Experiment 7

Student Name: Sikander Singh Nanglu **UID: 22BET10031**
Branch: BE-IT **Section/Group: 22BET-IOT-701/A**
Semester: 6 **Date of Performance: 19/3/2025**
Subject Name:AP2 **Subject Code: 22ITP-351**

1. **Aim:** Implement the following problem:- Climbing Stairs, Best Time to Buy and Sell a Stock, Maximum Subarray, House Robber, Jump Game, Unique Paths, Coin Change, Longest Increasing Subsequence, Maximum Product Subarray, Decode Ways, Best Time to Buy and Sell a Stock with Cooldown, Perfect Squares, Word Break, Word Break 2.

2. **Objective:** Solve dynamic programming problems related to optimization, counting paths, and subsequence/subarray challenges. Find the best possible outcome in scenarios like stock trading, coin change, decoding messages, and breaking words efficiently.

3. **Implementation/Code:**

(A) Climbing Stairs

```
class Solution {
public:
int climbStairs(int n) {
if (n <= 2) return n;
int first = 1, second = 2, ways;
for (int i = 3; i <= n; i++) {
ways = first + second;
first = second;
second = ways;
}
return ways;
}
};
```

(B) Best Time to Buy and Sell a Stock

```
class Solution {
public:
int maxProfit(vector<int>& prices) {
int minPrice = INT_MAX, maxProfit = 0;
for (int price : prices) {
minPrice = min(minPrice, price);  // Update the minimum price
maxProfit = max(maxProfit, price - minPrice);  // Update max profit
}
return maxProfit;
}
};
```

(C) Maximum Subarray

```
class Solution {
public:
int maxSubArray(vector<int>& nums) {
int maxSum = nums[0], currentSum = nums[0];
for (int i = 1; i < nums.size(); i++) {
currentSum = max(nums[i], currentSum + nums[i]); // Extend or restart subarray
maxSum = max(maxSum, currentSum); // Update max sum found
}
return maxSum;
}
};
```

(D) House Robber

```
class Solution {
public:
int rob(vector<int>& nums) {
if (nums.empty()) return 0;
if (nums.size() == 1) return nums[0];
int prev2 = 0, prev1 = 0;
for (int num : nums) {
int temp = max(prev1, prev2 + num);
prev2 = prev1;
prev1 = temp;
}
return prev1;
}
};
```

(E) Jump Game

```
class Solution {
public:
bool canJump(vector<int>& nums) {
int maxReach = 0;
for (int i = 0; i < nums.size(); i++) {
if (i > maxReach) return false; // If we can't reach index i
maxReach = max(maxReach, i + nums[i]); // Update max reachable index
 }
return true;
}
};
```

(F) Unique Paths

```
class Solution {
public:
int uniquePaths(int m, int n) {
vector<vector<int>> dp(m, vector<int>(n, 1));
for (int i = 1; i < m; i++) {
for (int j = 1; j < n; j++) {
dp[i][j] = dp[i - 1][j] + dp[i][j - 1];
}
}
return dp[m - 1][n - 1];

}

};
```

(G) Coin Change

```
class Solution {

public:

int coinChange(vector<int>& coins, int amount) {

vector<int> dp(amount + 1, INT_MAX);

dp[0] = 0;

for (int coin : coins) {

for (int j = coin; j <= amount; j++) {

if (dp[j - coin] != INT_MAX) {

dp[j] = min(dp[j], dp[j - coin] + 1);

}

}

}

return dp[amount] == INT_MAX ? -1 : dp[amount];

}

};
```

(H)Longest Increasing Subsequence

```
class Solution {

public:
```

```
int lengthOfLIS(vector<int>& nums) {

 int n = nums.size();

vector<int> dp(n, 1);

int maxLength = 1;

for (int i = 1; i < n; i++) {

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

if (nums[i] > nums[j]) {

dp[i] = max(dp[i], dp[j] + 1);

}

}

maxLength = max(maxLength, dp[i]);

}

return maxLength;

}

};
```

(I) Maximum Product Subarray

```
class Solution {

public:

int maxProduct(vector<int>& nums) {

int n = nums.size();

int maxProd = nums[0], minProd = nums[0], result = nums[0];
for (int i = 1; i < n; i++) {
if (nums[i] < 0) {
swap(maxProd, minProd);
}
maxProd = max(nums[i], maxProd * nums[i]);
minProd = min(nums[i], minProd * nums[i]);
result = max(result, maxProd);
}
return result;
}
};
```

