Description | Editorial | Solutions | Submissions

70. Climbing Stairs

Solved

Easy | Topics | Companies | Hint

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?

Example 1:

> **Input:** n = 2
> **Output:** 2
> **Explanation:** There are two ways to climb to the top.
> 1. 1 step + 1 step
> 2. 2 steps

Example 2:

> **Input:** n = 3
> **Output:** 3
> **Explanation:** There are three ways to climb to the top.
> 1. 1 step + 1 step + 1 step
> 2. 1 step + 2 steps
> 3. 2 steps + 1 step

23K | 448 | 334 Online

Code

C++ | Auto

```cpp
class Solution {
public:
  int climbStairs(int n) {
    int prev1 = 1;
    int prev2 = 1;

    for (int i = 2; i <= n; ++i) {
      const int dp = prev1 + prev2;
      prev2 = prev1;
      prev1 = dp;
    }

    return prev1;
  }
};
```

Saved | Ln 1, Col 1

Testcase | Test Result

Accepted Runtime: 0 ms

Case 1 | Case 2

Input

Description | Accepted | Editorial | Solutions | Submissions

55. Jump Game

Solved

Medium | Topics | Companies

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.*

Example 1:

> **Input:** nums = [2,3,1,1,4]
> **Output:** true
> **Explanation:** Jump 1 step from index 0 to 1, then 3 steps to the last index.

Example 2:

> **Input:** nums = [3,2,1,0,4]
> **Output:** false
> **Explanation:** You will always arrive at index 3 no matter what. Its maximum jump length is 0, which makes it impossible to reach the last index.

Constraints:

20.4K | 327 | 317 Online

Code

C++ | Auto

```cpp
class Solution {
public:
  bool canJump(vector<int>& nums) {
    int i = 0;

    for (int reach = 0; i < nums.size() && i <= reach; ++i)
      reach = max(reach, i + nums[i]);

    return i == nums.size();
  }
};
```

Saved | Ln 11, Col 3

Testcase | Test Result

Accepted Runtime: 0 ms

Case 1 | Case 2

Input

Description | Editorial | Solutions | Submissions

53. Maximum Subarray

Solved

Medium | Topics | Companies

Given an integer array `nums`, find the subarray with the largest sum, and return *its sum*.

Example 1:

```
Input: nums = [-2,1,-3,4,-1,2,1,-5,4]
Output: 6
Explanation: The subarray [4,-1,2,1] has the largest sum 6.
```

Example 2:

```
Input: nums = [1]
Output: 1
Explanation: The subarray [1] has the largest sum 1.
```

Example 3:

```
Input: nums = [5,4,-1,7,8]
Output: 23
Explanation: The subarray [5,4,-1,7,8] has the largest sum 23.
```

Constraints:

35.4K | 349 | 450 Online

Code

C++ | Auto

```cpp
class Solution {
public:
    int maxSubArray(vector<int>& nums) {
     int sum=nums[0],maxsum=nums[0];
     for(int i=1;i<nums.size();i++){
        sum=max(nums[i],sum+nums[i]);
        maxsum=max(maxsum,sum);
     }
        return maxsum;
      }

};
```

Saved | Ln 1, Col 1

Testcase | Test Result

Accepted Runtime: 0 ms

Case 1 | Case 2 | Case 3

Input

Description | Accepted | Editorial | Solutions | Submissions

322. Coin Change

Solved

Medium | Topics | Companies

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.

Example 1:

```
Input: coins = [1,2,5], amount = 11
Output: 3
Explanation: 11 = 5 + 5 + 1
```

Example 2:

```
Input: coins = [2], amount = 3
Output: -1
```

Example 3:

```
Input: coins = [1], amount = 0
Output: 0
```

19.7K | 164 | 315 Online

Code

C++ | Auto

```cpp
class Solution {
public:
  int coinChange(vector<int>& coins, int amount) {
    // dp[i] := the minimum number of coins to make up i
    vector<int> dp(amount + 1, amount + 1);
    dp[0] = 0;

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

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

Saved | Ln 14, Col 3

Testcase | Test Result

Accepted Runtime: 0 ms

• Case 1 • Case 2 • Case 3

Input