



## Experiment 7

**Student Name: Harsh**

**Branch: CSE**

**Semester: 6**

**Subject Name: AP Lab 2**

**UID: 22BCS12488**

**Section/Group: NTPP 603B**

**Date of Performance: 21 /03/25**

**Subject Code: 22CSP-351**

### 1. Aim:

- a. Maximum Subarray
- b. Unique Path
- c. Longest Increasing subsequence
- d. Climbing Stairs

### 2. Code:

```
a. class Solution {  
    public int maxSubArray(int[] nums) {  
        int maxSum = nums[0];  
        int currentSum = nums[0];  
  
        for (int i = 1; i < nums.length; i++) {  
            currentSum = Math.max(nums[i], currentSum + nums[i]);  
            maxSum = Math.max(maxSum, currentSum);  
        }  
  
        return maxSum;  
    }  
}
```

```
b. class Solution {  
    public int uniquePaths(int m, int n) {  
        int[][] dp = new int[m][n];  
  
        for (int i = 0; i < m; i++) {  
            dp[i][0] = 1;  
        }  
    }  
}
```

```
        for (int j = 0; j < n; j++) {
            dp[0][j] = 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];
    }
}
```

c.

```
class Solution {
    public int lengthOfLIS(int[] nums) {
        if (nums.length == 0) return 0;

        int[] dp = new int[nums.length];
        int maxLength = 1;

        for (int i = 0; i < nums.length; i++) {
            dp[i] = 1;
            for (int j = 0; j < i; j++) {
                if (nums[i] > nums[j]) {
                    dp[i] = Math.max(dp[i], dp[j] + 1);
                }
            }
            maxLength = Math.max(maxLength, dp[i]);
        }
        return maxLength;
    }
}
```

d. 

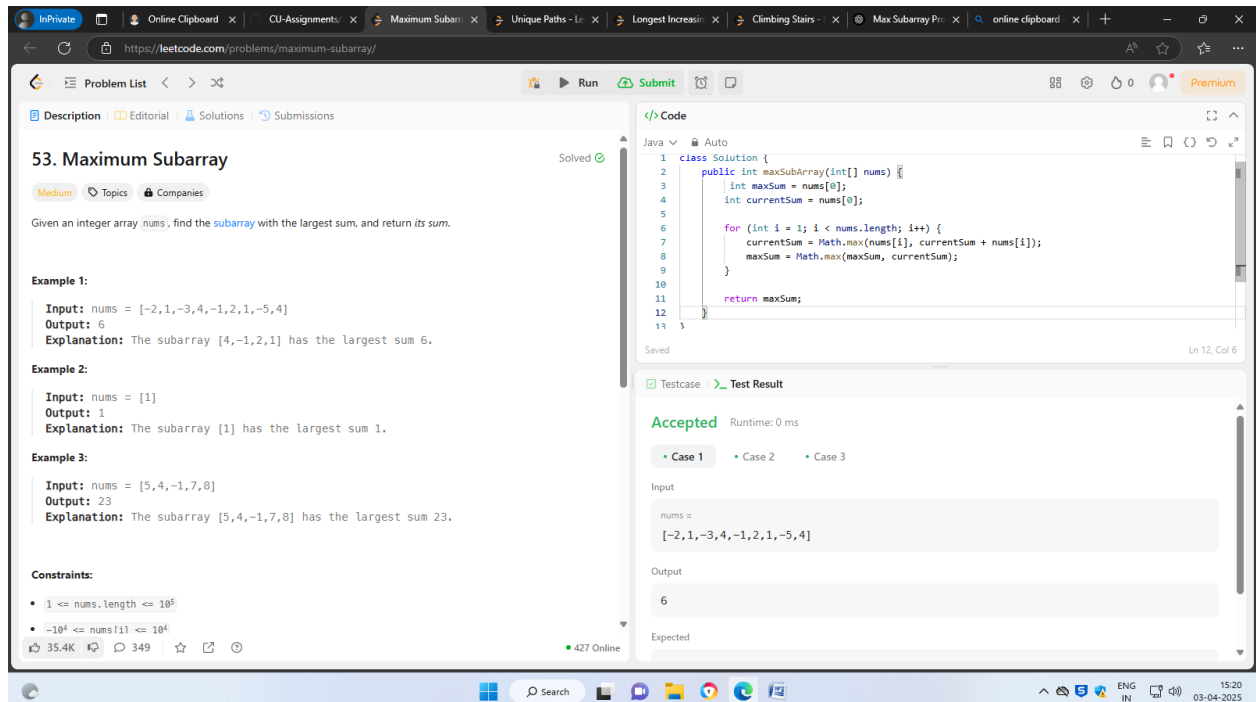
```
class Solution {
    public int climbStairs(int n) {
        if (n <= 2) return n;

        int first = 1, second = 2;
```

```
        for (int i = 3; i <= n; i++) {  
            int temp = first + second;  
            first = second;  
            second = temp;  
        }  
  
        return second;  
    }  
}
```

