Experiment-7

Student Name: Abhinav Paswan **UID:** 22BET10332
Branch: BE-IT **Section/Group:** 22BET-IOT-701(A)
Semester: 6th **Date of Performance:** 19-03-2025
Subject Name: AP LAB-II **Subject Code:** 22ITP-351

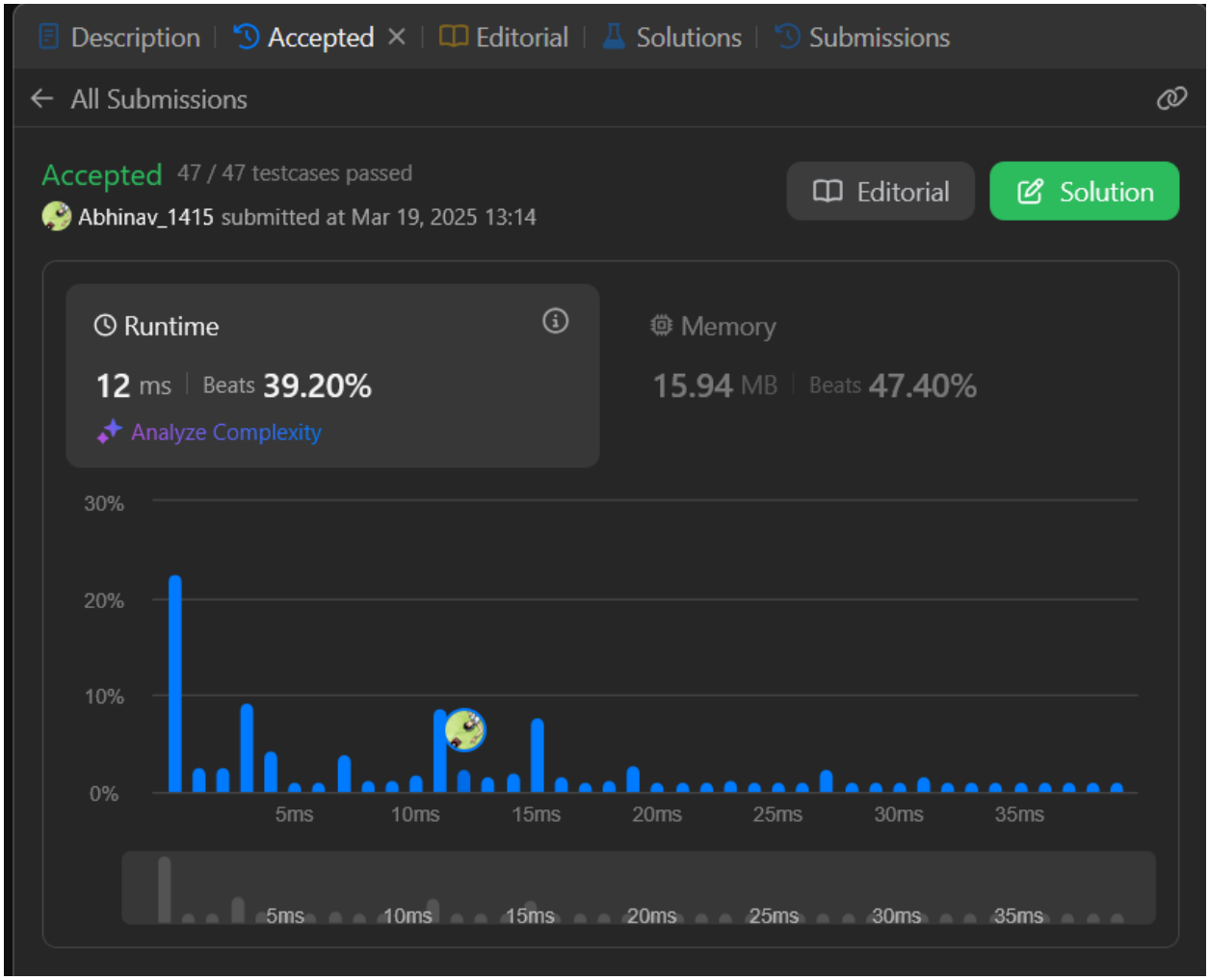
Problem 1:-

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.
Note that the same word in the dictionary may be reused multiple times in the segmentation.

