Experiment 7

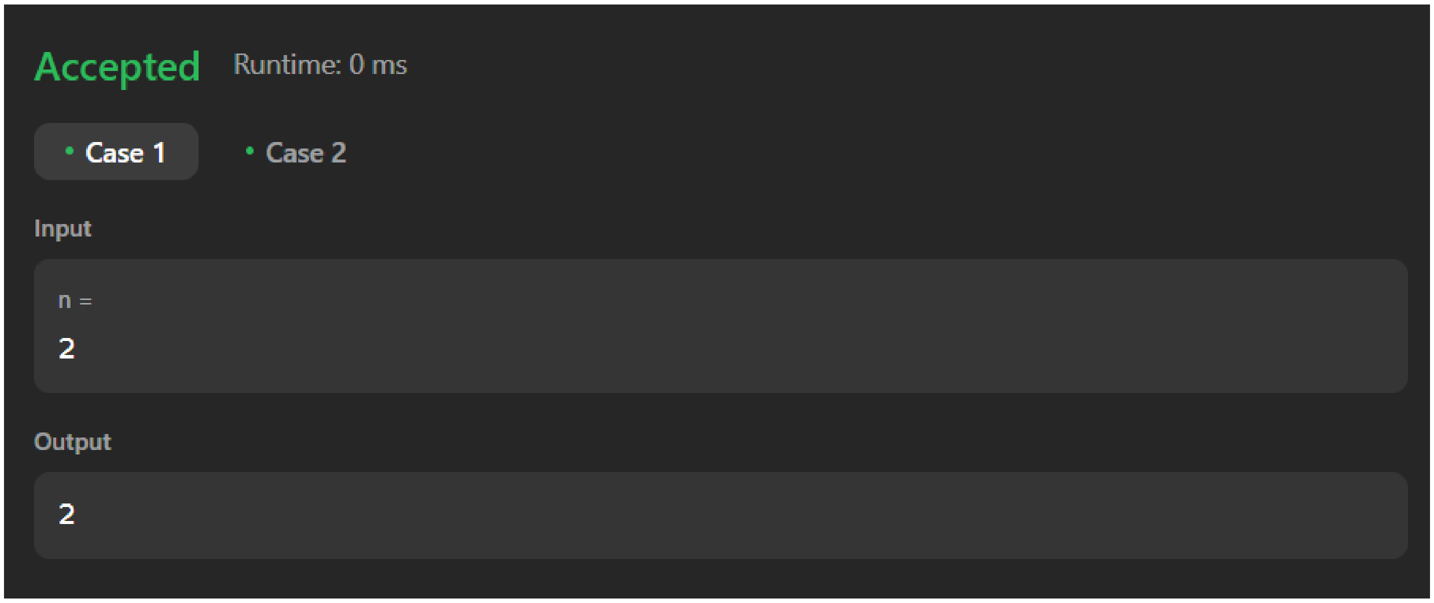
Name: Prashant **UID: 22BET10055**
Branch: BE-IT **Section/Group: 22BET_701-A**
Semester: 6 **Date of Performance: 26-02-25**
Subject Name: Advanced Programming Lab-2 **Subject Code: 22ITP-351**

Problem 1. Climbing Stairs- You are climbing a staircase. It takes n steps to reach the top.Each time you can either climb 1 or 2 steps. In how many distinct ways can you climb to the top?

Code:

```
class Solution {
public:
    int climbStairs(int n) {
        if (n == 0 || n == 1) {
            return 1;
        }
        return climbStairs(n-1) + climbStairs(n-2);
    }
};
```

Output:



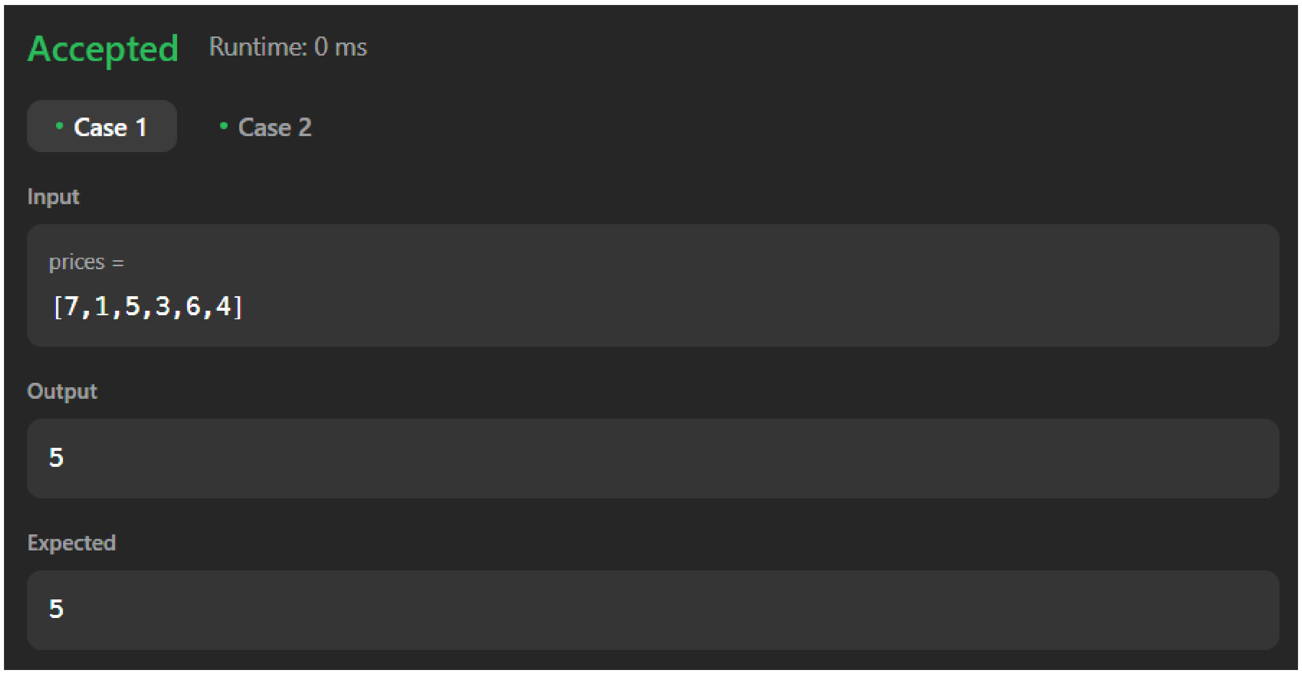
```
Accepted    Runtime: 0 ms

• Case 1    • Case 2

Input
n =
2

Output
2
```

Problem 2. Best Time to Buy and Sell a Stock- You are given an array prices where prices[i] is the price of a given stock on the ith day.You want to maximize your profit by choosing a single day to buy one stock and choosing a different day in the future to sell that stock.

