

## AP ASSIGNMENT – 4

Name – Deepak Sharma

UID – 22BCS13187

Section –IOT\_606-B

1. Longest Nice Substring: <https://leetcode.com/problems/longest-nice-substring/description/>

CODE:

```
class Solution {
```

```
public:
```

```
    string longestNiceSubstring(string s) {
```

```
        int n = s.size();
```

```
        string longest = "";
```

```
        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() > longest.length()) {
```

```
                        longest = sub;
```

```
                    }
```

```
                }
```

```
            }
```

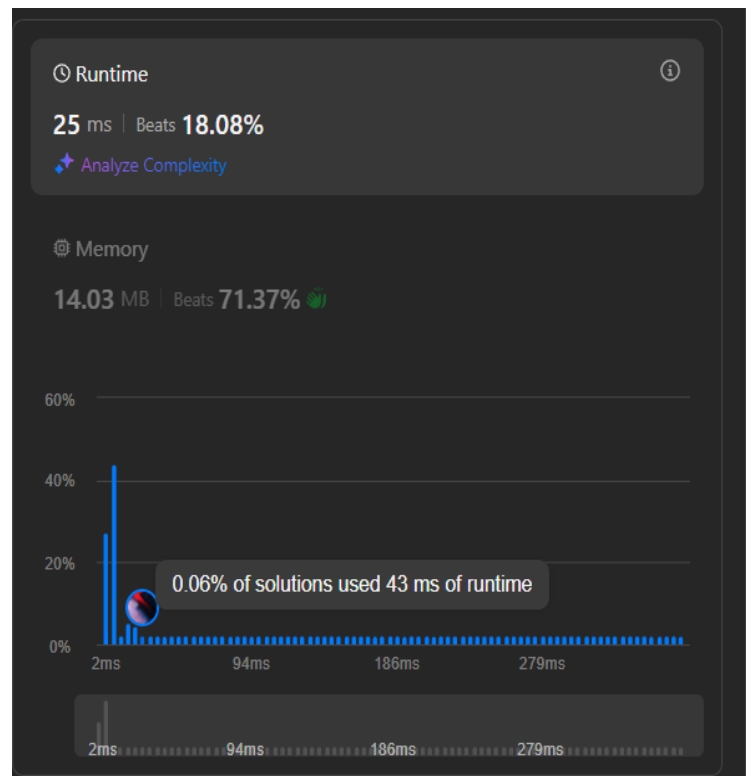
```
        }
```

```
        return longest;
```

```
}
```

private:

```
bool isNice(const string& s) {  
    int lower = 0, upper = 0;  
    for (char c : s) {  
        if (islower(c)) {  
            lower |= (1 << (c -  
'a'));  
        } else {  
            upper |= (1 << (c -  
'A'));  
        }  
    }  
    return lower == upper;  
}  
};
```



