Experiment 7

Student Name: Navkirat Singh
UID: 22BET101232
Branch: IT
Section/Group: 22BET_IOT-701/A
Semester: 6th
Date of Performance: 20/03/2025
Subject Name: Advance Programming-II
Subject Code: 22ITP-367

Problem: 1: Climbing Stairs

Problem Statement: 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?

1. **Objective:** Find in how many distinct ways can you climb to the top? .
2. **Code:**

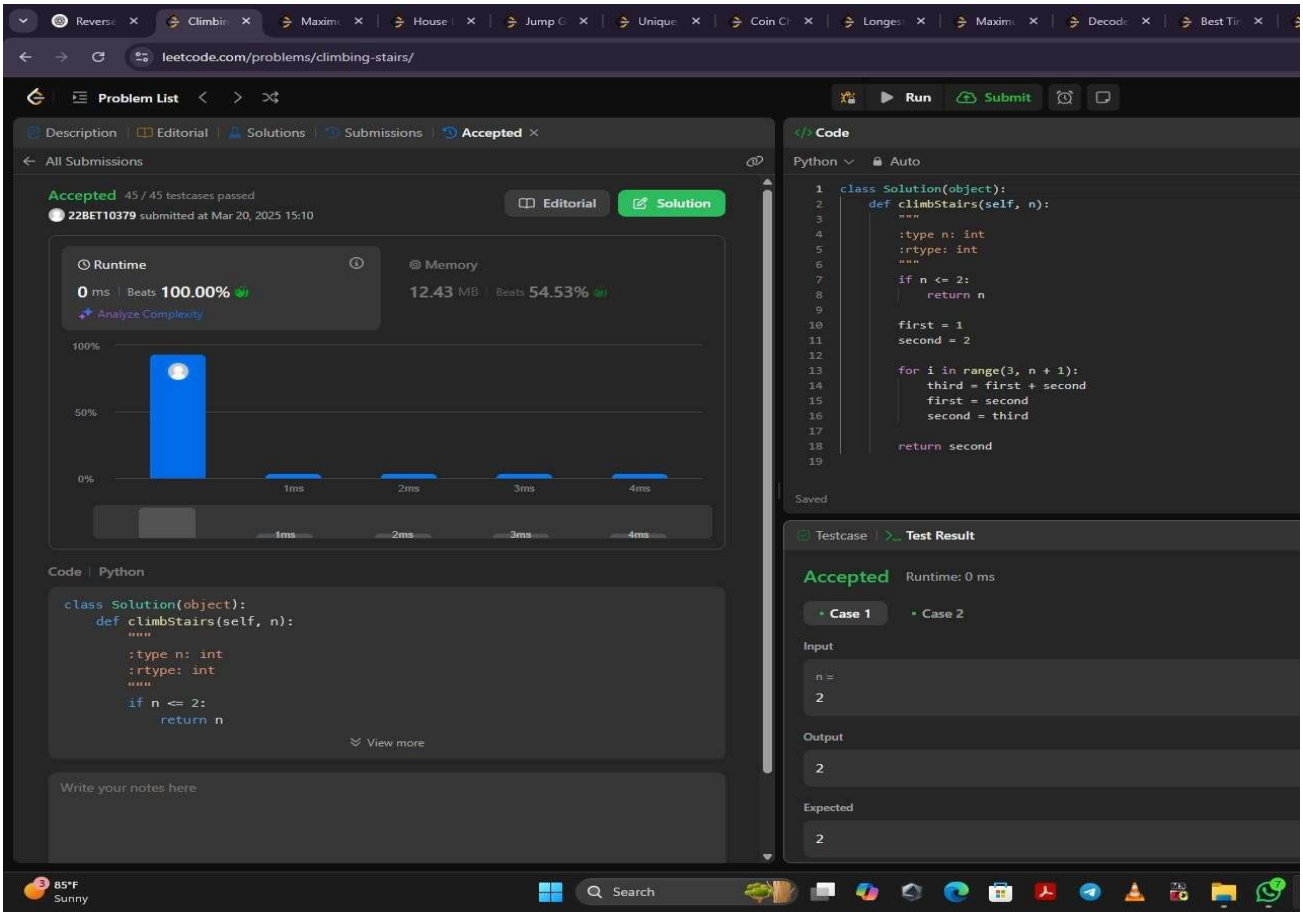
```
class Solution(object):
    def climbStairs(self, n):
        """
        :type n: int
        :rtype: int
        """
        if n <= 2:
            return n

        first = 1
        second = 2

        for i in range(3, n + 1):
            third = first + second
            first = second
            second = third

        return second
```

3. Result:



Problem 2: Best Time to Buy and Sell a Stock

Problem Statement : 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.

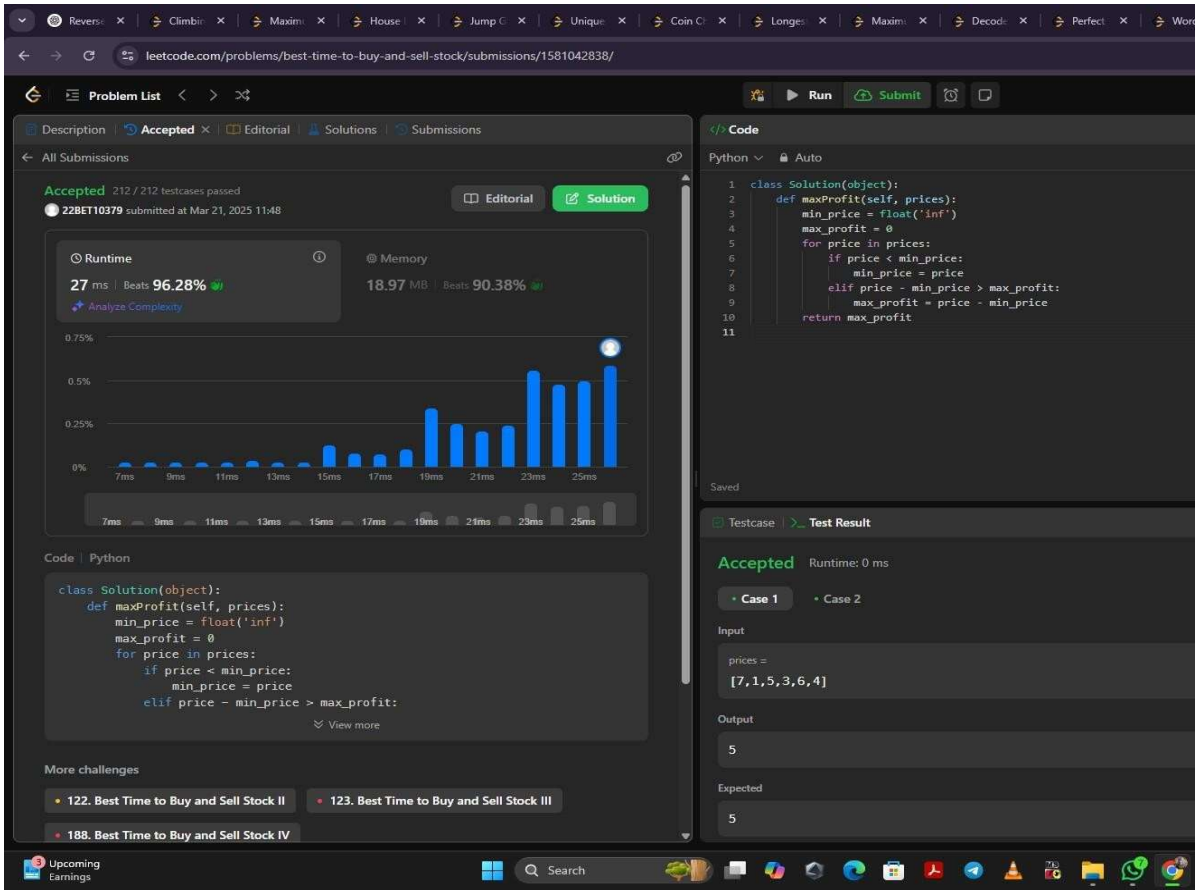
Return the maximum profit you can achieve from this transaction. If you cannot achieve any profit, return 0.

