There are some spherical balloons taped onto a flat wall that represents the XY-plane. The balloons are represented as a 2D integer array points where points[i] = [xstart, xend] denotes a balloon whose horizontal diameter stretches between xstart and xend. You do not know the exact y-coordinates of the balloons.Arrows can be shot up directly vertically (in the positive y-direction) from different points along the x-axis. A balloon with xstart and xend is burst by an arrow shot at x if xstart <= x <= xend. There is no limit to the number of arrows that can be shot. A shot arrow keeps traveling up infinitely, bursting any balloons in its path.Given the array points, return the minimum number of arrows that must be shot to burst all balloons.Example 1:Input: points = [[10,16],[2,8],[1,6],[7,12]]Output: 2Explanation: The balloons can be burst by 2 arrows:- Shoot an arrow at x = 6, bursting the balloons [2,8] and [1,6].- Shoot an arrow at x = 11, bursting the balloons [10,16] and [7,12].Example 2:Input: points = [[1,2],[3,4],[5,6],[7,8]]Output: 4Explanation: One arrow needs to be shot for each balloon for a total of 4 arrows.Example 3:constraint :

1 <= points.length <= 105points[i].length == 2-231 <= xstart < xend <= 231 – 1

Answer:

#pragma GCC optimize("O3", "unroll-loops")

class Solution {

public:

    int findMinArrowShots(vector<vector<int>>& points) {

        int n=points.size();

        sort(points.begin(), points.end());

        int arrows = 0;

        vector<int> prev =points[0];

        for (int i=1; i<n; i++) {

            auto& curr = points[i];

            if (curr[0] > prev[1]) {

                arrows++;

                prev = curr;

            }

            else

                prev[1] = min(prev[1], curr[1]);

        }

        arrows++; // Add the last arrow for the final balloon

    //    cout << "arrows=" << arrows << "\n====\n";

        return arrows;

    }

};

auto init = []() {

    ios::sync\_with\_stdio(0);

    cin.tie(0);

    cout.tie(0);

    return 'c';

}();