ASSIGNMENT 3

Student Name: Varun Sharma **UID: 22BCS15992**
Branch: B.E-C.S.E **Section/Group: 606-B**
Semester: 6th **Date of Performance: 17/02/25**
Subject Name: Advanced Programming Lab-2 **Subject Code: 22CSP-351**

1. 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.

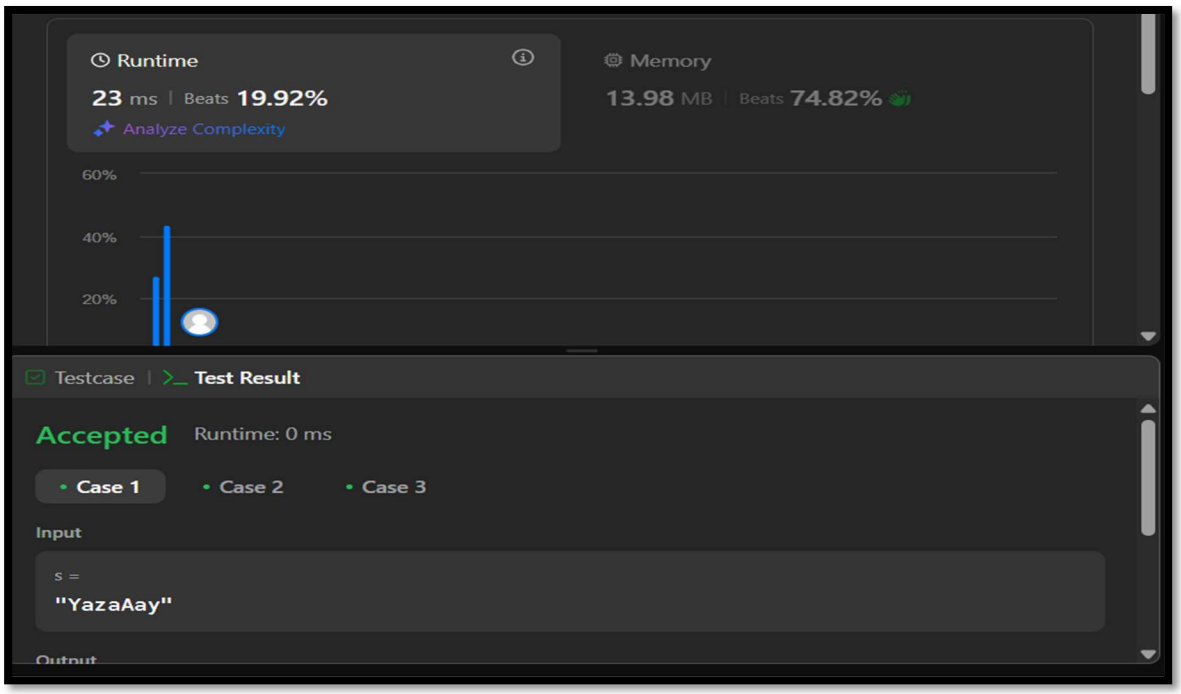
Code Snippet

```
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;

   }
};
```

Submission



2. **Reverse Bits**

Reverse bits of a given 32 bits unsigned integer.

Note:

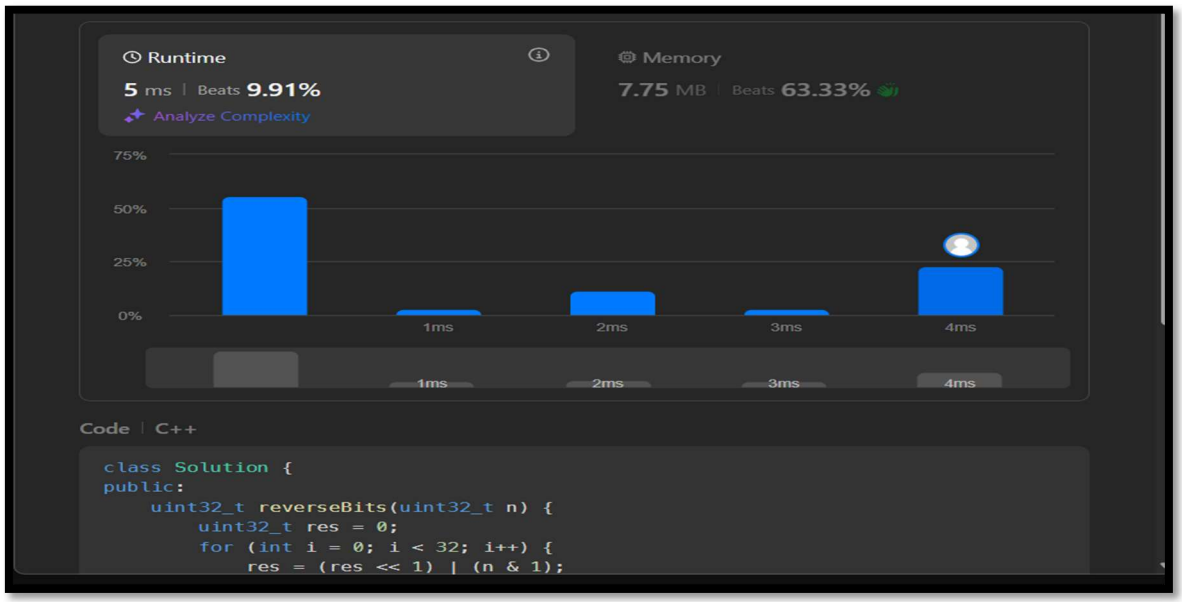
Note that in some languages, such as Java, there is no unsigned integer type. In this case, both input and output will be given as a signed integer type. They should not affect your implementation, as the integer's internal binary representation is the same, whether it is signed or unsigned.

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.

Code Snippet

```
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);
            n >>= 1;
        }
        return res;
    }
};
```

Submission



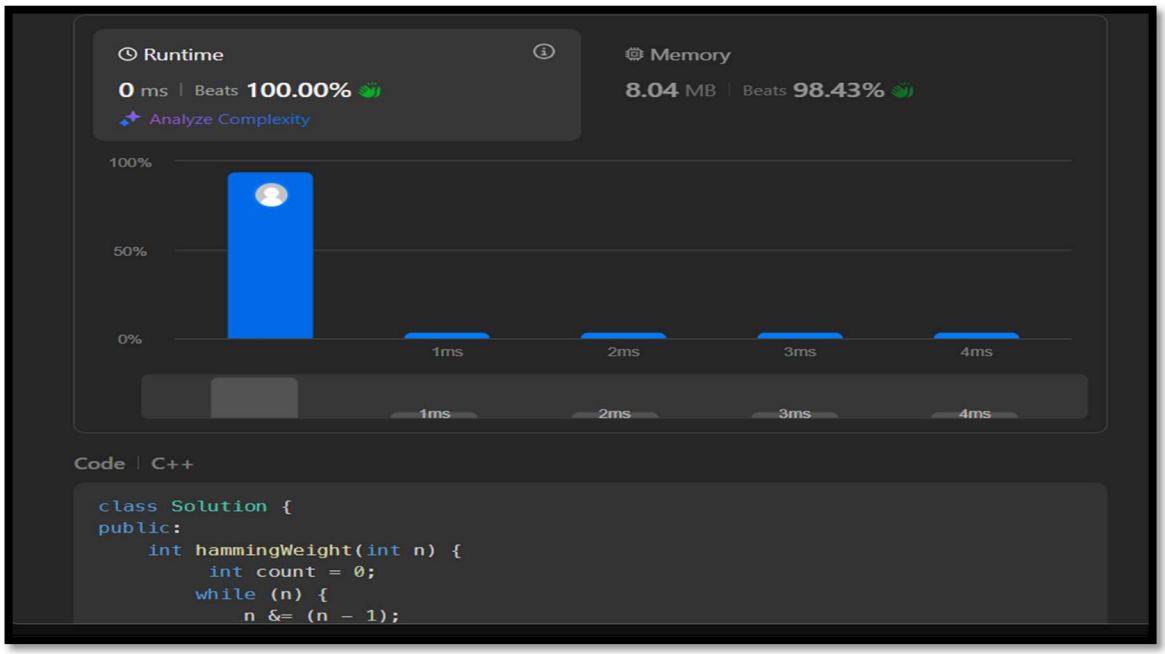
3. Number of 1 Bits

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).

