



DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

Assignment-4

Student Name: Kanish Rohilla

UID: 22BCS16248

Branch: BE- CSE-General

Section/Group: 22BCS-IOT-606/B

Semester: 6th

Date of Submission: 17/03/25

Subject Name: Advanced Programming Lab-2

Subject Code: 22CSP-351

Submitted to: Ms. Pratima Sonali Horo(E18304)

Question-1: Longest Nice Substring:

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.

Answer:

```
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;
                    }
                }
            }
        }
    }
}
```



DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

return ans;

}};

1763. Longest Nice Substring

Easy Topics Companies Hint

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: $s = \text{"YazaAay"}$
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: $s = \text{"Bb"}$
Output: "Bb"
Explanation: "Bb" is a nice string because both 'B' and 'b' appear. The whole string is a substring.

```
6         return false;
7     }
8     if(isupper(c)&&str.find(tolower(c))==string::npos){
9         return false;
10    }
11    }
12    return true;
13 }
```

Accepted Runtime: 0 ms

Case 1 Case 2 Case 3

Input

$s =$

"YazaAay"

Output

"aAa"

Expected

Question-2: Reverse Bits:

```
class Solution {
public:
    uint32_t reverseBits(uint32_t n) {
        uint32_t res = 0;
        for (int i = 0; i < 32; i++) {
            res = (res << 1) | (n & 1); // Shift result left, add LSB of n
            n >>= 1; // Shift n right
        }
        return res;
    }
};
```

190. Reverse Bits

Easy Topics Companies

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.

```
1 class Solution {
2 public:
3     uint32_t reverseBits(uint32_t n) {
4         uint32_t res = 0;
5         for (int i = 0; i < 32; i++) {
6             res = (res << 1) | (n & 1); // Shift result left, add LSB of n
7             n >>= 1; // Shift n right
8         }
9         return res;
10    }
11 }
```

