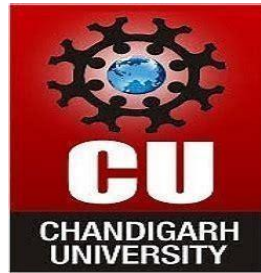


# **UNIVERSITY INSTITUTE OF ENGINEERING**

**Department of Computer Science & Engineering**

**(BE-CSE/IT-5<sup>th</sup> Sem)**



**Subject Name: AP LAB-II**

**Subject Code: 22CSP-351**

**Submitted to:**

Faculty name

Anjali Thakur

**Submitted by:**

Name: Ayush Kumar Mayank

UID: 22BCS11193

Section: 22BCS-IOT-614

Group: B

# 1. Problem - House RObber:

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 n;
    int fxn(int i, vector<int>& nums, vector<int>& dp) {
        if (i >= n) return 0; // Base case: No houses left
        if (dp[i] != -1) return dp[i]; // Memoization

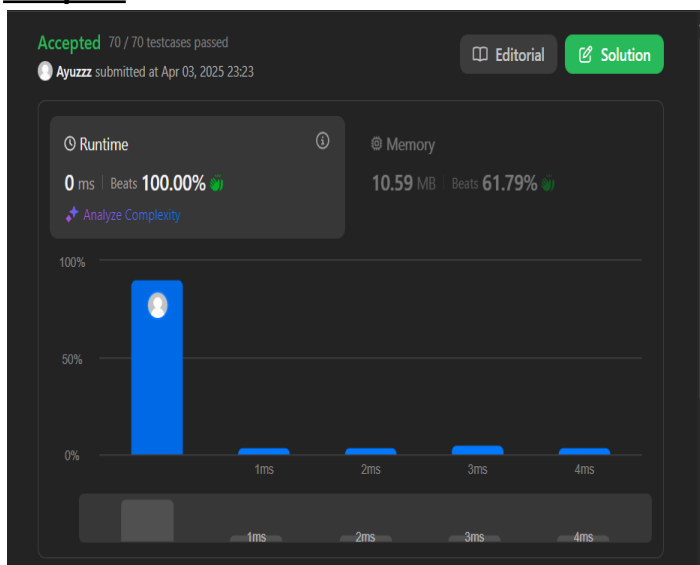
        // Option 1: Rob this house and skip the next
        int amount = fxn(i + 2, nums, dp) + nums[i];

        // Option 2: Skip this house
        int no_amount = fxn(i + 1, nums, dp);

        // Store and return the maximum amount possible
        return dp[i] = max(amount, no_amount);
    }

    int rob(vector<int>& nums) {
        n = nums.size();
        vector<int> dp(n + 1, -1);
        return fxn(0, nums, dp);
    }
};
```

## Output:



```
1 class Solution {
2 public:
3     int n;
4     int fxn(int i, vector<int>& nums, vector<int>& dp) {
5         if (i >= n) return 0; // Base case: No houses left
6         if (dp[i] != -1) return dp[i]; // Memoization
7
8         // Option 1: Rob this house and skip the next
9         int amount = fxn(i + 2, nums, dp) + nums[i];
10
11        // Option 2: Skip this house
12        int no_amount = fxn(i + 1, nums, dp);
13
14        // Store and return the maximum amount possible
15        return dp[i] = max(amount, no_amount);
16    }
17
18    int rob(vector<int>& nums) {
19        n = nums.size();
20        vector<int> dp(n + 1, -1);
21        return fxn(0, nums, dp);
22    }
23 };
```

## 2.Problem -: Jump Game

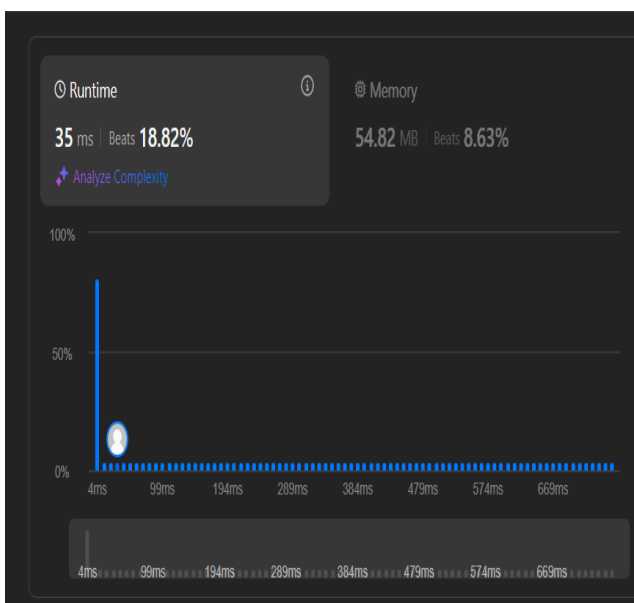
You are given an integer array `nums`. You are initially positioned at the array's first index, and each element in the array represents your maximum jump length at that position.

Return `true` if you can reach the last index, or `false` otherwise.

### Code:

