**Leetcode 3394: Check if Grid can be Cut into Sections**

**Problem Statement**

You are given an integer n representing the dimensions of an n x n grid, with the origin at the bottom-left corner of the grid. You are also given a 2D array of coordinates rectangles, where rectangles[i] is in the form [startx, starty, endx, endy], representing a rectangle on the grid. Each rectangle is defined as follows:

* (startx, starty): The bottom-left corner of the rectangle.
* (endx, endy): The top-right corner of the rectangle.

Note that the rectangles **do not overlap**. Your task is to determine if it is possible to make either two horizontal or two vertical cuts on the grid such that:

1. Each of the three resulting sections formed by the cuts contains at least one rectangle.
2. Every rectangle belongs to exactly one section.

Return true if such cuts can be made; otherwise, return false.

**Example 1:**

**Input:**

n = 5

rectangles = [[1,0,5,2],[0,2,2,4],[3,2,5,3],[0,4,4,5]]

**Output:**

true

**Explanation:**  
We can make horizontal cuts at y = 2 and y = 4.

**Example 2:**

**Input:**

n = 4

rectangles = [[0,0,1,1],[2,0,3,4],[0,2,2,3],[3,0,4,3]]

**Output:**

true

**Explanation:**  
We can make vertical cuts at x = 2 and x = 3.

**Example 3:**

**Input:**

n = 4

rectangles = [[0,2,2,4],[1,0,3,2],[2,2,3,4],[3,0,4,2],[3,2,4,4]]

**Output:**

false

**Explanation:**  
We cannot make two horizontal or two vertical cuts that satisfy the conditions.

**Solution Approach**

To solve this problem efficiently, we need to check if we can make two valid cuts in either the **horizontal** or **vertical** direction. The idea is:

1. Extract x-coordinates and y-coordinates from the given rectangles.
2. Sort and analyze intervals to determine if at least **three sections** can be formed.
3. If valid cuts exist in either direction, return true.

**C++ Implementation**

#include <vector>

#include <algorithm>

using namespace std;

class Solution {

public:

// Function to check if valid cuts can be made

bool checkValidCuts(int n, vector<vector<int>>& rectangles) {

vector<pair<int, int>> verticals; // Stores y-coordinates for vertical cuts

vector<pair<int, int>> horizontals; // Stores x-coordinates for horizontal cuts

// Extract x and y intervals from rectangles

for (auto& r : rectangles) {

horizontals.push\_back({r[0], r[2]}); // Store x-range of each rectangle

verticals.push\_back({r[1], r[3]}); // Store y-range of each rectangle

}

// Check if valid vertical or horizontal cuts exist

return check(verticals) || check(horizontals);

}

private:

// Function to check if we can make two cuts that create three valid sections

bool check(vector<pair<int, int>>& intervals) {

ranges::sort(intervals); // Sort intervals by starting coordinate

int r = 0, res = 0;

// Iterate through intervals to determine if at least three sections exist

for (auto interval : intervals) {

if (interval.first >= r) { // New section starts

res++;

}

r = max(r, interval.second); // Update the max right boundary

}

return res >= 3; // There should be at least three sections

}

};

**Explanation of Code**

1. **Extracting Intervals:**
   * We create two lists, one for horizontal and one for vertical cuts.
   * Each rectangle contributes an (x1, x2) and (y1, y2) pair.
2. **Sorting the Intervals:**
   * We sort the intervals based on their starting coordinate.
3. **Checking Valid Cuts:**
   * We iterate through the sorted intervals to find at least three separate sections.
   * If we find two valid cut points, we return true, otherwise false.

**Time Complexity Analysis**

* **Sorting the intervals**: O(m log m) where m = rectangles.size().
* **Iterating through the intervals**: O(m).
* **Overall complexity**: O(m log m), which is efficient for large inputs.

**Conclusion**

This approach efficiently determines whether we can divide the grid into three sections using two horizontal or vertical cuts. If either direction satisfies the condition, we return true; otherwise, we return false.