1. **Objective:** To Find the maximum profit by buying one stock and selling that stock in a single day.

2. Code:

```
class Solution(object):
    def maxProfit(self, prices):
        min_price = float('inf')
        max_profit = 0
        for price in prices:
            if price < min_price:
                min_price = price
            elif price - min_price > max_profit:
                max_profit = price - min_price
        return max_profit
```

3. Result:



Problem 3: Maximum Subarray

Problem Statement: Given an integer array nums, find the subarray with the largest sum, and return its sum.

1. **Objective:** Find the subarray with the largest sum, and return its sum.

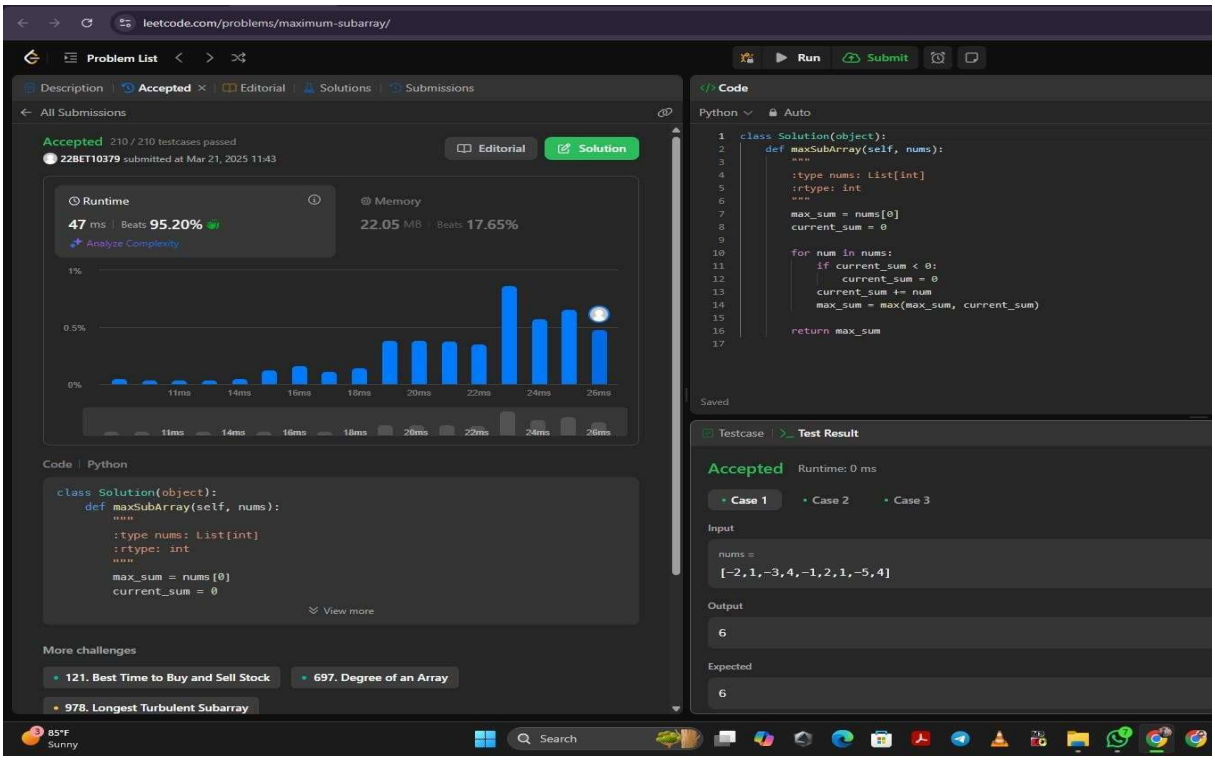
2. **Code:**

```
class Solution(object):
    def maxSubArray(self, nums):
        """
        :type nums: List[int]
        :rtype: int
        """
        max_sum = nums[0]
        current_sum = 0

        for num in nums:
            if current_sum < 0:
                current_sum = 0
            current_sum += num
            max_sum = max(max_sum, current_sum)

        return max_sum
```

3. **Result:**



Problem 4: House Robber

Problem Statement: 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.

Given an integer array nums representing the amount of money of each house, return the maximum amount of money you can rob tonight without alerting the police.

1. **Objective:** To return the maximum amount of money you can rob tonight without alerting the police.
2. **Code:**