```
class Solution {
    vector<int> memo;
public:
    bool canJump(vector<int>& nums) {
        int n=nums.size();
        vector<int> dp(n,0);
        dp[0]=true;

        for(int i=1;i<n;i++){
            for(int j=i-1;j>=0;j--){
                if(dp[j] && j+nums[j]>=i){
                    dp[i]=true;
                    break;
                }
            }
        }
        return dp[n-1];
    }
};
```



```
1 class Solution {
2     vector<int> memo;
3 public:
4     bool canJump(vector<int>& nums) {
5         int n=nums.size();
6         vector<int> dp(n,0);
7         dp[0]=true;
8
9         for(int i=1;i<n;i++){
10             for(int j=i-1;j>=0;j--){
11                 if(dp[j] && j+nums[j]>=i){
12                     dp[i]=true;
13                     break;
14                 }
15             }
16         }
17         return dp[n-1];
18     }
19 };
```

## 206. Problem - Maximum Product Subarray:

Given an integer array `nums`, find a subarray that has the largest product, and return the product.

### Code:

```
class Solution {
public:
    int maxProduct(vector<int>& nums) {
        int n = nums.size();
        int maxP = nums[0];
        int minP = nums[0];
        int result = nums[0];

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

            maxP = max({nums[i], maxP * nums[i], minP * nums[i]});
            minP = min({nums[i], tempMax * nums[i], minP * nums[i]});

            result = max(result, maxP);
        }
        return result;
    }
};
```

### Output:

Accepted 190 / 190 testcases passed

Ayuzz submitted at Apr 03, 2025 23:18

Editorial Solution

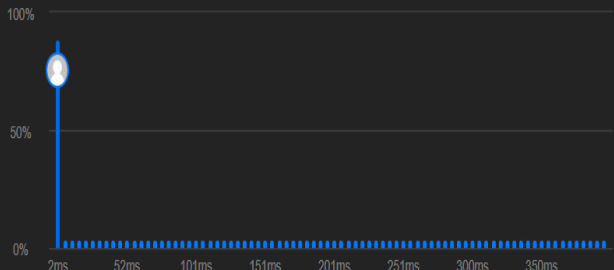
Runtime

0 ms | Beats 100.00%

Analyze Complexity

Memory

17.73 MB | Beats 57.75%



2ms 52ms 101ms 151ms 201ms 251ms 300ms 350ms

```
1 class Solution {
2 public:
3     int maxProduct(vector<int>& nums) {
4         int n = nums.size();
5         int maxP = nums[0];
6         int minP = nums[0];
7         int result = nums[0];
8
9         for(int i=1; i<n; i++) {
10             int tempMax = maxP;
11
12             maxP = max({nums[i], maxP * nums[i], minP * nums[i]});
13             minP = min({nums[i], tempMax * nums[i], minP * nums[i]});
14
15             result = max(result, maxP);
16         }
17         return result;
18     }
19 };
```

Saved

Ln 19, Col 3

## 2095.Problem - Coin Change:

### Code:

```
class Solution {
public:
    int coinChange(vector<int>& coins, int n) {
        // creating the base dp array, with first value set to 0
        int dp[++n];
        dp[0] = 0;
        // more convenient to have the coins sorted
        sort(begin(coins), end(coins));
        // populating our dp array
        for (int i = 1; i < n; i++) {
            // setting dp[0] base value to 1, 0 for all the rest
            dp[i] = INT_MAX;
            for (int c: coins) {
                if (i - c < 0) break;
                // if it was a previously not reached cell, we do not add use it
                if (dp[i - c] != INT_MAX) dp[i] = min(dp[i], 1 + dp[i - c]);
            }
        }
        return dp[--n] == INT_MAX ? -1 : dp[n];
    }
}
```

### Output:

