

WORKSHEET

Student Name: Ishu

UID: 22BCS15695

Branch: CSE

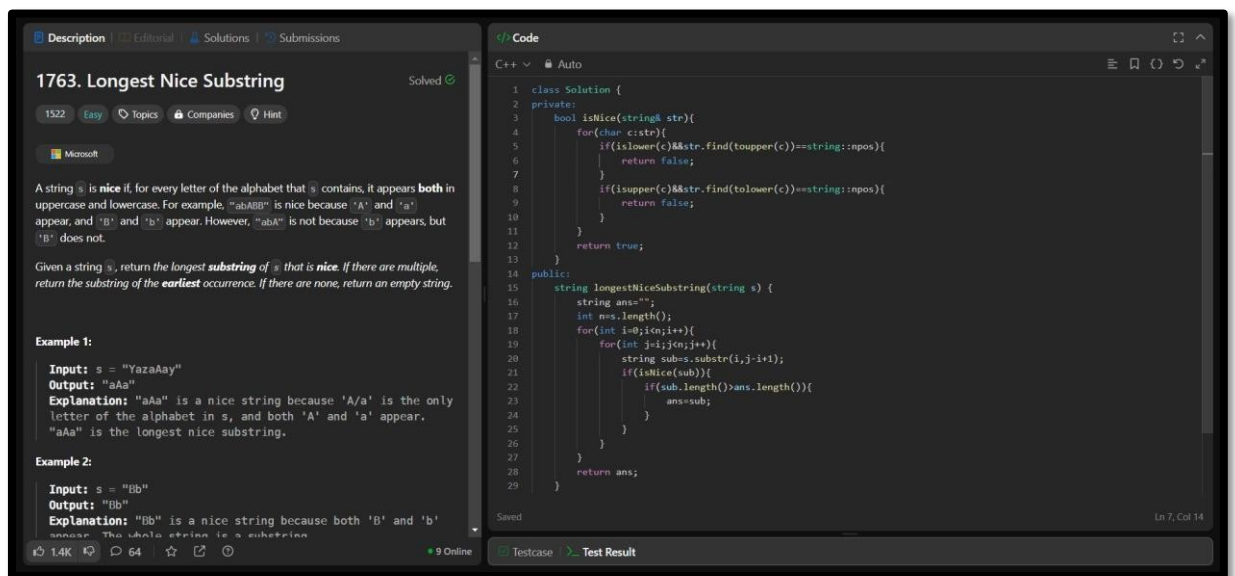
Section/Group: 609-'B'

Semester: 6th

Date of Submission: 26/07/24

Subject Name: AP LAB

1. Longest Nice Substring:



1763. Longest Nice Substring Solved

1522 Easy Topics Companies Hint

Microsoft

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' appear. The whole string is a substring.

```
1 class Solution {
2 private:
3     bool isNice(string& str){
4         for(char c:str){
5             if((islower(c)&&str.find(toupper(c))==string::npos){
6                 return false;
7             }
8             if((isupper(c)&&str.find(tolower(c))==string::npos){
9                 return false;
10            }
11        }
12        return true;
13    }
14 public:
15     string longestNiceSubstring(string s) {
16         string ans="";
17         int n=s.length();
18         for(int i=0;i<n;i++){
19             for(int j=i+1;j<n;j++){
20                 string sub=s.substr(i,j-i+1);
21                 if(isNice(sub)){
22                     if(sub.length()>ans.length()){
23                         ans=sub;
24                     }
25                 }
26             }
27         }
28         return ans;
29     }
30 }
```

```
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(islice(sub)){
```

```
if(sub.length()>ans.length())
```

```
ans=sub;}
```

```
}
```

```
}
```

```
}
```

```
return ans;
```

```
}
```

2. Reverse Bits

The screenshot displays the LeetCode interface for the problem "190. Reverse Bits". The problem description on the left states: "Reverse bits of a given 32 bits unsigned integer." It includes a note about signed vs. unsigned integers and an example where the input is 43261596 (binary 000000101001010000011101001100) and the output is 964176192 (binary 001110010111000001010010000000).