Code:

```
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];
}
};
```

OUTPUT:


Description
Accepted
Editorial
Solutions
Submissions
All Submissions
Accepted 47 / 47 testcases passed
Abhinav_1415 submitted at Mar 19, 2025 13:14
Editorial
Solution
Runtime
12 ms | Beats 39.20%
Analyze Complexity
Memory
15.94 MB | Beats 47.40%
30%
20%
10%
0%
5ms
10ms
15ms
20ms
25ms
30ms
35ms

Problem 2 :-

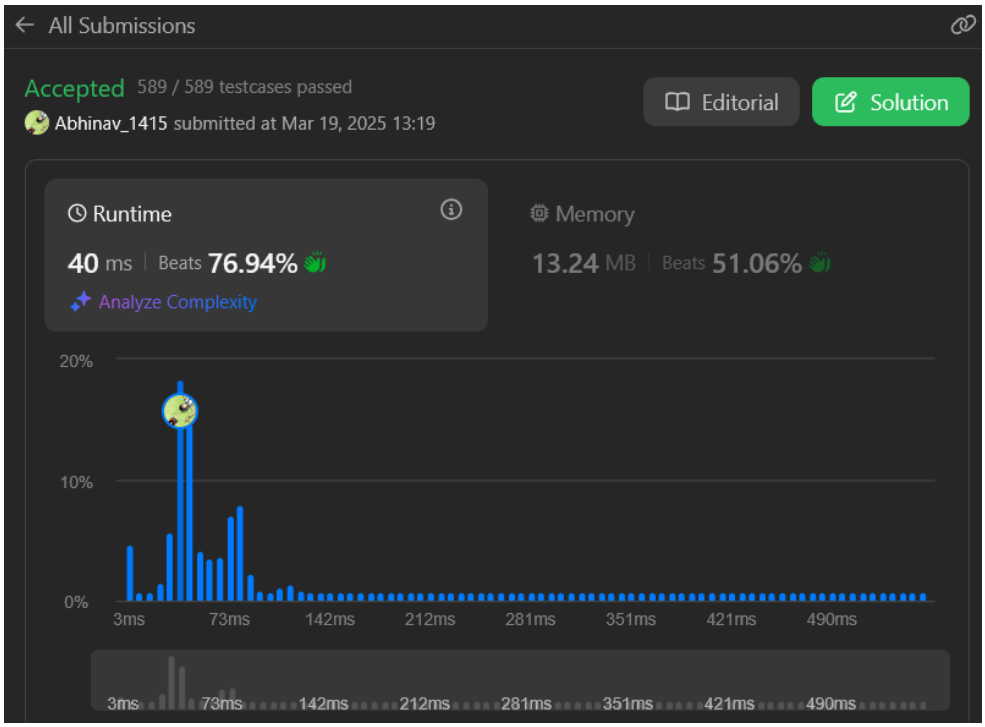
Given an integer n, return the least number of perfect square numbers that sum to n.

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:



Problem 3:-

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'

However, while decoding the message, you realize that there are many different ways you can decode the message because some codes are contained in other codes ("2" and "5" vs "25").

For example, "11106" can be decoded into:

"AAJF" with the grouping (1, 1, 10, 6)
"KJF" with the grouping (11, 10, 6)
The grouping (1, 11, 06) is invalid because "06" is not a valid code (only "6" is valid).

Note: there may be strings that are impossible to decode.

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.

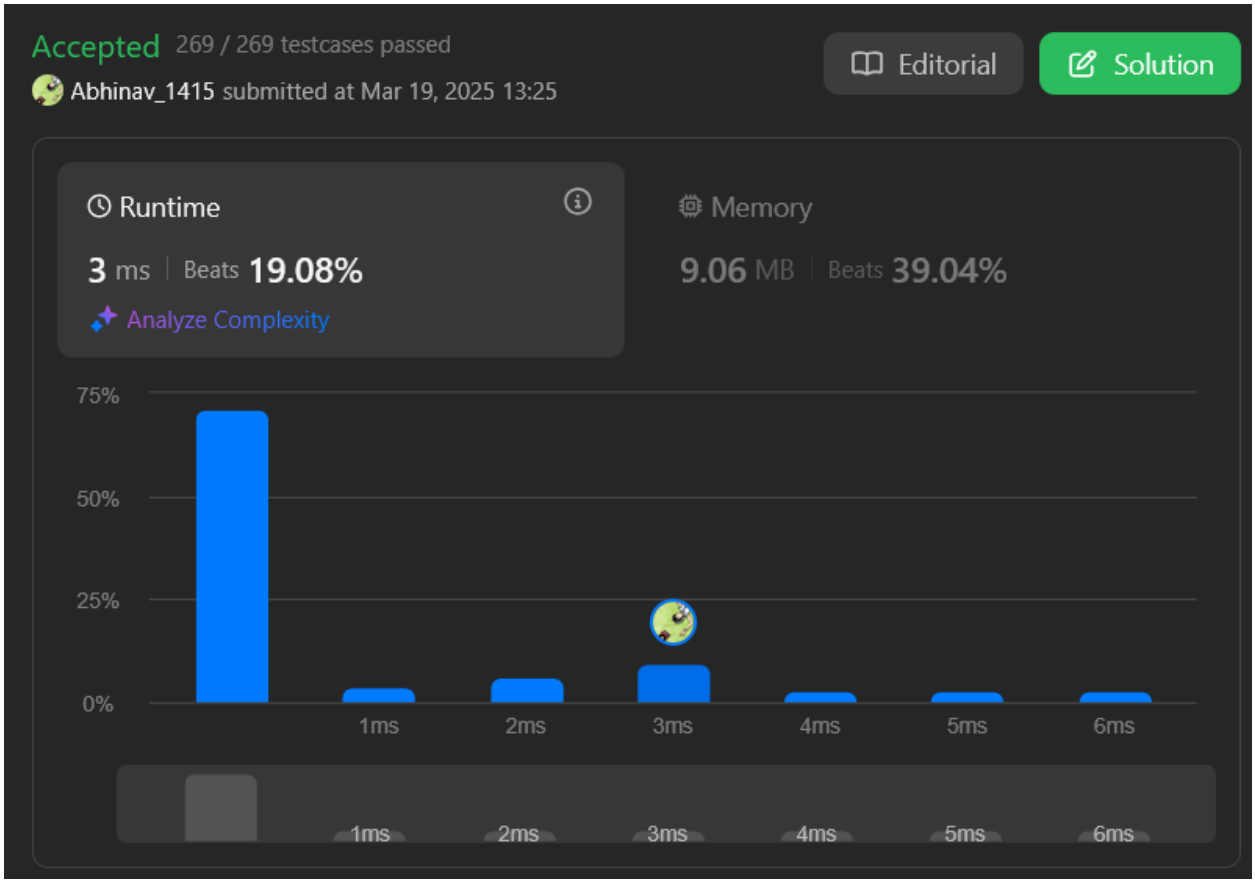
The test cases are generated so that the answer fits in a 32-bit integer.

Code:

```
class Solution {
public:
    int numDecodings(string s) {
    if (s.empty() || s[0] == '0') return 0;

    int n = s.size();
    vector<int> dp(n + 1, 0);
    dp[0] = 1;
    dp[1] = (s[0] != '0') ? 1 : 0;

    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];
}
};
```

OUTPUT:



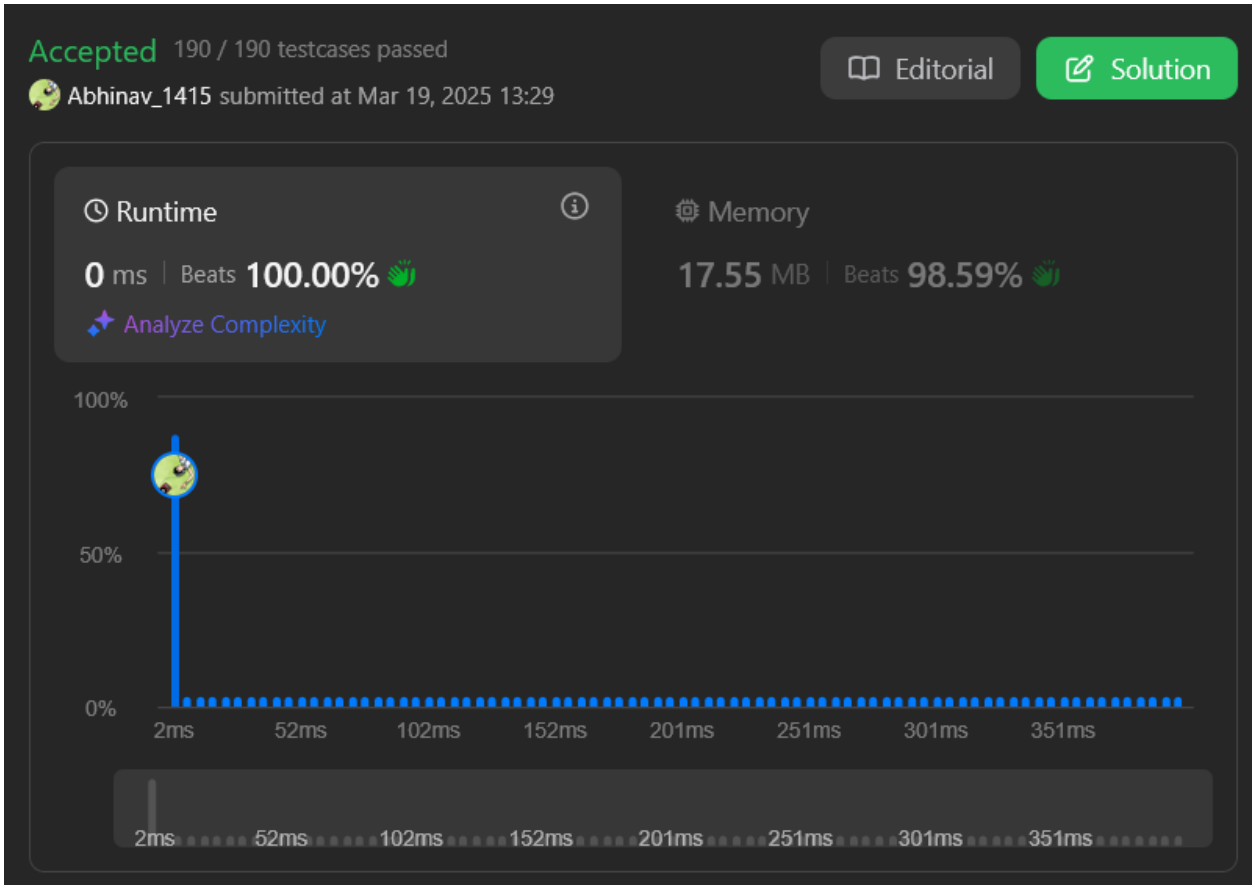
Problem 4:-

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) {
    if (nums.empty()) return 0;

    int maxProd = nums[0], minProd = nums[0], result = nums[0];

    for (int i = 1; i < nums.size(); ++i) {
        if (nums[i] < 0) swap(maxProd, minProd);

        maxProd = max(nums[i], nums[i] * maxProd);
        minProd = min(nums[i], nums[i] * minProd);