Code:

```
class Solution {
public:
    int maxProfit(std::vector<int>& prices) {
        int buy = prices[0];
        int profit = 0;
        for (int i = 1; i < prices.size(); i++) {
            if (prices[i] < buy) {
                buy = prices[i];
            } else if (prices[i] - buy > profit) {
                profit = prices[i] - buy;
            }
        }
        return profit;
    }
};
```

Output:



Accepted Runtime: 0 ms

Input

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

Output

```
5
```

Expected

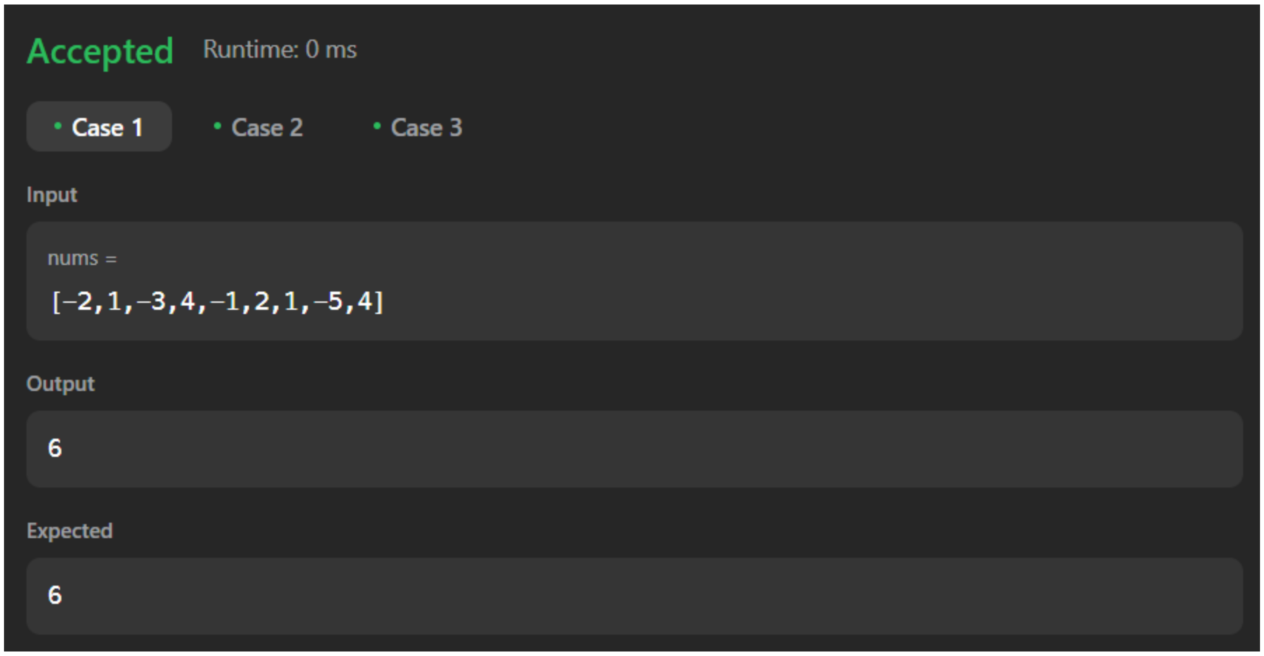
```
5
```

Problem 3. Maximum Subarray. - Given an integer array nums, find the subarray with the largest sum, and return its sum.

Code:

```
class Solution {
public:
    int maxSubArray(vector<int>& nums) {
        int n = size(nums), ans = INT_MIN;
        for(int i = 0; i < n; i++)
            for(int j = i, curSum = 0; j < n ; j++)
                curSum += nums[j],
                ans = max(ans, curSum);
        return ans;
    }
};
```

Output:



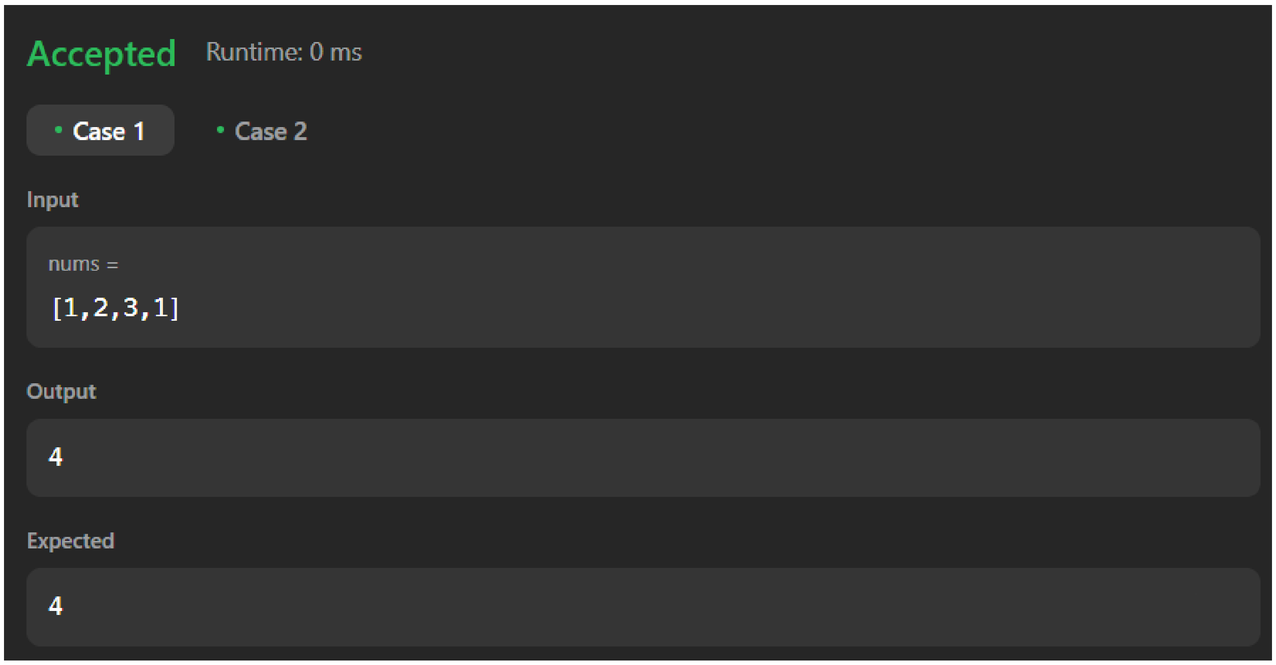
```
Input
nums =
[-2,1,-3,4,-1,2,1,-5,4]
Output
6
Expected
6
```

