# Experiment 4

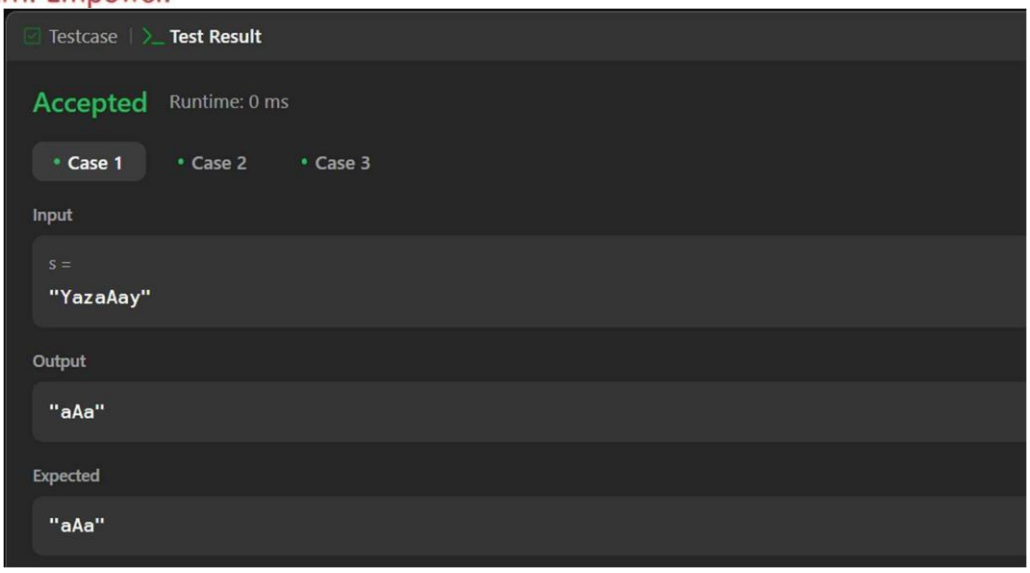
**Student Name: Aman Chaudhary** **UID: 22BET10222**
**Branch: BE-IT** **Section/Group: 22BET_IOT_701/A**
**Semester: 6th** **Date of Performance:14-2-25**
**Subject Name: Advanced Programming Lab-2** **Subject Code: 22ITP-351**

**Problem 1**. Given a string s, return the longest substring of s that is nice. If there are multiple, return the substring of the earliest occurrence. If there are none, return an empty string.

**Code:**

```
#include <unordered_set>
#include <string>
using namespace std;

class Solution {
public:
   string longestNiceSubstring(string s) {
      if (s.length() < 2) return "";

      unordered_set<char> charSet(s.begin(), s.end());

      for (int i = 0; i < s.length(); i++) {
         if (charSet.count(tolower(s[i])) && charSet.count(toupper(s[i]))) {
            continue;
         }

         string left = longestNiceSubstring(s.substr(0, i));
         string right = longestNiceSubstring(s.substr(i + 1));

         return (left.length() >= right.length()) ? left : right;
      }

      return s;
   }
};
```

**Output:**

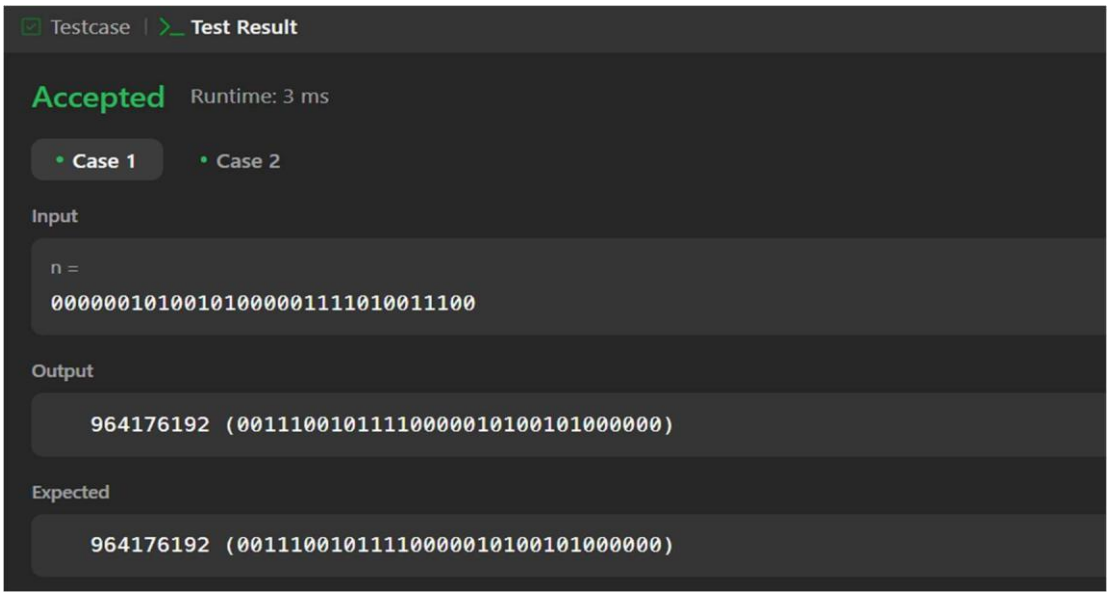



**Problem 2**. Reverse bits of a given 32 bits unsigned integer.

**Code:**

```
class Solution {
public:
  uint32_t
    reverseBits(uint32_t n)
    { uint32_t result = 0;
    for (int i = 0; i < 32;
      i++) {  result = (result
      << 1) | (n & 1); n >>=
      1;
    }
    return result;
```

**Output:**

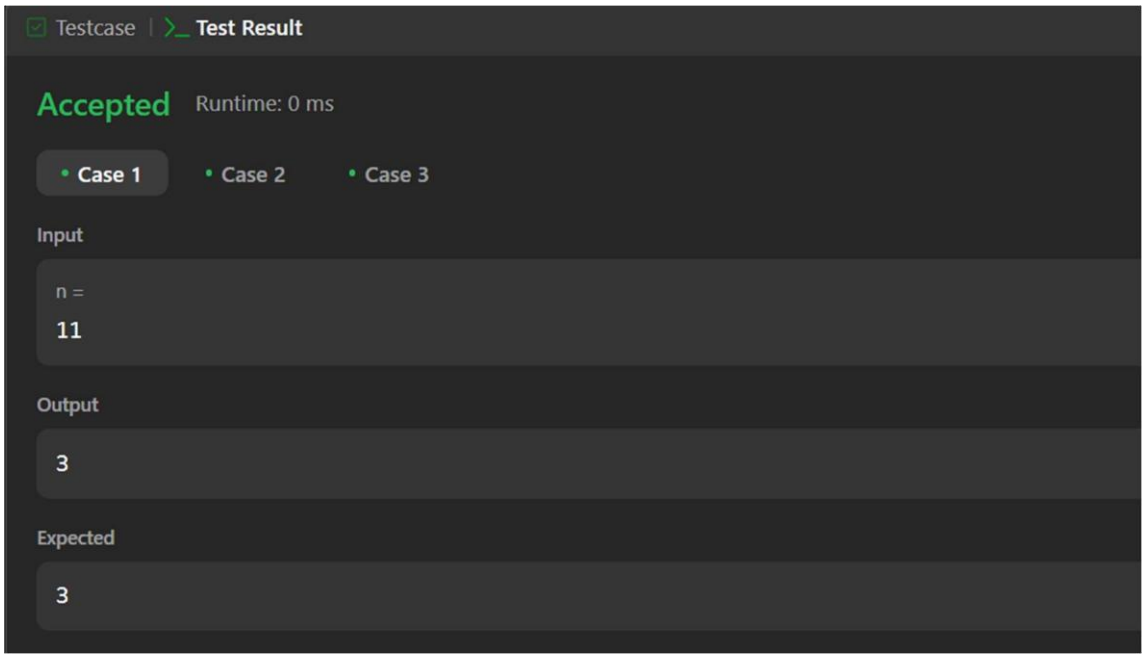



**Problem 3**. Given a positive integer n, write a function that returns the number of set bits in its binary representation (also known as the Hamming weight).

**Code:**

```
#include <cstdint>

class Solution {
public:
   int hammingWeight(int n) {
      int count = 0;
      while (n != 0) {
         count += (n & 1);
         n >>= 1;
      }
      return count;
   }
};
```

**Output:**



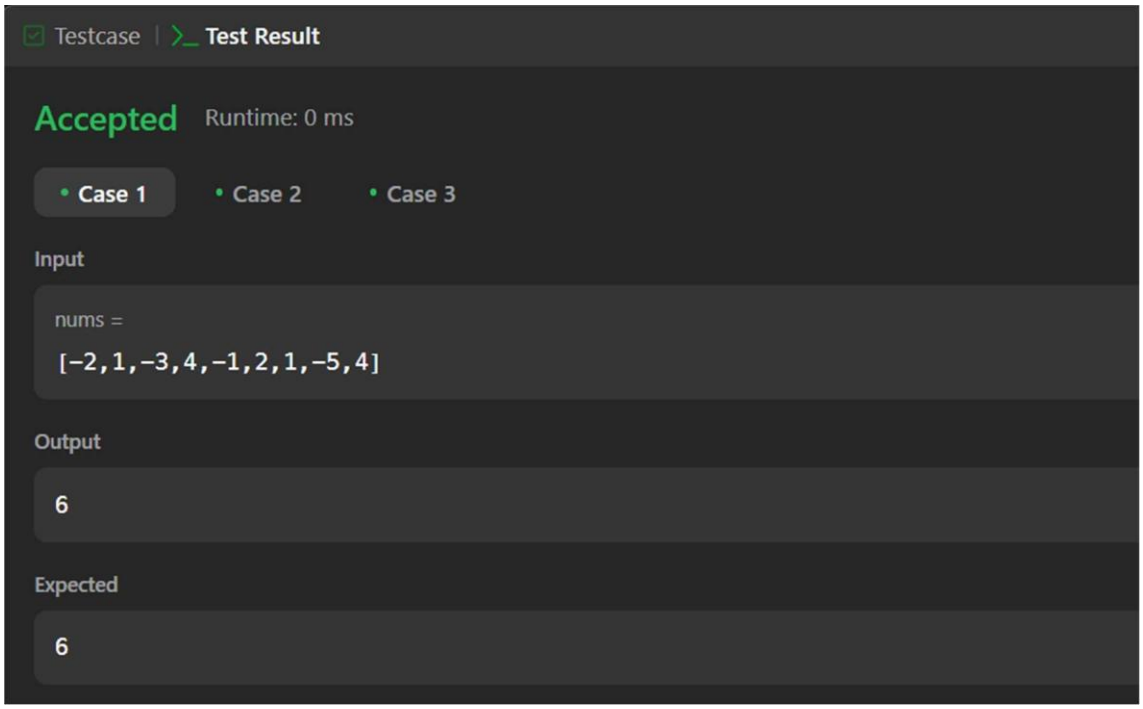

**Problem 4**. Given an integer array nums, find the subarray with the largest sum, and return *its sum*.

**Code:**

```
#include <vector>
#include <algorithm>

using namespace std;

class Solution {
public:
    int maxSubArray(vector<int>& nums) {
        int maxsum = nums[0];
        int currsum = 0;

        for (int num : nums) {
            if (currsum < 0) {
                currsum = 0;
            }
            currsum += num;
            maxsum = max(maxsum, currsum);
        }

        return maxsum;
```

```
        }
    };
```

**Output:**




**Problem 5**. Write an efficient algorithm that searches for a value target in an m x n integer matrix. This matrix has the following properties:

- Integers in each row are sorted in ascending from left to right.
- Integers in each column are sorted in ascending from top to bottom.

**Code:**

```
class Solution {
public:
    bool searchMatrix(vector<vector<int>>& matrix, int target) {
      if (matrix.empty() || matrix[0].empty()) return false;

        int rows = matrix.size();
        int cols = matrix[0].size();
        int left = 0, right = rows * cols - 1;

        while (left <= right) {
            int mid = left + (right - left) / 2;
            int row = mid / cols;
            int col = mid % cols;

            if (matrix[row][col] == target)
                return true;
            else if (matrix[row][col] < target)
                left = mid + 1;
```

```
            else
                right = mid - 1;
        }
        return false;
    }
```

**Output:**

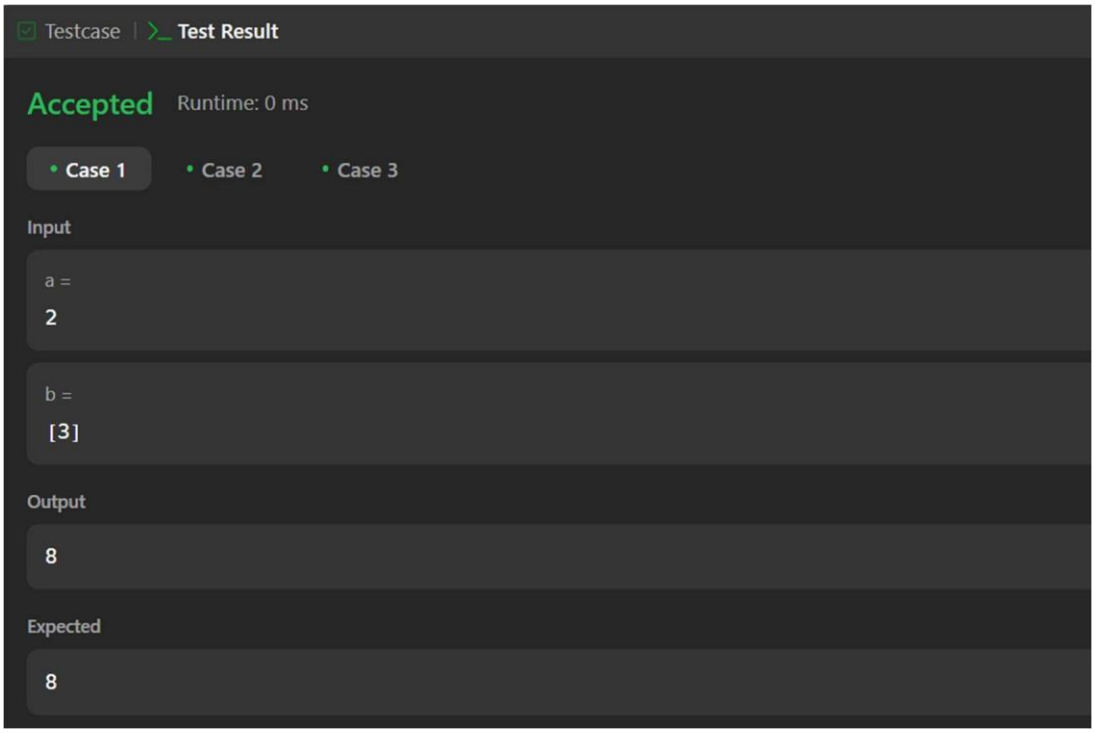
**Problem 6**. Write an efficient algorithm that searches for a value target in an m x n integer matrix matrix. This matrix has the following properties:

- Integers in each row are sorted in ascending from left to right.
- Integers in each column are sorted in ascending from top to bottom.

## Code:

```
#include <vector>

using namespace std;

class Solution {
public:
   const int MOD = 1337;

   int modPow(int a, int exp, int mod) {
      int result = 1;
      a %= mod;
      while (exp > 0) {
         if (exp % 2 == 1)
            result = (result * a) % mod;
         a = (a * a) % mod;
         exp /= 2;
      }
      return result;
   }

   int superPow(int a, vector<int>& b) {
      if (b.empty()) return 1;

      int lastDigit = b.back();
      b.pop_back();

      int part1 = modPow(a, lastDigit, MOD);
      int part2 = modPow(superPow(a, b), 10, MOD);

      return (part1 * part2) % MOD;
   }
}
```

**Output:**



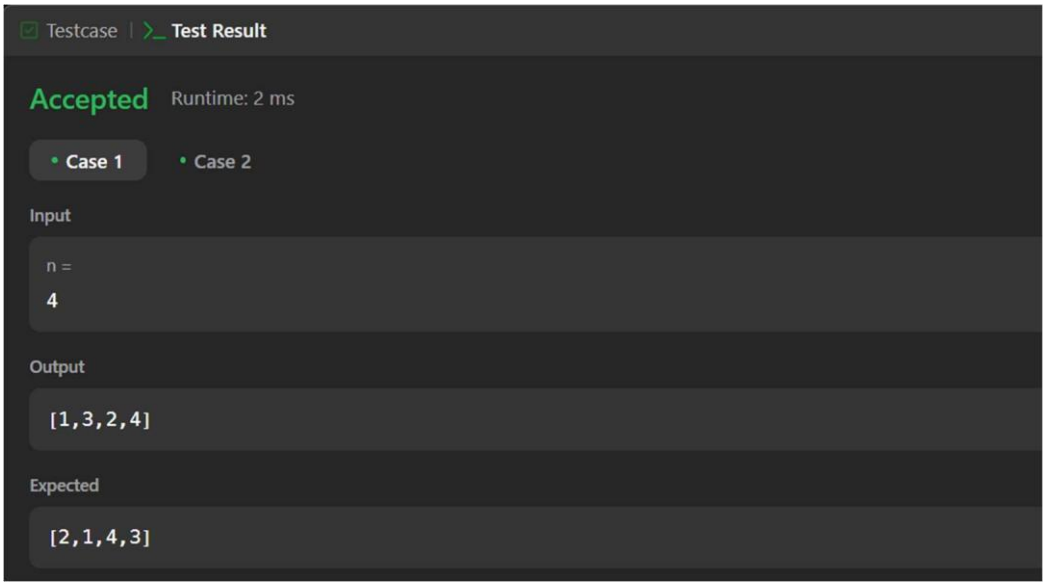

**Problem 7**. Given the integer n, return *any* ***beautiful*** *array* nums *of length* n. There will be at least one valid answer for the given n.

**Code:**

```
#include <vector>

using namespace std;

class Solution {
public:
    vector<int> beautifulArray(int n) {
        vector<int> result = {1};
        while (result.size() < n) {
            vector<int> temp;
            for (int num : result) {
                if (2 * num - 1 <= n)
                    temp.push_back(2 * num - 1);
            }
            for (int num : result) {
                if (2 * num <= n)
                    temp.push_back(2 * num);
            }
            result = temp;
        }
        return result;
```

```
    }
};
```

**Output:**




**Problem 8**. Given a positive integer n, write a function that returns the number of set bits in its binary representation (also known as the Hamming weight).

## Code:

```
#include <cstdint>

class Solution {
public:
    int hammingWeight(int n) {
        int count = 0;
        while (n != 0) {
            count += (n & 1);
            n >>= 1;
        }
        return count;
    }
};
```

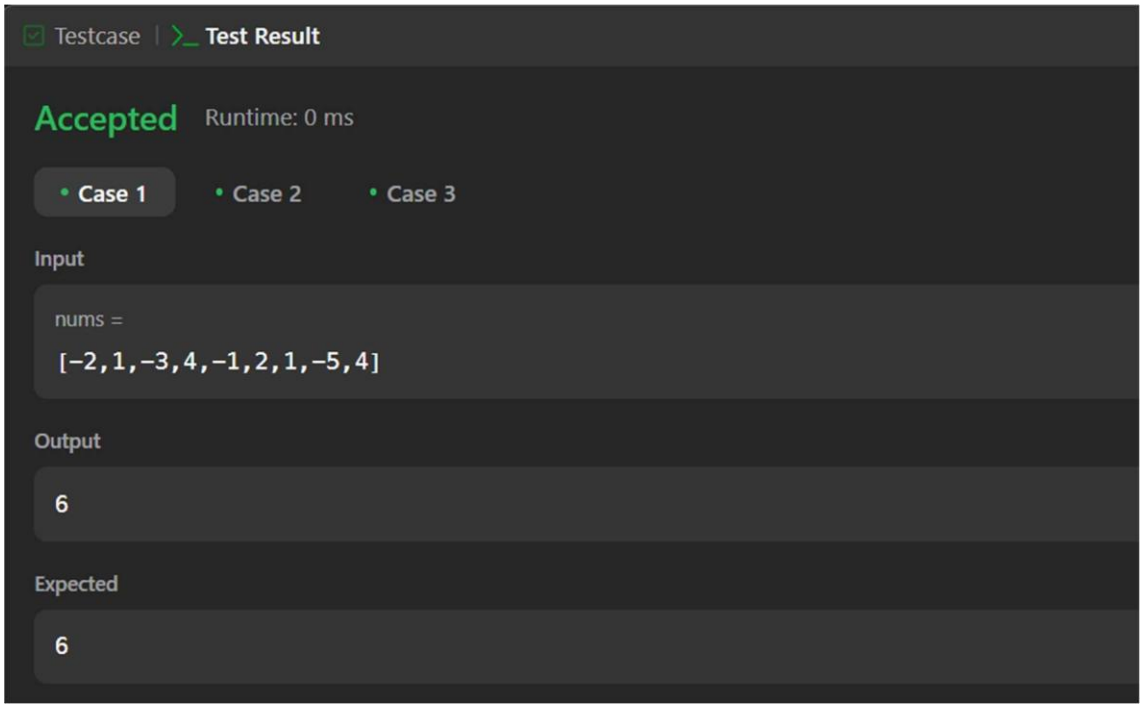
**Output:**


**Problem 9**. Given an integer array nums, find the subarray with the largest sum, and return *its sum*.

## Code:

```
#include <vector>
#include <algorithm>

using namespace std;

class Solution {
public:
   int maxSubArray(vector<int>& nums) {
      int maxsum = nums[0];
      int currsum = 0;

      for (int num : nums) {
         if (currsum < 0) {
            currsum = 0;
         }
         currsum += num;
         maxsum = max(maxsum, currsum);
      }

      return maxsum;
   }
};
```

**Output:**




**Problem 10**. Write an efficient algorithm that searches for a value target in an m x n integer matrix. This matrix has the following properties:

- Integers in each row are sorted in ascending from left to right.
- Integers in each column are sorted in ascending from top to bottom.

**Code:**

```
#include <vector>

using namespace std;

class Solution {
public:
   bool searchMatrix(vector<vector<int>>& matrix, int target) {
      if (matrix.empty() || matrix[0].empty()) return false;

      int rows = matrix.size();
      int cols = matrix[0].size();
      int left = 0, right = rows * cols - 1;

      while (left <= right) {
         int mid = left + (right - left) / 2;
         int row = mid / cols;
         int col = mid % cols;

         if (matrix[row][col] == target) return true;
         else if (matrix[row][col] < target) left = mid + 1;
         else right = mid - 1;
      }

      return false;
   }
};
```

**Output:**


Testcase
Test Result
Accepted
Runtime: 3 ms
Case 1
Case 2
Input
matrix =
[[1,4,7,11,15],[2,5,8,12,19],[3,6,9,16,22],[10,13,14,17,24],[18,21,23,26,30]]
target =
5
Output
true
Expected
true