



# DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

## Experiment: -7

**Student Name** Shalini Kumari

**UID** 22BET10202

**Branch:** CSE

**Section/Group** 22BET\_IOT-703/A

**Semester:** 6<sup>th</sup>

**Date of Performance:** 10/03/2025

**Subject Name:** Advanced Programming Lab-2 **Subject Code:** 22CSP-351

### Problem -1

**1. Aim:** Climbing Stairs

**2. Objective:**

- **Understanding the Problem:** To understand how to solve the staircase problem using a simple mathematical pattern based on previous steps.
- **Using Fibonacci Sequence:** To learn how the Fibonacci sequence helps calculate the number of ways to climb stairs.
- **Practicing Code Efficiency:** To practice using loops and variables to write efficient and clean code by updating values and avoiding repetitive calculations, which improves performance.
- **Handling Edge Cases:** To understand how to handle edge cases like small values and ensure correct output.
- **Improving Problem-Solving:** To improve problem-solving skills by applying dynamic programming concepts

**3. Implementation/Code:**

```
class Solution {
public:
    int climbStairs(int n) {
        if (n==1) return 1;
        int a=1,b=2;
        for (int i=3;i<=n;i++) {
            int temp =a+b;
            a=b;
            b=temp;
        }
        return b;
    }
};
```

## 4. Output:

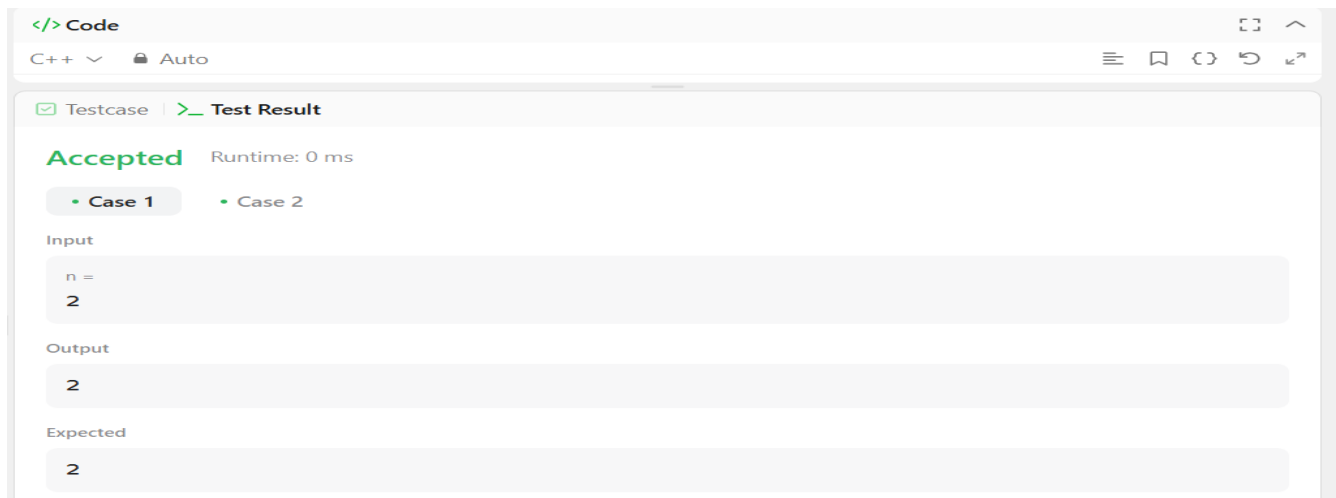


Figure 1

## 5. Learning Outcome:

- **Pattern Recognition:** You will understand how to find patterns and use them to solve coding problems.
- **Applying Fibonacci:** You will learn how to apply the Fibonacci sequence in real-life scenarios.
- **Enhancing Coding Skills:** You will improve your coding skills by practicing loops and updating variables.
- **Simplifying Problems:** You will understand how to simplify problems by breaking them into smaller steps.
- **Building Confidence:** You will become more confident in solving mathematical problems using code.

### Problem-2

#### 1. Aim: Maximum Subarray

#### 2. Objectives:

- **Understanding the Problem:** To understand how to find the subarray with the largest sum from a given integer array using a logical approach.
- **Using Kadane's Algorithm:** To learn how Kadane's algorithm helps find the maximum subarray sum by efficiently updating current and maximum sums.
- **Practicing Efficient Coding:** To practice using loops and conditions to update the sum quickly and avoid unnecessary calculations.
- **Handling Negative Numbers:** To understand how to handle both positive and negative values while calculating the maximum sum.
- **Exploring Advanced Approaches:** To explore the divide and conquer method for solving the problem more efficiently with deeper understanding.