Problem 4. House Robber- You are a professional robber planning to rob houses along a street. Each house has a certain amount of money stashed, the only constraint stopping you from robbing each of them is that adjacent houses have security systems connected and it will automatically contact the police if two adjacent houses were broken into on the same night.

Code:

```
class Solution {
public:
    int rob(vector<int>& nums) {
        int prevRob = 0;
        int maxRob = 0;

        for (int curValue : nums) {
            int temp = max(maxRob, prevRob + curValue);
            prevRob = maxRob;
            maxRob = temp;
        }

        return maxRob;
    }
};
```

Output:


Accepted Runtime: 0 ms

Case 1 Case 2

Input

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

Output

```
4
```

Expected

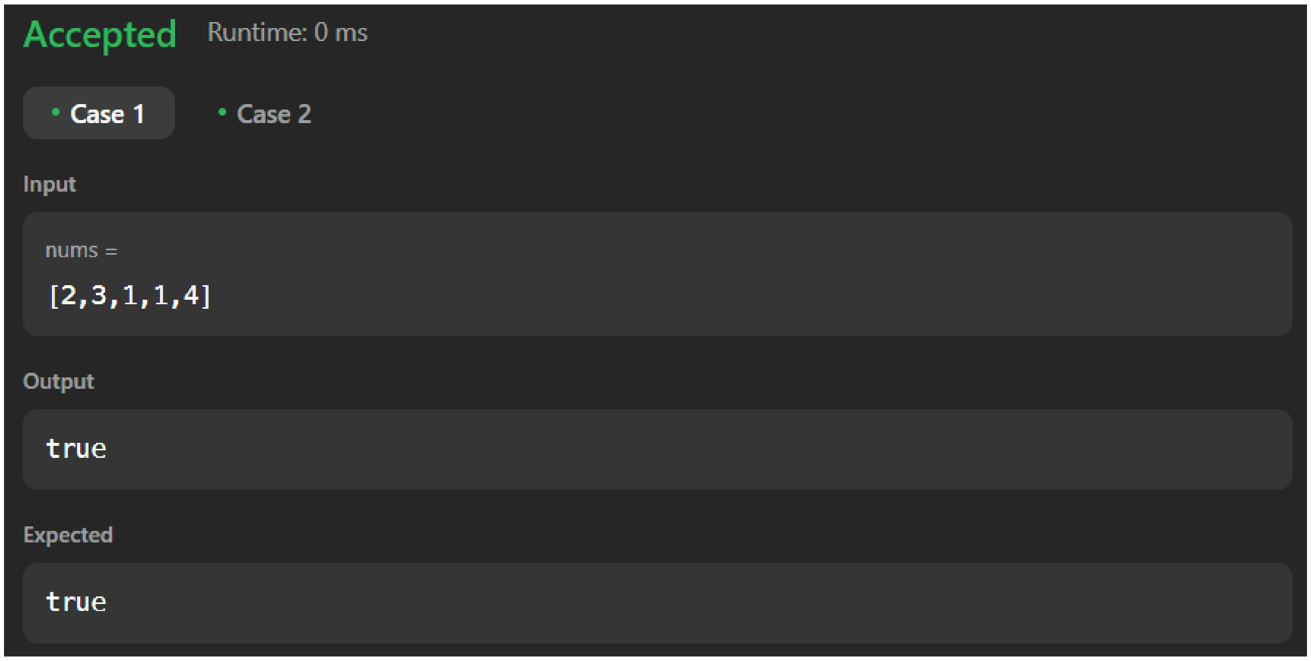
```
4
```

Problem 5. Jump Game- You are given an integer array nums. You are initially positioned at the array's first index, and each element in the array represents your maximum jump length at that position.Return true if you can reach the last index, or false otherwise.

Code:

```
class Solution {
public:
    bool canJump(vector<int>& nums) {
        int goal = nums.size() - 1;
        for (int i = nums.size() - 1; i >= 0; i--) {
            if (i + nums[i] >= goal) {
                goal = i;
            }
        }
        return goal == 0;
    }
}
```

Output:



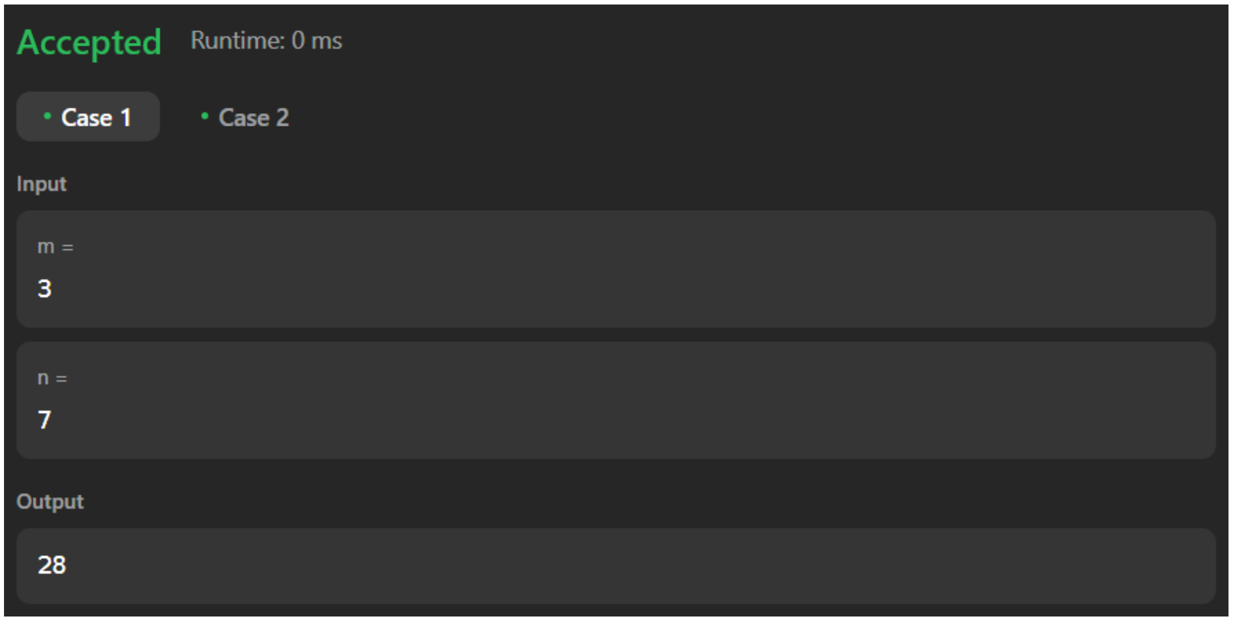
```
Accepted   Runtime: 0 ms

Case 1   Case 2

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

Output
true

Expected
true
```

Problem 6. Unique Paths-There is a robot on an m x n grid. The robot is initially located at the top-left corner (i.e., grid[0][0]). The robot tries to move to the bottom-right corner (i.e., grid[m - 1][n - 1]). The robot can only move either down or right at any point in time.

Code:

```
class Solution {
public:
    int uniquePaths(int m, int n) {
        std::vector<int> aboveRow(n, 1);

        for (int row = 1; row < m; row++) {
            std::vector<int> currentRow(n, 1);
            for (int col = 1; col < n; col++) {
                currentRow[col] = currentRow[col - 1] + aboveRow[col];
            }
            aboveRow = currentRow;
        }

        return aboveRow[n - 1];
    }
};
```

Output:



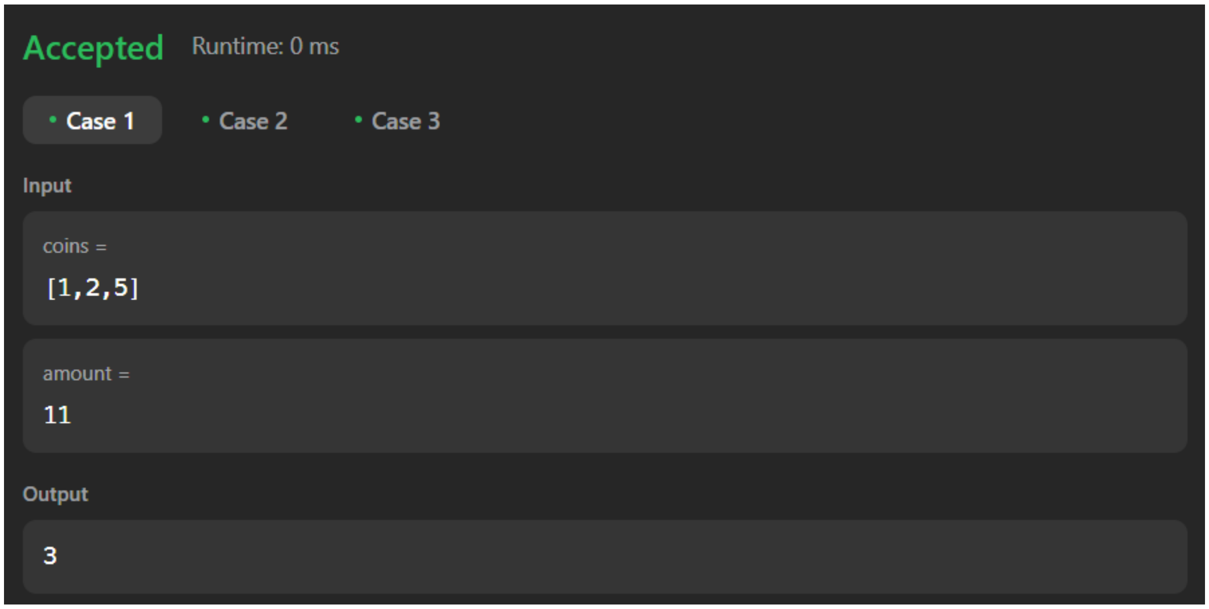
```
Accepted   Runtime: 0 ms
• Case 1   • Case 2
Input
m =
3
n =
7
Output
28
```

Problem 7. Coin Change- You are given an integer array coins representing coins of different denominations and an integer amount representing a total amount of money.

Code:

```
class Solution {
public:
    vector<int> dp;

    long solve(vector<int>& coins, int rem) {
        if (rem == 0) return 0;
        if (rem < 0) return INT_MAX;
        if (dp[rem] != -1) return dp[rem];
        long res = INT_MAX;
        for (int coin : coins) {
            res = min(res, 1 + solve(coins, rem - coin));
        }
        return dp[rem] = res;
    }
    int coinChange(vector<int>& coins, int amount) {
        dp.resize(amount + 1, -1);
        int res = solve(coins, amount);
        return res == INT_MAX ? -1 : res;
    }
};
```

Output:



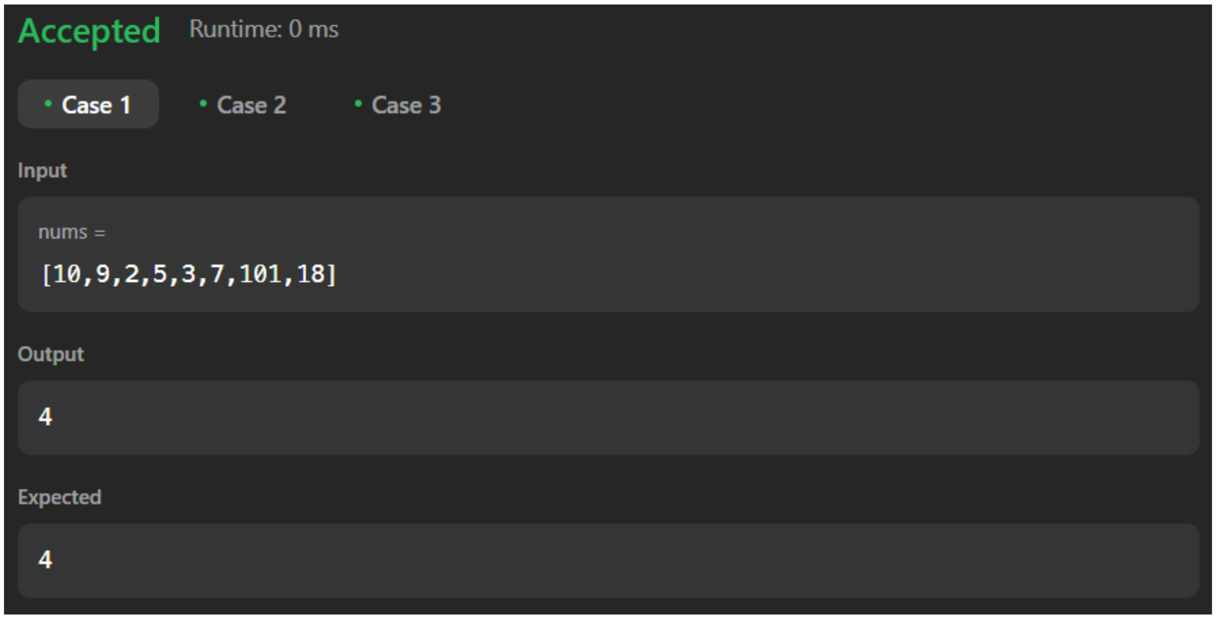
```
Accepted   Runtime: 0 ms
• Case 1   • Case 2   • Case 3
Input
coins =
[1,2,5]
amount =
11
Output
3
```

Problem 8. Longest Increasing Subsequence- Given an integer array nums, return the length of the longest strictly increasing subsequence.

Code:

```
class Solution {
public:
    int lengthOfLIS(std::vector<int>& nums) {
        if (nums.empty()) {
            return 0;
        }

        int n = nums.size();
        std::vector<int> dp(n, 1);

        for (int i = 1; i < n; ++i) {
            for (int j = 0; j < i; ++j) {
                if (nums[i] > nums[j]) {
                    dp[i] = std::max(dp[i], dp[j] + 1);
                }
            }
        }

        return *std::max_element(dp.begin(), dp.end());
    }
};
```

Output:



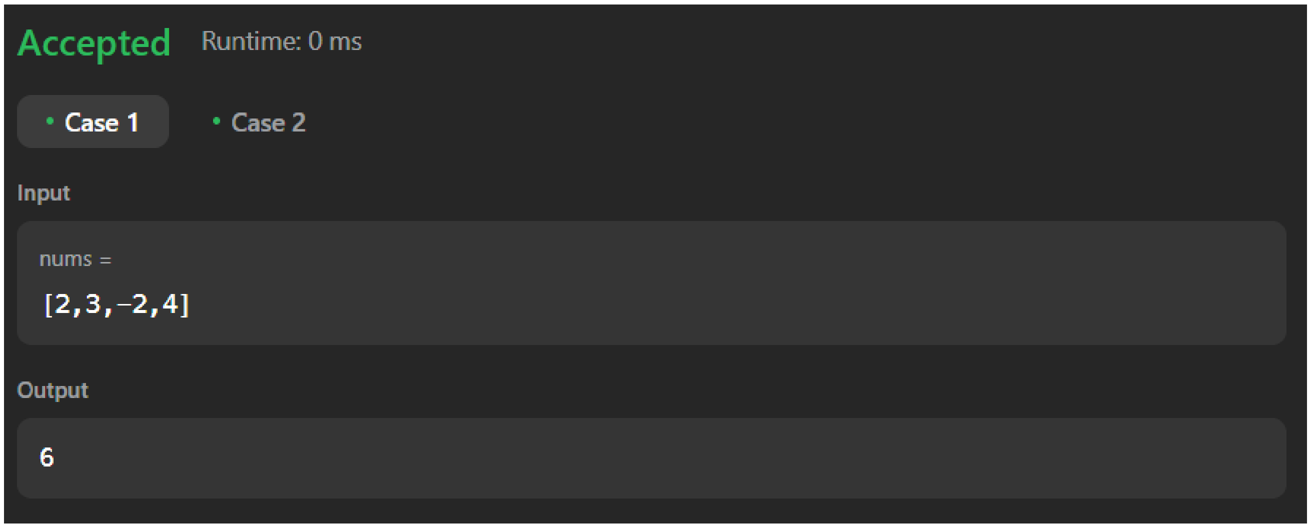
```
Accepted   Runtime: 0 ms

• Case 1   • Case 2   • Case 3

Input
nums =
[10,9,2,5,3,7,101,18]

Output
4

Expected
4
```

Problem 9. Maximum Product Subarray- Given an integer array nums, find a subarray that has the largest product, and return the product.The test cases are generated so that the answer will fit in a 32-bit integer.