Accepted Runtime: 0 ms

Case 1 Case 2

Input

$n =$

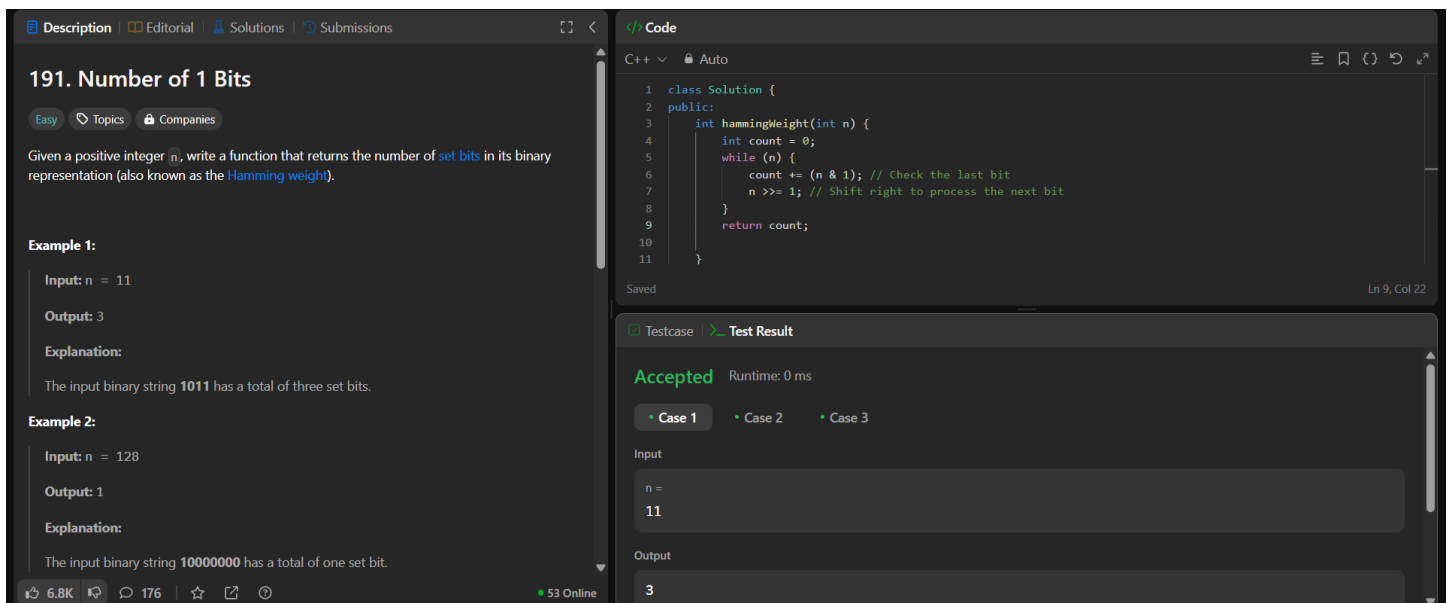
00000010100101000001111010011100

Output

Question-3: Number of 1 Bits: 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](#)).

Answer:

```
class Solution {
public:
    int hammingWeight(int n) {
        int count = 0;
        while (n) {
            count += (n & 1); // Check the last bit
            n >>= 1; // Shift right to process the next bit
        }
        return count;
    }
};
```



The screenshot displays a coding platform interface for the problem "191. Number of 1 Bits". The left panel shows the problem description, which asks for a function to return the number of set bits in the binary representation of a positive integer n . It includes two examples: Example 1 with input $n = 11$ and output 3, and Example 2 with input $n = 128$ and output 1. The right panel shows the C++ code for the solution, which is the same as the one provided in the text. Below the code, the test results are shown as "Accepted" with a runtime of 0 ms. The interface also includes tabs for "Description", "Editorial", "Solutions", and "Submissions", and a bottom status bar indicating 6.8K likes, 176 comments, and 53 online users.

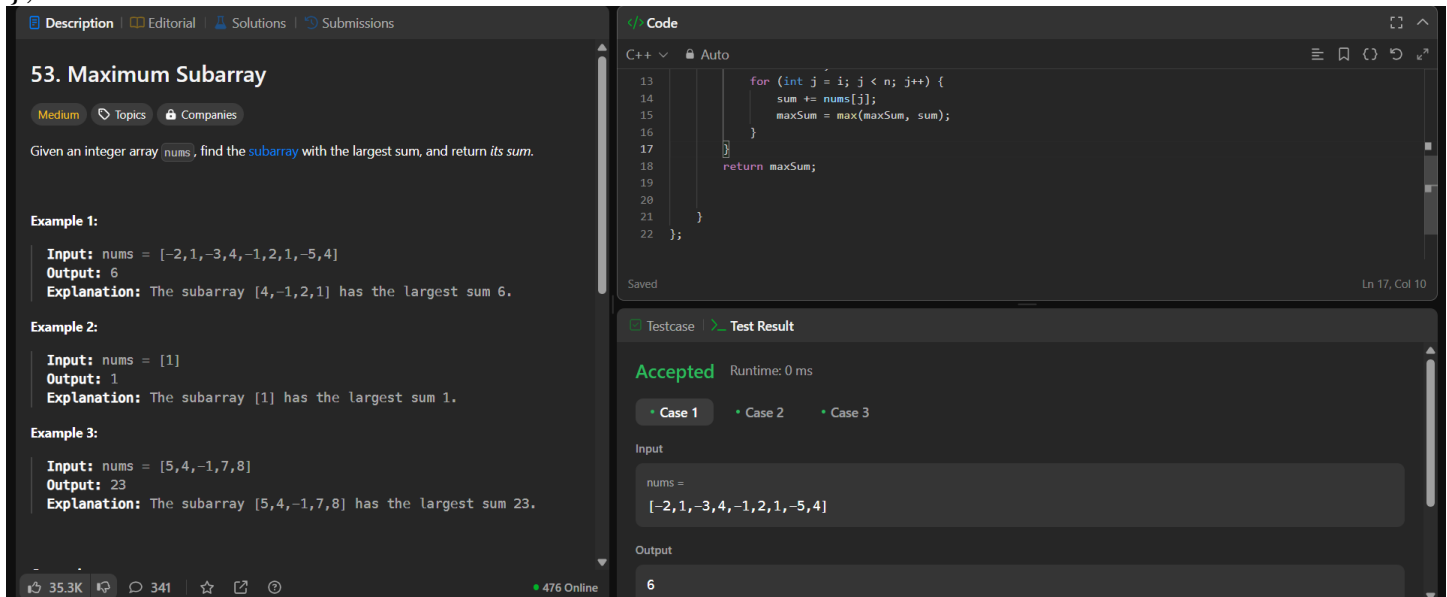
Question-4: Maximum Subarray:

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

Answer:

```
class Solution {
public:
    int maxSubArray(vector<int>& nums) {
        // int maxSum = nums[0], curSum = 0;
```

```
// for (int num : nums) {
//     curSum = max(num, curSum + num);
//     maxSum = max(maxSum, curSum);
// }
// return maxSum;
int maxSum = nums[0], n = nums.size();
for (int i = 0; i < n; i++) {
    int sum = 0;
    for (int j = i; j < n; j++) {
        sum += nums[j];
        maxSum = max(maxSum, sum);
    }
}
return maxSum;
};
```



The screenshot shows a coding platform interface. On the left, the problem description for '53. Maximum Subarray' is visible, including examples and constraints. On the right, the code editor shows the C++ solution. Below the code editor, the test results are displayed, showing 'Accepted' status and the input/output for the first test case.

53. Maximum Subarray

Medium Topics Companies

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.

Code Editor:

```
C++
13     for (int j = i; j < n; j++) {
14         sum += nums[j];
15         maxSum = max(maxSum, sum);
16     }
17 }
18 return maxSum;
19 }
20 };
21
22
```

Test Results:

Accepted Runtime: 0 ms

Case 1 Case 2 Case 3

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

Output: 6

Question-5: 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.

Answer:

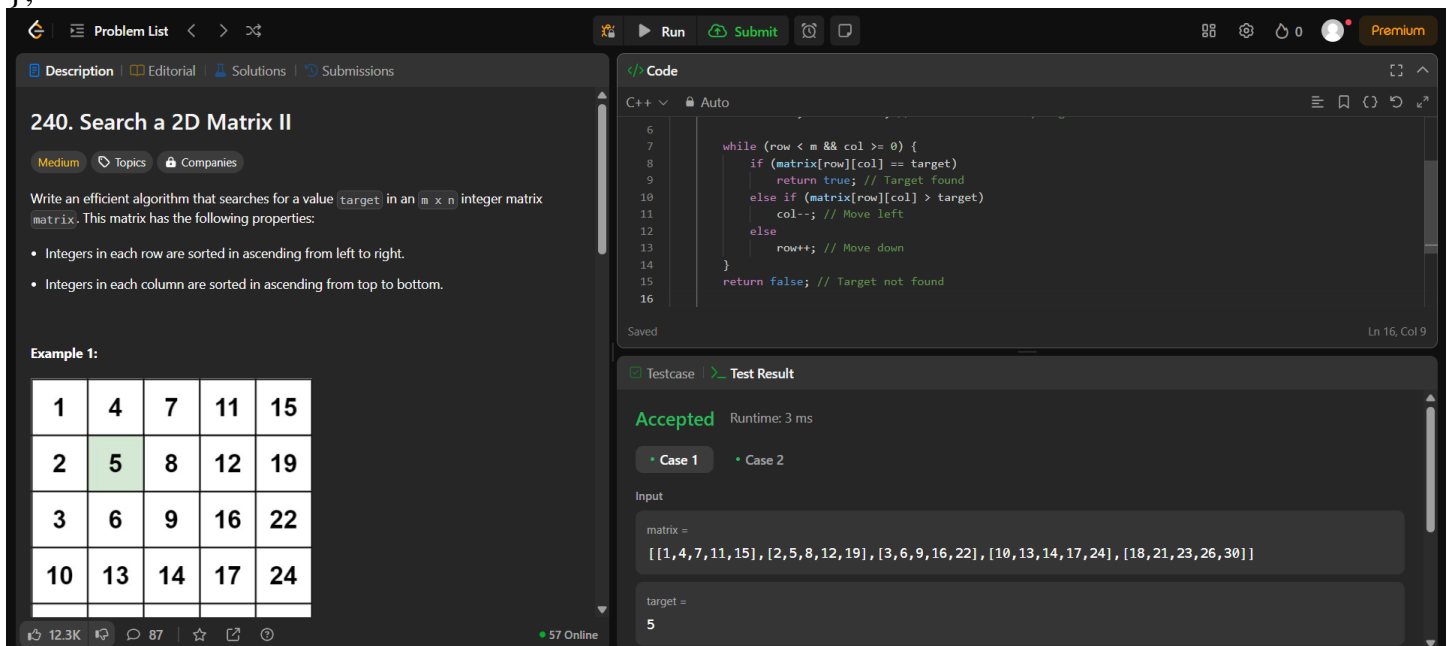
```
class Solution {
```

public:

```
bool searchMatrix(vector<vector<int>>& matrix, int target) {
    int m = matrix.size(), n = matrix[0].size();
