Editorial for Fenwick Tree - Haircut Problem

Description of Problem

1. Functionality

   We are given an array A of length N, with $0 \leq A[i] \leq N$. For each j from 0 to N−1, we define A' by cutting all values in A greater than j down to exactly j and we want the number of inversions ($m < n$ and $A'[m] > A'[n]'$) in A′.

   For example, A=[5,2,3,3,0], when j=3, for each number > 3 in A, we cut it to 3 and add it to A'. So we have A'=[3,2,3,3,0]. The number of inversions in A' is 5.

2. Implementation Requirement
   (a) It is not allowed to use brute force comparison. But the brute force comparison could be used to verify test results.
   (b) It is required to use Fenwick Tree for quick update and query

3. Input and Output
   (a) Input:
      - The first line is an integer N.
      - The second line contains $A_1, A_2, \dots, A_N$.

   (b) Output:
      - For each of $j = 0, 1, \dots N - 1$, output the number of inversions on a new line

Solution of Problem

You can first notice that as j increases, the answer is nondecreasing. When we move from j to j+1, we only allow elements to become larger, never smaller, so existing inversions stay and new inversions may appear. So instead of recomputing the answer from scratch for each j, we can think about how the answer changes when j increases by 1.

Consider a fixed inversion in the original array, a pair (i, k) with i < k and A[i] > A[k]. For which j does this pair contribute as an inversion after cutting?

Let x = A[i] and y = A[k], with x > y. For a given j:

if j < y, then both values are cut to j, so they are equal, not an inversion
if j = y, then both become y, still not an inversion
if j > y, then Ak stays y, and Ai becomes min(x, j) ≥ y+1, so the pair is an inversion

So this inversion starts to appear exactly when j becomes y+1, and then stays forever for all larger j. That means:

answer[0] = 0
For j ≥ 1, answer[j] = answer[j−1] + (number of original inversions whose right element has value j−1)

Define cnt[v] = number of pairs (i, k) with i < k, A[i] > A[k], and A[k] = v. *If v is not in the array, cnt[v]=0*

Then the output is:

ans[0] = 0
ans[1] = ans[0] + cnt[0]
ans[2] = ans[1] + cnt[1]
...
ans[j] = ans[j−1] + cnt[j−1]

So the core task is to compute cnt[v] for all v from 0 to N−1.

We can compute these counts by sweeping values in increasing order and using a Fenwick tree over positions to count, for each index i with A[i] = v, how many earlier elements are greater than v. After counting, we also update the Fenwick tree at position i to mark A[i] = v as processed, so that this occurrence of v will be treated as a ‘non-greater’ element in all future queries.

Conceptually, this problem is easiest to think about in terms of prefix sums. For each position, we want to know how many earlier elements satisfy some condition. A Fenwick tree is just a data structure that lets us maintain these prefix sums dynamically in O(log N) per update and query, so we keep the same prefix-sum idea but in an efficient form.

Complexity: the Fenwick operations are O(N log N), and memory is O(N).

Implementation of Solution

The solution is implemented in C++. The file is haircut.cpp under the directory bit_problems.

Test Cases

The testfile is called haircut_testcases.