The right side of the image shows a C++ solution in a code editor. The code defines a class `Solution` with a method `reverseBits` that takes an integer `n` and returns its bit-reversed value. The implementation uses a loop to swap bits from the outside in.

Below the code, the "Test Result" section shows "Accepted" with a runtime of 2 ms. It displays the input `n = 000000101001010000011101001100` and the output `964176192 (001110010111000001010010000000)`.

```

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;
        }
    }
}

```

3. Number of 1 Bits:

The screenshot shows the LeetCode interface for the problem "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*)." Examples provided are: Example 1: Input $n = 11$, Output 3; Example 2: Input $n = 128$, Output 1. The C++ code in the editor is as follows:

```

1 class Solution {
2 public:
3     int hammingWeight(uint32_t n) {
4         int ans = 0;
5
6         for (int i = 0; i < 32; ++i)
7             if ((n >> i) & 1)
8                 ++ans;
9
10        return ans;
11    }
12 };

```

The test results show "Accepted" with a runtime of 0 ms. The test case input is $n = 11$ and the output is 3.

```

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;
    }
};

```

4. Maximum Subarray:

The screenshot displays a coding interface for the 'Maximum Subarray' problem. On the left, the problem description is shown, including the title '53. Maximum Subarray', difficulty 'Medium', and a 'Solved' status. The description asks to find the subarray with the largest sum. Three examples are provided: Example 1 with input [-2,1,-3,4,-1,2,1,-5,4] and output 6; Example 2 with input [1] and output 1; and Example 3 with input [5,4,-1,7,8] and output 23. On the right, the C++ code is shown in a dark-themed editor. The code defines a class 'Solution' with a public method 'maxSubArray' that takes a vector of integers and returns the maximum sum. The implementation uses a loop to calculate the sum of the current subarray and updates the maximum sum if the current sum is greater. The 'Test Result' section shows the code is 'Accepted' with a runtime of 0 ms. The input field contains the same array as Example 1, and the output field shows the result 6.

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.

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

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

```
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;
            }
        }
    }
}
```

5. Search a 2D Matrix II:

The screenshot shows the LeetCode problem page for "240. Search a 2D Matrix II" and a C++ solution. The problem description states: "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

The C++ solution is as follows:

```
1 class Solution {
2 public:
3     bool searchMatrix(vector<vector<int>>& matrix, int target) {
4         int row = matrix.size();
5         int col = matrix[0].size();
6
7         int rowIndex = 0;
8         int colIndex = col-1;
9
10        while(rowIndex < row && colIndex >= 0){
11            int element = matrix[rowIndex][colIndex];
12
13            if(element == target){
14                return true;
15            }
16        }
17        return false;
18    }
19 }
```

The test result shows "Accepted" with a runtime of 2 ms. The input matrix is `[[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]]` and the target is `5`.

```
class Solution {
public: bool searchMatrix(vector<vector<int>>& matrix, int target) {
    for (int i = 0; i < matrix.size(); i++) {
        for (int j = 0; j < matrix[i].size(); j++) {
            if (matrix[i][j] == target) {
                return true;
            }
        }
    }
    return false;
}
```

6. Super Pow:

The screenshot shows the LeetCode interface for problem 372, "Super Pow". The problem description states: "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." Examples provided are: 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: $1 \leq a < 2^{31}$, $1 \leq b.length \leq 2000$, $0 \leq b[i] < 10$. The code editor shows a C++ solution using a recursive function `modPow` to calculate the power modulo 1337.

```
12  
13  
14 private:  
15     static constexpr int kMod = 1337;  
16  
17     long modPow(long x, long n) {  
18         if (n == 0)  
19             return 1;  
20         if (n % 2 == 1)  
21             return x * modPow(x % kMod, (n - 1) / 2) % kMod;  
22         return modPow(x * x % kMod, n / 2) % kMod;  
23     };
```

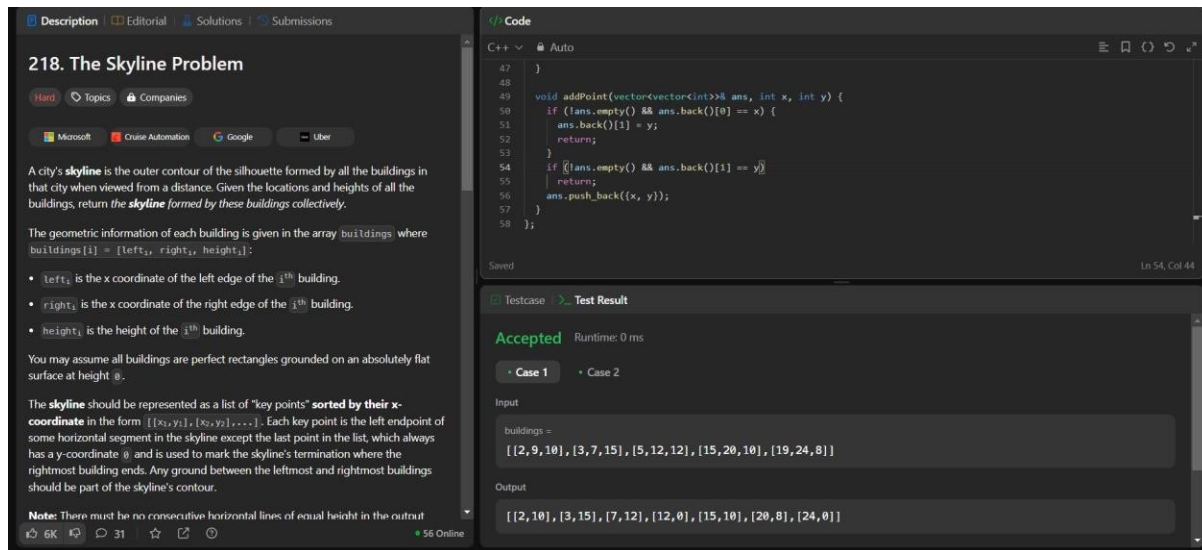
```
class Solution {  
    private: int solve(int base, int power, int mod) {  
        int ans = 1; while (power > 0) {  
            if (power & 1) { ans = (ans * base) % mod; }  
            base = (base * base) % mod; power >>= 1; }  
        return ans; }  
    public: int superPow(int a, vector<int>& b) {  
        a%=1337; int n = b.size();  
        int m = 1140; int expi = 0; for(int i : b){ expi = (expi*10+i)%m; }  
        if (expi == 0) { expi = m;  
        }  
        return solve(a,expi,1337); } };
```

7. Beautiful Array:

The screenshot shows the LeetCode interface for problem 932, 'Beautiful Array'. The problem description states that an array `nums` of length `n` is beautiful if it is a permutation of integers in the range `[1, n]` and for every $0 \leq i < j < n$, there is no index `k` with $j < k < i$ where $2 * nums[k] == nums[i] + nums[j]$. The goal is to return any beautiful array of length `n`. Examples provided are: Example 1: Input `n = 4`, Output `[2,1,4,3]`; Example 2: Input `n = 5`, Output `[3,1,2,5,4]`. Constraints include `1 <= n <= 1000`. The C++ code on the right implements a recursive divide-and-conquer solution, partitioning the array into two halves and ensuring the beautiful property is maintained.

```
class Solution {
public: int partition(vector<int> &v, int start, int end, int mask) {
    int j = start; for(int i = start; i <= end; i++) {
        if((v[i] & mask) != 0) { swap(v[i], v[j]); j++; } }
    return j; }
    void sort(vector<int> & v, int start, int end, int mask) {
        if(start >= end) return;
        int mid = partition(v, start, end, mask);
        sort(v, start, mid - 1, mask << 1); sort(v, mid, end, mask << 1); }
    vector<int> beautifulArray(int N) {
        vector<int> ans;
        for(int i = 0; i < N; i++) ans.push_back(i + 1);
        sort(ans, 0, N - 1, 1); return ans; } };
};
```

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] = [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.
- `heighti` 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.