    int row = 0, col = n - 1; // Start from the top-right corner
```

```
    while (row < m && col >= 0) {
        if (matrix[row][col] == target)
            return true; // Target found
        else if (matrix[row][col] > target)
            col--; // Move left
        else
            row++; // Move down
    }
    return false; // Target not found
```

```
};
```



The screenshot shows a coding platform interface with the following components:

- Problem List:** A sidebar on the left showing the problem list.
- Description:** The main area on the left containing the problem statement for "240. Search a 2D Matrix II". It includes a "Medium" difficulty tag, "Topics", and "Companies". The problem asks to write an efficient algorithm to search for a target in an m x n integer matrix where rows and columns are sorted in ascending order. It also provides an example matrix and its properties.
- Code:** The central area showing the C++ code solution. The code implements a search algorithm starting from the top-right corner and moving left or down based on the comparison with the target.
- Testcase / Test Result:** The bottom right area showing the test results. It indicates that the solution is "Accepted" with a runtime of 3 ms. It also shows the input matrix and target value used for testing.

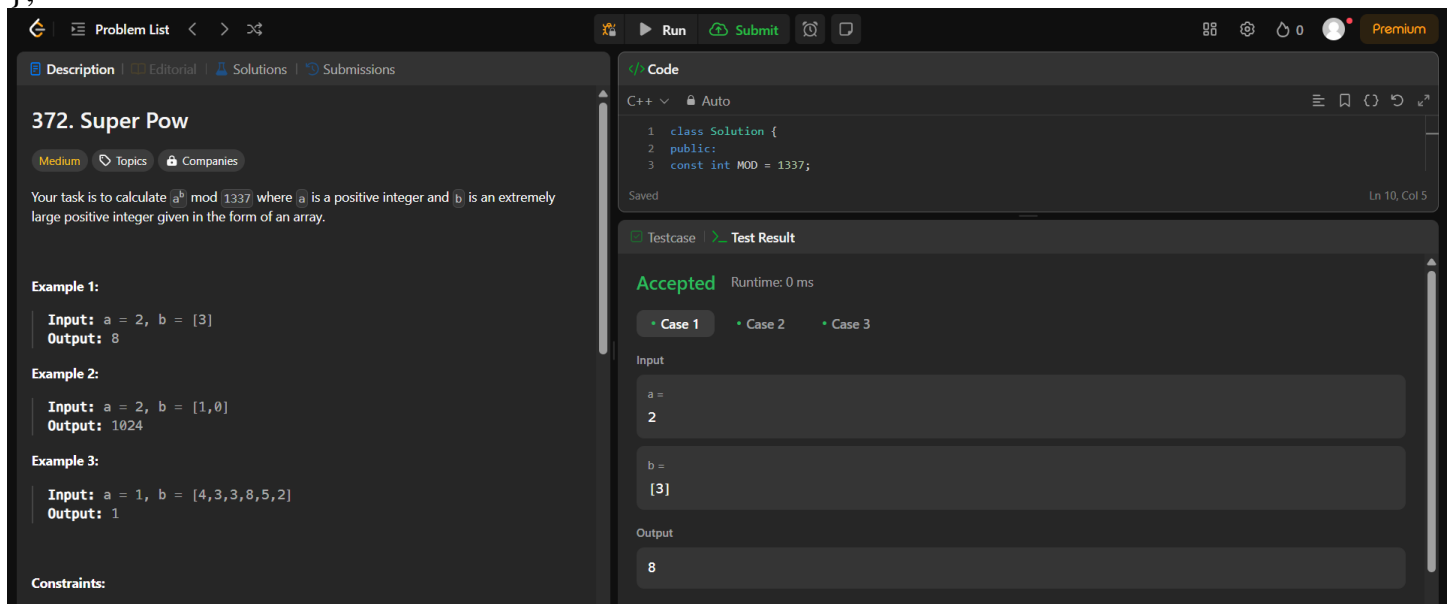
Question-6: Super Pow: Your task is to calculate $a^b \text{ mod } 1337$ where a is a positive integer and b is an extremely large positive integer given in the form of an array.