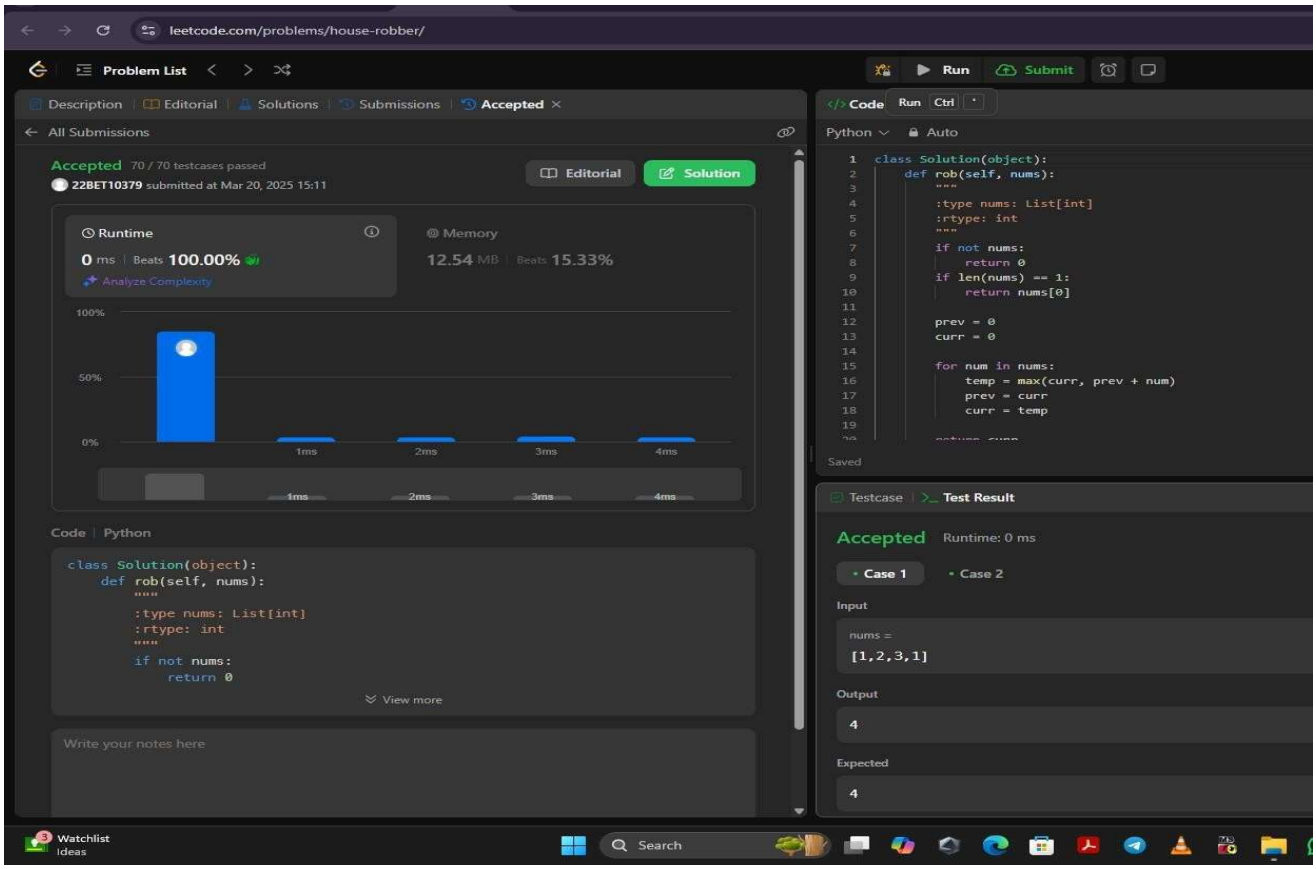
```
class Solution(object):
    def rob(self, nums):
        """
        :type nums: List[int]
        :rtype: int
        """
        if not nums:
            return 0
        if len(nums) == 1:
            return nums[0]

        prev = 0
        curr = 0

        for num in nums:
            temp = max(curr, prev + num)
            prev = curr
            curr = temp

        return curr
```

Result:



Problem 5: Jump Game

Problem Statement: 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.

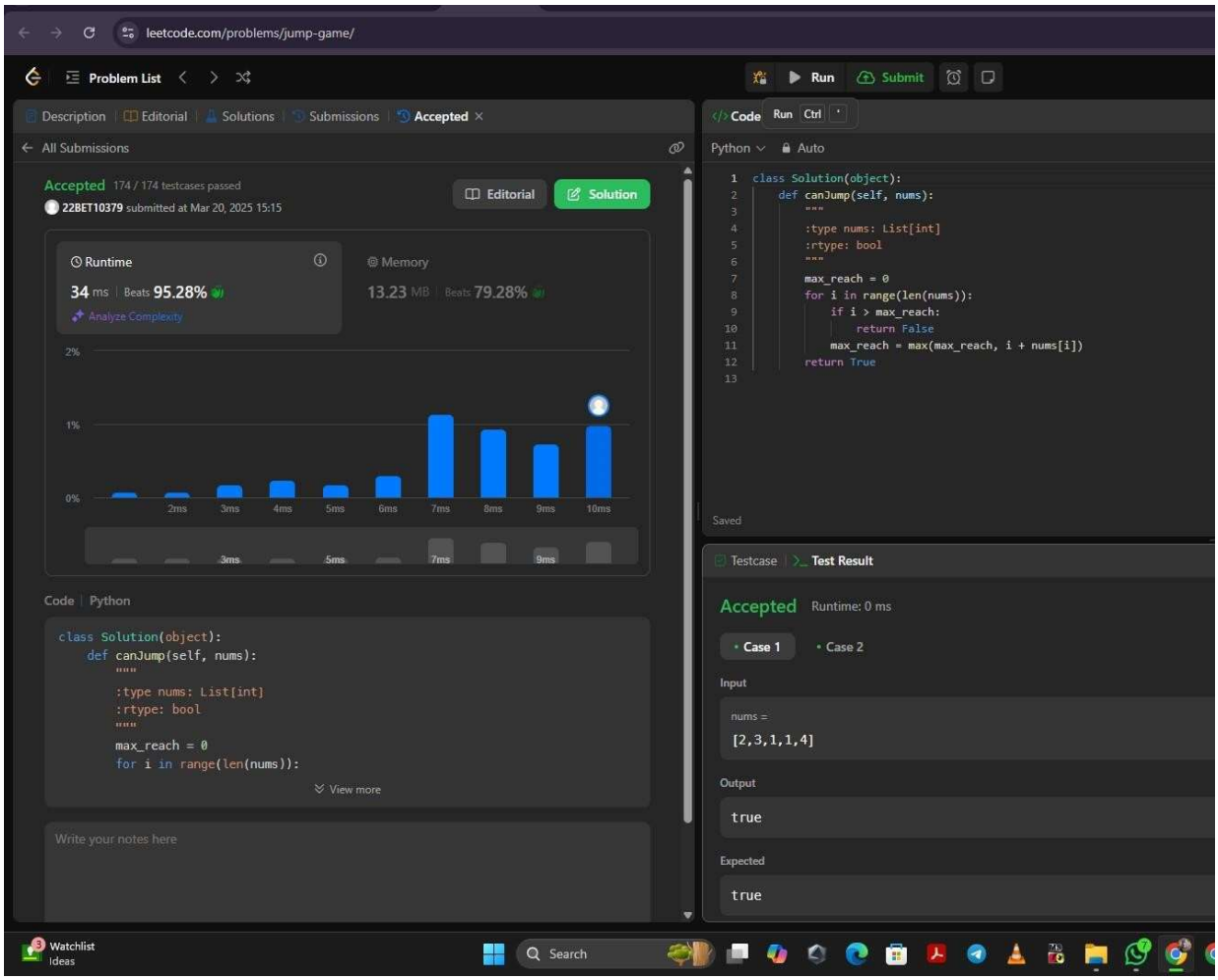
1. **Objective:** To find the your maximum jump length at that position.

