EXPERIMENT7

Student Name: Omdev Kumar **UID:**22BET10206

Branch: BE -IT **Section/Group:**22BET_IOT-703(B)

Semester: 6th **Subject Code:** 22ITP-351

PROBLEM-1

AIM:-

Climbing Stairs

CODE:-

```
class Solution {
    public int climbStairs(int n) {
        if (n == 0 || n == 1) {
            return 1;
        }

        int[] dp = new int[n+1];
        dp[0] = dp[1] = 1;

        for (int i = 2; i <= n; i++) {
            dp[i] = dp[i-1] + dp[i-2];
        }
        return dp[n];
    }
}
```

OUTPUT:-

PROBLEM-2

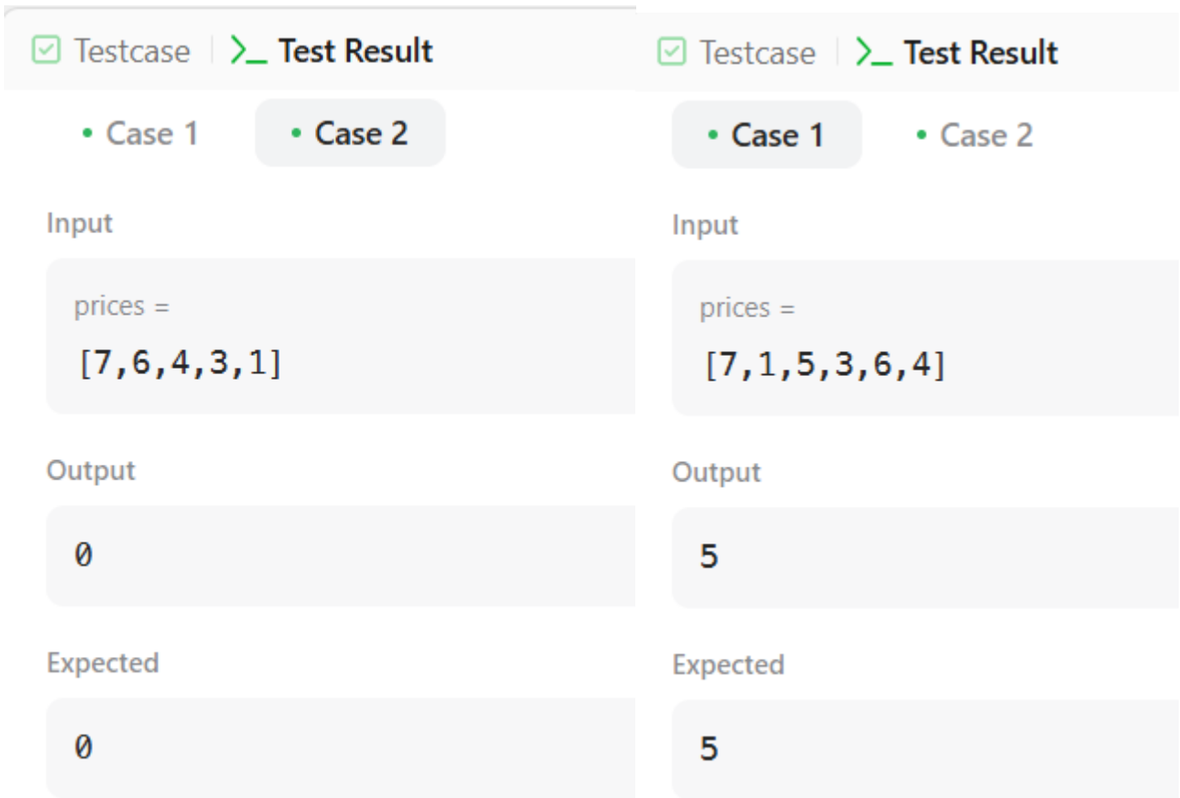
AIM:-

Best Time to Buy and Sell a Stock

CODE:-

```
class Solution {
    public int maxProfit(int[] prices) {
        int buy = prices[0];
        int profit = 0;
        for (int i = 1; i < prices.length; i++) {
            if (prices[i] < buy) {
                buy = prices[i];
            } else if (prices[i] - buy > profit) {
                profit = prices[i] - buy;
            }
        }
        return profit;
    }
}
```

}**OUTPUT:-**



PROBLEM-3

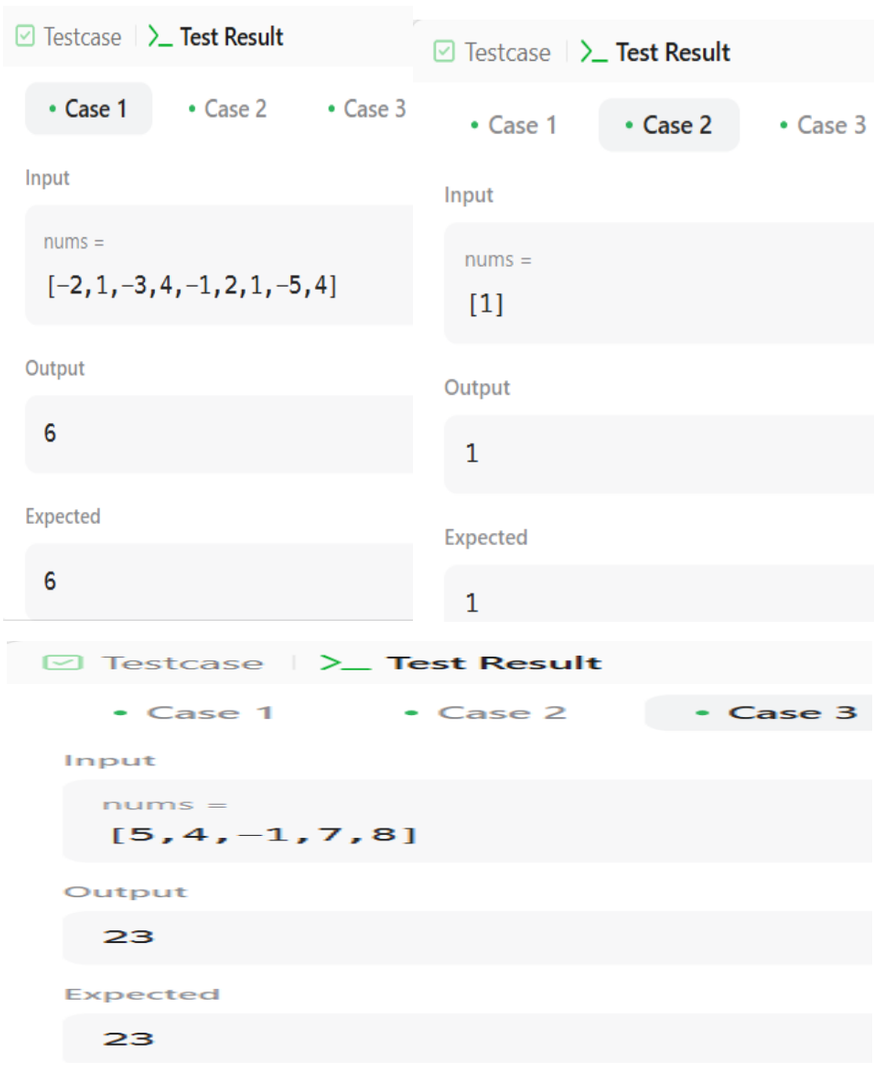
AIM:-

Maximum Subarray

CODE:-

```
class Solution {
    public int maxSubArray(int[] nums) {
        int res = nums[0];
        int total = 0;

        for (int n : nums) {
            if (total < 0) {
                total = 0;
            }

            total += n;
            res = Math.max(res, total);
        }

        return res;
    }
}
}
```

OUTPUT:



PROBLEM-4

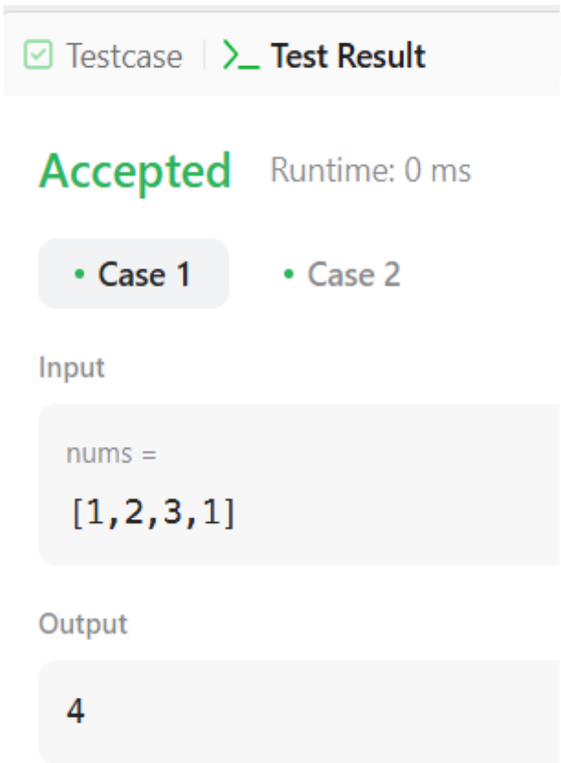
AIM:-

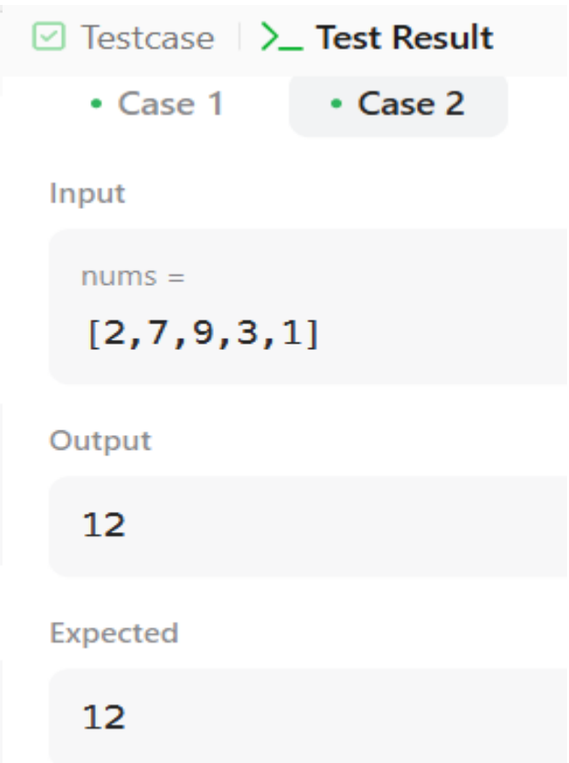
House Robber

CODE:-

```
class Solution {
 public int rob(int[] nums) {
     int n = nums.length;

     if (n == 1) {
         return nums[0];
     }

     int[] dp = new int[n];

     dp[0] = nums[0];
     dp[1] = Math.max(nums[0], nums[1]);

     for (int i = 2; i < n; i++) {
         dp[i] = Math.max(dp[i - 1], nums[i] + dp[i - 2]);
     }

     return dp[n - 1];
 }
}
```

OUTPUT:-





PROBLEM-5

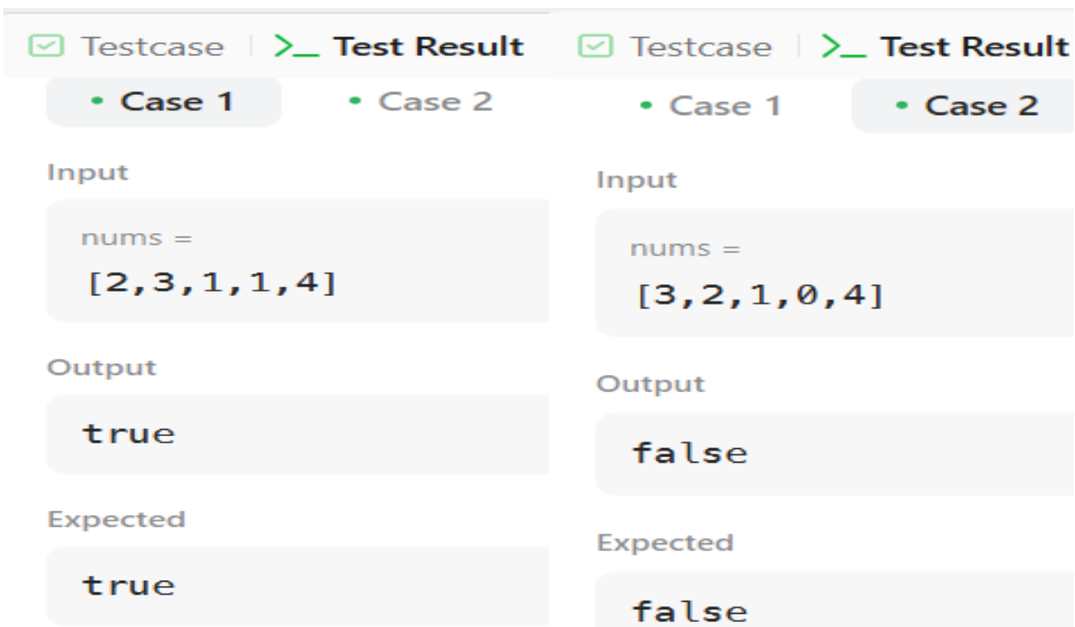
AIM:-

Jump Game

CODE:-

```
class Solution {
    public boolean canJump(int[] nums) {
        int goal = nums.length - 1;

        for (int i = nums.length - 2; i >= 0; i--) {
            if (i + nums[i] >= goal) {
                goal = i;
            }
        }

        return goal == 0;
    }
}
```

OUTPUT:-



PROBLEM-6

AIM:-

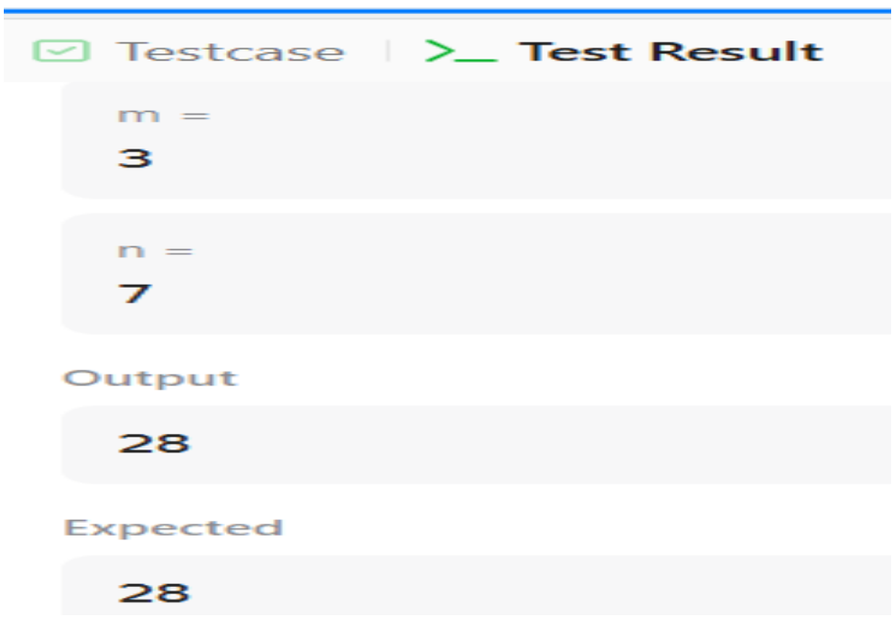
Unique Paths

CODE:-

```
class Solution {
    public int uniquePaths(int m, int n) {
        int[] aboveRow = new int[n];
        Arrays.fill(aboveRow, 1);

        for (int row = 1; row < m; row++) {
```

```
            int[] currentRow = new int[n];
            Arrays.fill(currentRow, 1);
            for (int col = 1; col < n; col++) {
                currentRow[col] = currentRow[col - 1] + aboveRow[col];
            }
            aboveRow = currentRow;
        }

        return aboveRow[n - 1];
    }
}
```

OUTPUT:-



PROBLEM-7

AIM:-

Coin Change

CODE:-

```
import java.util.*;

public class Solution {
    public static int coinChange(int[] coins, int target) {
        if (target == 0)
            return 0;

        int n = coins.length;
        if (n == 1)
            return target % coins[0] == 0 ? target / coins[0] : -1;

        Arrays.sort(coins);

        int minCoin = coins[0];
        if (target == minCoin)
            return 1;
        int idx = 1, gcdVal = minCoin;
        while (idx < n && target >= coins[idx]) {
            if (target == coins[idx])
                return 1;
            gcdVal = gcd(coins[idx], gcdVal);
            coins[idx] -= minCoin;
            idx++;
        }
        if (target % gcdVal != 0)
            return -1;

        int minVal = (target - 1) / (coins[idx - 1] + minCoin) + 1;
        int maxVal = target / minCoin;
        for (int i = minVal; i <= maxVal; i++) {
            if (findCombination(coins, 1, idx - 1, target - i * minCoin, i))
                return i;
        }
        return -1;
    }
```

```
    private static boolean findCombination(int[] coins, int left, int right, int target, int maxCoins) {
        if (target == 0)
            return true;
        if (target < coins[left] || target / coins[right] > maxCoins)
            return false;
        if (target % coins[right] == 0)
            return true;
        if (left == right)
            return false;
        for (int k = target / coins[right] + 1; k-- > 0; ) {
            if (findCombination(coins, left, right - 1, target - k * coins[right], maxCoins - k))
                return true;
        }
        return false;
    }

    private static int gcd(int a, int b) {
        while (b != 0) {
            int temp = b;
            b = a % b;
            a = temp;
        }
        return a;
    }
}
```