## 2. Reverse Bits: <https://leetcode.com/problems/reverse-bits/description/>

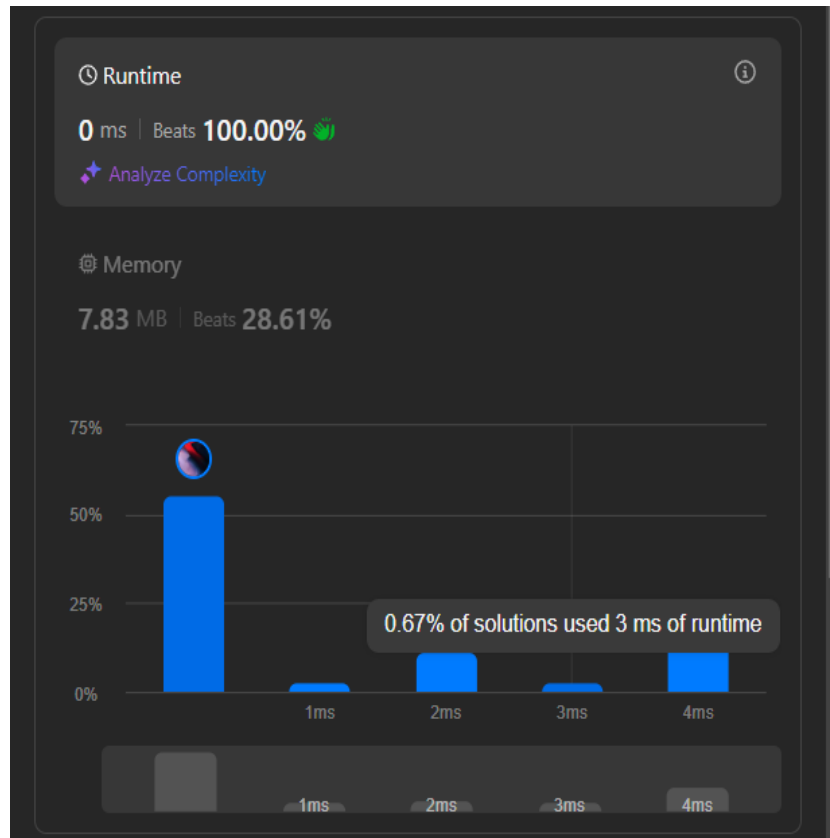
CODE:

```
class Solution {  
public:  
    uint32_t reverseBits(uint32_t n) {  
        uint32_t result = 0;  
        for (int i = 0; i < 32; i++) {
```

```

        result = (result << 1) | (n & 1);
        n >>= 1;
    }
    return result;
}
};

```



### 3. Number of 1 Bits: <https://leetcode.com/problems/number-of-1-bits/description/>

CODE:

```

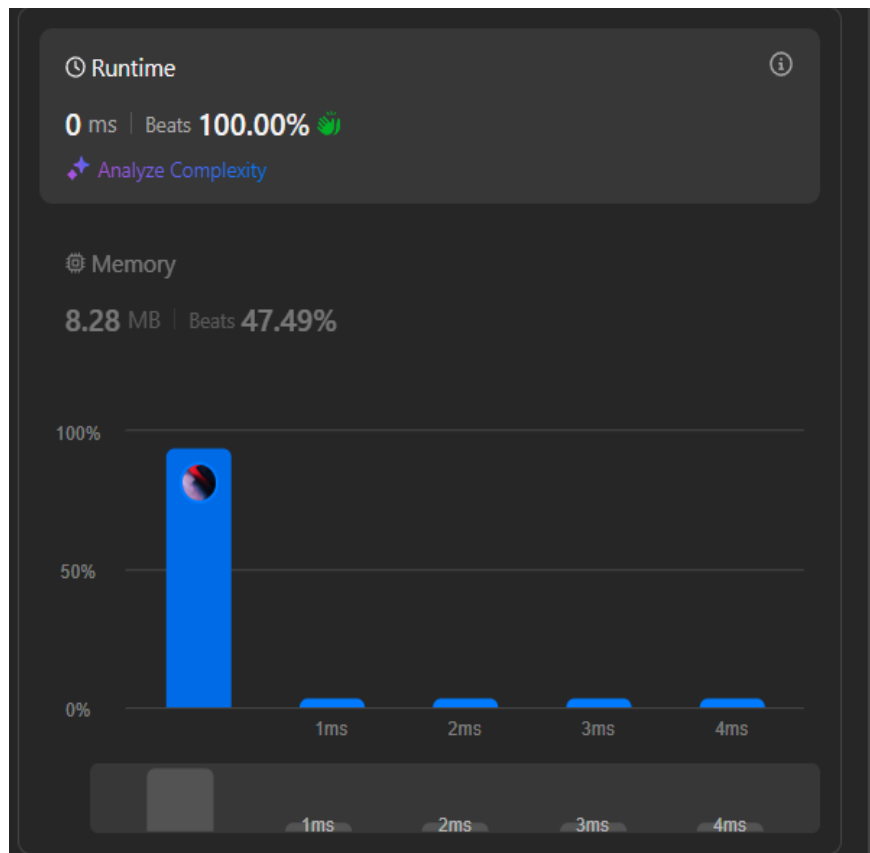
class Solution {
public:
    int hammingWeight(int n) {
        int count = 0;
        while (n) {
            count += (n & 1);