(J) Decode Ways

```
class Solution {
public:
int numDecodings(string s) {
int n = s.size();
if (s[0] == '0') return 0;
vector<int> dp(n + 1, 0);
dp[0] = 1;
dp[1] = 1;
for (int i = 2; i <= n; i++) {
int oneDigit = s[i - 1] - '0';
int twoDigit = stoi(s.substr(i - 2, 2));
if (oneDigit >= 1) {
dp[i] += dp[i - 1];
}
if (twoDigit >= 10 && twoDigit <= 26) {
dp[i] += dp[i - 2];
}
}
return dp[n];
}
};
```

(K) Best Time to Buy and Sell a Stock with Cooldown

```
class Solution {
public:
int maxProfit(vector<int>& prices) {
int n = prices.size();
if (n == 0) return 0;
vector<int> buy(n, 0), sell(n, 0), cooldown(n, 0);
buy[0] = -prices[0];
sell[0] = 0;
cooldown[0] = 0;
for (int i = 1; i < n; i++) {
```

```
buy[i] = max(buy[i - 1], cooldown[i - 1] - prices[i]);
sell[i] = max(sell[i - 1], buy[i - 1] + prices[i]);
cooldown[i] = max(cooldown[i - 1], sell[i - 1]);
}
return sell[n - 1];
}
};
```

(L)Perfect squares

```
class Solution {
public:
int numSquares(int n) {
vector<int> dp(n + 1, INT_MAX);
dp[0] = 0;
for (int i = 1; i <= n; i++) {
for (int j = 1; j * j <= i; j++) {
dp[i] = min(dp[i], dp[i - j * j] + 1);
}
}
return dp[n];
}
};
```

(M) Word Break

```
class Solution {
public:
bool wordBreak(string s, vector<string>& wordDict) {
unordered_set<string> wordSet(wordDict.begin(), wordDict.end());
int n = s.size();
vector<bool> dp(n + 1, false);
dp[0] = true;
for (int i = 1; i <= n; i++) {
for (int j = 0; j < i; j++) {
if (dp[j] && wordSet.find(s.substr(j, i - j)) != wordSet.end()) {
dp[i] = true;
break;
}
}
}
return dp[n];
}
};
```

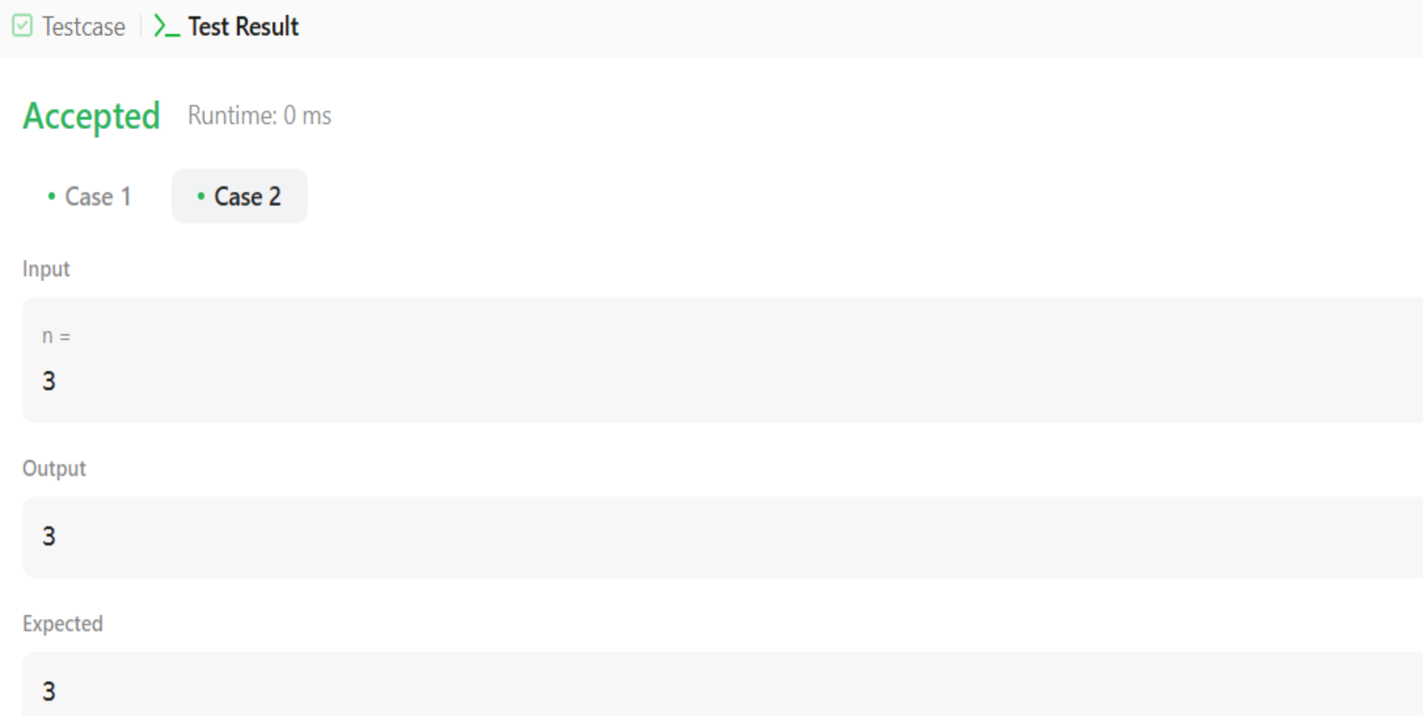
(N) Word Break 2

```
class Solution {
public:
```

```
unordered_map<string, vector<string>> memo;

vector<string> wordBreak(string s, vector<string>& wordDict) {

unordered_set<string> wordSet(wordDict.begin(), wordDict.end());

return helper(s, wordSet);

}

vector<string> helper(string s, unordered_set<string>& wordSet) {

if (memo.find(s) != memo.end()) return memo[s];

if (s.empty()) return {""};

vector<string> res;

for (int i = 1; i <= s.size(); i++) {

string word = s.substr(0, i);

if (wordSet.count(word)) {

vector<string> suffixes = helper(s.substr(i), wordSet);

for (string suffix : suffixes) {

res.push_back(word + (suffix.empty() ? "" : " " + suffix));

}

}

}

return memo[s] = res;

}

};
```