2. **Code:**

```
class Solution(object):
    def canJump(self, nums):
        """
        :type nums: List[int]
        :rtype: bool
        """
        max_reach = 0
```

```
for i in range(len(nums)):
        if i > max_reach:
            return False
        max_reach = max(max_reach, i + nums[i])
    return True
```

3. Result:



Problem 6:Unique Paths

Problem Statement: 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.

Given the two integers m and n, return the number of possible unique paths that the robot can take to reach the bottom-right corner.

The test cases are generated so that the answer will be less than or equal to 2 * 109.

1. Objective: Find the two integers m and n, return the number of possible unique paths that the robot can take to reach the bottom-right corner.

The test cases are generated so that the answer will be less than or equal to 2 * 109.

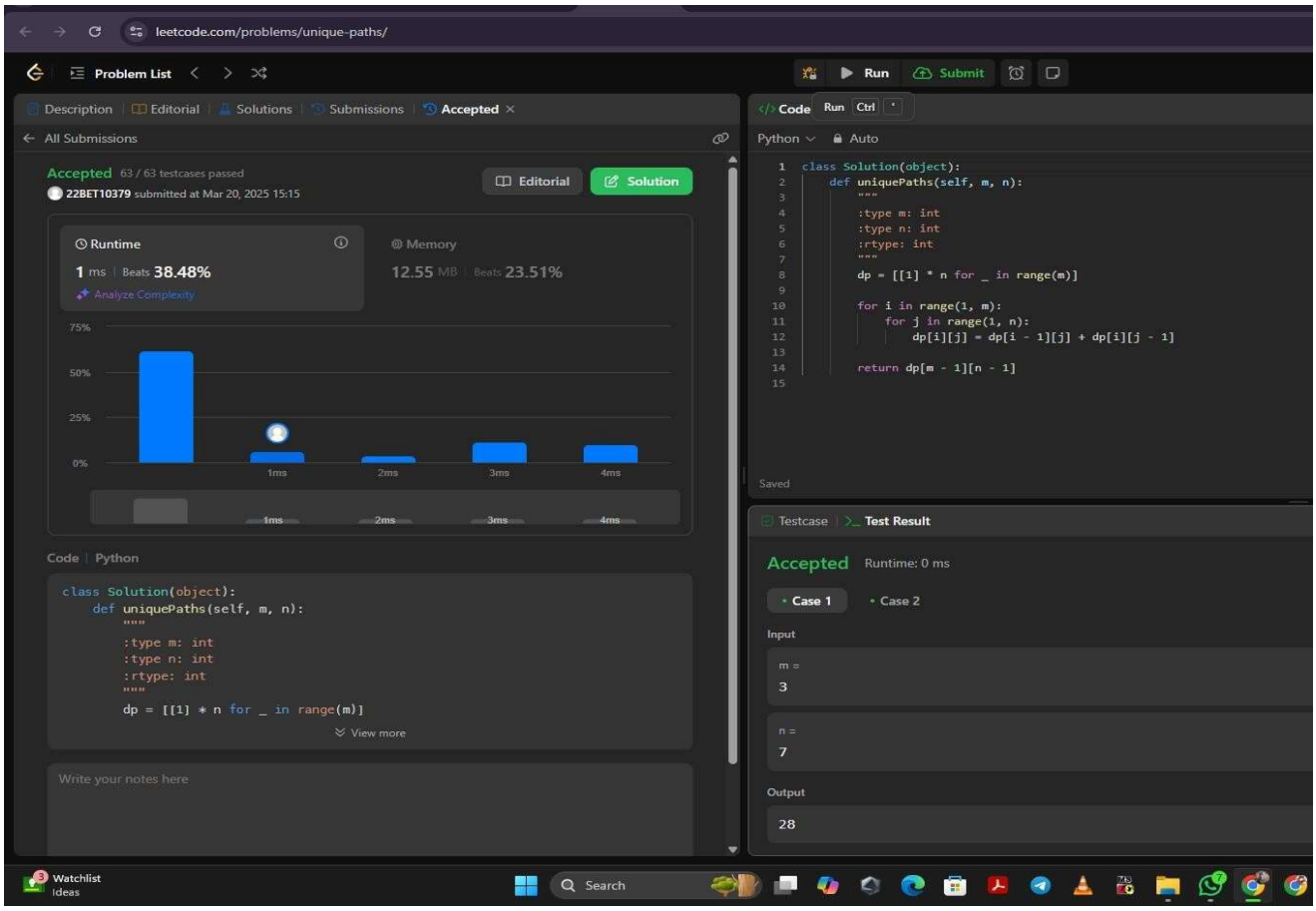
2. Code:

```
class Solution(object):
    def uniquePaths(self, m, n):
        """
        :type m: int
        :type n: int
        :rtype: int
        """
        dp = [[1] * n for _ in range(m)]

        for i in range(1, m):
            for j in range(1, n):
                dp[i][j] = dp[i - 1][j] + dp[i][j - 1]

        return dp[m - 1][n - 1]
```

3. Result:



Problem 7: Coin Change

Problem Statement: You are given an integer array coins representing coins of different denominations and an integer amount representing a total amount of money.

Return the fewest number of coins that you need to make up that amount. If that amount of money cannot be made up by any combination of the coins, return -1.

You may assume that you have an infinite number of each kind of coin.

1. Objective:

If that amount of money cannot be made up by any combination of the coins, return -1.

