

ASSIGNMENT-4(AP)

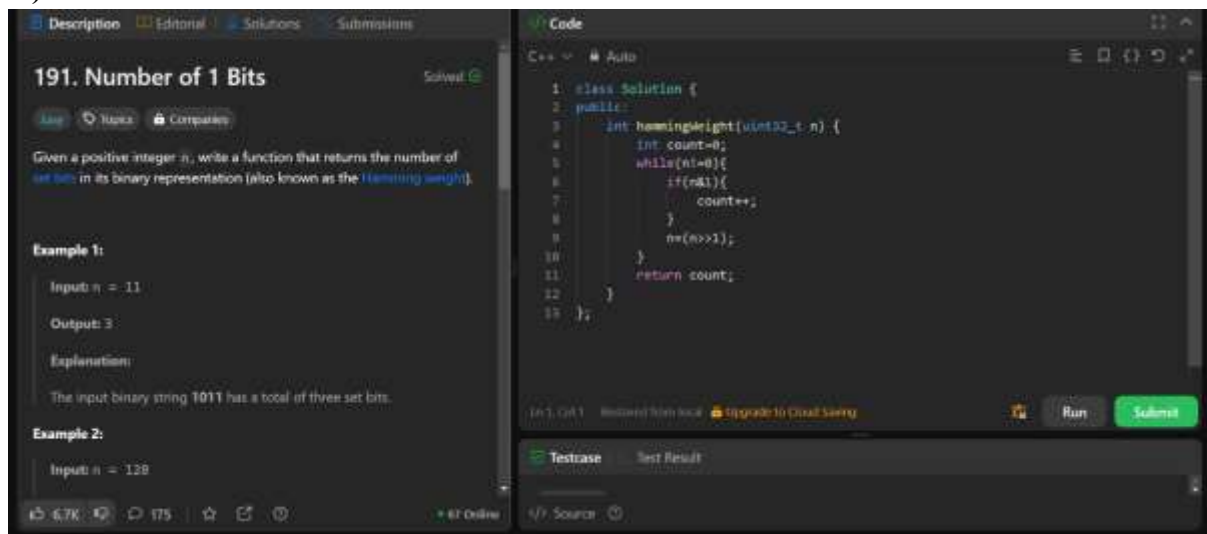
Name: Saharsh Kumar

UID: 22BCS14059

Section: 22BCS_FL_IOT-604

Group: A

1.) Number of 1 Bits

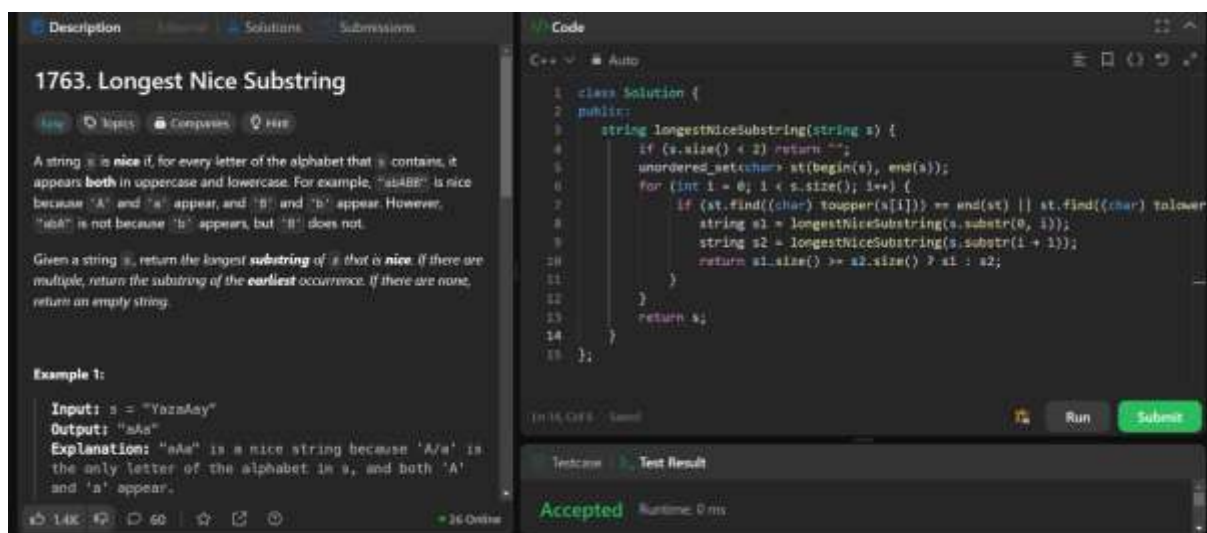


The screenshot shows a coding problem interface for "191. Number of 1 Bits". The problem description states: "Given a positive integer n , write a function that returns the number of set bits in its binary representation (also known as the **Hamming weight**)." Example 1 shows input $n = 11$ and output 3, with an explanation that the binary string 1011 has three set bits. Example 2 shows input $n = 128$. The solution code in C++ is as follows:

```
1 class Solution {
2 public:
3     int hammingWeight(uint32_t n) {
4         int count=0;
5         while(n!=0){
6             if(n&1){
7                 count++;
8             }
9             n=(n>>1);
10        }
11        return count;
12    }
13};
```

The code is submitted and the test case is passed, resulting in an "Accepted" status with a runtime of 0 ms.

