Documentation

Problem Statement

The problem requires reconstructing a queue based on the given attributes of people. Each person is represented as [h, k], where h is the height, and k is the number of people in front who have a height greater than or equal to h. Based on these constraints, the goal is to return the queue in the correct order.

Intuition

The problem can be efficiently solved by sorting and placing each person in the correct position. Since taller people affect shorter ones' placement, it is beneficial to process taller individuals first and insert them into the queue accordingly.

Key Observations

Sorting by height in descending order ensures that inserting individuals does not affect the placement of taller individuals already positioned. Sorting by k in ascending order ensures that individuals are placed at the right index without disrupting others.

Approach

The algorithm sorts the people array by height in descending order and by k in ascending order for individuals with the same height. Then, each person is inserted into the result list at the index specified by k. This ensures that all conditions are satisfied efficiently.

Edge Cases

The solution must handle cases where all people have the same height but different k values, cases where k is zero for all individuals, and cases where the input has only one person.

Complexity Analysis

The time complexity is O(n log n) due to sorting, followed by $O(n^2)$ for inserting elements at specific positions in the list. The space complexity is O(n) for storing the reconstructed queue.

Alternative Approaches

One alternative approach is using a segment tree or binary indexed tree to track available positions, but this would add unnecessary complexity for most cases.

Test Cases

Several test cases should be considered, including minimal input cases, cases with increasing k values, cases where all heights are different, and cases where k values are randomly assigned.

Final Thoughts

This problem demonstrates the power of sorting combined with greedy insertion. Understanding how the sorting order influences placement is crucial in solving such problems efficiently. The approach balances simplicity and efficiency, making it an optimal solution for real-world scenarios.