```

```

        n >>= 1;
    }
    return count;
}
};

```



#### 4. Maximum Subarray: <https://leetcode.com/problems/maximum-subarray/description/>

CODE:

```

class Solution {
public:
    int maxSubArray(vector<int>& nums) {
        int res=nums[0];
        int maxend=nums[0];

        for(int i=1;i<nums.size();i++){

```

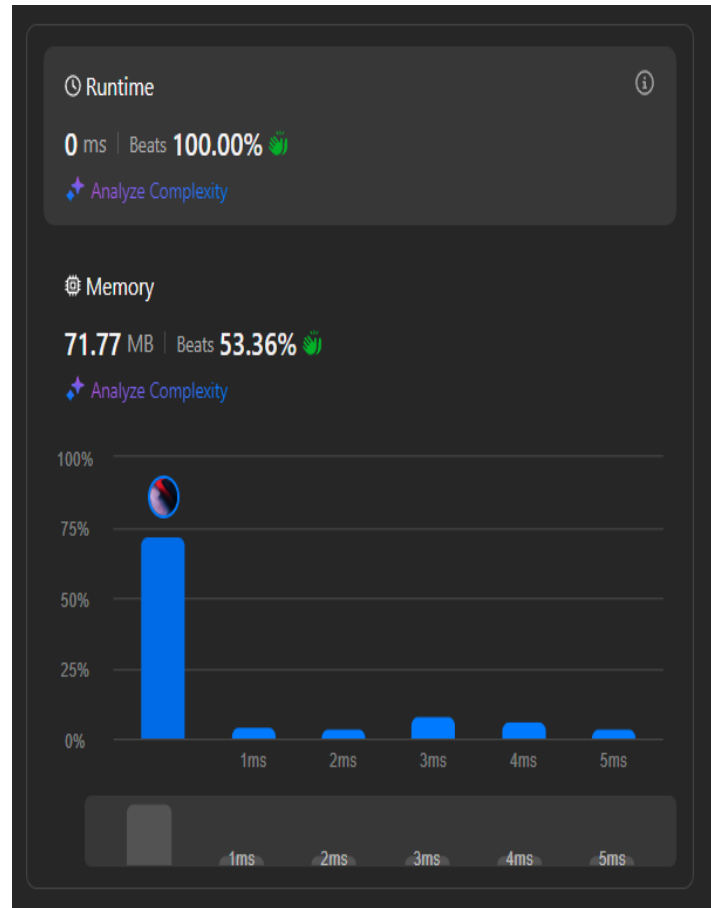
```

        maxend=max(maxend+nums[i],nums[i]);

        res=max(res,maxend);
    }

    return res;
}
};

```



5. Search a 2D Matrix II: <https://leetcode.com/problems/search-a-2d-matrix-ii/description/>

CODE:

```

class Solution {
public:
    bool searchMatrix(vector<vector<int>>& matrix, int target) {
        if (matrix.empty() || matrix[0].empty()) return false;
    }
}