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]]`

```

47 }
48
49 void addPoint(vector<vector<int>>& ans, int x, int y) {
50     if (!ans.empty() && ans.back()[0] == x) {
51         ans.back()[1] = y;
52         return;
53     }
54     if (!ans.empty() && ans.back()[1] == y)
55         return;
56     ans.push_back({x, y});
57 }
58

```

Testcase: **Accepted** Runtime: 0 ms

Case 1 Case 2

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]]`

```

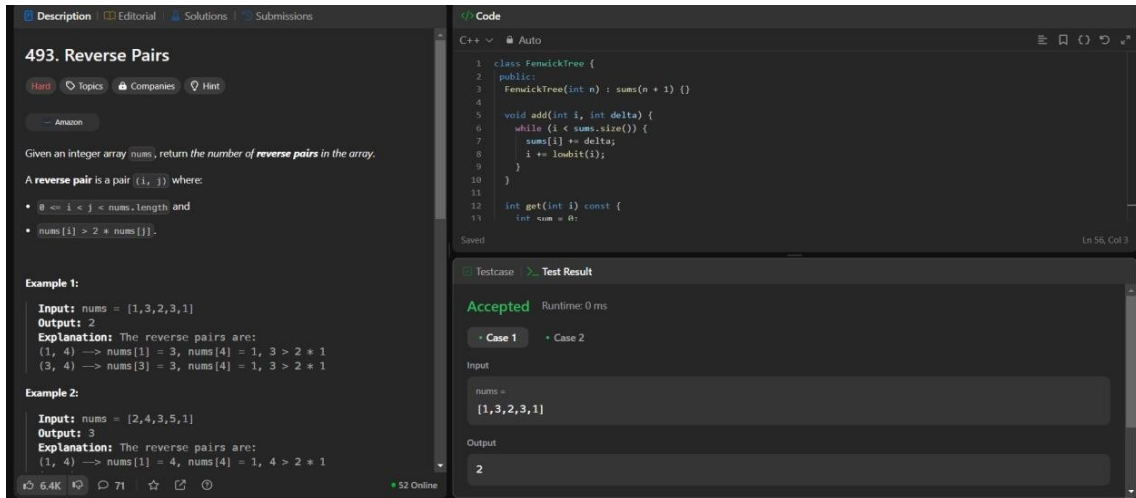
class Solution {
public: vector<vector<int>> getSkyline(vector<vector<int>>& buildings) {
    int edge_idx = 0;
    vector<pair<int, int>> edges;
    priority_queue<pair<int, int>> pq;
    vector<vector<int>> skyline;
    for (int i = 0; i < buildings.size(); ++i) {
        const auto &b = buildings[i];
        edges.emplace_back(b[0], i);
        edges.emplace_back(b[1], i);
        std::sort(edges.begin(), edges.end());
        while (edge_idx < edges.size()) {
            int curr_height; const auto &[curr_x, _] = edges[edge_idx];

```



```
while (edge_idx < edges.size() && curr_x == edges[edge_idx].first) {
    const auto &[_ , building_idx] = edges[edge_idx];
    const auto &b = buildings[building_idx];
    if (b[0] == curr_x) pq.emplace(b[2], b[1]); ++edge_idx; }
    while (!pq.empty() && pq.top().second <= curr_x) pq.pop();
    curr_height = pq.empty() ? 0 : pq.top().first; if (skyline.empty() ||
    skyline.back()[1] != curr_height) skyline.push_back({curr_x, curr_height}); }
    return skyline; } };
```

9. Reverse Pairs:



493. Reverse Pairs

Hard Topics Companies Hint

Amazon

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:
 $(1, 4) \rightarrow \text{nums}[1] = 4, \text{nums}[4] = 1, 4 > 2 * 1$

6.4K 71 52 Online

Code

```
C++
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;
    }
};
```

