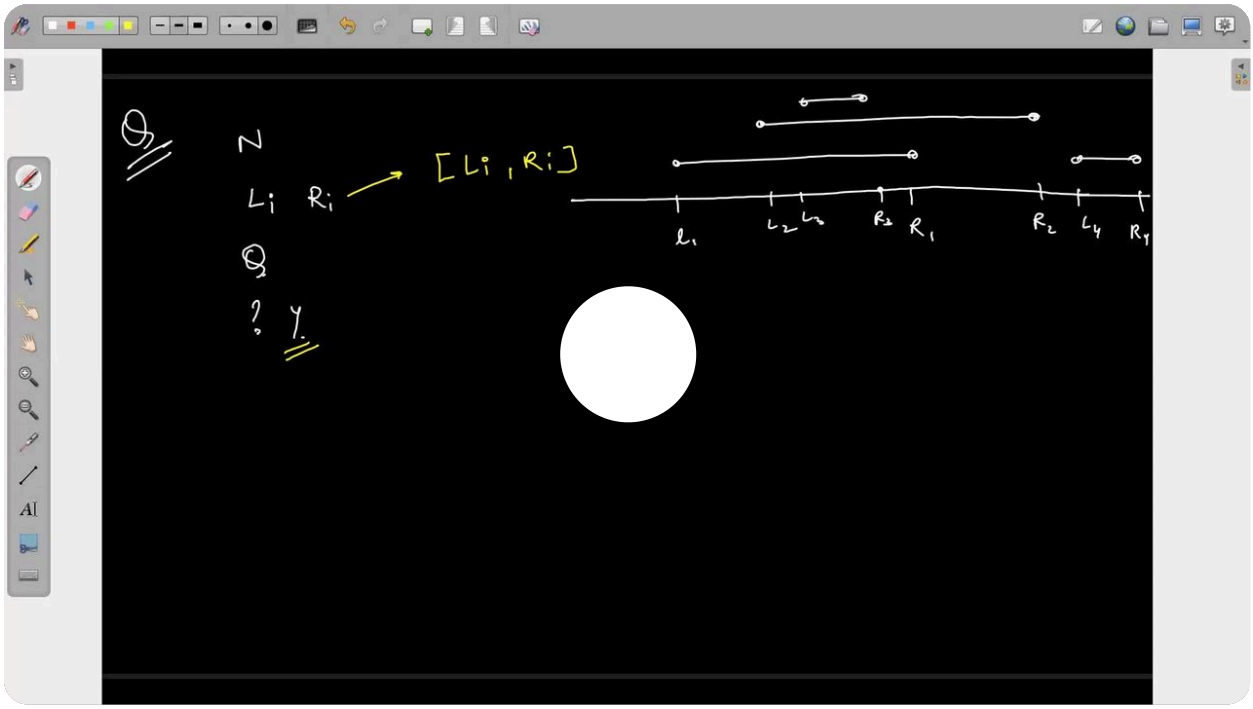
Range Maintenance Ideas - 1



Problem Statement:

We are given N ranges varies from Li to Ri on number line. Then you are given Q queries in form of ? Y, which means you have to return how many ranges passes through this points. Constraints N, Q <=1e5 Li, Ri, Y <=1e9

Approach:

To efficiently solve the problem of counting the number of ranges that pass through a given point Y, we can use a binary search-based approach. We'll preprocess the given ranges to sort them based on their left and right endpoints. Then, for each query point Y, we'll use binary search to find the number of ranges that pass through Y.

Here's the detailed approach:

1. Sort the given ranges based on their left endpoints.
2. For each query point Y:
    - Perform a binary search to find the number of ranges whose right endpoint is less than Y (i.e., ranges ending before Y). This can be done using the lower_bound function in C++ or

the bisect_left function in Python.
    - Perform a binary search to find the number of ranges whose left endpoint is greater than Y (i.e., ranges starting after Y). This can be done using the upper_bound function in C++ or the bisect_right function in Python.
3. The total number of ranges passing through Y is given by: Total ranges - (number of ranges ending before Y + number of ranges starting after Y).

Code:

Copy

```cpp
#include <bits/stdc++.h>
using namespace std;

int count_ranges( vector<int> &starts, vector<int>& ends, int Y) {
    // Find the number of ranges that start at or before Y
    int start_count = starts.size() - (upper_bound(starts.begin(), starts.end(), Y) - starts.begin());

    // Find the number of ranges that ends before Y
    int end_count = (upper_bound(ends.begin(), ends.end(), Y) - ends.begin());

    // The number of ranges passing through Y is the overlap
    return (starts.size() - (start_count + end_count));
}

int main() {
    vector<pair<int, int>> ranges = {{1, 5}, {3, 7}, {6, 9}, {8, 11}};
    vector<int> queries = {3, 6, 8};


    vector<int> starts, ends;
    for (const auto& range : ranges) {
        starts.push_back(range.first);
        ends.push_back(range.second);
    }
    sort(starts.begin(), starts.end());
    sort(ends.begin(), ends.end());

    for (int Y : queries) {
        cout << count_ranges(starts, ends, Y) << endl;
    }
}
```

Time Complexity:

This implementation efficiently counts the number of ranges passing through each query point Y. The time complexity for each query is O(log N), where N is the number of ranges, due to the binary search operations.