Answer:

```
class Solution {
public:
    const int MOD = 1337;
```

```
int powerMod(int x, int y) {
    int result = 1;
    x %= MOD;
    while (y > 0) {
        if (y % 2 == 1) result = (result * x) % MOD;
        x = (x * x) % MOD;
        y /= 2;
    }
    return result;
}

int superPow(int a, vector<int>& b) {
    if (b.empty()) return 1;
    int lastDigit = b.back();
    b.pop_back();
    return (powerMod(superPow(a, b), 10) * powerMod(a, lastDigit)) % MOD;
}
};
```



372. Super Pow

Medium Topics Companies

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

Constraints:

Code

```
C++
1 class Solution {
2 public:
3     const int MOD = 1337;
4 }
```

Testcase | Test Result

Accepted Runtime: 0 ms

Case 1 **Case 2** **Case 3**

Input

$a =$
2

$b =$
[3]

Output
8

Question-7: 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 $0 \leq i < j < n$, there is no index `k` with $i < k < j$ where $2 * \text{nums}[k] == \text{nums}[i] + \text{nums}[j]$.

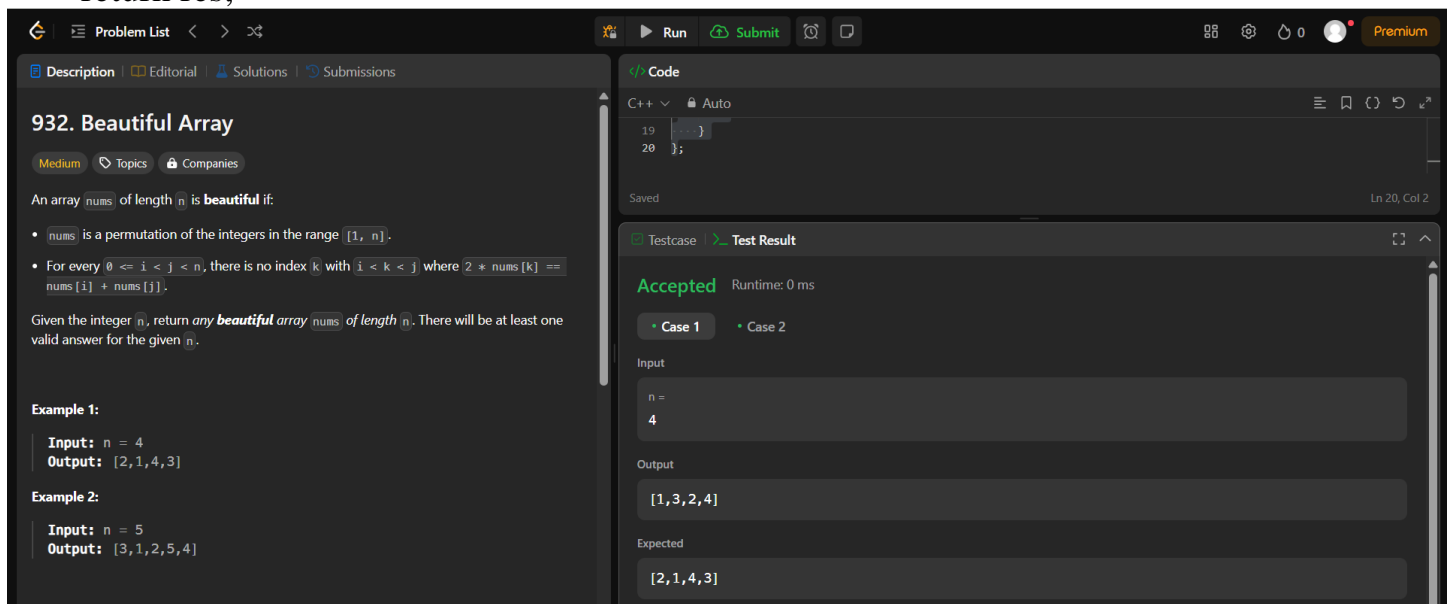
Given the integer `n`, return *any beautiful array* `nums` of length `n`. There will be at least one valid answer for the given `n`.