### 3. Implementation/Code:

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

### 4. Output:



Figure 2

### 5. Learning Outcomes

- **Better Problem-Solving Skills:** You will learn how to analyze array problems and develop a logical approach to find the largest sum.
- **Understanding Kadane's Algorithm:** You will understand how Kadane's algorithm works and why it is effective for finding maximum subarray sums.
- **Writing Clean and Fast Code:** You will improve your ability to write efficient code by properly using loops and conditions.
- **Handling Edge Cases:** You will be able to handle cases with mixed positive and negative numbers confidently.
- **Applying Advanced Methods:** You will gain experience in using the divide and conquer approach to solve complex array problems.

## Problem: -3

### 1. Aim: Jump Game

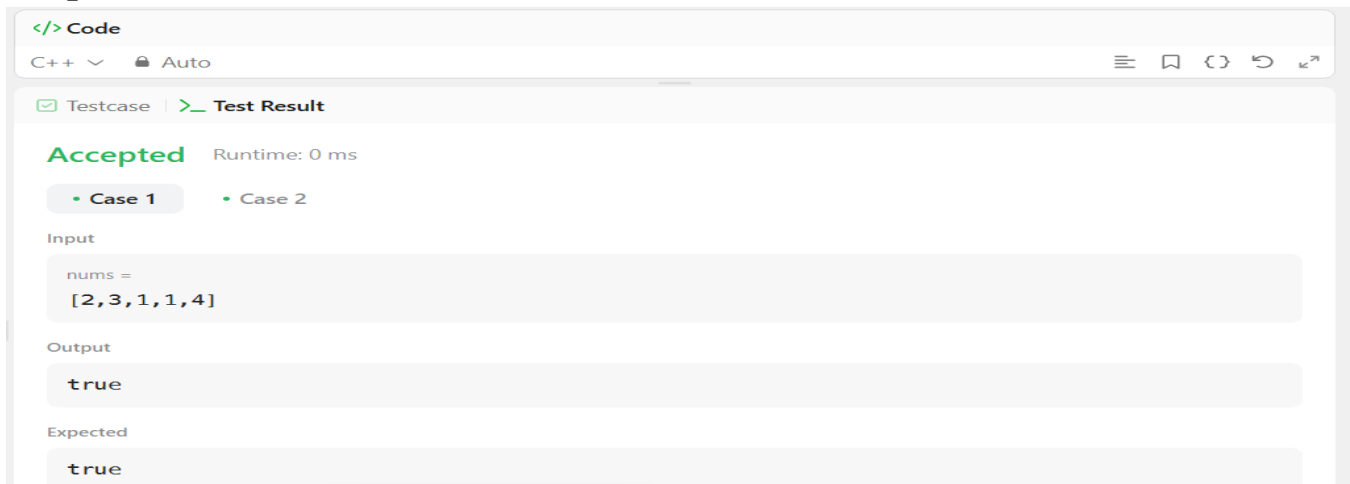
### 2. Objectives:

- **Understanding the Problem:** To understand how to check if you can reach the last index using jump values in the array.
- **Using Greedy Approach:** To learn how the greedy approach helps in finding the maximum reachable index at each step.
- **Practicing Efficient Code:** To practice writing efficient code using loops and conditions to reduce calculations.
- **Handling Stuck Positions:** To understand how to handle cases where progress is blocked due to zero jump value.
- **Optimizing Performance:** To improve performance by stopping early once the last index is reachable.

### 3. Implementation/Code:

```
class Solution {  
public:  
    bool canJump(vector<int>& nums) {  
        int maxReach=0;  
        for (int i=0; i<nums.size();i++) {  
            if (i > maxReach) return false;  
            maxReach=max(maxReach,i+nums[i]);  
            if (maxReach>=nums.size()-1) return true;  
        }  
        return false;  
    }  
};
```

### 4. Output:



The screenshot shows a C++ IDE with the following content:

```
</> Code  
C++ v Auto  
Testcase | Test Result  
Accepted Runtime: 0 ms  
• Case 1 • Case 2  
Input  
nums =  
[2,3,1,1,4]  
Output  
true  
Expected  
true
```

Figure 3



# DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

## 5. Learning Outcomes:

- **Better Problem Solving:** You will learn how to solve array-based movement problems step-by-step.
- **Understanding Greedy Method:** You will understand how the greedy approach helps in making the best jump decision.
- **Writing Clean Code:** You will improve your coding skills by writing simple and optimized code.
- **Handling Edge Cases:** You will know how to handle cases where movement is blocked by zero jump value.
- **Improving Efficiency:** You will learn to write faster solutions by reducing unnecessary calculations.