4. Output:

Climbing Stairs



Accepted Runtime: 0 ms

Input

```
n =
3
```

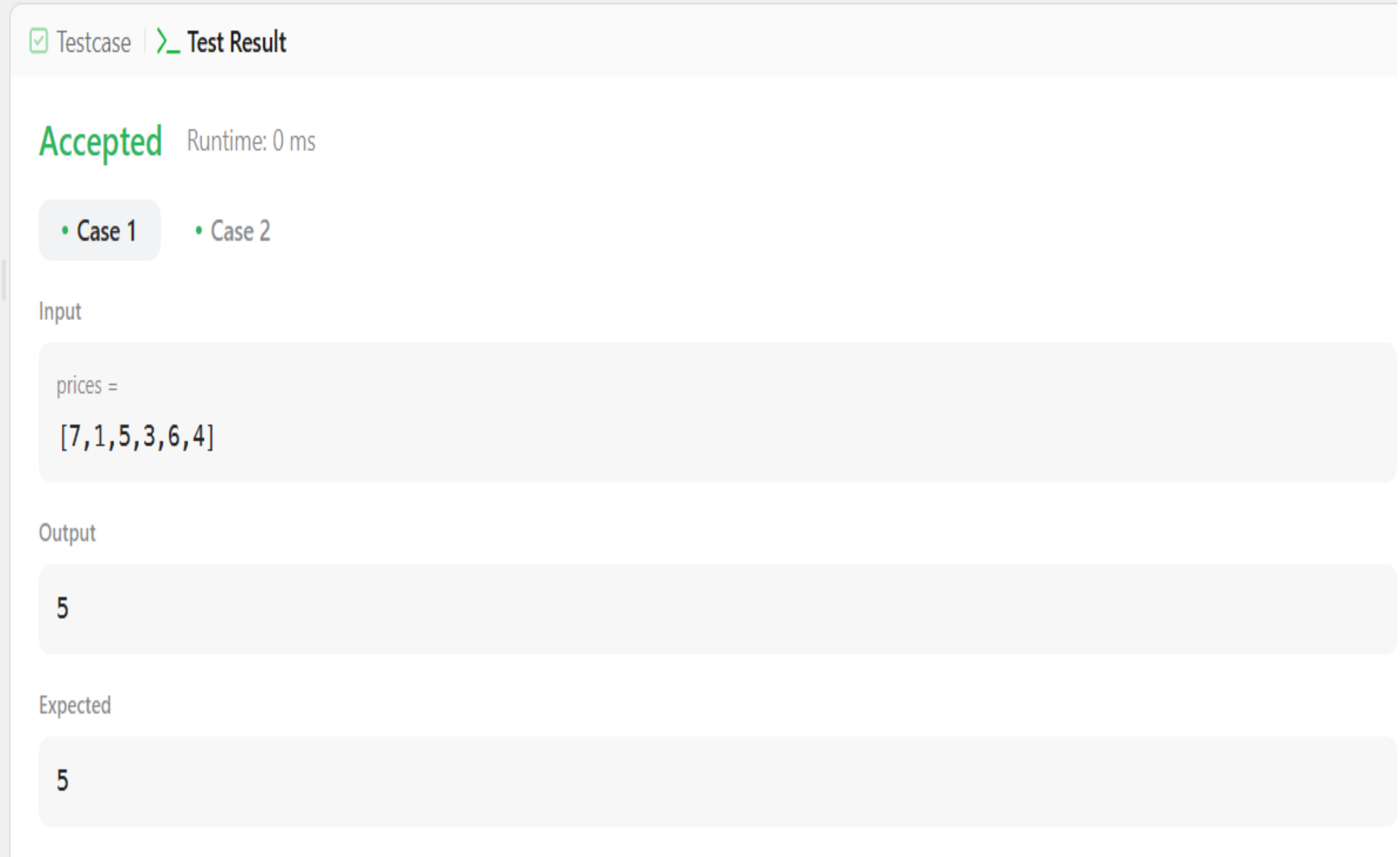
Output

```
3
```

Expected

```
3
```

Best Time to Buy and Sell a Stock



Accepted Runtime: 0 ms

Input

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

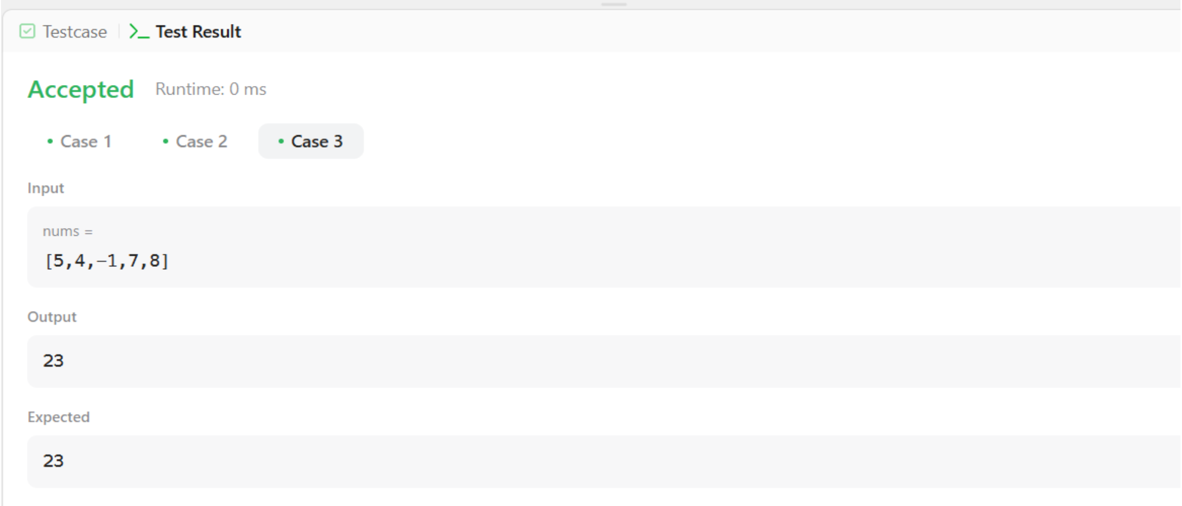
Output

```
5
```

Expected

```
5
```

Maximum Subarray



Accepted Runtime: 0 ms

Input

```
nums =
[5,4,-1,7,8]
```

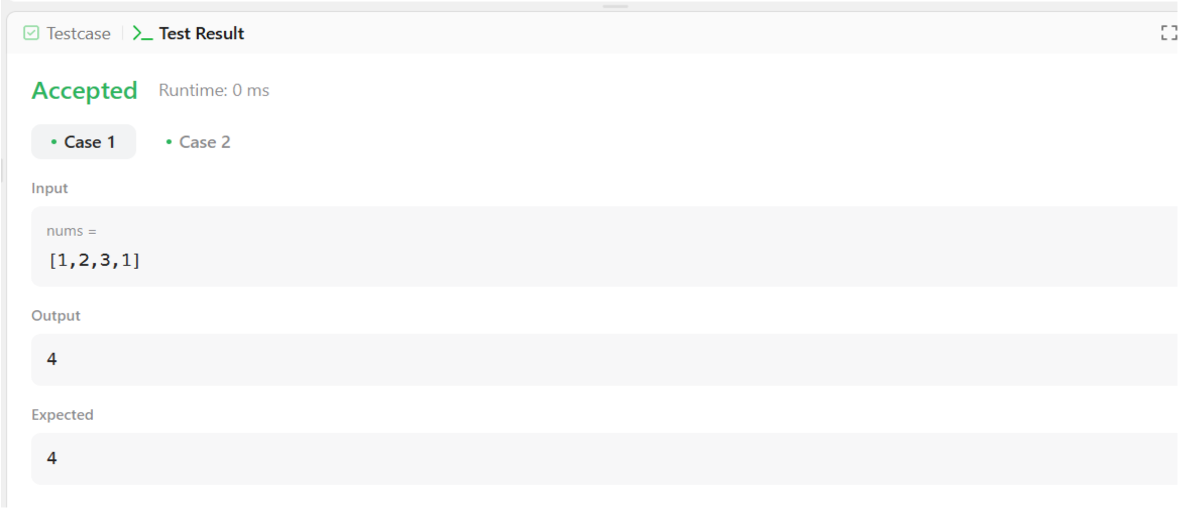
Output

```
23
```

Expected

```
23
```

House Robber



Accepted Runtime: 0 ms

Input

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

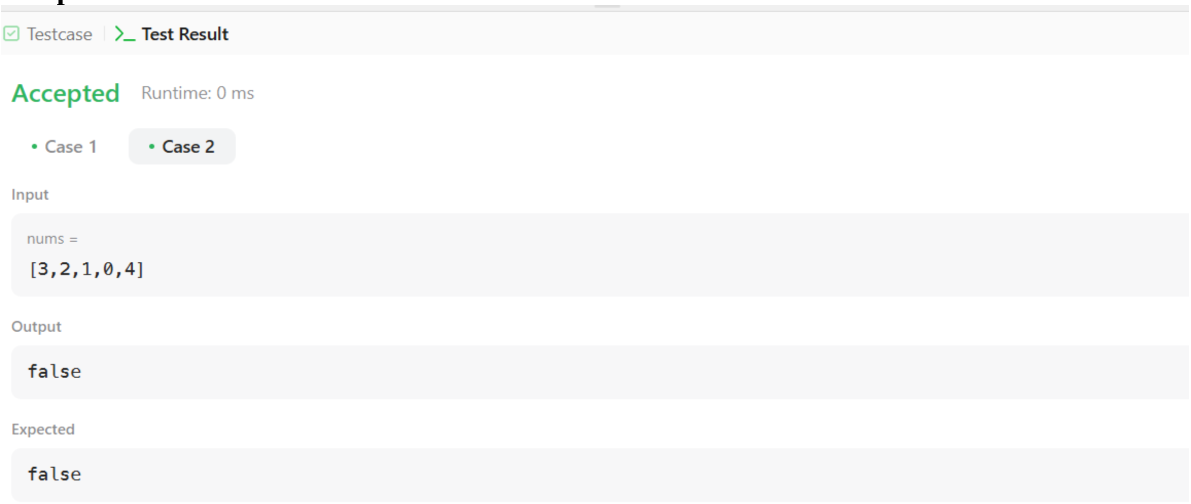
Output

```
4
```

Expected

```
4
```

Jump Game



Accepted Runtime: 0 ms

Input

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