Answer:

```
vector<int> res = {1}; // Start with base case
```

```
while (res.size() < n) {
    vector<int> temp;
    for (int num : res) {
        if (num * 2 - 1 <= n) temp.push_back(num * 2 - 1); // Odd numbers
    }
    for (int num : res) {
        if (num * 2 <= n) temp.push_back(num * 2); // Even numbers
    }
    res = temp;
}
```

```
return res;
```



The screenshot shows a coding platform interface. On the left, the problem description for "932. Beautiful Array" is visible. It states that an array `nums` of length `n` is beautiful if:

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

 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]`
 On the right, the code editor shows a C++ solution:


```
19 vector<int> res = {1};
20 return res;
```

 Below the code editor, the test result is shown as "Accepted" with a runtime of 0 ms. The input field shows `n = 4`, and the output field shows `[1, 3, 2, 4]`. The expected output is `[2, 1, 4, 3]`.

Question-8: 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] = [lefti, righti, heighti]`:

`lefti` is the x coordinate of the left edge of the `i`th building.

`righti` 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 $[[x_1, y_1], [x_2, y_2], \dots]$. 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 skyline. For instance, $[\dots, [2, 3], [4, 5], [7, 5], [11, 5], [12, 7], \dots]$ is not acceptable; the three lines of height 5 should be merged into one in the final output as such: $[\dots, [2, 3], [4, 5], [12, 7], \dots]$

Answer:

```
class Solution {
public:
    vector<vector<int>> getSkyline(vector<vector<int>>& buildings) {
        vector<pair<int, int>> events;
        vector<vector<int>> result;

        // Step 1: Convert buildings into events (start and end points)
        for (auto& b : buildings) {
            events.emplace_back(b[0], -b[2]); // Start event (negative height to differentiate start)
            events.emplace_back(b[1], b[2]); // End event (positive height)
        }

        // Step 2: Sort events (first by x, then by height)
        sort(events.begin(), events.end());

        // Step 3: Process events with a max heap (multiset for ordered heights)
        multiset<int> heights = {0}; // Store heights (with ground level)
        int prevMax = 0;

        for (auto& [x, h] : events) {
            if (h < 0) heights.insert(-h); // Start event (add height)
            else heights.erase(heights.find(h)); // End event (remove height)

            int currMax = *heights.rbegin(); // Get max height
```




DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

```
// Step 4: If height changes, add to result
    if (currMax != prevMax) {
        result.push_back({x, currMax});
        prevMax = currMax;
    }
}

return result;

}

};
```

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] = [lefti, righti, heighti]`:

- `lefti` is the x coordinate of the left edge of the `ith` building.
- `righti` is the x coordinate of the right edge of the `ith` building.
- `heighti` is the height of the `ith` 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 skyline. For instance, `[[1,2],[2,3],[3,4],[4,5],[5,6],[6,7],[7,8],[8,9],[9,10]]` is not acceptable: the three lines of height 4, 5 and 6 are consecutive and should be merged into one line segment `[[1,2],[2,3],[3,4],[4,5],[5,6],[6,7],[7,8],[8,9],[9,10]]`.