2. Code:

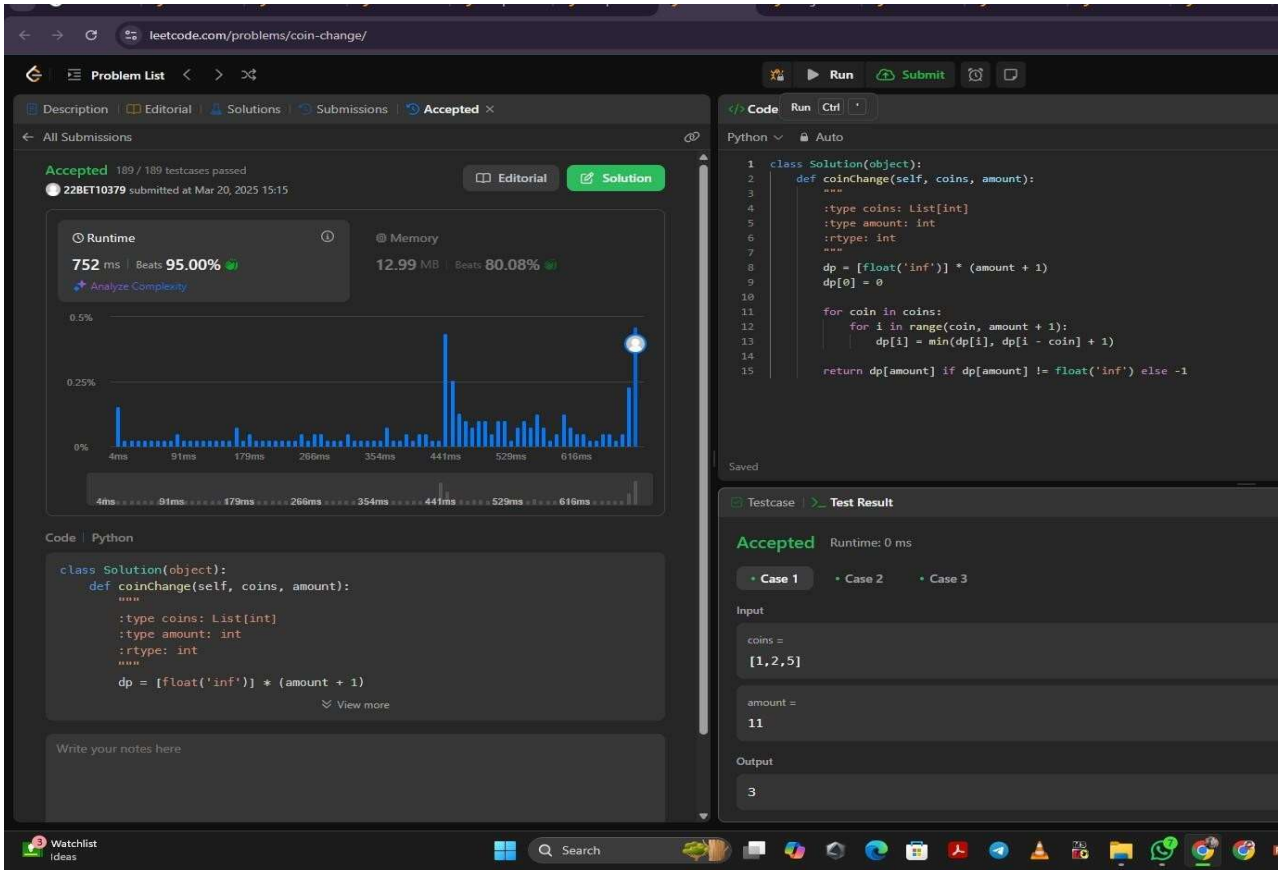
```
class Solution(object):
    def coinChange(self, coins, amount):
        """
        :type coins: List[int]
        :type amount: int
        :rtype: int
        """
```

```
dp = [float('inf')] * (amount + 1)
dp[0] = 0

for coin in coins:
    for i in range(coin, amount + 1):
        dp[i] = min(dp[i], dp[i - coin] + 1)

return dp[amount] if dp[amount] != float('inf') else -1
```

3. Result:



Problem 8: Longest Increasing Subsequence

Problem Statement: Given an integer array nums, return the length of the longest strictly increasing subsequence.

1. Objective:

Find an integer array nums, return the length of the longest strictly increasing subsequence.

2. Code:

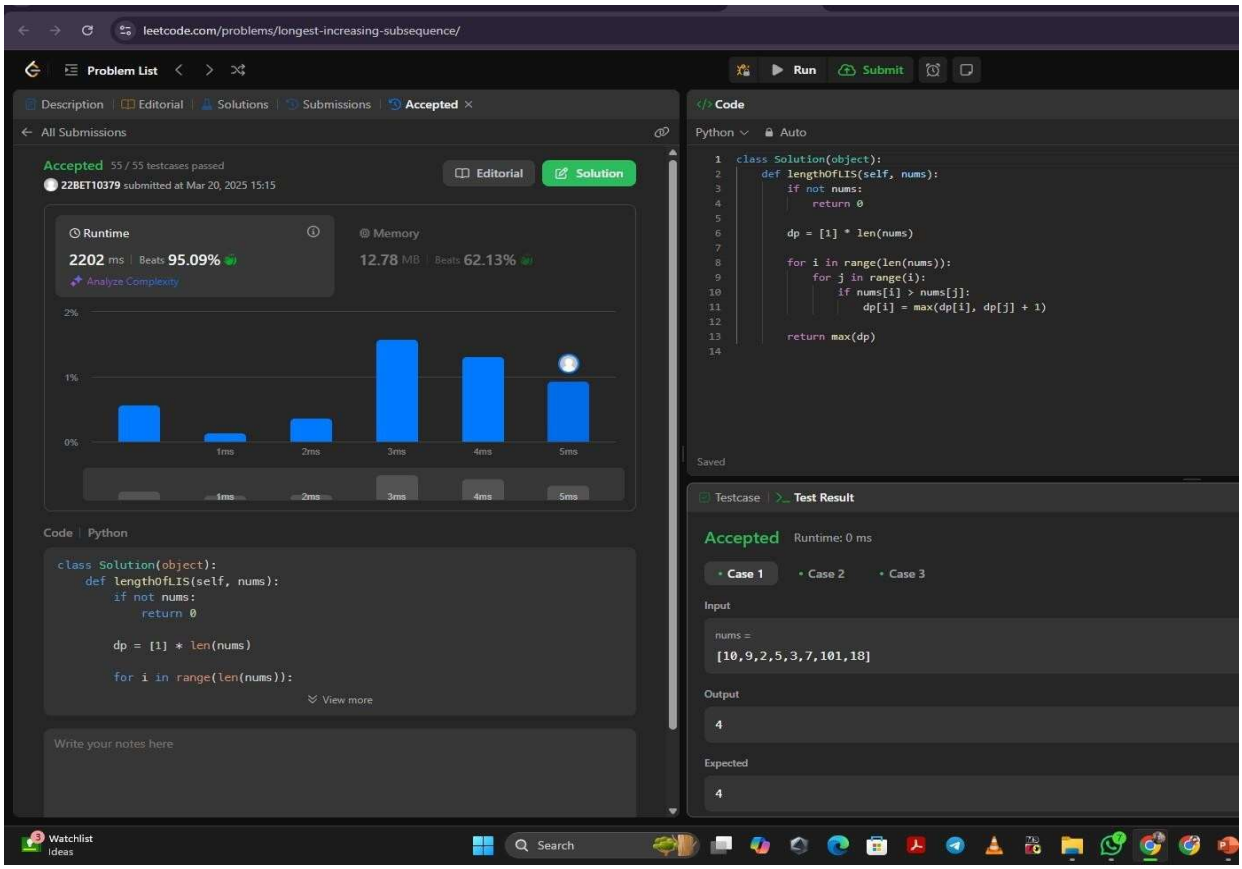
```
class Solution(object):
    def lengthOfLIS(self, nums):
        if not nums:
            return 0

        dp = [1] * len(nums)

        for i in range(len(nums)):
            for j in range(i):
                if nums[i] > nums[j]:
                    dp[i] = max(dp[i], dp[j] + 1)

        return max(dp)
```

3. Result:



Problem 9: Maximum Product Subarray

Problem Statement: 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.

