Description | Editorial | Solutions | Submissions

70. Climbing Stairs

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
```

Constraints:

- `1 <= n <= 45`

Code

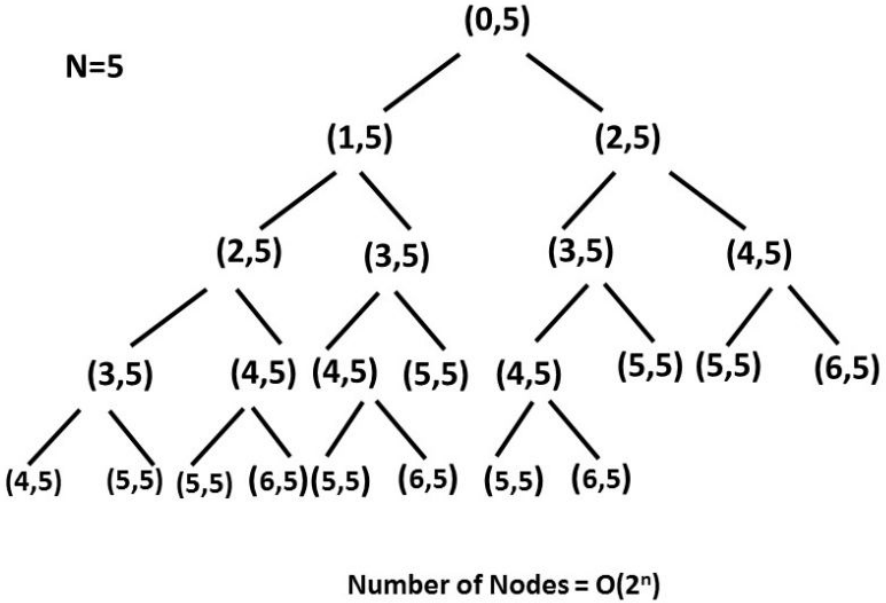
C++ • Auto

```cpp
class Solution {
public:
    int climbStairs(int n) {

    }
};
```

Approach 1: Brute Force

```python
# Python3
class Solution:
    def climbStairs(self, n: int) -> int:
        return self.climb_Stairs(0, n)

    def climb_Stairs(self, i: int, n: int) -> int:
        if i > n:
            return 0
        if i == n:
            return 1
        return self.climb_Stairs(i + 1, n) + self.climb_Stairs(i + 2, n)
```

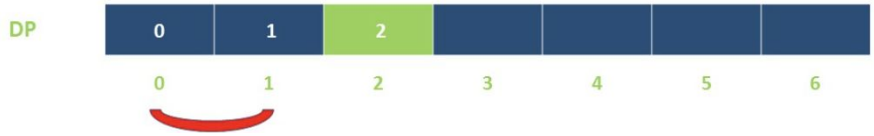
- Time complexity : $O(2^n)$.



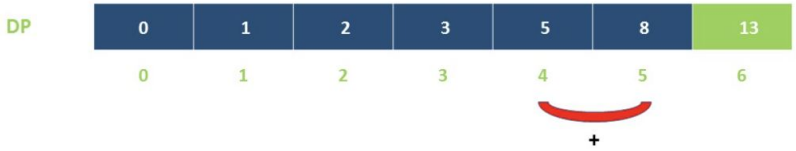
N=6

N=6

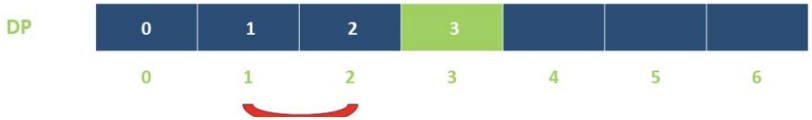
N=6



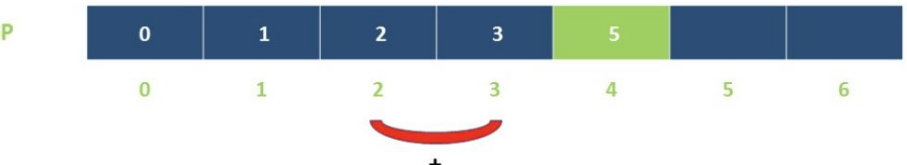
N=6



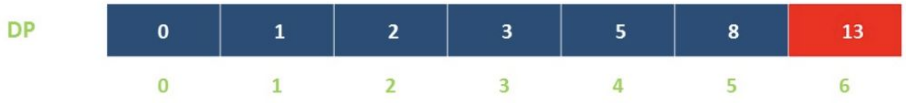
N=6



N=6



N=6



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
```