Accepted Runtime: 0 ms

Case 1 Case 2

Input

`buildings =`

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

Output

Question-9: 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:

- $0 \leq i < j < \text{nums.length}$ and
- $\text{nums}[i] > 2 * \text{nums}[j]$.

Answer:

```
class Solution {
public:
    int mergeAndCount(vector<int>& nums, int left, int mid, int right) {
        int count = 0, j = mid + 1;
```

```
// Count reverse pairs
for (int i = left; i <= mid; i++) {
    while (j <= right && nums[i] > 2LL * nums[j]) j++;
    count += (j - (mid + 1));
}

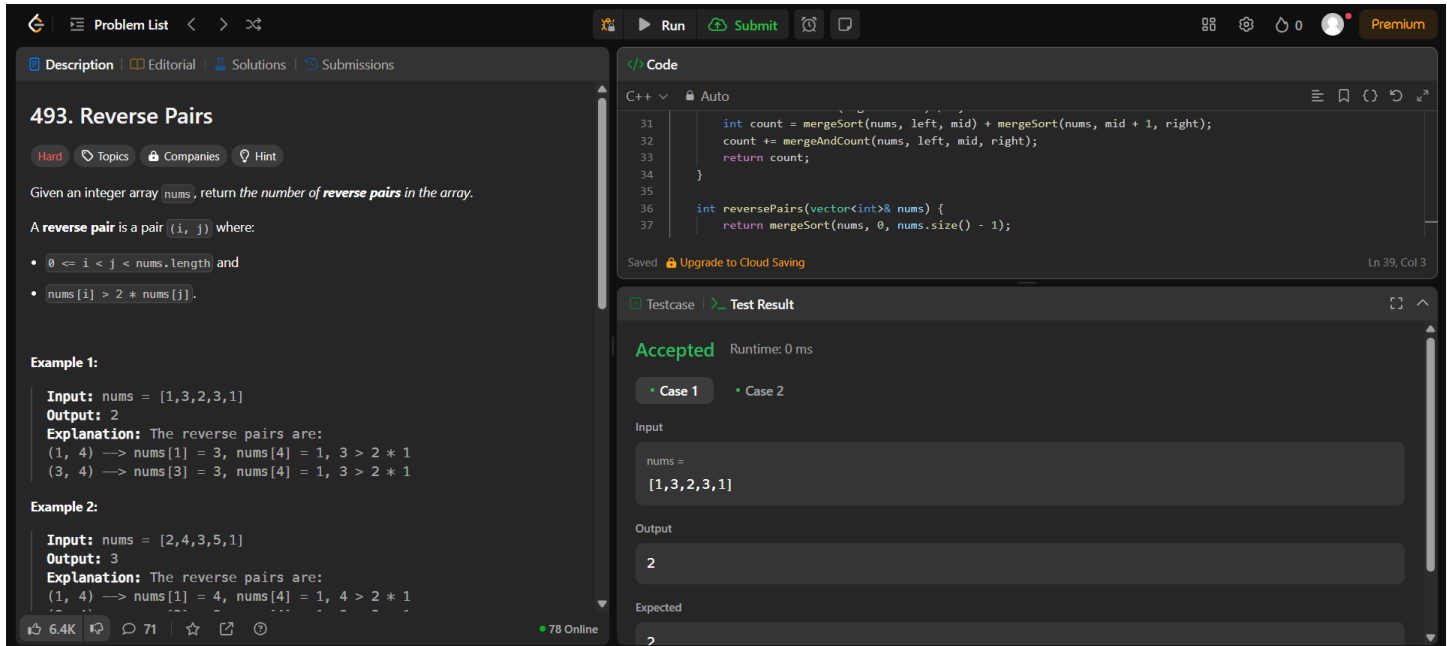
// Merge step
vector<int> temp;
int i = left, k = mid + 1;
while (i <= mid && k <= right) {
    if (nums[i] <= nums[k]) temp.push_back(nums[i++]);
    else temp.push_back(nums[k++]);
}
while (i <= mid) temp.push_back(nums[i++]);
while (k <= right) temp.push_back(nums[k++]);

// Copy back sorted elements
for (int i = left; i <= right; i++) nums[i] = temp[i - left];

return count;
}

int mergeSort(vector<int>& nums, int left, int right) {
    if (left >= right) return 0;
    int mid = left + (right - left) / 2;
    int count = mergeSort(nums, left, mid) + mergeSort(nums, mid + 1, right);
    count += mergeAndCount(nums, left, mid, right);
    return count;
}

int reversePairs(vector<int>& nums) {
    return mergeSort(nums, 0, nums.size() - 1);
}
};
```