### 3. Output:

a.



The screenshot displays the LeetCode interface for the problem "53. Maximum Subarray". The problem description states: "Given an integer array `nums`, find the **subarray** with the largest sum, and return its sum." Examples provided include: Example 1 (Input: [-2,1,-3,4,-1,2,1,-5,4], Output: 6), Example 2 (Input: [1], Output: 1), and Example 3 (Input: [5,4,-1,7,8], Output: 23). Constraints specify  $1 \leq \text{nums.length} \leq 10^5$  and  $-10^4 \leq \text{nums}[i] \leq 10^4$ . The Java solution code is shown in the editor, implementing a Kadane's algorithm. The test results section shows "Accepted" with a runtime of 0 ms for Case 1, where the input is [-2,1,-3,4,-1,2,1,-5,4] and the output is 6.



# DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

b.

**62. Unique Paths**

Medium Topics Companies

There is a robot on an  $m \times n$  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:  $m = 3, n = 7$   
Output: 28

17.3K 182 194 Online

```
class Solution {
    public int uniquePaths(int m, int n) {
        int[][] dp = new int[m][n];
        for (int i = 0; i < m; i++) {
            dp[i][0] = 1;
        }
        for (int j = 0; j < n; j++) {
            dp[0][j] = 1;
        }
    }
}
```

Accepted Runtime: 0 ms

Case 1 Case 2

Input

m = 3  
n = 7

Output

28

c.

**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]`  
Output: 1

**Constraints:**

- $1 \leq \text{nums.length} \leq 2500$
- $-10^4 \leq \text{nums}[i] \leq 10^4$

21.6K 220 289 Online

```
class Solution {
    public int lengthOfLIS(int[] nums) {
        if (nums.length == 0) return 0;
        int[] dp = new int[nums.length];
        int maxLength = 1;
        for (int i = 0; i < nums.length; i++) {
            dp[i] = 1;
            for (int j = 0; j < i; j++) {
                if (nums[i] > nums[j]) {
                    dp[i] = Math.max(dp[i], dp[j] + 1);
                }
            }
        }
    }
}
```

Accepted Runtime: 0 ms

Case 1 Case 2 Case 3

Input

nums = [10, 9, 2, 5, 3, 7, 101, 18]

Output

4

Expected



# DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

d.

The screenshot displays the LeetCode interface for the '70. Climbing Stairs' problem. The problem description states: '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?'. Two examples are provided: Example 1 with  $n=2$  and output 2, and Example 2 with  $n=3$  and output 3. The constraints specify  $1 \leq n \leq 45$ . The code editor shows a Java solution using a loop to calculate the number of ways. The test result is 'Accepted' with a runtime of 0 ms.

**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:**

- $1 \leq n \leq 45$

23K 448 299 Online

```
1 class Solution {
2     public int climbStairs(int n) {
3         if (n <= 2) return n;
4
5         int first = 1, second = 2;
6
7         for (int i = 3; i <= n; i++) {
8             int temp = first + second;
9             first = second;
10            second = temp;
11        }
12    }
```

Accepted Runtime: 0 ms

Case 1 Case 2

Input

$n =$

2

Output

2

Expected