1. **Objective:** Find 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.

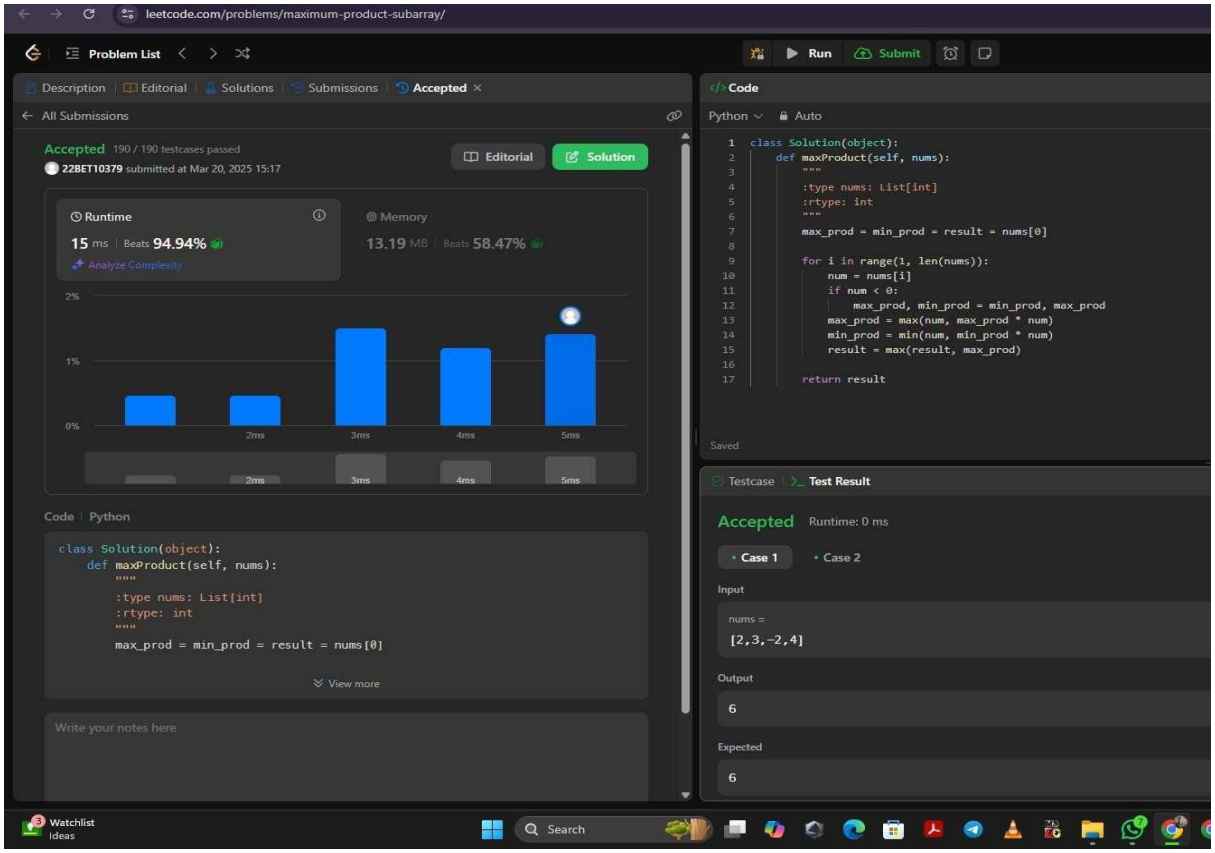
2. **Code:**

```
class Solution(object):
    def maxProduct(self, nums):
        """
        :type nums: List[int]
        :rtype: int
        """
        max_prod = min_prod = result = nums[0]

        for i in range(1, len(nums)):
            num = nums[i]
            if num < 0:
                max_prod, min_prod = min_prod, max_prod
            max_prod = max(num, max_prod * num)
            min_prod = min(num, min_prod * num)
            result = max(result, max_prod)

        return result
```

3. Result:



Problem 10: Decode Ways

Problem Statement: Given a string s containing only digits, return the number of ways to decode it. If the entire string cannot be decoded in any valid way, return 0.

The test cases are generated so that the answer fits in a 32-bit integer..

1. **Objective:** Find the string s containing only digits, return the number of ways to decode it. If the entire string cannot be decoded in any valid way, return 0.

   The test cases are generated so that the answer fits in a 32-bit integer.