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:

- $0 \leq i < j < \text{nums.length}$ and
- $\text{nums}[i] > 2 * \text{nums}[j]$.

Example 1:

Input: `nums = [1,3,2,3,1]`
Output: 2
Explanation: The reverse pairs are:
 $(1, 4) \rightarrow \text{nums}[1] = 3, \text{nums}[4] = 1, 3 > 2 * 1$
 $(3, 4) \rightarrow \text{nums}[3] = 3, \text{nums}[4] = 1, 3 > 2 * 1$

Example 2:

Input: `nums = [2,4,3,5,1]`
Output: 3
Explanation: The reverse pairs are:
 $(1, 4) \rightarrow \text{nums}[1] = 4, \text{nums}[4] = 1, 4 > 2 * 1$
 $(2, 4) \rightarrow \text{nums}[2] = 3, \text{nums}[4] = 1, 3 > 2 * 1$
 $(3, 4) \rightarrow \text{nums}[3] = 5, \text{nums}[4] = 1, 5 > 2 * 1$

Code:

```
C++
int count = mergeSort(nums, left, mid) + mergeSort(nums, mid + 1, right);
count += mergeAndCount(nums, left, mid, right);
return count;

int reversePairs(vector<int>& nums) {
    return mergeSort(nums, 0, nums.size() - 1);
}
```

Testcase: Accepted Runtime: 0 ms

Case 1:

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

Output: 2

Expected: 2

Question 10: 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.

Solution:

```
class Solution {
public:
    int lengthOfLIS(vector<int>& nums, int k) {
        int maxVal = *max_element(nums.begin(), nums.end());
        vector<int> segTree(maxVal + 1, 0);

        auto query = [&](int l, int r) {
            int res = 0;
            while (r > 0) {
                res = max(res, segTree[r]);
                r -= (r & -r);
            }
            return res;
        };

        return query(0, maxVal);
    }
};
```



DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

```
auto update = [&](int idx, int val) {  
    while (idx <= maxVal) {  
        segTree[idx] = max(segTree[idx], val);  
        idx += (idx & -idx);  
    }  
};
```

```
int res = 0;  
for (int num : nums) {  
    int best = query(max(1, num - k), num - 1);  
    int newLength = best + 1;  
    update(num, newLength);  
    res = max(res, newLength);  
}  
return res;  
}  
};
```

Description | Editorial | Solutions | Submissions

Hard | Topics | Companies | Hint

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.

Example 2:

Input: `nums = [7,4,5,1,8,12,4,7]`, `k = 5`

917 | 26 | 11 Online

Code

```
C++ | Auto
```

```
1 class Solution {  
2 public:  
3     int lengthOfLIS(vector<int>& nums, int k) {  
4         int maxVal = *max_element(nums.begin(), nums.end());  
5         vector<int> segTree(maxVal + 1, 0);  
6  
7         // Function to query the max value in the range [1, r]  
8         auto query = [&](int l, int r) {
```

Saved | Ln 7, Col 5

Testcase | **Test Result**

Input

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

`k =`
`3`

Output

`6`

Expected