Code:

```
class Solution {
public:
    int maxProduct(vector<int>& nums) {
        int res = *max_element(nums.begin(), nums.end());
        int curMax = 1, curMin = 1;

        for (int n : nums) {
            int temp = curMax * n;
            curMax = max({temp, curMin * n, n});
            curMin = min({temp, curMin * n, n});

            res = max(res, curMax);
        }

        return res;
    }
};
```

Output:



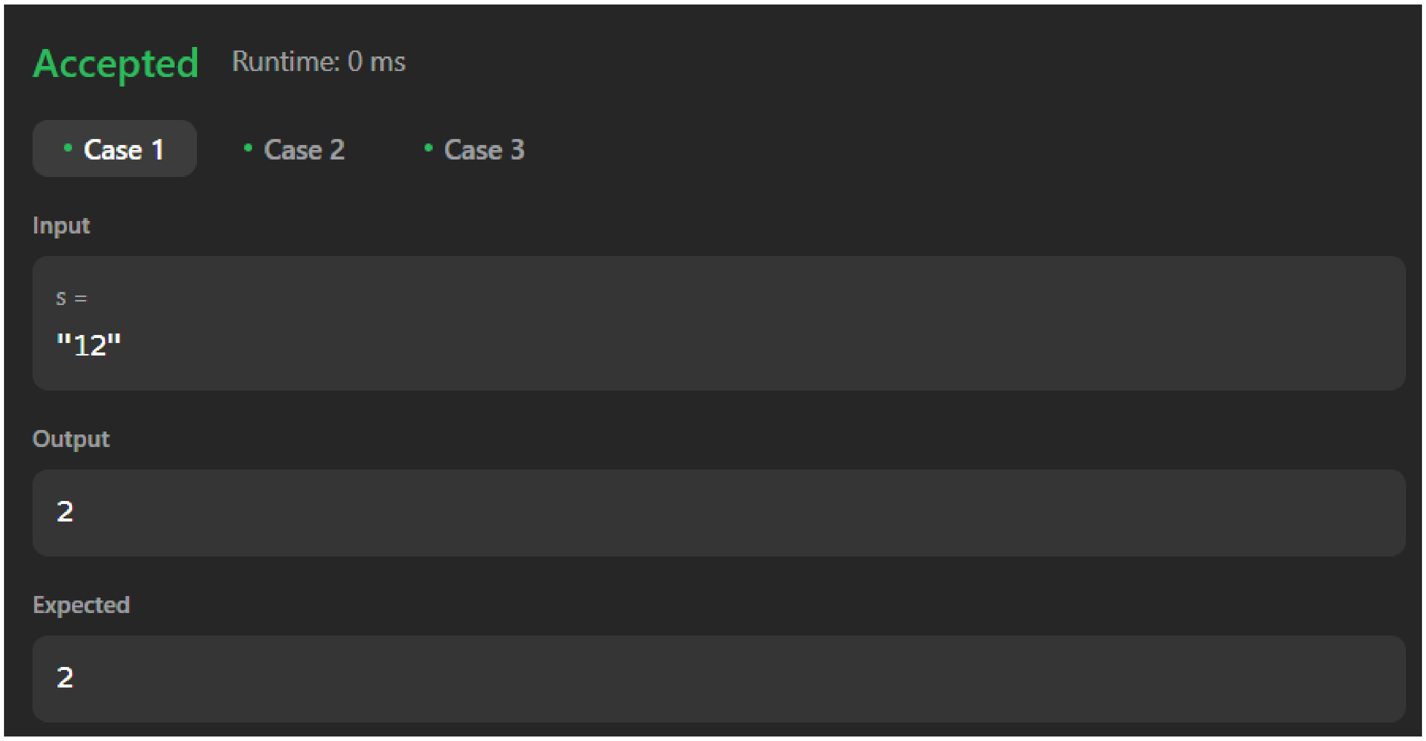
```
Accepted   Runtime: 0 ms

• Case 1   • Case 2

Input
nums =
[2,3,-2,4]

Output
6
```

Problem 10. Decode Ways- You have intercepted a secret message encoded as a string of numbers. The message is decoded via the following mapping:"1" -> 'A', "2" -> 'B' , "25" -> 'Y', "26" -> 'Z'.

Code:

```
class Solution {
public:
    int numDecodings(const string& s) {
        int n = s.size(), dp = 0, dp1 = 1, dp2 = 0;
        for (int i = n - 1; i >= 0; --i) {
            if (s[i] != '0') // Single digit
                dp += dp1;
            if (i+1 < s.size() && (s[i] == '1' || s[i] == '2' && s[i+1] <= '6')) // Two digits
                dp += dp2;
            dp2 = dp1;
            dp1 = dp;
            dp = 0;
        }
        return dp1;
    }
};
```

Output:



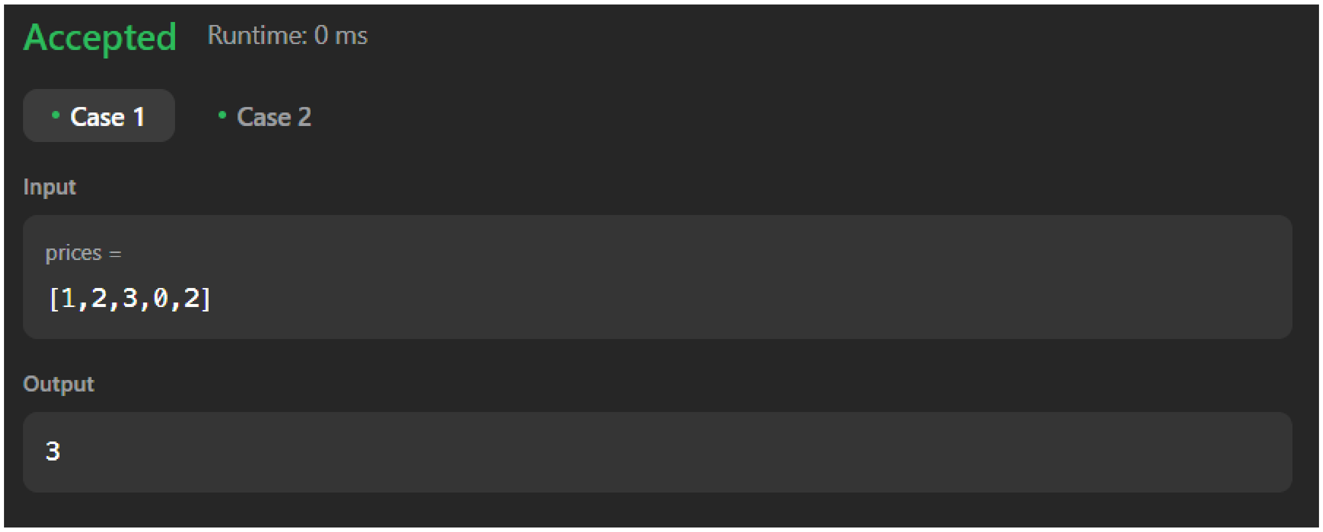
```
Accepted   Runtime: 0 ms

• Case 1   • Case 2   • Case 3

Input
s =
"12"

Output
2

Expected
2
```

