# Experiment: -7

**Student Name: Harsh Kumar**
**Branch: B.E-CSE**
**Semester: 6**
**Subject Name: AP-II**

**UID: 22BCS15183**
**Section/Group: 640`B**
**Date of Performance:10/03/2025**
**Subject Code:22CSP-351**

## Problem -1

1. **Aim:** Climbing Stairs

2. **Objective:**

The staircase problem uses a pattern like the Fibonacci sequence to find ways to climb stairs efficiently. It teaches how to write clean code with loops, handle edge cases, and apply dynamic programming to improve problem-solving skills.

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:

The maximum subarray problem focuses on finding the subarray with the largest sum using a logical approach. Kadane's Algorithm helps efficiently update current and maximum sums using simple loops and conditions, making the code clean and fast. It also teaches how to handle negative numbers effectively. Additionally, exploring divide and conquer methods deepens understanding and introduces more advanced problem-solving techniques.

## 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:

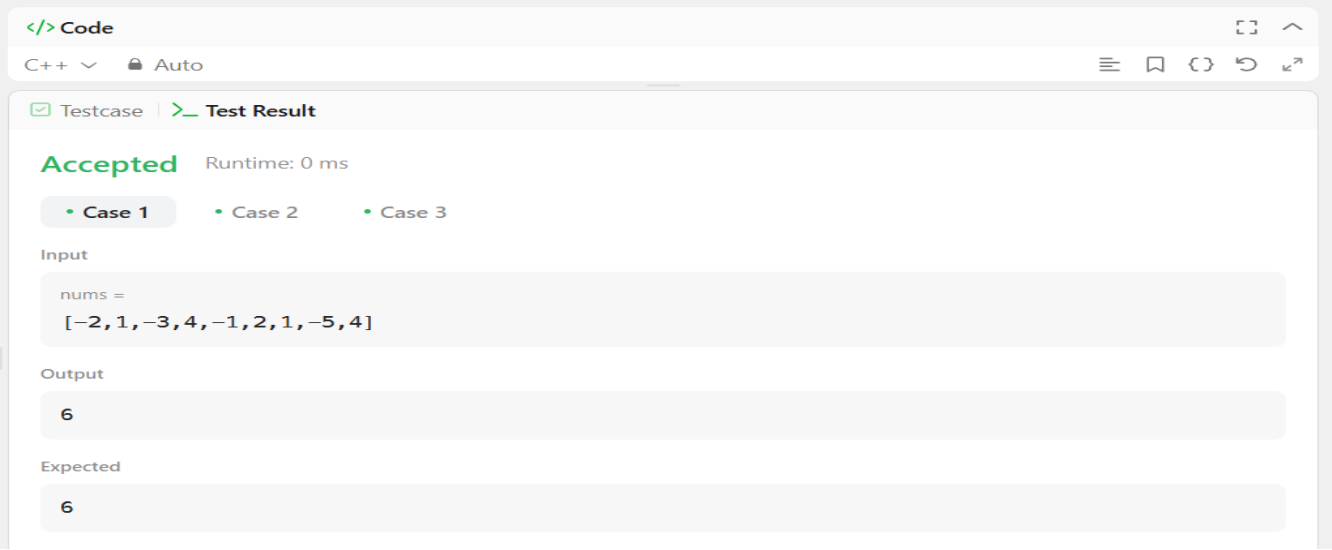


**Figure 2**

## 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:**

The jump game problem teaches how to check if the last index is reachable using jump values in an array. The greedy approach helps track the farthest reachable index efficiently. It involves writing optimized code with loops, handling stuck positions like zero jumps, and improving performance by exiting early when the goal is reached.

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:**

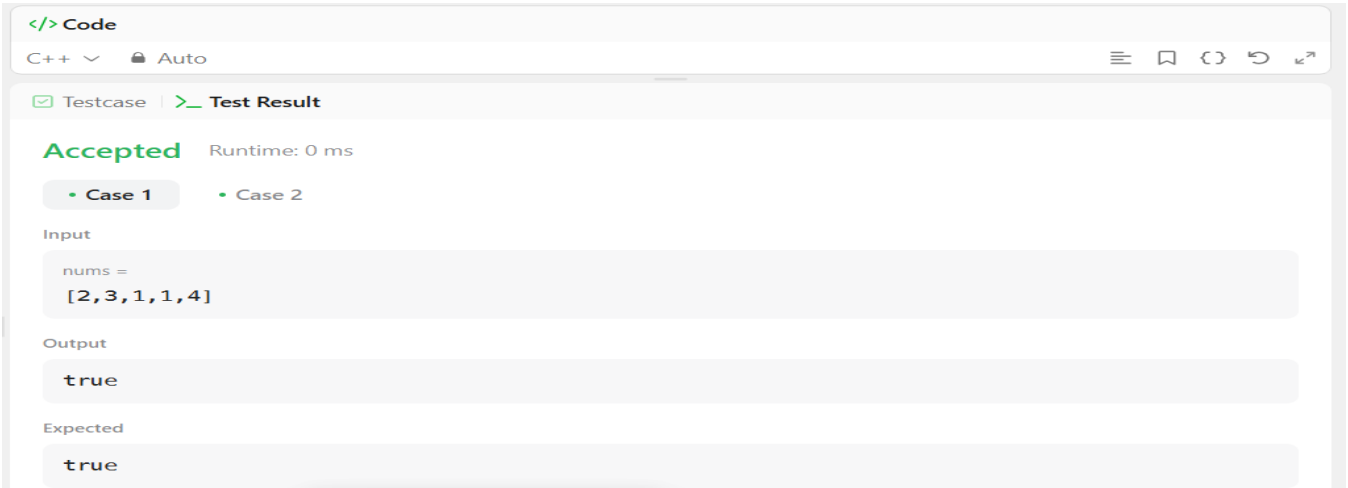


**Figure 3**

## 5. LearningOutcomes:

- **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.