        result = max(result, maxProd);
    }

    return result;
}
};
```

OUTPUT:


Accepted 190 / 190 testcases passed
Abhinav_1415 submitted at Mar 19, 2025 13:29
Editorial
Solution
Runtime
0 ms | Beats 100.00%
Analyze Complexity
Memory
17.55 MB | Beats 98.59%
100%
50%
0%
2ms
52ms
102ms
152ms
201ms
251ms
301ms
351ms

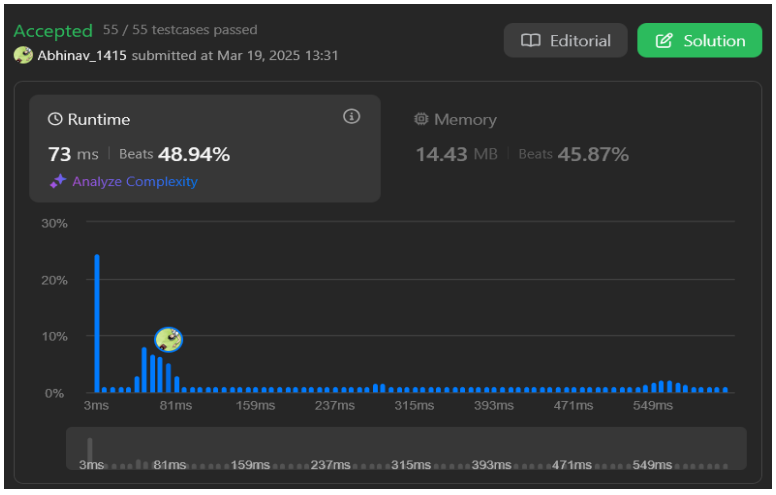
Problem 5:-

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

Code:-

```
class Solution {
public:
    int lengthOfLIS(vector<int>& nums) {
    int n = nums.size();
    if (n == 0) return 0;

    vector<int> dp(n, 1);
    int maxLength = 1;

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

    return maxLength;
}
};
```

OUTPUT:



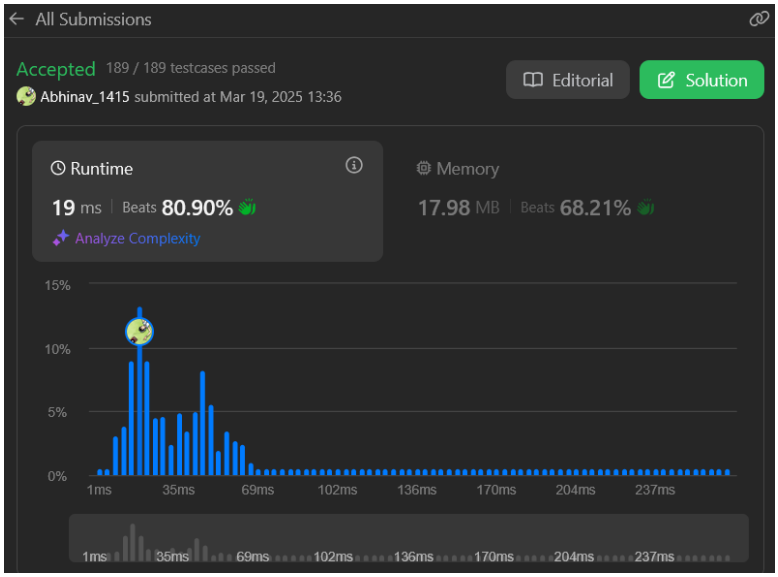
Problem 6:

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

Code:

```
class Solution {
public:
  int coinChange(vector<int>& coins, int amount) {
   vector<int> dp(amount + 1, amount + 1);
   dp[0] = 0;

   for (int i = 1; i <= amount; ++i) {
      for (int coin : coins) {
         if (i >= coin) {
            dp[i] = min(dp[i], dp[i - coin] + 1);
         }
      }
   }