```

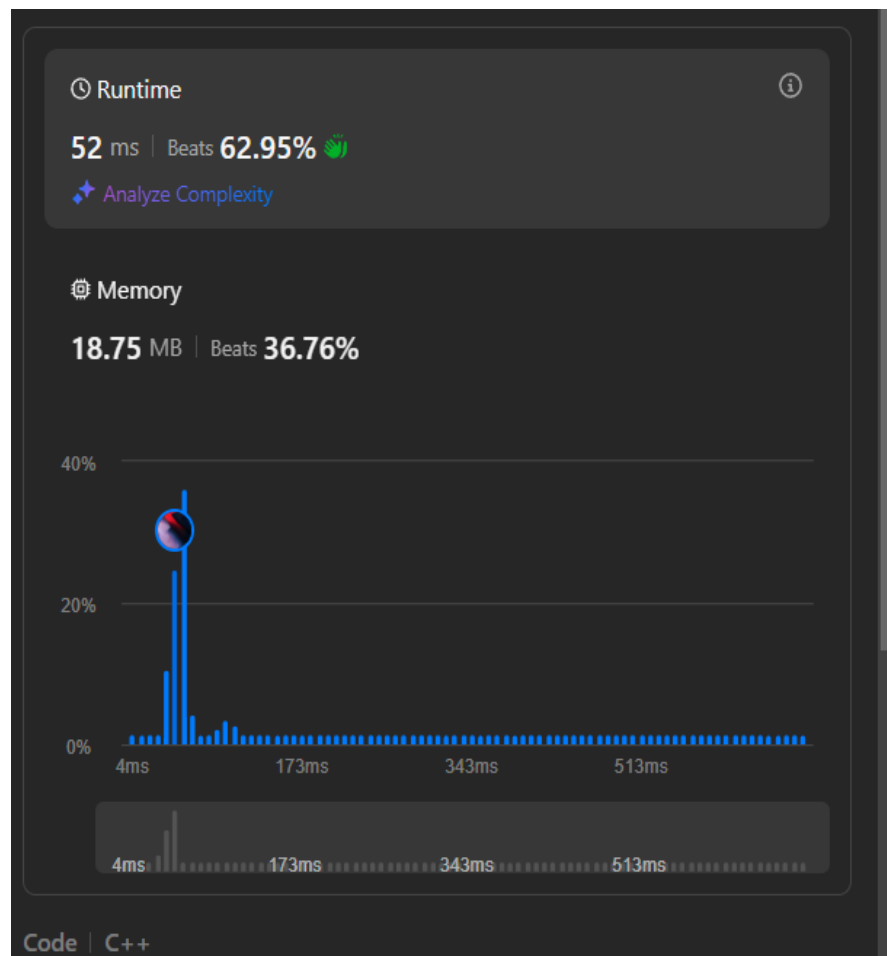
```
int rows = matrix.size();
int cols = matrix[0].size();

int row = 0, col = cols - 1;

while (row < rows && col >= 0) {
    if (matrix[row][col] == target) return true;
    else if (matrix[row][col] > target) col--;
    else row++;
}

return false;
}
```

```
};
```



6. Super Pow: <https://leetcode.com/problems/super-pow/description/>

CODE:

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

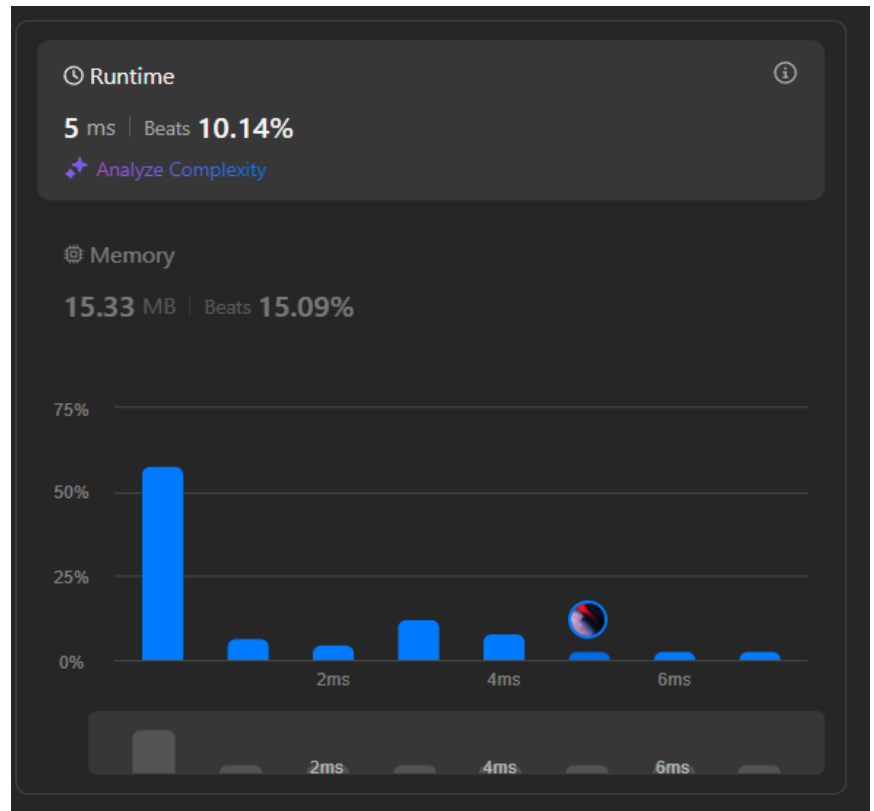
    int modPow(int x, int n) {
        int res = 1;
        x %= MOD;
        while (n) {
            if (n % 2) res = (res * x) % MOD;
            x = (x * x) % MOD;
            n /= 2;
        }
        return res;
    }

