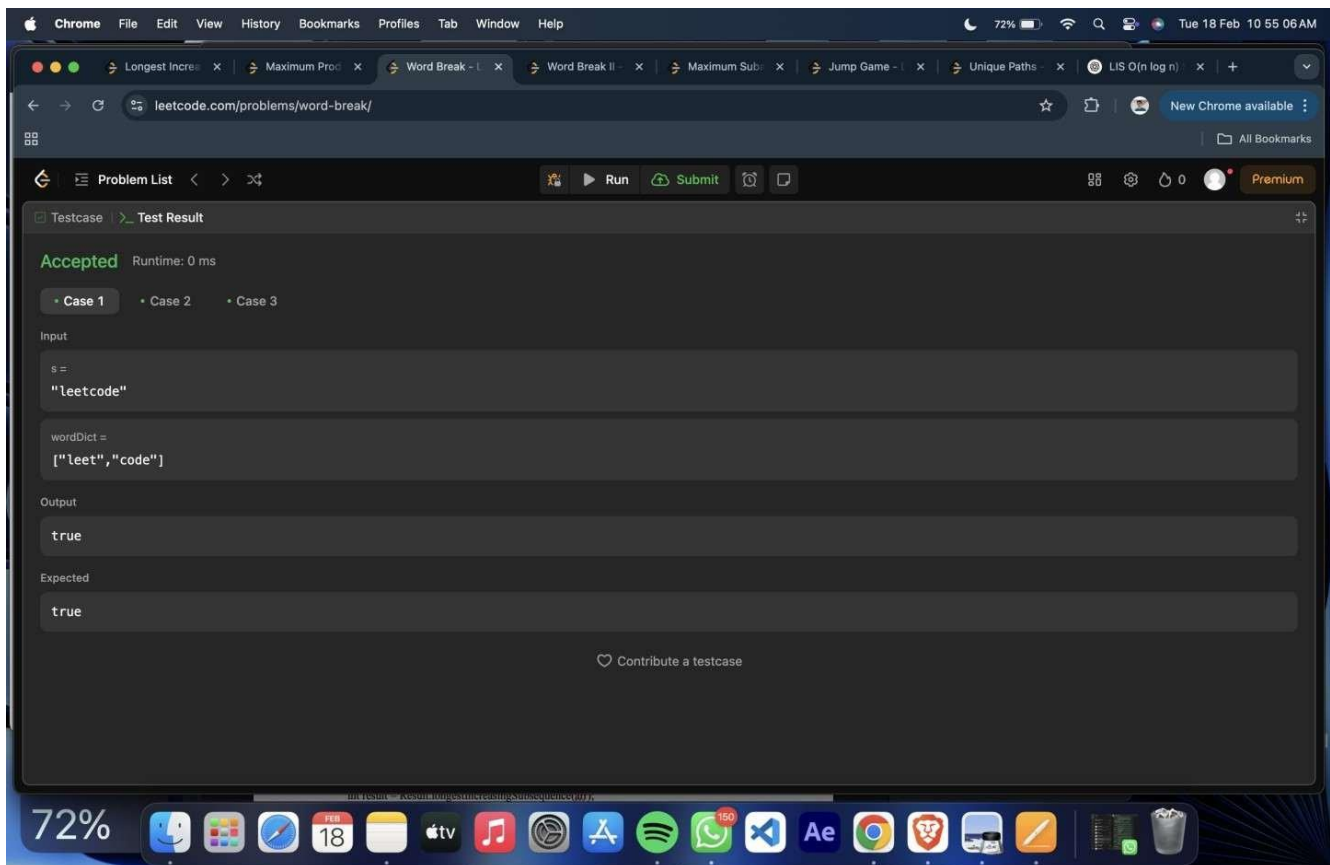
Problem 4
Aim:

Coin Change

Code:

```java
import java.util.Arrays;


class Solution {
    public int coinChange(int[] coins, int amount) {
        int max = amount + 1;  // A large value representing "infinity"
        int[] dp = new int[amount + 1];        Arrays.fill(dp, max);
        dp[0] = 0;  // Base case: 0 coins needed for amount 0

        for (int coin : coins) {
            for (int i = coin; i <= amount; i++) {
                dp[i] = Math.min(dp[i], 1 + dp[i - coin]);  // DP transition
            }
        }
        return dp[amount] == max ? -1 : dp[amount];
    } }
```

Output:

Problem 5

Aim: House Robber

Code:

```java
class Solution { public int
rob(int[] nums) { int n =
nums.length;

if (n == 1) { return
nums[0];
}

int[] dp = new int[n];

dp[0] = nums[0];
dp[1] = Math.max(nums[0], nums[1]);

for (int i = 2; i < n; i++) {
dp[i] = Math.max(dp[i - 1], nums[i] + dp[i - 2]);
}

return dp[n - 1];
}
}
```
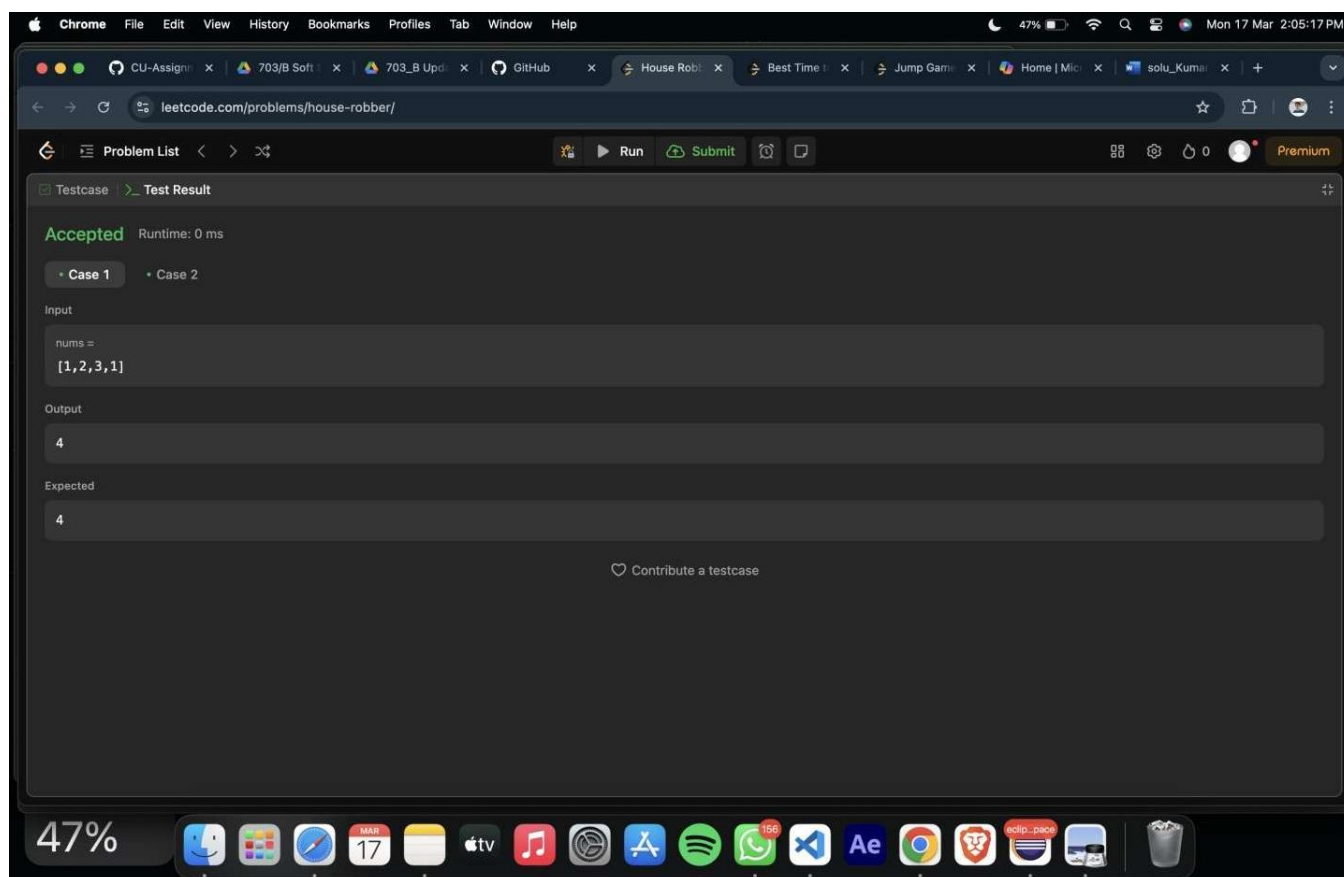
Output:

Problem 6

Aim:

Best Time to Buy and Sell Stock **Code:**

```
class Solution {
public int maxProfit(int[] prices) { int
buyPrice = prices[0];
int profit = 0;

for (int i = 1; i < prices.length; i++) {
if (buyPrice > prices[i]) { buyPrice =
prices[i];
}

profit = Math.max(profit, prices[i] - buyPrice);
}

return profit;
}
}
```

Output:

☑ Testcase | >_ Test Result

Accepted   Runtime: 0 ms

• Case 1   • Case 2

Input

prices =
[7,1,5,3,6,4]

Output

5

Expected

5

☑ Testcase | >_ Test Result

Accepted   Runtime: 0 ms

• Case 1   • Case 2

Input

prices =
[7,6,4,3,1]

Output

0

Expected

0

Case 1   Case 2   Case 3

Problem 7
Aim:

Beautiful Array **Code:**

```cpp
class Solution { public:      int partition(vector<int> &v, int start, int end, int mask)
  {
    int j = start;
    for(int i = start; i <= end; i++)
    {
      if((v[i] & mask) != 0)
      {
        swap(v[i], v[j]);
        j++;
      }
    }
    return j;
  }

  void sort(vector<int> & v, int start, int end, int mask)
  {
    if(start >= end) return;
    int mid = partition(v, start, end, mask);
    sort(v, start, mid - 1, mask << 1);
    sort(v, mid, end, mask << 1);
  }

  vector<int> beautifulArray(int N) {
    vector<int> ans;
    for(int i = 0; i < N; i++) ans.push_back(i + 1);
    sort(ans, 0, N - 1, 1);          return ans;
  }
};
```

Output:

Accepted Runtime: 0 ms	**Accepted** Runtime: 0 ms
• Case 1 • Case 2	• Case 1 • Case 2
Input	Input
n = 4	n = 5
Output	Output
[3,1,2,4]	[3,5,1,2,4]
Expected	Expected
[2,1,4,3]	[3,1,2,5,4]
Case 1	Case 2

Problem 8

Aim: Jump

Game **Code:**

```
class Solution {
public boolean canJump(int[] nums) { int
goal = nums.length - 1;

for (int i = nums.length - 2; i >= 0; i--) {
if (i + nums[i] >= goal) {
goal = i;
}
}

return goal == 0;
}
}
```

☑ Testcase | >_ Test Result

Accepted Runtime: 0 ms

• Case 1   • Case 2

Input

```
nums =
[3,2,1,0,4]
```

Output

```
false
```

Expected

```
false
```

☑ Testcase | >_ Test Result

Accepted Runtime: 0 ms

• Case 1   • Case 2

Input

```
nums =
[2,3,1,1,4]
```

Output

```
true
```

Expected

```
true
```