Code Snippet

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

Submissions



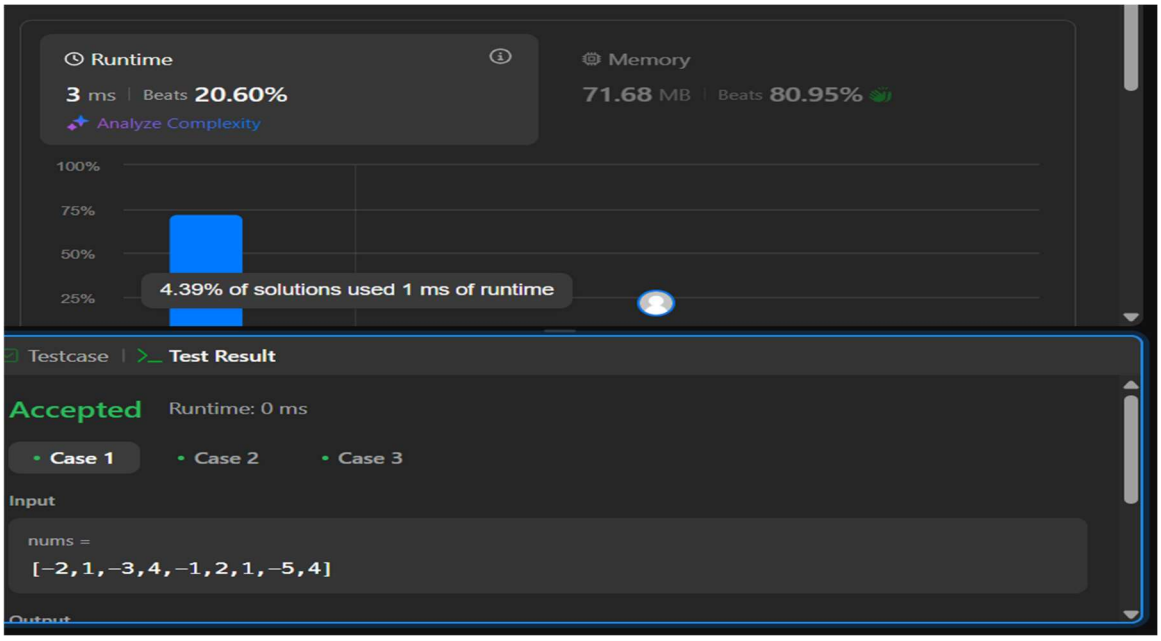
4. **Maximum Subarray**

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

Code Snippet

```
class Solution {
public:
    int maxSubArray(vector<int>& nums) {
        int sum=0;
        int maxi=INT_MIN;
        for(int i=0;i<nums.size();i++){
            sum+=nums[i];
            maxi=max(sum,maxi);
            if(sum<0){
                sum=0;
            }
        }
        return maxi;
    }
};
```

Submission



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:

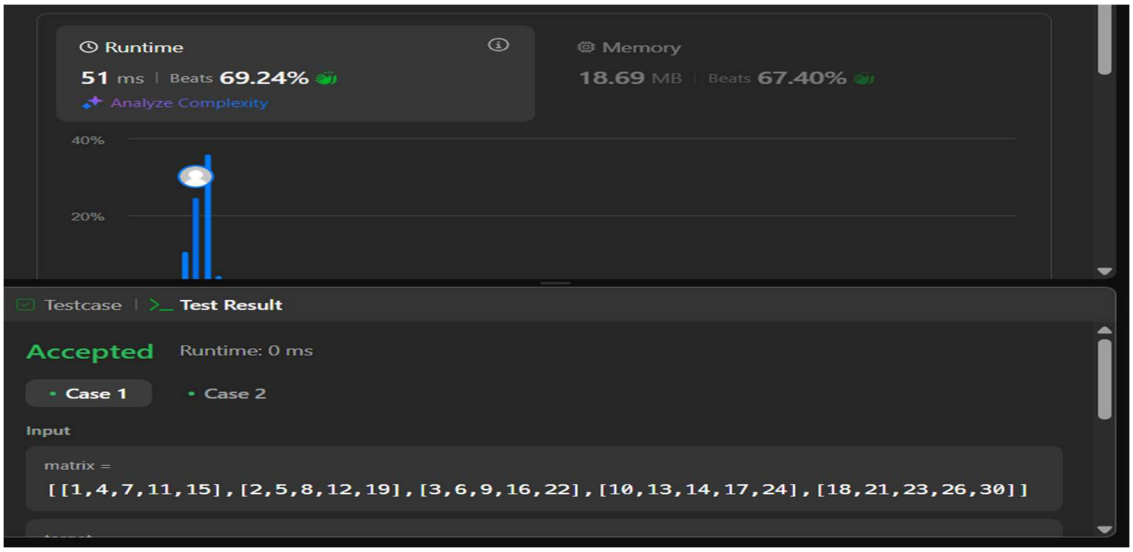
Integers in each row are sorted in ascending from left to right.

Integers in each column are sorted in ascending from top to bottom.

Code Snippet

```
class Solution {
public:
   bool searchMatrix(vector<vector<int>>& matrix, int target) {
       int n = matrix.size();
   int m = matrix[0].size();
   int row = 0, col = m - 1;
   while (row < n && col >= 0) {
      if (matrix[row][col] == target) return true;
      else if (matrix[row][col] < target) row++;
      else col--;
   }
   return false;
   }
};
```

Submission



6. **Super Pow**

Your task is to calculate ab mod 1337 where a is a positive integer and b is an extremely large positive integer given in the form of an array.

Code Snippet

```
class Solution {
public:
    static constexpr int MOD = 1337;
    int modPow(int a, int exp) {
        int res = 1;
        a %= MOD;
        while (exp) {
            if (exp % 2) res = (res * a) % MOD;
            a = (a * a) % MOD;
            exp /= 2;
        }
        return res;
    }
    int superPow(int a, vector<int>& b) {
        int res = 1;
        for (int digit : b) {
            res = modPow(res, 10) * modPow(a, digit) % MOD;
        }
        return res;
    }
};
```

Submission

7. Beautiful Array

An array nums of length n is beautiful if:

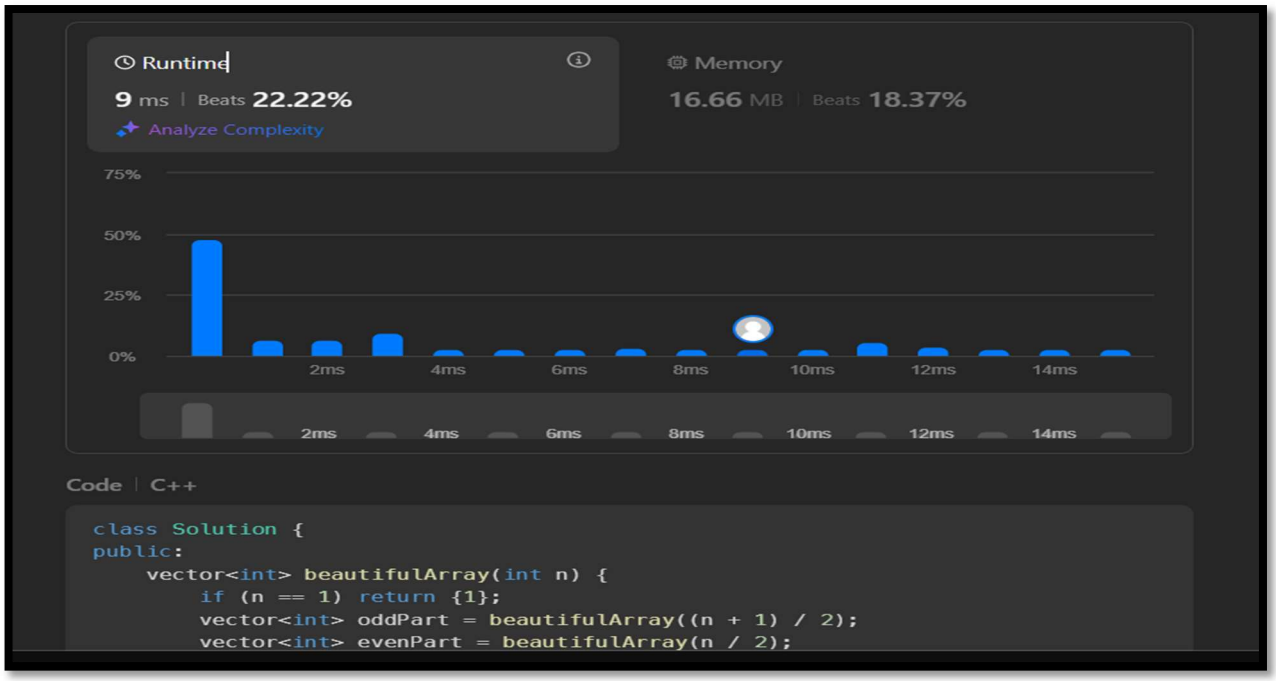
- nums is a permutation of the integers in the range [1, n].
- For every 0 <= i < j < n, there is no index k with i < k < j where 2 * nums[k] == nums[i] + nums[j].

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

Code Snippet

```
class Solution {
public:
    vector<int> beautifulArray(int n) {
        if (n == 1) return {1};
        vector<int> oddPart = beautifulArray((n + 1) / 2);
        vector<int> evenPart = beautifulArray(n / 2);
        vector<int> result;
        for (int x : oddPart) result.push_back(2 * x - 1);
        for (int x : evenPart) result.push_back(2 * x);
        return result;
    }
};
```

Submission



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 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.

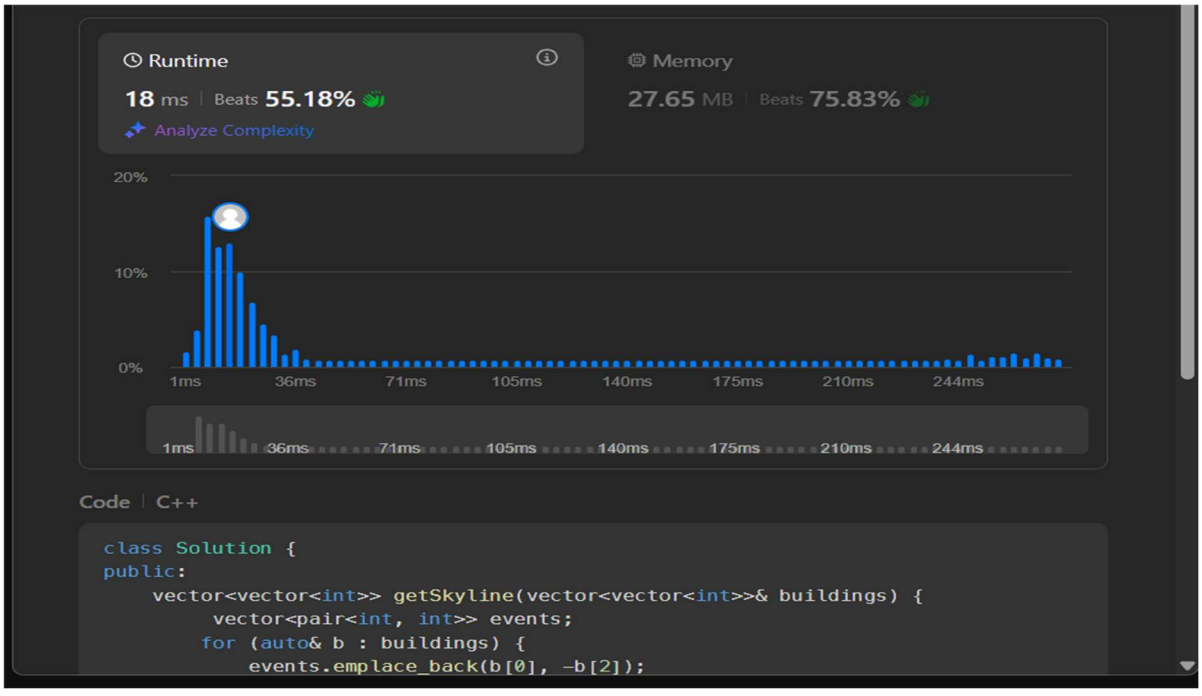
Note: There must be no consecutive horizontal lines of equal height in the output skyline. For instance, [...,[2 3],[4 5],[7 5],[11 5],[12 7],...] is not acceptable; the three lines of height 5 should be merged into one in the final output as such: [...,[2 3],[4 5],[12 7],...]