Code

Python3 | Auto

```python
# Python3
class Solution:
    def climbStairs(self, n: int) -> int:
        if n == 1:
            return 1
        dp = [0 for _ in range(n + 1)]
        dp[1] = 1
        dp[2] = 2
        for i in range(3, n + 1):
            dp[i] = dp[i - 1] + dp[i - 2]
        return dp[n]
```

Description | Editorial | Solutions | Submissions

300. Longest Increasing Subsequence

Medium | Topics | Companies

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

Example 1:

```
Input: nums = [10,9,2,5,3,7,101,18]
Output: 4
Explanation: The longest increasing subsequence is [2,3,7,101], therefore the length is 4.
```

Example 2:

```
Input: nums = [0,1,0,3,2,3]
Output: 4
```

Example 3:

```
Input: nums = [7,7,7,7,7,7,7]
Output: 1
```

Constraints:

- `1 <= nums.length <= 2500`
- `-10^4 <= nums[i] <= 10^4`

Follow up: Can you come up with an algorithm that runs in `O(n log(n))` time complexity?

Code

C++ • Auto

```cpp
class Solution {
public:
    int lengthOfLIS(vector<int>& nums) {

    }
};
```

Saved

Ln 1, Col 1

A Framework to Solve Dynamic Programming Problems

1. First, we need some function or array that represents the answer to the problem from a given state
2. Second, we need a way to transition between states, such as dp[5] and dp[7]. This is called a recurrence relation
3. The third component is: we need a base case. For this problem, we can initialize every element of dp to 1, since every element on its own is technically an increasing subsequence.

nums

10	9	2	5	3	7

dp

1	1	1	1	1	1

dp[i] represents the length of the longest increasing subsequence that ends at index i

Description | Editorial | Solutions | Submissions

300. Longest Increasing Subsequence

Solved

Medium | Topics | Companies

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

Example 1:

```
Input: nums = [10,9,2,5,3,7,101,18]
Output: 4
Explanation: The longest increasing subsequence is [2,3,7,101], therefore the length is 4.
```

Example 2:

```
Input: nums = [0,1,0,3,2,3]
Output: 4
```

Example 3:

```
Input: nums = [7,7,7,7,7,7,7]
Output: 1
```

Constraints:

- `1 <= nums.length <= 2500`
- `-10^4 <= nums[i] <= 10^4`

Follow up: Can you come up with an algorithm that runs in `O(n log(n))` time complexity?

Code

Python3 | Auto

```python
class Solution:
    def lengthOfLIS(self, nums: List[int]) -> int:
        dp = [1] * len(nums)
        for i in range(1, len(nums)):
            for j in range(i):
                if nums[i] > nums[j]:
                    dp[i] = max(dp[i], dp[j] + 1)

        return max(dp)
```

Description | Editorial | Solutions | Submissions

62. Unique Paths

Solved

Medium | Topics | Companies

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`.

Example 1:


```
Input: m = 3, n = 7
Output: 28
```

Example 2:

```
Input: m = 3, n = 2
Output: 3
Explanation: From the top-left corner, there are a total of 3 ways to reach the bottom-right corner:
1. Right -> Down -> Down
2. Down -> Down -> Right
3. Down -> Right -> Down
```

Code

C++ • Auto

```cpp
class Solution {
public:
    int uniquePaths(int m, int n) {

    }
};
```

Saved

Ln 1, Col 1