2. Code:

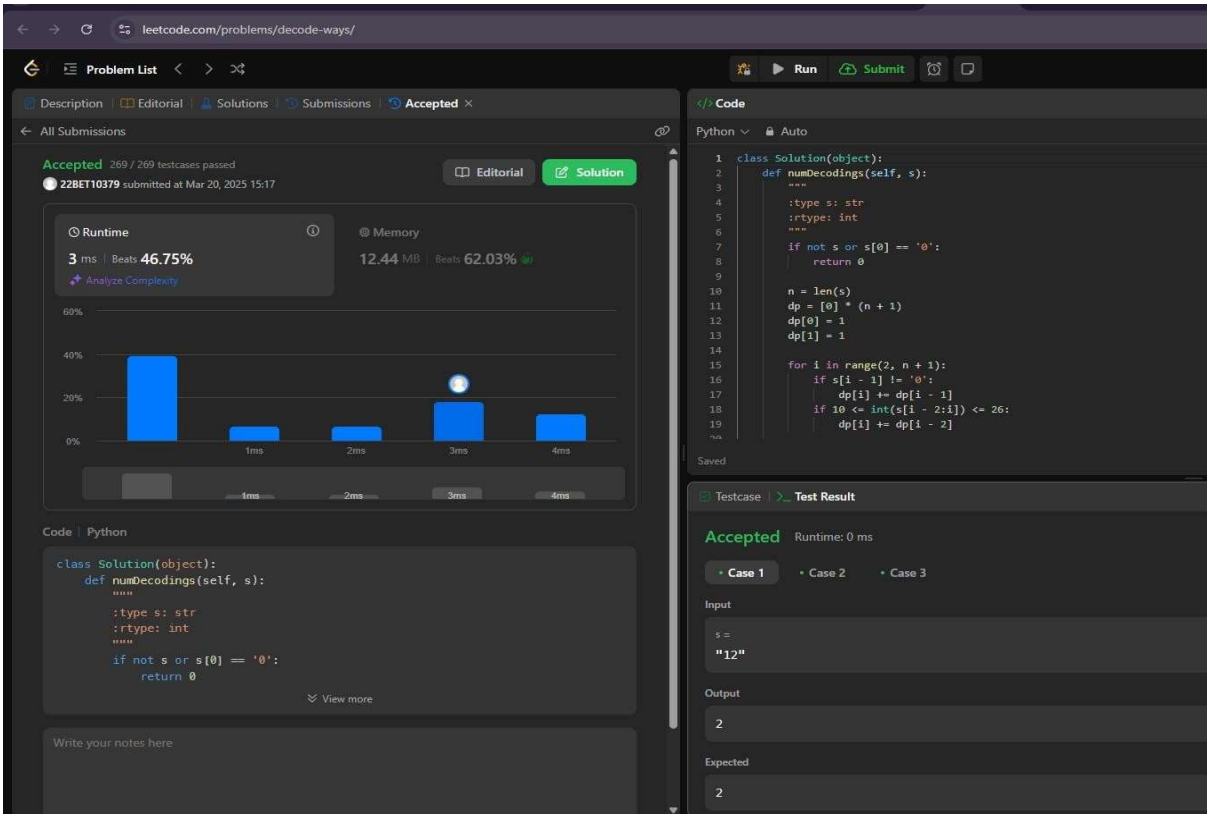
```
class Solution(object):
    def numDecodings(self, s):
        """
        :type s: str
        :rtype: int
        """
        if not s or s[0] == '0':
            return 0

        n = len(s)
        dp = [0] * (n + 1)
        dp[0] = 1
        dp[1] = 1

        for i in range(2, n + 1):
            if s[i - 1] != '0':
                dp[i] += dp[i - 1]
            if 10 <= int(s[i - 2:i]) <= 26:
                dp[i] += dp[i - 2]

        return dp[n]
```

3. Result:



Problem 11: Perfect Squares

Problem Statement: 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.

1. **Objective:** Find an integer n, return the least number of perfect square numbers that sum to n.

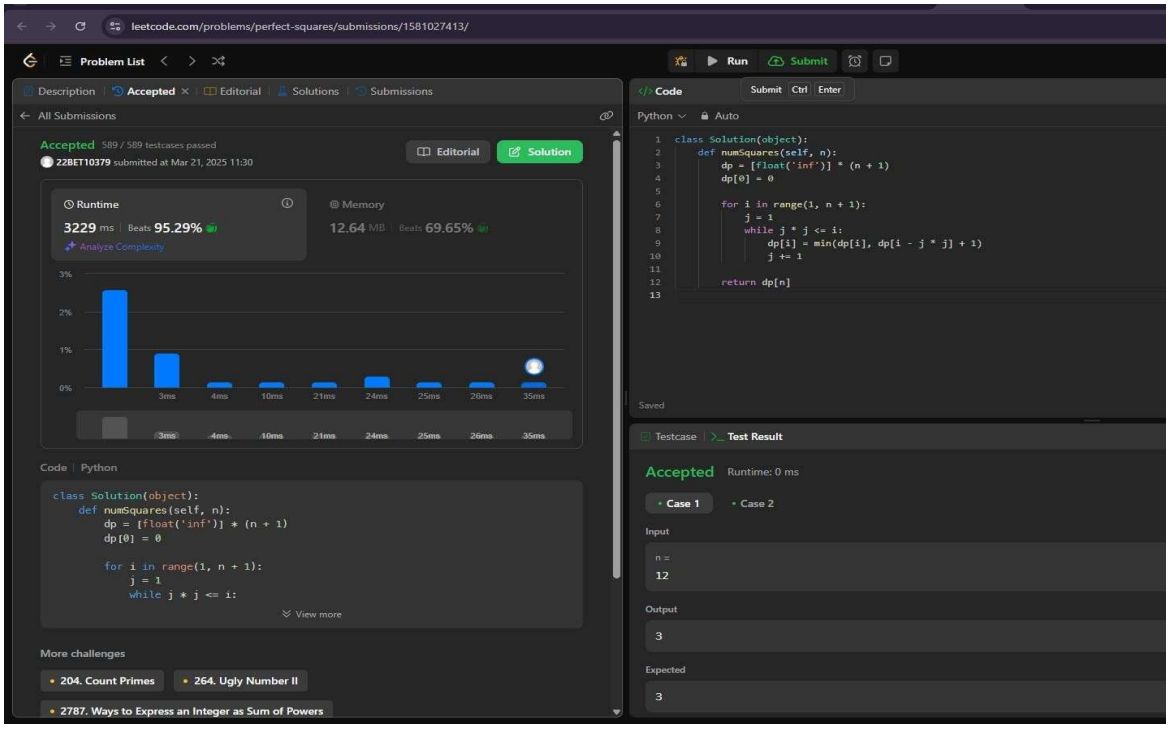
2. **Code:**

```
class Solution(object):
    def numSquares(self, n):
        dp = [float('inf')] * (n + 1)
        dp[0] = 0

        for i in range(1, n + 1):
            j = 1
            while j * j <= i:
                dp[i] = min(dp[i], dp[i - j * j] + 1)
                j += 1

        return dp[n]
```

3. **Result:**



Problem 12: Word Break

Problem Statement: 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.

1. **Objective:** Find the 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.