   return (dp[amount] == amount + 1) ? -1 : dp[amount];
}
};
```

OUTPUT:



Problem 7:

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.

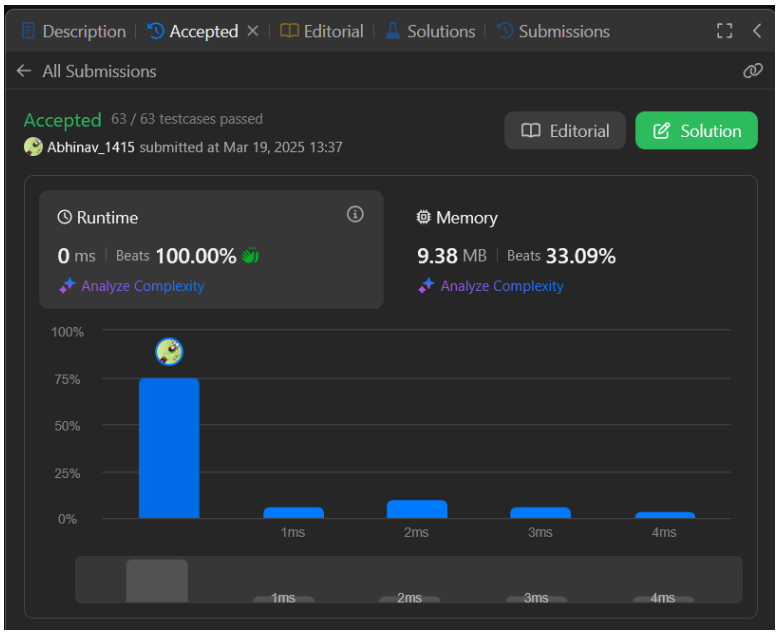
Given the two integers m and n, return *the number of possible unique paths that the robot can take to reach the bottom-right corner*.

The test cases are generated so that the answer will be less than or equal to $2 * 10^9$.

Code:

```
#include <vector>
using namespace std;
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];
    }
};
```

OUTPUT:



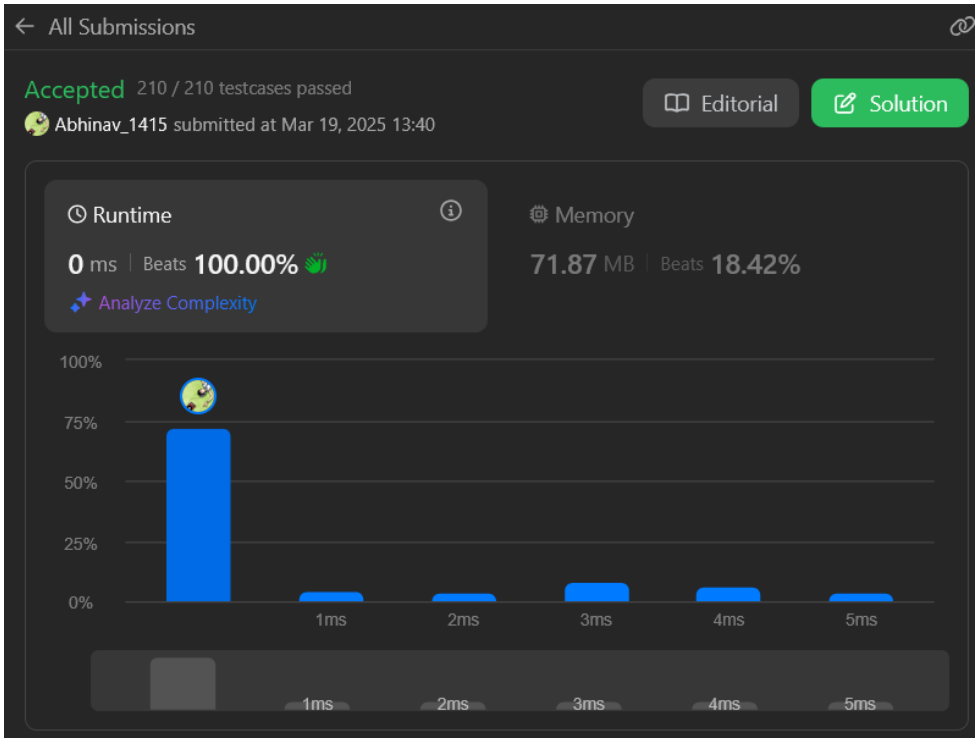
Problem 8:

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 maxSum = nums[0];
    int currentSum = nums[0];

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

    return maxSum;
}
};
```

OUTPUT:



Problem 9:

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*.

CODE:

```
class Solution {
public:
  int rob(vector<int>& nums) {
   if (nums.empty()) return 0;
   if (nums.size() == 1) return nums[0];

   int prev2 = 0;
   int prev1 = 0;

   for (int num : nums) {
      int current = max(prev1, num + prev2);
      prev2 = prev1;
      prev1 = current;
   }

   return prev1;
}
};
```

OUTPUT: