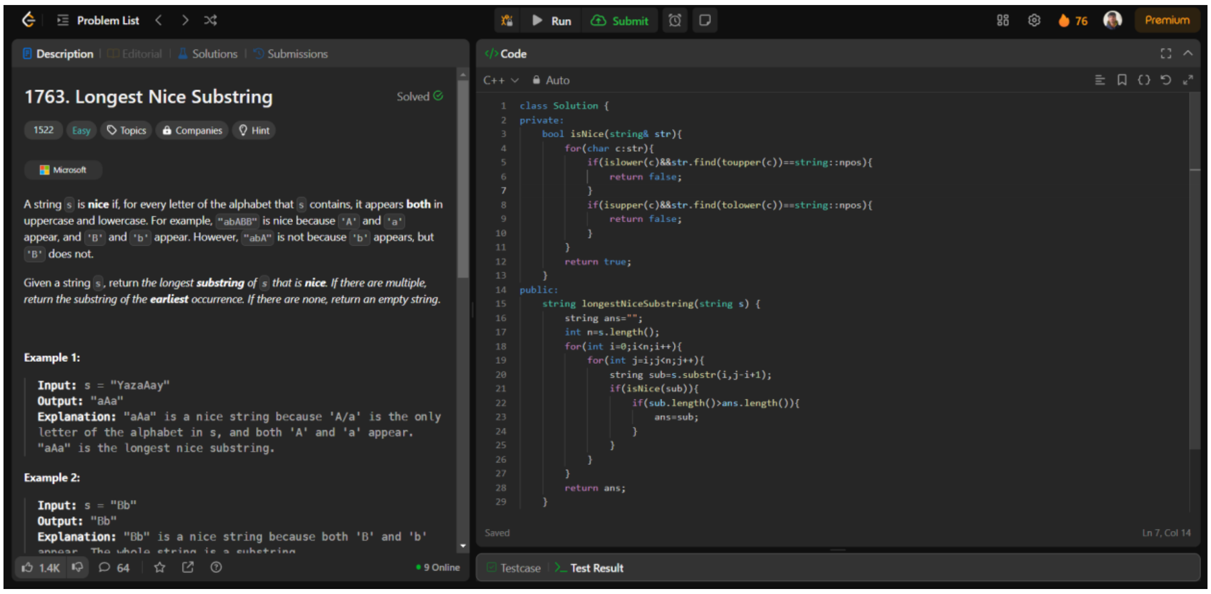
Name: - Chakshit Guleria

UID: - 22BCS16813

Sec/Group: - IOT_609/B

AP worksheet

1. **Longest Nice Substring:**



1763. Longest Nice Substring

A string `s` is **nice** if, for every letter of the alphabet that `s` contains, it appears **both** in uppercase and lowercase. For example, `"abABB"` is nice because `'A'` and `'a'` appear, and `'B'` and `'b'` appear. However, `"abA"` is not because `'b'` appears, but `'B'` does not.

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.*

Example 1:

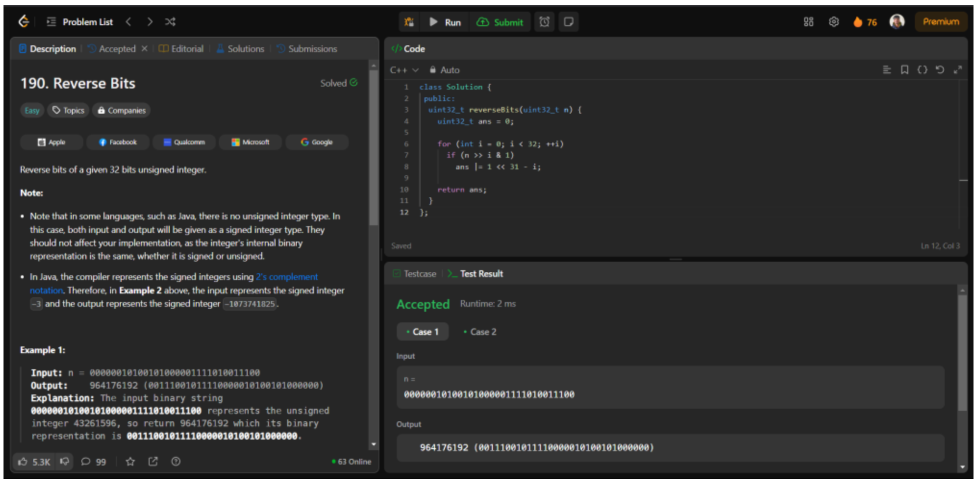
```
Input: s = "YazaAay"
Output: "aAa"
Explanation: "aAa" is a nice string because 'A/a' is the only
letter of the alphabet in s, and both 'A' and 'a' appear.
"aAa" is the longest nice substring.
```

Example 2:

```
Input: s = "Bb"
Output: "Bb"
Explanation: "Bb" is a nice string because both 'B' and 'b'
```

```cpp
class Solution {
private:
    bool isNice(string& str){
        for(char c:str){
            if(islower(c)&&str.find(toupper(c))==string::npos){
                return false;
            }
            if(isupper(c)&&str.find(tolower(c))==string::npos){
                return false;
            }
        }
        return true;
    }
public:
    string longestNiceSubstring(string s) {
        string ans="";
        int n=s.length();
        for(int i=0;i<n;i++){
            for(int j=i;j<n;j++){
                string sub=s.substr(i,j-i+1);
                if(isNice(sub)){
                    if(sub.length()>ans.length()){
                        ans=sub;
                    }
                }
            }
        }
        return ans;
    }
```

2. **Reverse Bits**



190. Reverse Bits

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 2'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:

```
Input: n = 00000010100101000001111010011100
Output:    964176192 (00111001011110000010100101000000)
Explanation: The input binary string
00000010100101000001111010011100 represents the unsigned
integer 43261596, so return 964176192 which its binary
representation is 00111001011110000010100101000000.
```

```cpp
class Solution {
 public:
  uint32_t reverseBits(uint32_t n) {
    uint32_t ans = 0;

    for (int i = 0; i < 32; ++i)
      if (n >> i & 1)
        ans |= 1 << 31 - i;

    return ans;
  }
};
```

Accepted Runtime: 2 ms

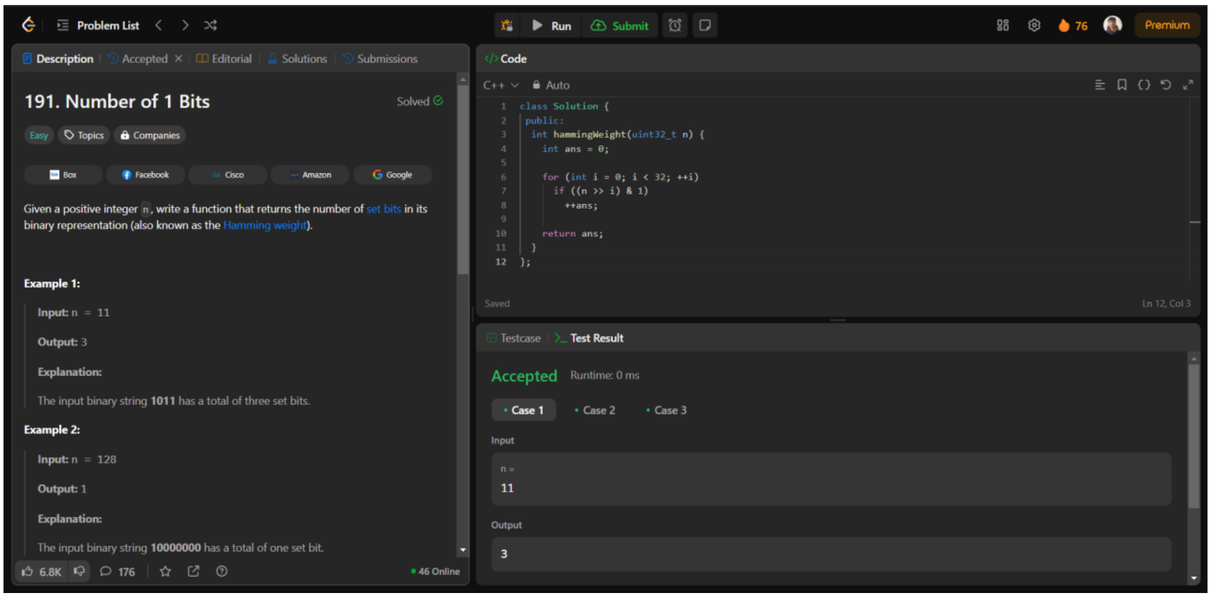
Input

```
n =
00000010100101000001111010011100
```

Output

```
964176192 (00111001011110000010100101000000)
```

3. **Number of 1 Bits:**



191. Number of 1 Bits

Solved

Easy · Topics · Companies

Box · Facebook · Cisco · Amazon · Google

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:

Input: n = 11

Output: 3

Explanation:

The input binary string **1011** has a total of three set bits.

Example 2:

Input: n = 128

Output: 1

Explanation:

The input binary string **10000000** has a total of one set bit.

6.8K · 176 · 46 Online

Code · C++ · Auto

```cpp
class Solution {
 public:
  int hammingWeight(uint32_t n) {
    int ans = 0;

    for (int i = 0; i < 32; ++i)
      if ((n >> i) & 1)
        ++ans;

    return ans;
  }
};
```

Saved · Ln 12, Col 3

Testcase · Test Result

Accepted Runtime: 0 ms

Case 1 · Case 2 · Case 3

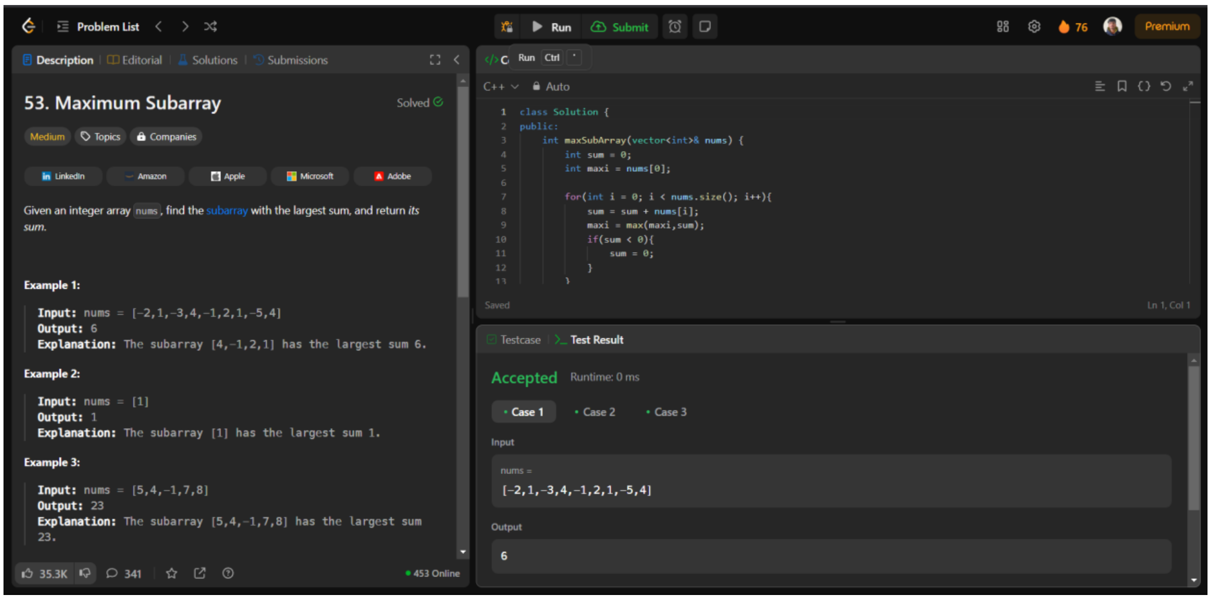
Input

```
n =
11
```

Output

```
3
```

4. Maximum Subarray:



53. Maximum Subarray

Solved

Medium · Topics · Companies

LinkedIn · Amazon · Apple · Microsoft · Adobe

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 [4,-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.
```

35.3K · 341 · 453 Online

Run Ctrl ' · C++ · Auto

```cpp
class Solution {
public:
    int maxSubArray(vector<int>& nums) {
        int sum = 0;
        int maxi = nums[0];

        for(int i = 0; i < nums.size(); i++){
            sum = sum + nums[i];
            maxi = max(maxi,sum);
            if(sum < 0){
                sum = 0;
            }
        }
```

Saved · Ln 1, Col 1

Testcase · Test Result

Accepted Runtime: 0 ms

Case 1 · Case 2 · Case 3

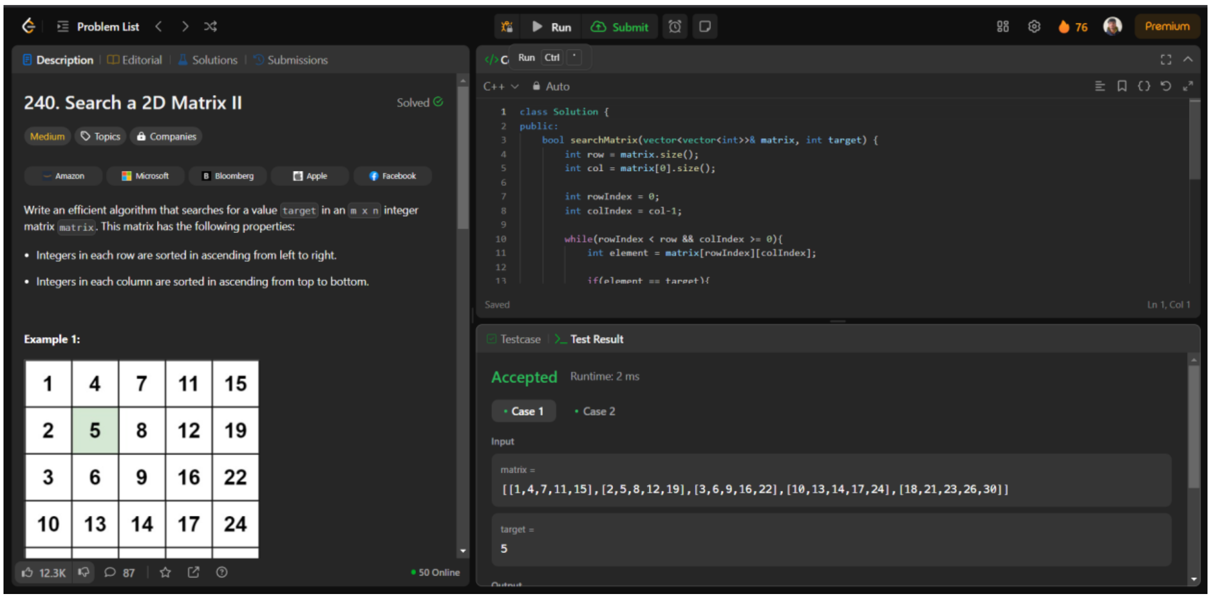
Input

```
nums =
[-2,1,-3,4,-1,2,1,-5,4]
```

Output

```
6
```

5. Search a 2D Matrix II:


240. Search a 2D Matrix II

Solved

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
10	13	14	17	24

```cpp
class Solution {
public:
    bool searchMatrix(vector<vector<int>>& matrix, int target) {
        int row = matrix.size();
        int col = matrix[0].size();

        int rowIndex = 0;
        int colIndex = col-1;

        while(rowIndex < row && colIndex >= 0){
            int element = matrix[rowIndex][colIndex];

            if(element == target){
```

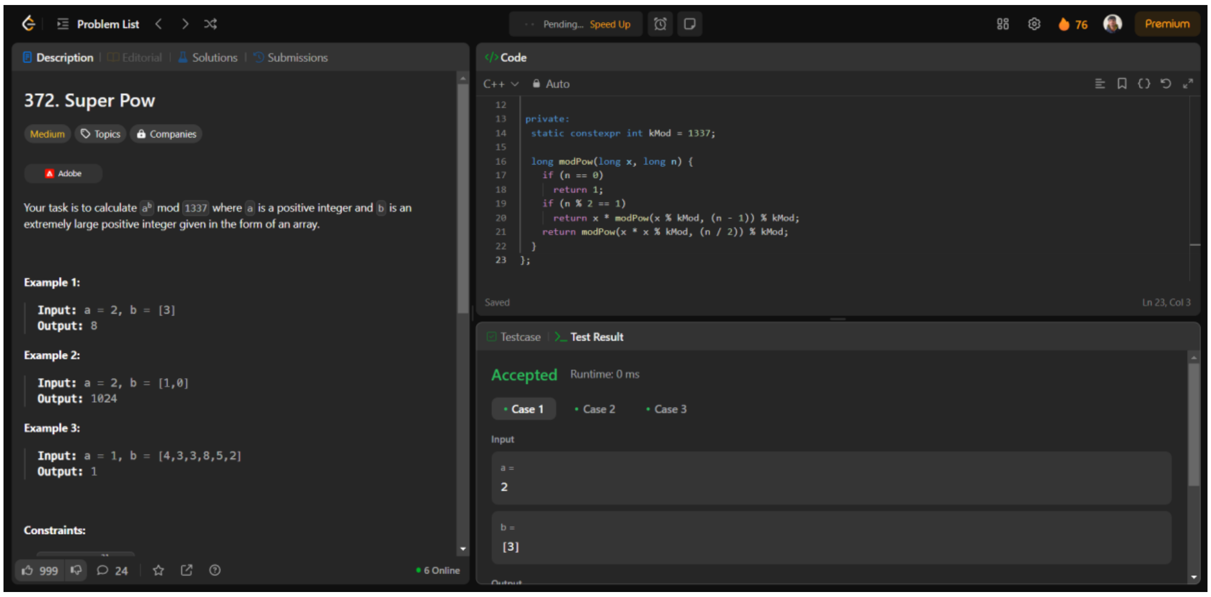
Accepted Runtime: 2 ms

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
```

6. Super Pow:


372. Super Pow

Your task is to calculate a^b mod `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
```

Constraints:

```cpp

 private:
  static constexpr int kMod = 1337;

  long modPow(long x, long n) {
    if (n == 0)
      return 1;
    if (n % 2 == 1)
      return x * modPow(x % kMod, (n - 1)) % kMod;
    return modPow(x * x % kMod, (n / 2)) % kMod;
  }
};
```

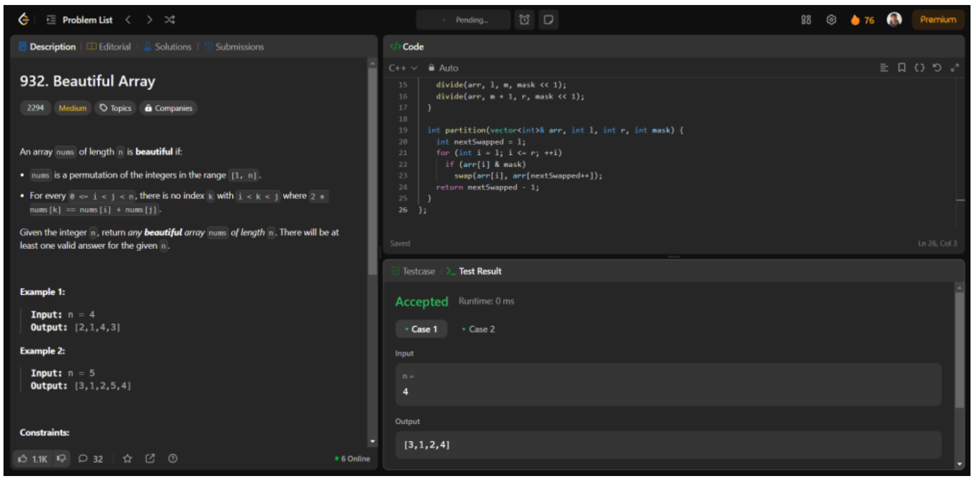
Accepted Runtime: 0 ms

Case 1 · Case 2 · Case 3

Input

```
a =
2
b =
[3]
```

7. Beautiful Array:


932. Beautiful Array

An array `nums` of length `n` is **beautiful** if:

- `nums` is a permutation of the integers in the range `[1, n]`.
- For every `0 <= i < j < n`, there is no index `k` with `i < k < j` where `2 * nums[k] == nums[i] + nums[j]`.

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

Example 1:

```
Input: n = 4
Output: [2,1,4,3]
```

Example 2:

```
Input: n = 5
Output: [3,1,2,5,4]
```

Constraints:

```cpp
    divide(arr, l, m, mask << 1);
    divide(arr, m + 1, r, mask << 1);
  }

  int partition(vector<int>& arr, int l, int r, int mask) {
    int nextSwapped = l;
    for (int i = l; i <= r; ++i)
      if (arr[i] & mask)
        swap(arr[i], arr[nextSwapped++]);
    return nextSwapped - 1;
  }
};
```

Accepted Runtime: 0 ms

Case 1 · Case 2

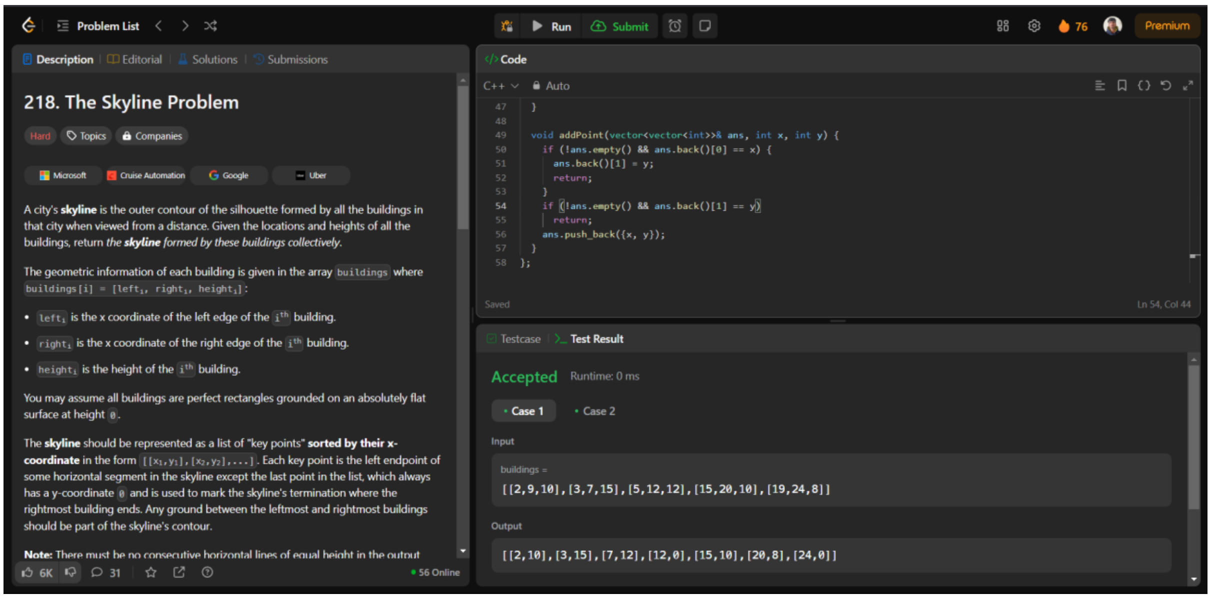
Input

```
n =
4
```

Output

```
[3,1,2,4]
```

8. The Skyline Problem:



218. The Skyline Problem

A city's **skyline** is the outer contour of the silhouette formed by all the buildings in that city when viewed from a distance. Given the locations and heights of all the buildings, return *the **skyline** formed by these buildings collectively*.

The geometric information of each building is given in the array `buildings` where `buildings[i] = [left_i, right_i, height_i]`:

- `left_i` is the x coordinate of the left edge of the `i^th` building.
- `right_i` is the x coordinate of the right edge of the `i^th` building.
- `height_i` is the height of the `i^th` building.

You may assume all buildings are perfect rectangles grounded on an absolutely flat surface at height `0`.

The **skyline** should be represented as a list of "key points" **sorted by their x-coordinate** in the form `[[x1,y1],[x2,y2],...]`. Each key point is the left endpoint of some horizontal segment in the skyline except the last point in the list, which always has a y-coordinate `0` and is used to mark the skyline's termination where the rightmost building ends. Any ground between the leftmost and rightmost buildings should be part of the skyline's contour.

Note: There must be no consecutive horizontal lines of equal height in the output

```
  }

  void addPoint(vector<vector<int>>& ans, int x, int y) {
    if (!ans.empty() && ans.back()[0] == x) {
      ans.back()[1] = y;
      return;
    }
    if (!ans.empty() && ans.back()[1] == y)
      return;
    ans.push_back({x, y});
  }
};
```

Saved Ln 54, Col 44

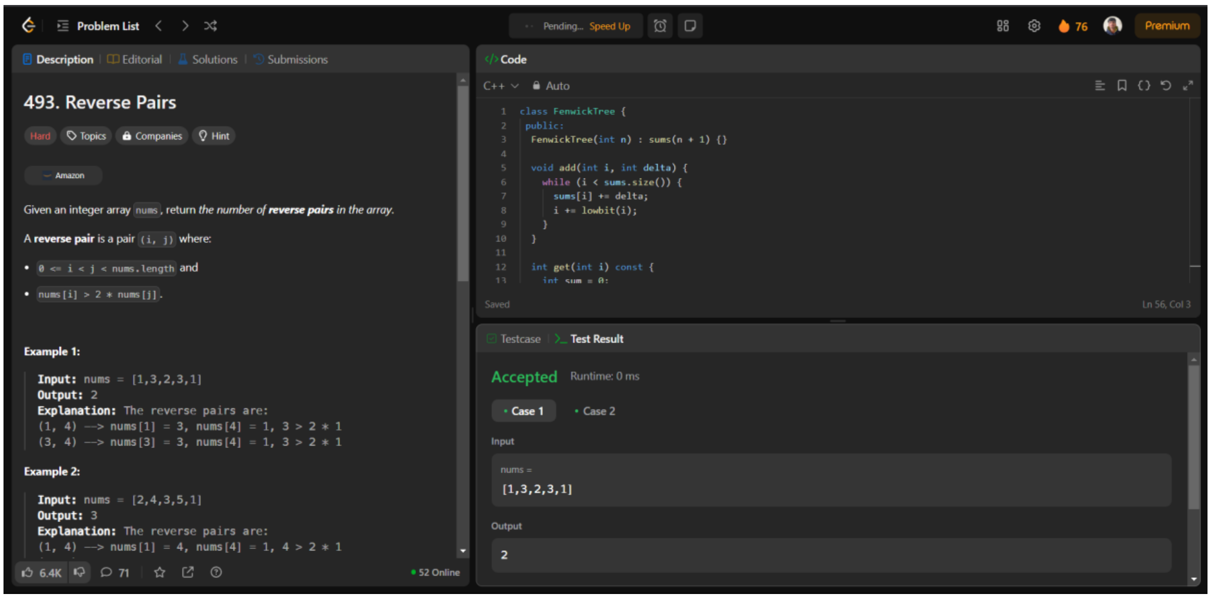
Accepted Runtime: 0 ms

Input

buildings =
```
[[2,9,10],[3,7,15],[5,12,12],[15,20,10],[19,24,8]]
```

Output
```
[[2,10],[3,15],[7,12],[12,0],[15,10],[20,8],[24,0]]
```

9. Reverse Pairs:



493. Reverse Pairs

Given an integer array `nums`, return *the number of **reverse pairs** in the array*.

A **reverse pair** is a pair `(i, j)` where:

- `0 <= i < j < nums.length` and
- `nums[i] > 2 * nums[j]`.

Example 1:

```
Input: nums = [1,3,2,3,1]
Output: 2
Explanation: The reverse pairs are:
(1, 4) --> nums[1] = 3, nums[4] = 1, 3 > 2 * 1
(3, 4) --> nums[3] = 3, nums[4] = 1, 3 > 2 * 1
```

Example 2:

```
Input: nums = [2,4,3,5,1]
Output: 3
Explanation: The reverse pairs are:
(1, 4) --> nums[1] = 4, nums[4] = 1, 4 > 2 * 1
```

```
class FenwickTree {
 public:
  FenwickTree(int n) : sums(n + 1) {}

  void add(int i, int delta) {
    while (i < sums.size()) {
      sums[i] += delta;
      i += lowbit(i);
    }
  }

  int get(int i) const {
    int sum = 0;
```

Saved Ln 56, Col 3

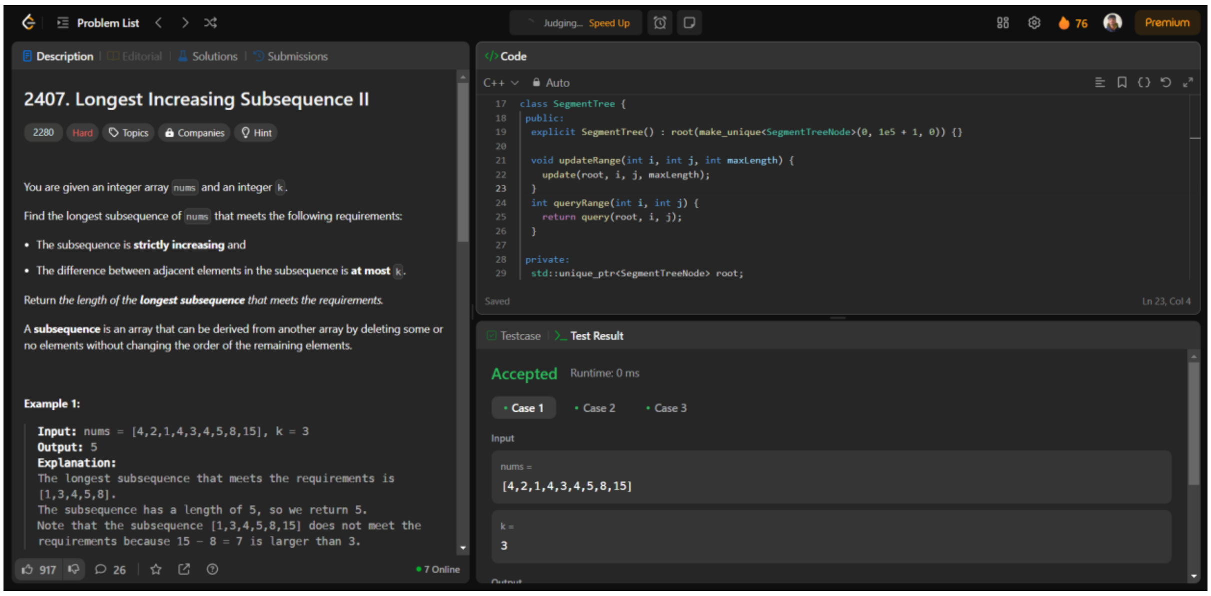
Accepted Runtime: 0 ms

Input

nums =
```
[1,3,2,3,1]
```

Output
```
2
```

10.Longest Increasing Subsequence II:



2407. Longest Increasing Subsequence II

You are given an integer array `nums` and an integer `k`.

Find the longest subsequence of `nums` that meets the following requirements:

- The subsequence is **strictly increasing** and
- The difference between adjacent elements in the subsequence is **at most** `k`.

Return *the length of the* ***longest subsequence*** *that meets the requirements.*

A **subsequence** is an array that can be derived from another array by deleting some or no elements without changing the order of the remaining elements.

Example 1:

```
Input: nums = [4,2,1,4,3,4,5,8,15], k = 3
Output: 5
Explanation:
The longest subsequence that meets the requirements is
[1,3,4,5,8].
The subsequence has a length of 5, so we return 5.
Note that the subsequence [1,3,4,5,8,15] does not meet the
requirements because 15 - 8 = 7 is larger than 3.
```

```cpp
class SegmentTree {
 public:
  explicit SegmentTree() : root(make_unique<SegmentTreeNode>(0, 1e5 + 1, 0)) {}

  void updateRange(int i, int j, int maxLength) {
    update(root, i, j, maxLength);
  }
  int queryRange(int i, int j) {
    return query(root, i, j);
  }

 private:
  std::unique_ptr<SegmentTreeNode> root;
```

Accepted Runtime: 0 ms

Input

nums =
[4,2,1,4,3,4,5,8,15]

k =
3