Testcase Test Result

Accepted Runtime: 0 ms

Case 1 Case 2

Input

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

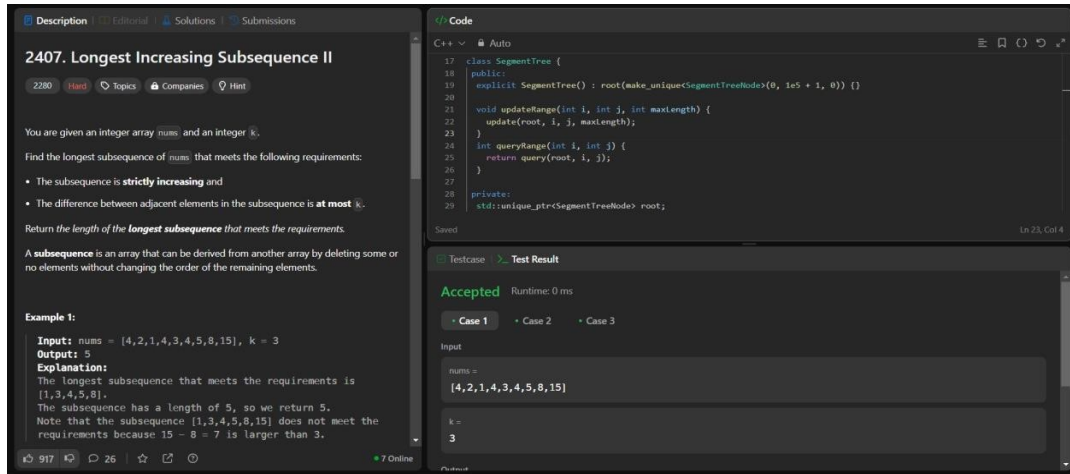
Output

`2`

```
class Solution {
private:
    void merge(vector<int>& nums, int low, int mid, int high, int&
reversePairsCount){
        int j = mid+1;
        for(int i=low; i<=mid; i++){
            while(j<=high && nums[i] > 2*(long long)nums[j]){
```

```
j++; }  
reversePairsCount += j-(mid+1);}  
int size = high-low+1;  
vector<int> temp(size, 0);  
int left = low, right = mid+1, k=0;  
while(left<=mid && right<=high){  
    if(nums[left] < nums[right]){  
        temp[k++] = nums[left++];}  
    else {temp[k++] = nums[right++]; }  
    while(left<=mid){temp[k++] = nums[left++]; }  
    while(right<=high){temp[k++] = nums[right++]; }  
    int m=0;  
    for(int i=low; i<=high; i++){  
        nums[i] = temp[m++];    } }  
void mergeSort(vector<int>& nums, int low, int high, int& reversePairsCount){  
    if(low >= high){  
        return;    }  
    int mid = (low + high) >> 1;  
    mergeSort(nums, low, mid, reversePairsCount);  
    mergeSort(nums, mid+1, high, reversePairsCount);  
    merge(nums, low, mid, high, reversePairsCount); }  
public: int reversePairs(vector<int>& nums) {  
    int reversePairsCount = 0;  
    mergeSort(nums, 0, nums.size()-1, reversePairsCount);  
    return reversePairsCount; } };
```

10. Longest Increasing Subsequence II:



2407. Longest Increasing Subsequence II

2280 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.

Accepted Runtime: 0 ms

Case 1 Case 2 Case 3

Input

`nums =`

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

`k =`

`3`

```
class MaxSegmentTree {
public: int n; vector<int> tree;
MaxSegmentTree(int n_) : n(n_) {
int size = (int)(ceil(log2(n))); size = (2 * pow(2, size)) - 1; tree =
vector<int>(size); }
int max_value() { return tree[0]; }
int query(int l, int r) { return query_util(0, l, r, 0, n - 1); }
int query_util(int i, int qL, int qR, int l, int r) {
if (l >= qL && r <= qR) return tree[i];
if (l > qR || r < qL) return INT_MIN; int m = (l + r) / 2;
return max(query_util(2 * i + 1, qL, qR, l, m), query_util(2 * i + 2, qL, qR, m + 1,
r)); }
void update(int i, int val) {
update_util(0, 0, n - 1, i, val); }
void update_util(int i, int l, int r, int pos, int val) {
if (pos < l || pos > r) return; if (l == r) {
tree[i] = max(val, tree[i]); return; } int m = (l + r) / 2;
update_util(2 * i + 1, l, m, pos, val);
update_util(2 * i + 2, m + 1, r, pos, val); tree[i] = max(tree[2 * i + 1], tree[2 * i +
2]);}
};
class Solution {
```



DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

```
public: int lengthOfLIS(vector<int>& nums, int k) {  
    MaxSegmentTree tree(1e5 + 1);  
    for (int i : nums) { int lower = max(0, i - k);  
        int cur = 1 + tree.query(lower, i - 1);  
        tree.update(i, cur); }  
    return tree.max_value(); } };
```