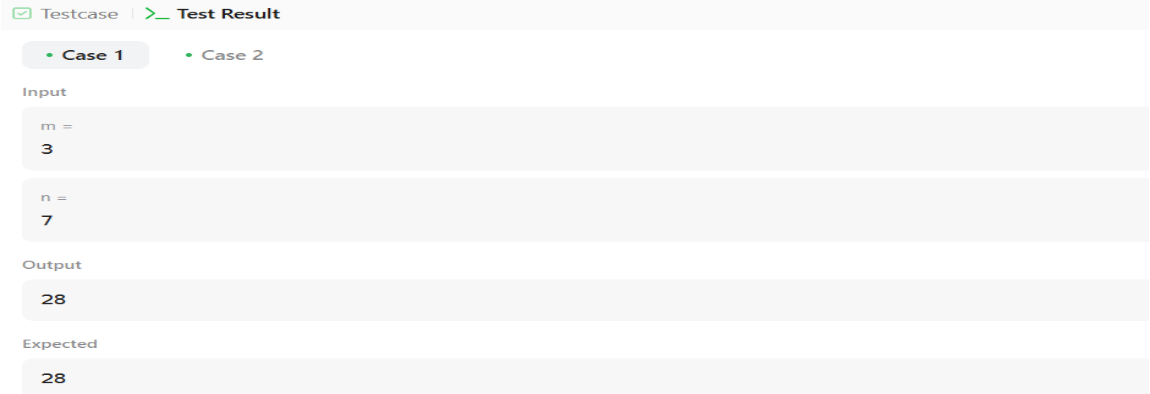
Output

```
false
```

Expected

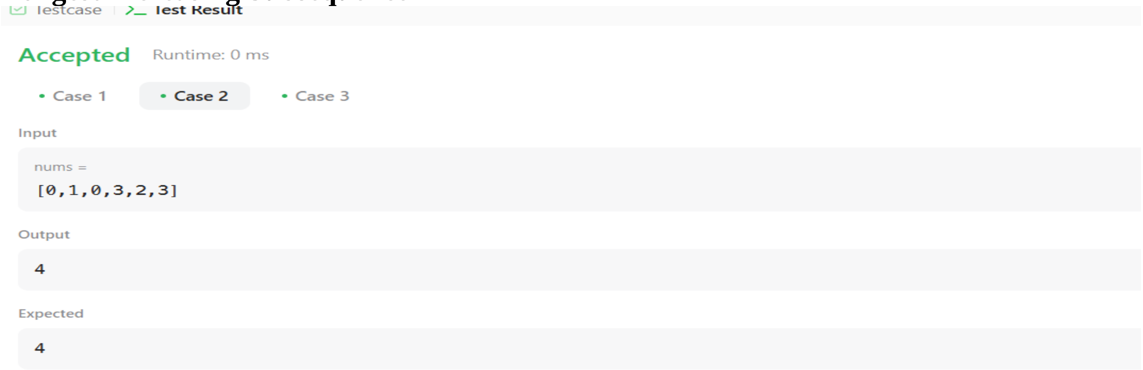
```
false
```

Unique Path



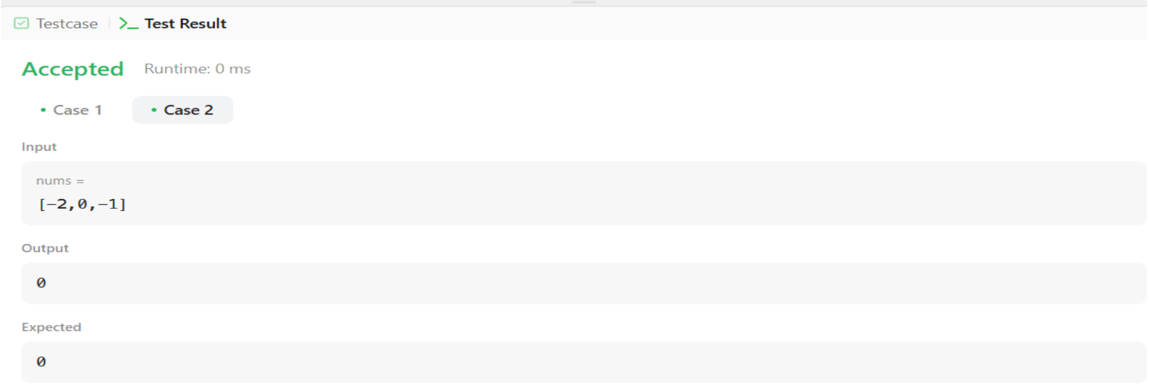
```
Input
m =
3
n =
7
Output
28
Expected
28
```

Longest Increasing Subsequence



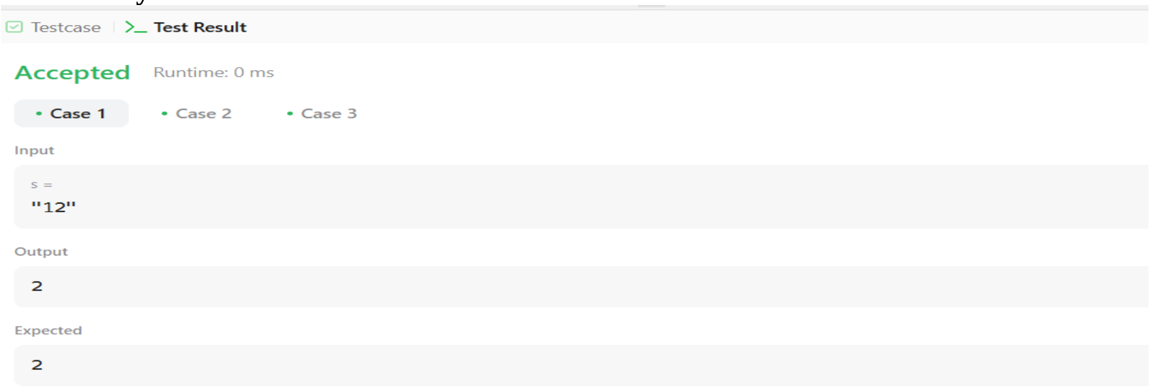
```
Accepted   Runtime: 0 ms
Input
nums =
[0,1,0,3,2,3]
Output
4
Expected
4
```