Problem 11. Best time to buy and Sell a Stock with Cooldown- You are given an array prices where prices[i] is the price of a given stock on the ith day.Find the maximum profit you can achieve. You may complete as many transactions as you like (i.e., buy one and sell one share of the stock multiple times) with the following restrictions:

Code:

```
class Solution {
public:
    int maxProfit(vector<int>& prices){
            if (prices.size() <= 1) return 0;
            vector<int> s0(prices.size(), 0);
            vector<int> s1(prices.size(), 0);
            vector<int> s2(prices.size(), 0);
            s1[0] = -prices[0];
            s0[0] = 0;
            s2[0] = INT_MIN;
            for (int i = 1; i < prices.size(); i++) {
                    s0[i] = max(s0[i - 1], s2[i - 1]);
                    s1[i] = max(s1[i - 1], s0[i - 1] - prices[i]);
                    s2[i] = s1[i - 1] + prices[i];
            }
            return max(s0[prices.size() - 1], s2[prices.size() - 1]);
    }
};
```

Output:



Accepted Runtime: 0 ms

• Case 1 • Case 2

Input

prices =

```
[1,2,3,0,2]
```

Output

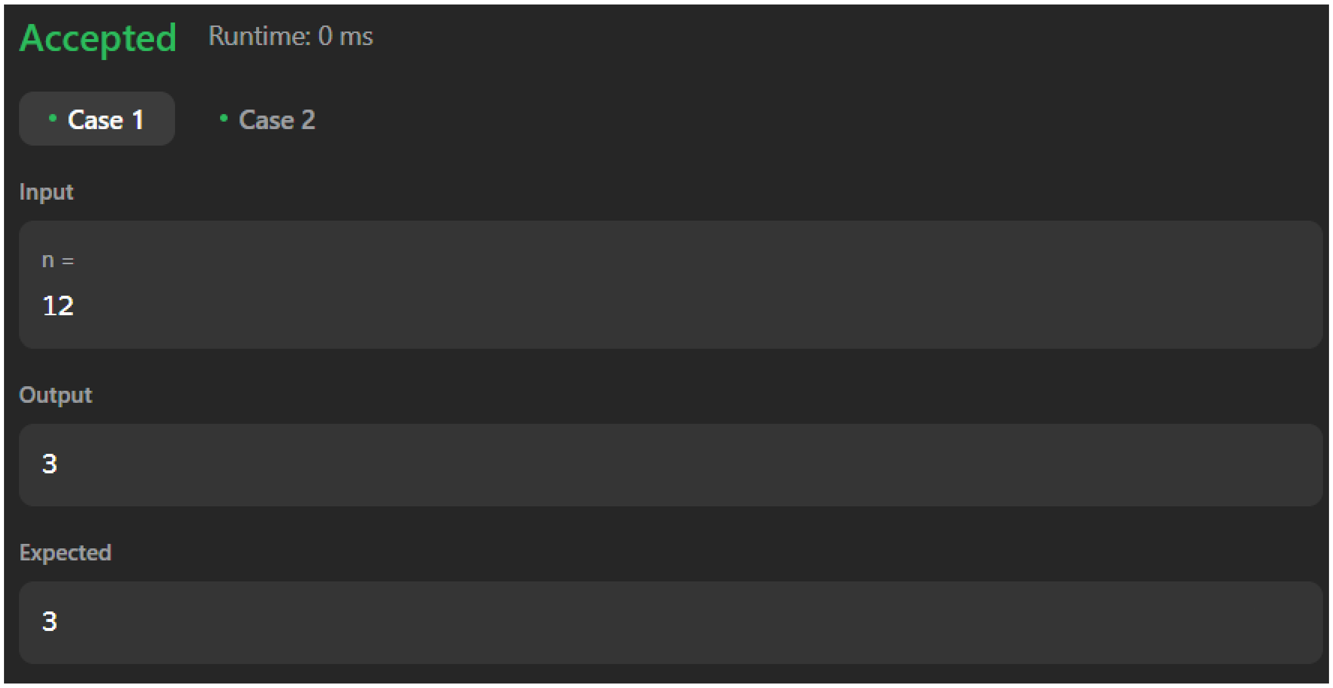
```
3
```

Problem 12. Perfect Squares- Given an integer n, return the least number of perfect square numbers that sum to n.A perfect square is an integer that is the square of an integer; in other words, it is the product of some integer with itself. For example, 1, 4, 9, and 16 are perfect squares while 3 and 11 are not.

Code:

```
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];
    }
};
```

Output:



Accepted Runtime: 0 ms

• Case 1 • Case 2

Input

```
n =
12
```

Output

```
3
```

Expected

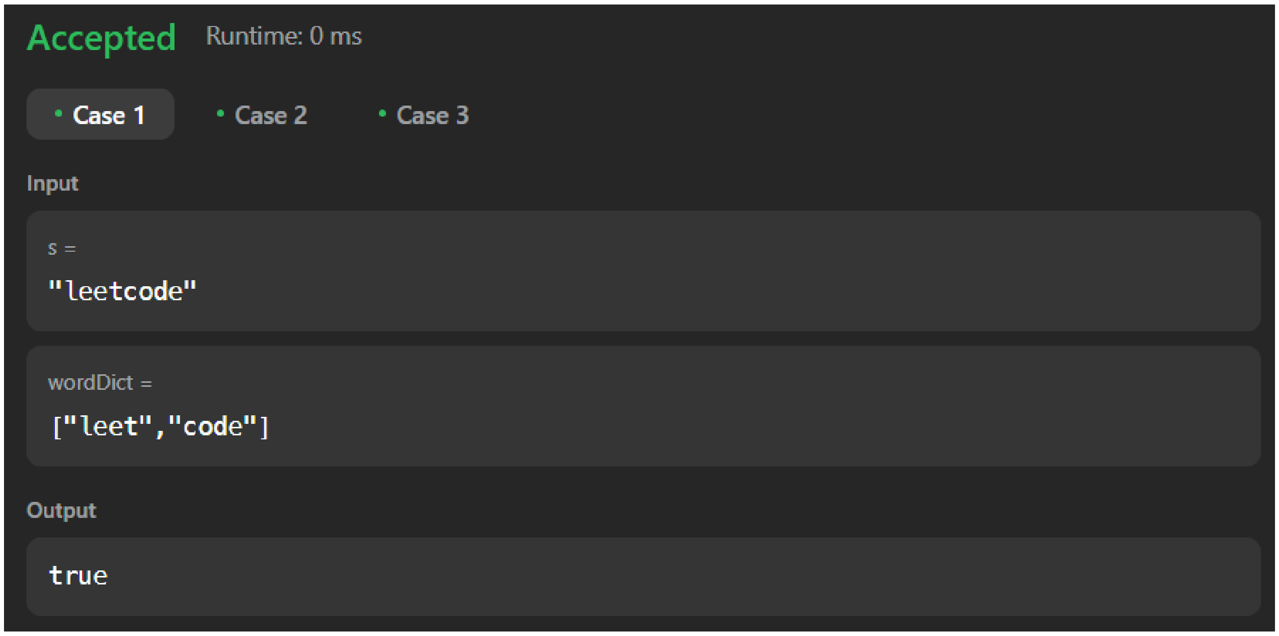
```
3
```

Problem 13. Word Break- Given a string s and a dictionary of strings wordDict, return true if s can be segmented into a space-separated sequence of one or more dictionary words.

Code:

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

Output:



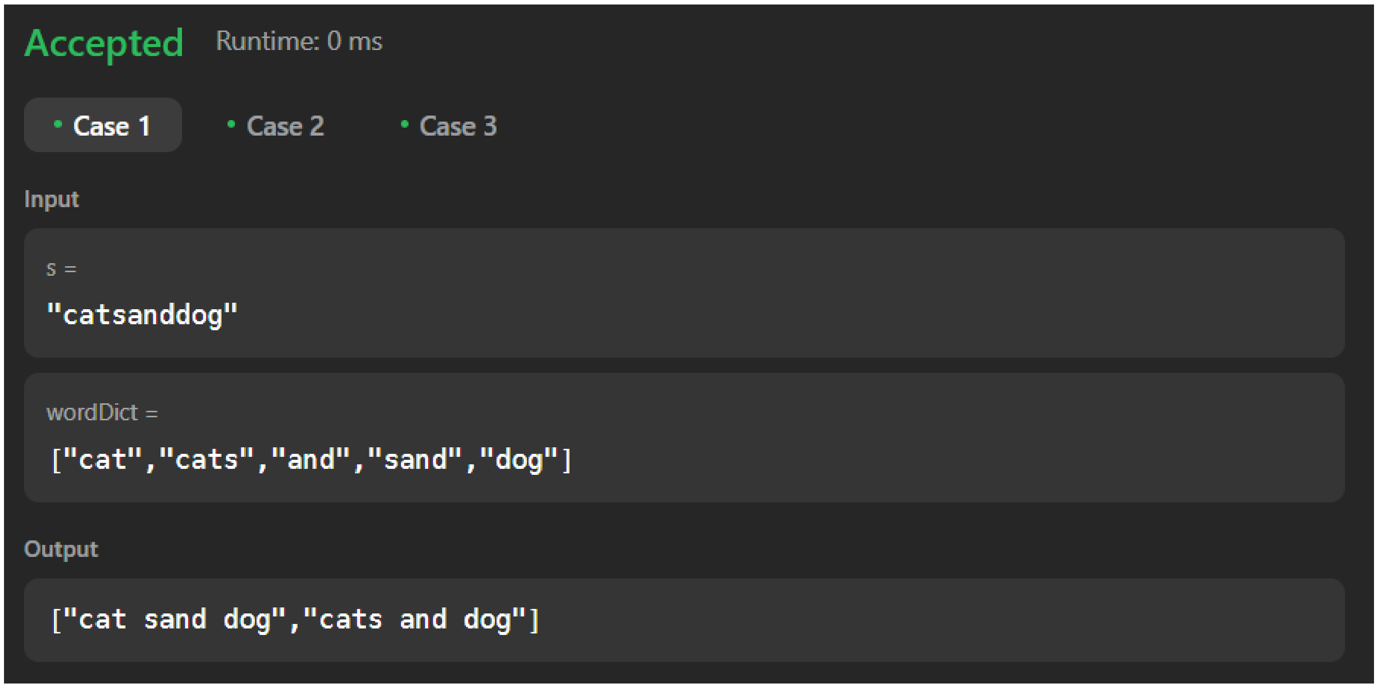
```
Accepted   Runtime: 0 ms

• Case 1   • Case 2   • Case 3

Input
s =
"leetcode"
wordDict =
["leet","code"]

Output
true
```

Problem 14. Word Break 2- Given a string s and a dictionary of strings wordDict, add spaces in s to construct a sentence where each word is a valid dictionary word. Return all such possible sentences in any order.

Code:

```
class Solution
{
public:
    unordered_map<string, vector<string>> m;
    vector<string> wordBreak(string s, vector<string>& wordDict)
    {
        if (m.count(s)) return m[s];
        unordered_set<string> dict(wordDict.begin(), wordDict.end());

        if (dict.count(s)) m[s].push_back(s);
        for (int i = 1; i < s.size(); i++)
        {
            string left = s.substr(0, i), right = s.substr(i);
            if (dict.count(left))
                for (string w : wordBreak(right, wordDict))
                    m[s].push_back(left + " " + w);
        }
        return m[s];
    }
};
```

Output:


Accepted Runtime: 0 ms

Case 1 Case 2 Case 3

Input

```
s =
"catsanddog"
```

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

Output

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