Code Snippet

```
class Solution {
public:
    vector<vector<int>> getSkyline(vector<vector<int>>& buildings) {
```

```
        vector<pair<int, int>> events;
        for (auto& b : buildings) {
            events.emplace_back(b[0], -b[2]);
            events.emplace_back(b[1], b[2]);
        }
        sort(events.begin(), events.end(), [](const pair<int, int>& a, const pair<int, int>& b) {
            if (a.first == b.first) return a.second < b.second;
            return a.first < b.first;
        });
        multiset<int> heights = {0};
        vector<vector<int>> skyline;
        int prevMaxHeight = 0;
        for (auto& [x, h] : events) {
            if (h < 0) heights.insert(-h);
            else heights.erase(heights.find(h));
            int curMaxHeight = *heights.rbegin();
            if (curMaxHeight != prevMaxHeight) {
                skyline.push_back({x, curMaxHeight});
                prevMaxHeight = curMaxHeight;
            }
        }
        return skyline;
    }
};
```

Submission



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

Code Snippet

```
class Solution {
public:
    int mergeAndCount(vector<int>& nums, int left, int mid, int right) {
        int count = 0, j = mid + 1;
        for (int i = left; i <= mid; i++) {
            while (j <= right && (long long)nums[i] > 2LL * nums[j]) j++;
            count += (j - (mid + 1));
        }
        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++]);

        for (int i = left; i <= right; i++) nums[i] = temp[i - left];
        return count;
    }
    int mergeSortAndCount(vector<int>& nums, int left, int right) {
        if (left >= right) return 0;
        int mid = left + (right - left) / 2;
        int count = mergeSortAndCount(nums, left, mid) +
                    mergeSortAndCount(nums, mid + 1, right) +
                    mergeAndCount(nums, left, mid, right);
        return count;
    }
    int reversePairs(vector<int>& nums) {
        return mergeSortAndCount(nums, 0, nums.size() - 1);
    }
};
```

Submission

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.

Code Snippet

```
class Solution {
public:
   class SegmentTree {
   vector<int> tree;
   int size;
public:
   SegmentTree(int n) {
      size = n + 1;
      tree.resize(2 * size, 0);
   }
   void update(int index, int value) {
      index += size;
      tree[index] = value;
      for (index /= 2; index > 0; index /= 2)
         tree[index] = max(tree[2 * index], tree[2 * index + 1]);
   }
   int query(int left, int right) {
      int res = 0;
      left += size, right += size;
      while (left <= right) {
         if (left % 2 == 1) res = max(res, tree[left++]);
         if (right % 2 == 0) res = max(res, tree[right--]);
         left /= 2, right /= 2;
```

```
        }
        return res;
    }
};
public:
    int lengthOfLIS(vector<int>& nums, int k) {
        int maxVal = *max_element(nums.begin(), nums.end());
        SegmentTree segTree(maxVal);

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

Submission