Maximum Product Subarray



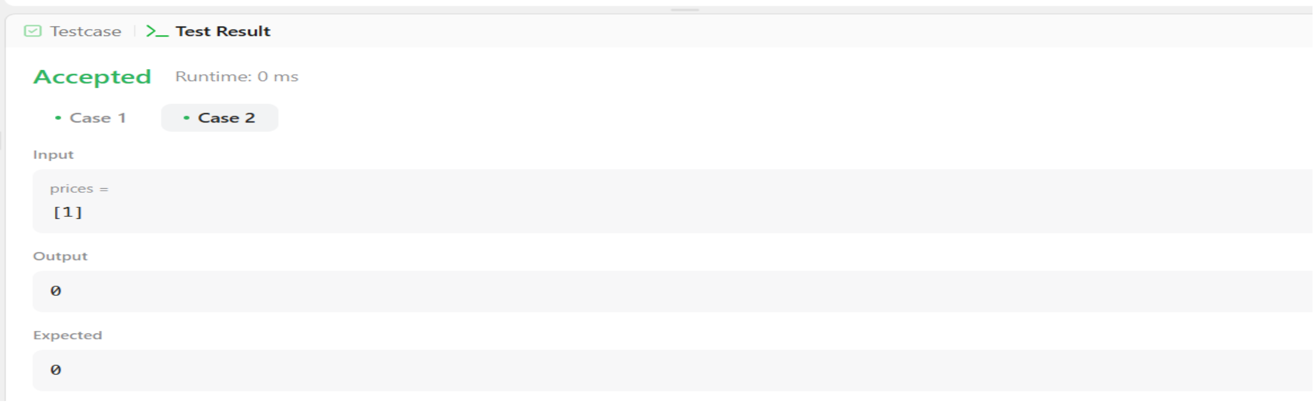
```
Accepted   Runtime: 0 ms
Input
nums =
[-2,0,-1]
Output
0
Expected
0
```

Decode ways



```
Accepted   Runtime: 0 ms
Input
s =
"12"
Output
2
Expected
2
```