    int superPow(int a, vector<int>& b) {
        int result = 1;
        for (int digit : b) {
            result = modPow(result, 10) * modPow(a, digit) %
MOD;
        }
    }
}
```

```

        return result;
    }
};

```



## 7. Beautiful Array: <https://leetcode.com/problems/beautiful-array/description/>

CODE:

```

class Solution {
public:
    vector<int> beautifulArray(int n) {
        vector<int> result = {1};

        while (result.size() < n) {
            vector<int> temp;
            for (int num : result) {

```

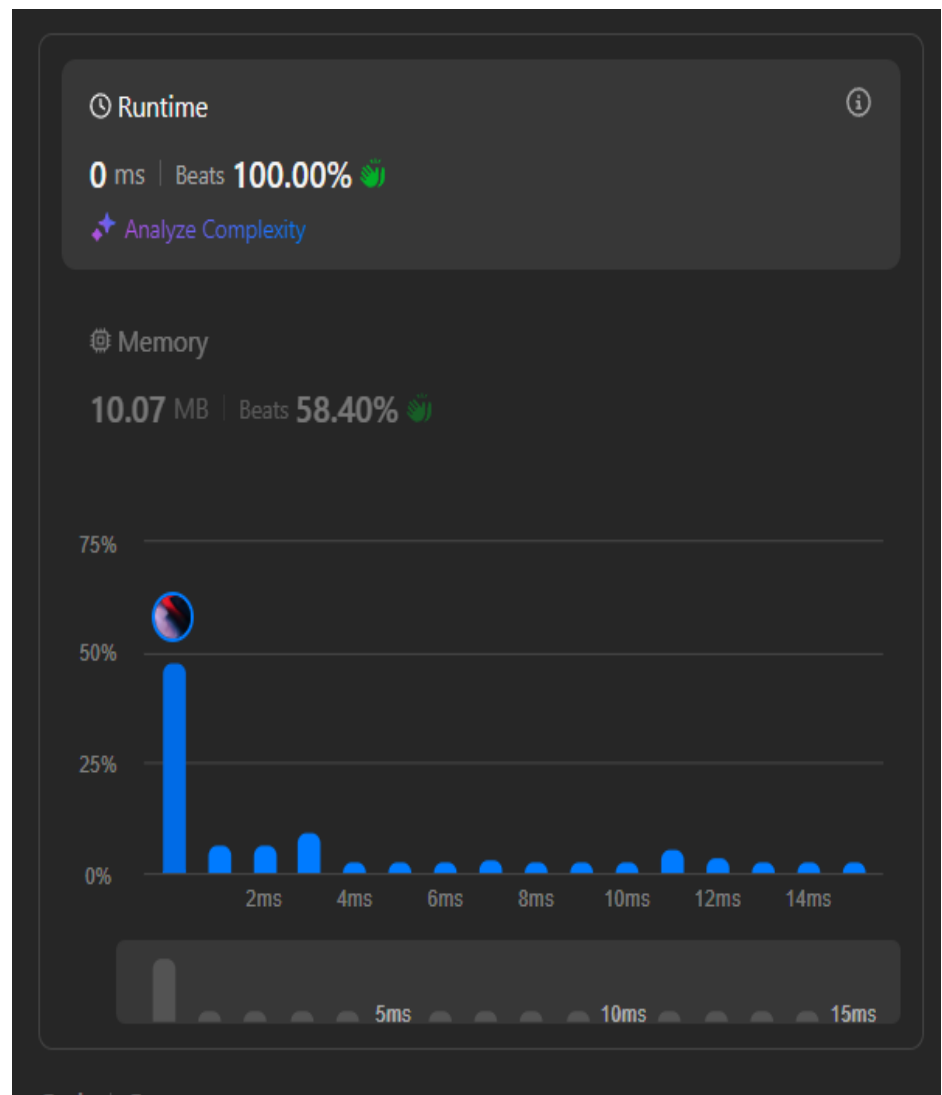


```

        if (2 * num - 1 <= n) temp.push_back(2 * num - 1); //
Odd numbers
    }
    for (int num : result) {
        if (2 * num <= n) temp.push_back(2 * num); // Even
numbers
    }
    result = temp;
}

return
result;
}
};

```



8. The Skyline Problem: <https://leetcode.com/problems/the-skyline-problem/description/>

CODE:

```
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
        for (auto& b : buildings) {
            events.emplace_back(b[0], -b[2]); // Left edge (negative
height for entering)
            events.emplace_back(b[1], b[2]); // Right edge (positive
height for leaving)
        }

        // Step 2: Sort events
        sort(events.begin(), events.end());

        // Step 3: Process events using max heap
        multiset<int> heights = {0}; // Initial ground height
        int prevMax = 0;
```

```

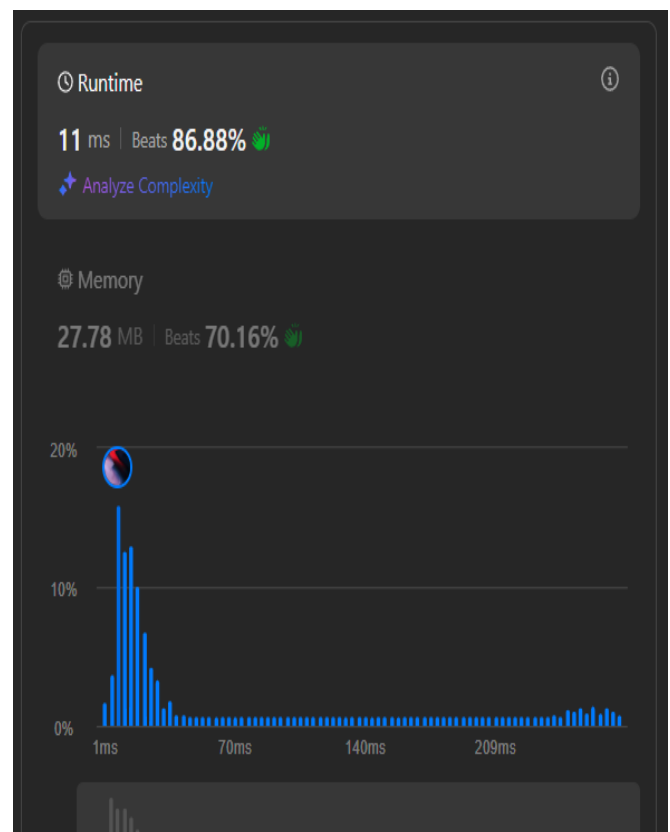
for (auto& [x, h] : events) {
    if (h < 0) {
        heights.insert(-h); // Entering event, add height
    } else {
        heights.erase(heights.find(h)); // Leaving event,
remove height
    }

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

    if (currMax != prevMax) { // If height changed, add key
point
        result.push_back({x, currMax});
        prevMax = currMax;
    }
}

return result;
}
};

```



9. Reverse Pairs: <https://leetcode.com/problems/reverse-pairs/description/>

CODE:

```
class Solution {
public:
    int mergeAndCount(vector<int>& nums, int left, int mid, int
right) {
        int count = 0;
        int 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 sorted elements back
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;
```

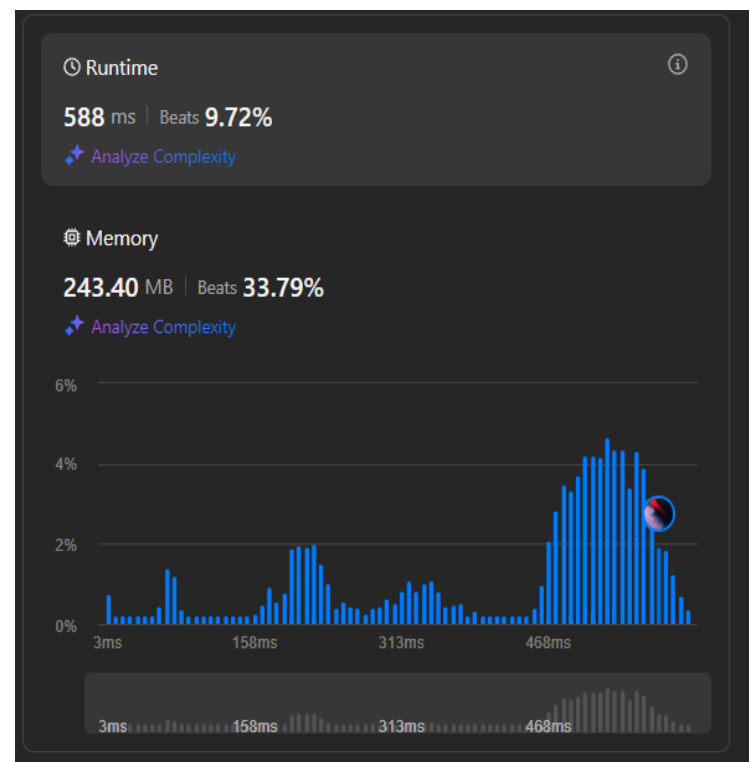
```
    int count = mergeSortAndCount(nums, left, mid) +
        mergeSortAndCount(nums, mid + 1, right) +
        mergeAndCount(nums, left, mid, right);
```

```

        return count;
    }

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

```



## 10. Longest Increasing Subsequence

II: <https://leetcode.com/problems/longest-increasing-subsequence-ii/description/>

CODE:

```

class Solution {
public:
    class SegmentTree {
    private:
        vector<int> tree;

```

```
int n;
```

```
void update(int idx, int l, int r, int pos, int value) {  
    if (l == r) {  
        tree[idx] = value;  
        return;  
    }  
    int mid = (l + r) / 2;  
    if (pos <= mid) {  
        update(2 * idx + 1, l, mid, pos, value);  
    } else {  
        update(2 * idx + 2, mid + 1, r, pos, value);  
    }  
    tree[idx] = max(tree[2 * idx + 1], tree[2 * idx + 2]);  
}
```

```
int query(int idx, int l, int r, int ql, int qr) {  
    if (ql > r || qr < l) return 0;  
    if (ql <= l && r <= qr) return tree[idx];  
    int mid = (l + r) / 2;  
    return max(query(2 * idx + 1, l, mid, ql, qr), query(2 *  
idx + 2, mid + 1, r, ql, qr));  
}
```

```
public:
```

```

SegmentTree(int size) {
    n = size;
    tree.resize(4 * size, 0);
}

void update(int pos, int value) {
    update(0, 0, n - 1, pos, value);
}

int query(int left, int right) {
    if (left > right) return 0;
    return query(0, 0, n - 1, left, right);
}

};

int lengthOfLIS(vector<int>& nums, int k) {
    int maxValue = *max_element(nums.begin(), nums.end());
    SegmentTree segTree(maxValue + 1);

    int maxLIS = 0;
    for (int num : nums) {
        int bestPrev = segTree.query(max(0, num - k), num - 1);
        int currLIS = bestPrev + 1;
        segTree.update(num, currLIS);
        maxLIS = max(maxLIS, currLIS);
    }
}

```



```
    }  
  
    return maxLIS;  
}  
};
```