2.) Longest Nice Substring



The screenshot shows a coding problem interface for "1763. Longest Nice Substring". The problem description states: "A string s is **nice** if, for every letter of the alphabet that s contains, it appears **both** in uppercase and lowercase. For example, "aBbBc" is nice because 'A' and 'a' appear, and 'B' and 'b' appear. However, "abB" is not because 'b' appears, but 'B' does not." Example 1 shows input $s = "YazsAay"$ and output "aAa", with an explanation that "aAa" is a nice string because 'A/a' is the only letter of the alphabet in s , and both 'A' and 'a' appear. The solution code in C++ is as follows:

```
1 class Solution {
2 public:
3     string longestNiceSubstring(string s) {
4         if (s.size() < 2) return "";
5         unordered_set<char> at(begin(s), end(s));
6         for (int i = 0; i < s.size(); i++) {
7             if (at.find((char) toupper(s[i])) == end(at) || at.find((char) tolower(s[i])) == end(at)) {
8                 string s1 = longestNiceSubstring(s.substr(0, i));
9                 string s2 = longestNiceSubstring(s.substr(i + 1));
10                return s1.size() >= s2.size() ? s1 : s2;
11            }
12        }
13        return s;
14    }
15};
```

The code is submitted and the test case is passed, resulting in an "Accepted" status with a runtime of 0 ms.

3.) Maximum Subarray

53. Maximum Subarray Solved

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

Example 1:
Input: `nums = [-2,1,-3,4,-1,2,1,-5,4]`
Output: 6
Explanation: The subarray `[-1,2,1]` has the largest sum 6.

Example 2:
Input: `nums = [1]`
Output: 1
Explanation: The subarray `[1]` has the largest sum 1.

Example 3:
Input: `nums = [5,4,-1,7,8]`
Output: 23
Explanation: The subarray `[5,4,-1,7,8]` has the largest sum 23.

12.3K 82 490 Online

```
1 class Solution {
2 public:
3     int maxSubArray(vector<int>& nums) {
4         int maxi=INT_MIN;
5         int sum=0;
6         for(int i=0;i<nums.size();i++){
7             sum+=nums[i];
8
9             if(sum>maxi){
10                 maxi=sum;
11             }
12             if(sum<0){
13                 sum=0;
14             }
15         }
16         return maxi;
17     }
```

Accepted Runtime: 0 ms

4.) Search a 2D Matrix II

240. Search a 2D Matrix II

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.

Example 1:

1	4	7	11	15
2	5	8	12	19
3	6	9	16	22

12.3K 82 33 Online

```
1 class Solution {
2 public:
3     bool searchMatrix(vector<vector<int>>& matrix, int target) {
4         int m = matrix.size(), n = m ? matrix[0].size() : 0, r = 0, c = 0;
5         while (r < m && c < n) {
6             if (matrix[r][c] == target) {
7                 return true;
8             }
9             matrix[r][c] > target ? c++ : r++;
10        }
11        return false;
12    }
```

Accepted Runtime: 3 ms

Case 1 Case 2

Input

5.) Super Pow

372. Super Pow

Medium

Your task is to calculate $a^b \bmod 1337$, where a is a positive integer and b is an extremely large positive integer given in the form of an array.

Example 1:

Input: $a = 2, b = [3]$
Output: 8

Example 2:

Input: $a = 2, b = [1,0]$
Output: 1024

Example 3:

Input: $a = 1, b = [4,3,3,8,5,2]$
Output: 1

```
1 class Solution {
2 public:
3     const int MOD = 1337;
4
5     int pow(int a, int b) {
6         int result = 1;
7         a %= MOD;
8         for (int i = 0; i < b; i++) {
9             result = (result * a) % MOD;
10        }
11        return result;
12    }
13
14    int superPow(int a, vector<int>& b) {
15        int result = 1;
16        for (int i = b.size() - 1; i >= 0; i--) {
17            result = (result * pow(a, b[i])) % MOD;
18        }
19        return result;
20    }
21 }
```

Accepted Runtime: 0 ms

6.) Reverse Bits

190. Reverse Bits

Easy

Reverse bits of a given 32 bits unsigned integer.

Note:

- Note that in some languages, such as Java, there is no unsigned integer type. In this case, both input and output will be given as a signed integer type. They should not affect your implementation, as the integer's internal binary representation is the same, whether it is signed or unsigned.
- In Java, the compiler represents the signed integers using [two's complement notation](#). Therefore, in **Example 2** above, the input represents the signed integer -3 and the output represents the signed integer -1073741825 .

Example 1:

```
1 class Solution {
2 public:
3     uint32_t reverseBits(uint32_t n) {
4         long int ans=0;
5         string n1="";
6         for(int i=0;i<32;n/=2,i++){
7             n1 = to_string((n%2)) + n1;
8         }
9         for(int i = 31;i>=0;i--){
10             ans+= pow(2,i)*((int)(n1[i])-48);
11             n1.erase(i);
12         }
13     }
14 }
```

Accepted Runtime: 0 ms

Case 1 Case 2