Best Time to Buy and Sell a Stock with Cooldown



Accepted Runtime: 0 ms

• Case 1 • Case 2

Input

prices =

```
[1]
```

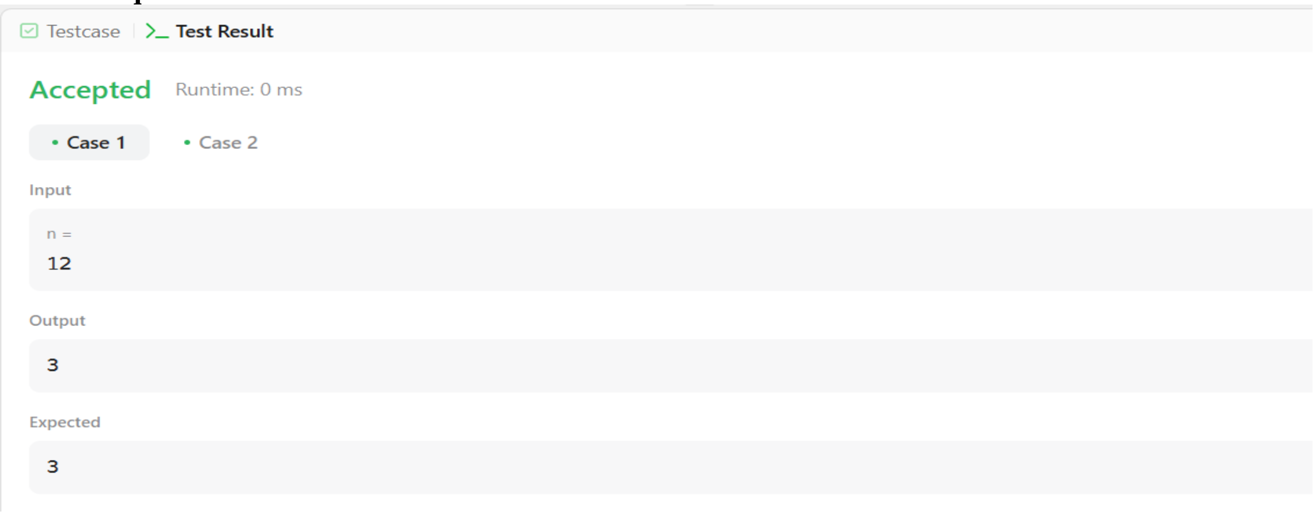
Output

```
0
```

Expected

```
0
```

Perfect Sequres



Accepted Runtime: 0 ms

• Case 1 • Case 2

Input

n =

```
12
```

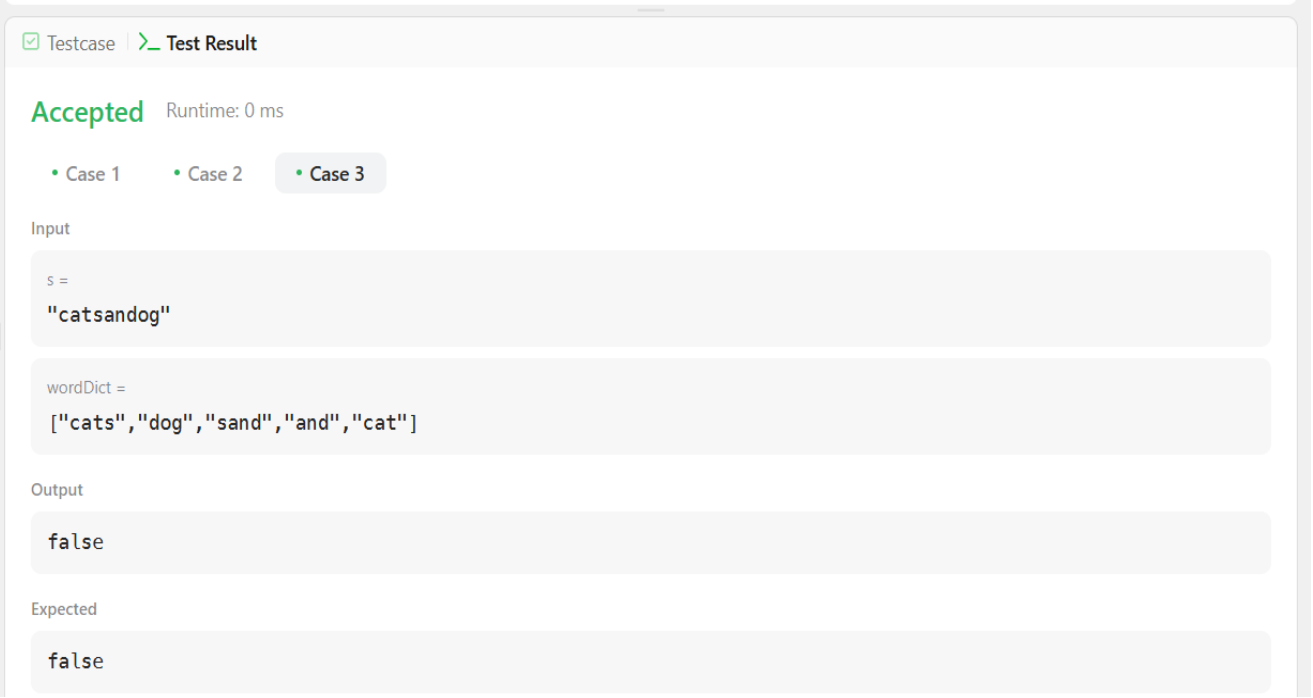
Output

```
3
```

Expected

```
3
```

Word Break



Accepted Runtime: 0 ms

• Case 1 • Case 2 • Case 3

Input

s =

```
"catsandog"
```

wordDict =

```
["cats","dog","sand","and","cat"]
```

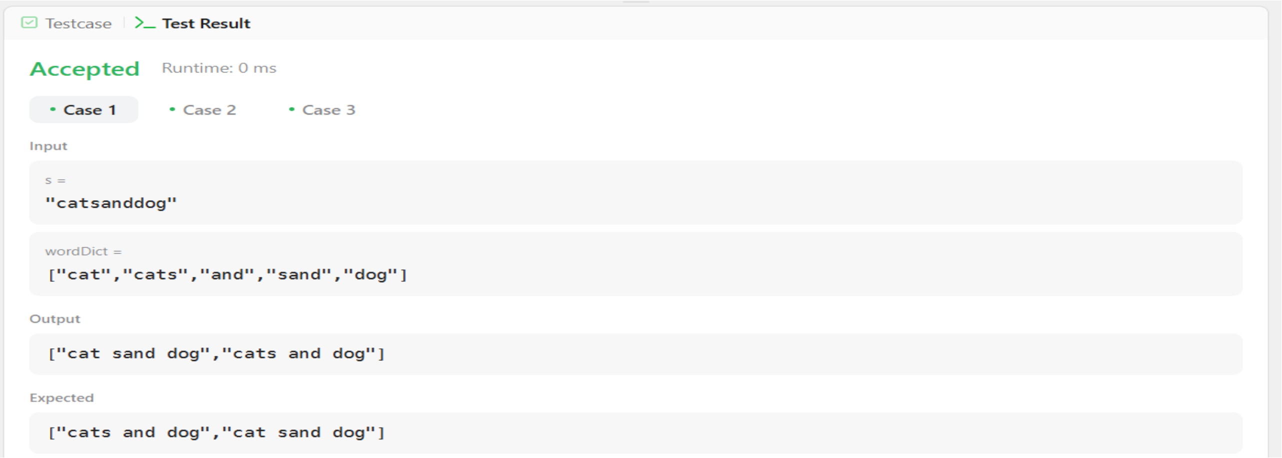
Output

```
false
```

Expected

```
false
```

Word Break 2



Accepted Runtime: 0 ms

Input

s =

```
"catsanddog"
```

wordDict =

```
["cat","cats","and","sand","dog"]
```

Output

```
["cat sand dog","cats and dog"]
```

Expected

```
["cats and dog","cat sand dog"]
```

5. Learning Outcomes:-

- Ability to analyze problems, evaluate information, and make logical decisions.
- Capability to identify, understand, and develop solutions to complex issues.
- Proficiency in expressing ideas clearly, both verbally and in writing
- Willingness to learn new skills and adjust to changing environments.
- Ability to work effectively with others in diverse environments.