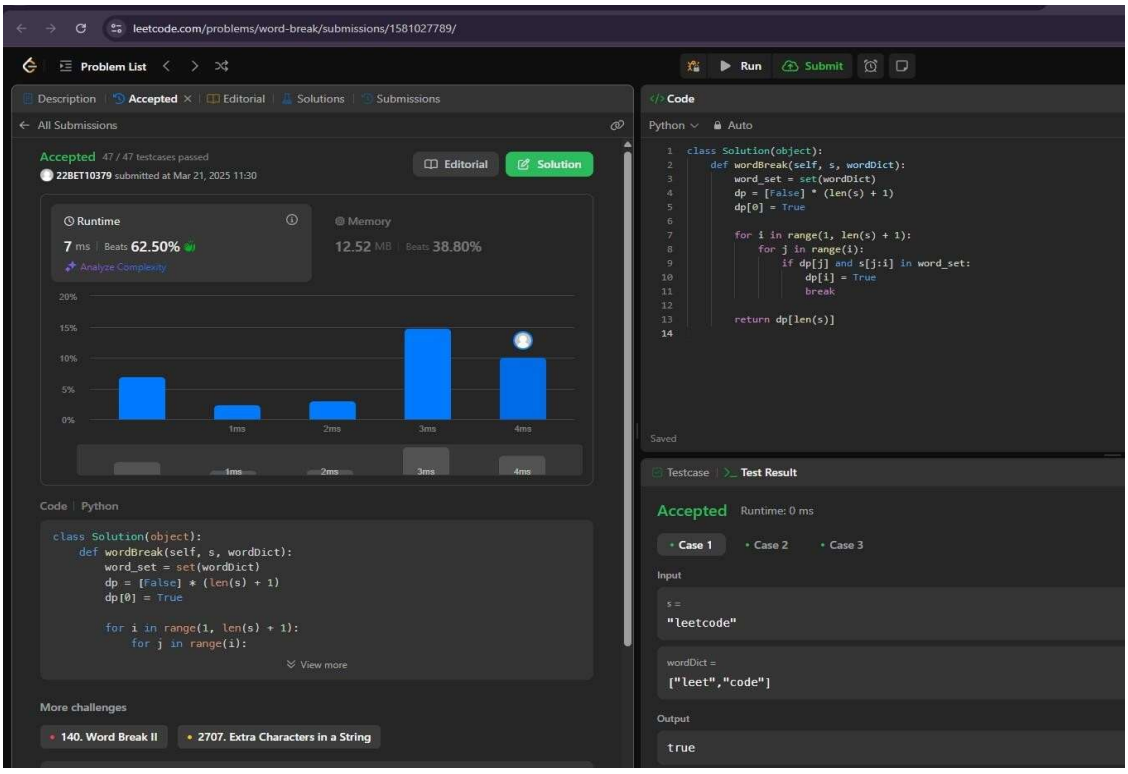
2. **Code:**

```
class Solution(object):
    def wordBreak(self, s, wordDict):
        word_set = set(wordDict)
        dp = [False] * (len(s) + 1)
        dp[0] = True

        for i in range(1, len(s) + 1):
            for j in range(i):
                if dp[j] and s[j:i] in word_set:
                    dp[i] = True
                    break

        return dp[len(s)]
```

3. **Result:**



Problem 13: Word Break 2

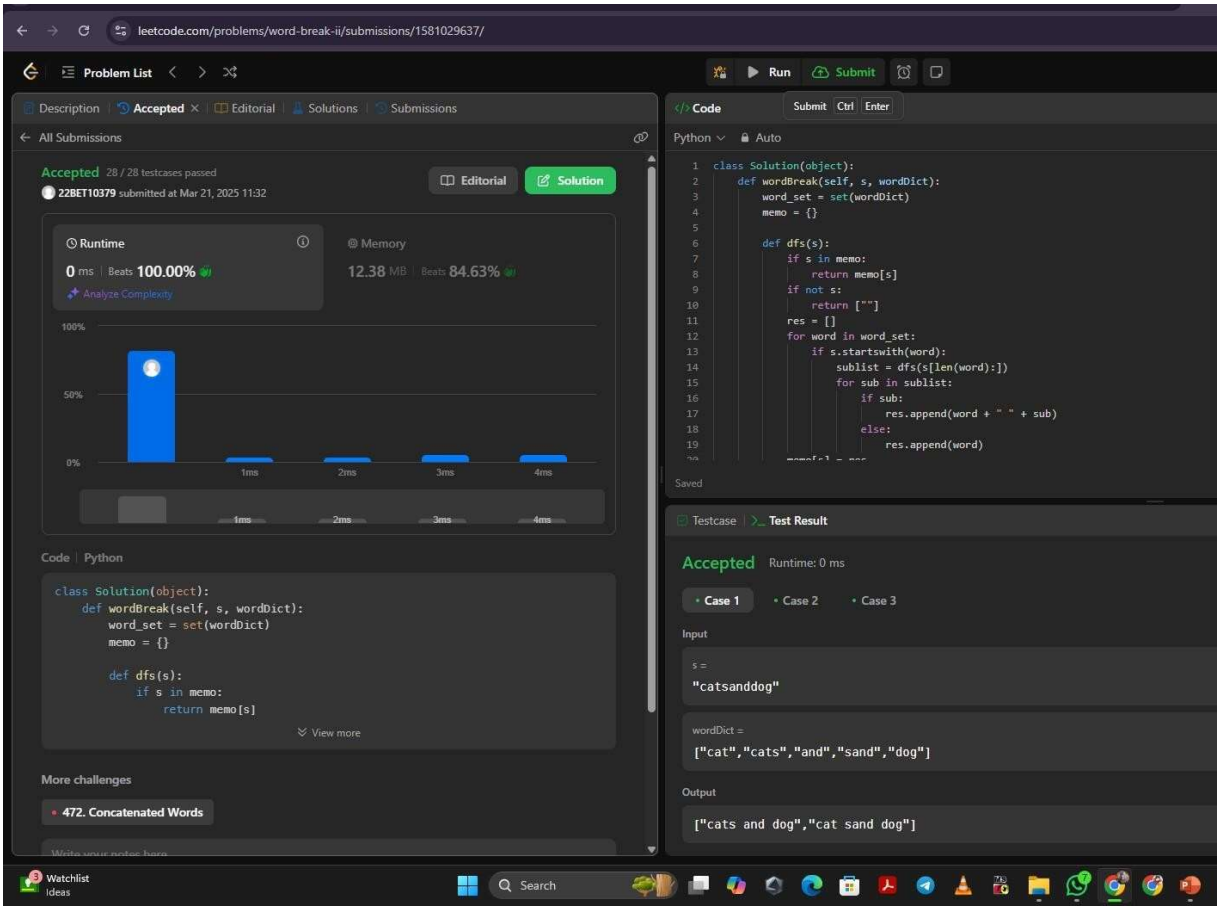
Problem Statement: Given a string s and a dictionary of strings wordDict, add spaces in s to construct a sentence where each word is a valid dictionary word. Return all such possible sentences in any order.

1. **Objective:** Find the 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.

2. **Code:**

```
class Solution(object):
    def wordBreak(self, s, wordDict):
        word_set = set(wordDict)
        memo = {}
        def dfs(s):
            if s in memo:
                return memo[s]
            if not s:
                return [""]
            res = []
            for word in word_set:
                if s.startswith(word):
                    sublist = dfs(s[len(word):])
                    for sub in sublist:
                        if sub:
                            res.append(word + " " + sub)
                        else:
                            res.append(word)
            memo[s] = res
            return res
        return dfs(s)
```

3. Result:



✅ Key Learning Outcomes:

- Recursive problem-solving using DFS.
- Optimization with memoization (dynamic programming).
- Efficient word lookup using sets.
- String manipulation and prefix checking.
- Generating all possible valid sentences from a word dictionary.