Continuous Internal Evaluation 2

Q-1. 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] = [x_{start}, x_{end}] denotes a balloon whose horizontal diameter stretches between x_{start} and x_{end}. 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 x_{start} and x_{end} is **burst** by an arrow shot at x if x_{start} <= x <= x_{end}. 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: 2

Explanation: 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: 4

Explanation: One arrow needs to be shot for each balloon for a total of 4 arrows.

Example 3:

Input: points = [[1,2],[2,3],[3,4],[4,5]]

Output: 2

Explanation: The balloons can be burst by 2 arrows:

- Shoot an arrow at x = 2, bursting the balloons [1,2] and [2,3].

- Shoot an arrow at x = 4, bursting the balloons [3,4] and [4,5].

Constraints:

- 1 <= points.length <= 10^5
- Points[i].length == 2
- -231 <= x_{start} < x_{end} <= $2^{31} - 1$

code :

```
#include <iostream>
#include <algorithm>
using namespace std;

struct Balloon {
    int start, end;
};
bool compare(Balloon a, Balloon b) {
    return a.end < b.end;
}

int findMinArrows(Balloon points[], int n) {
    if (n == 0) return 0;

    // Sort by ending point
    sort(points, points + n, compare);

    int arrows = 1;
```

```
    int arrowPos = points[0].end;

    for (int i = 1; i < n; i++) {
        if (points[i].start > arrowPos) {
            arrows++;
            arrowPos = points[i].end;
        }
    }

    return arrows;
}

int main() {
    int n;
    cout << "Enter number of balloons: ";
    cin >> n;

    Balloon points[100000];

    cout << "Enter the balloon intervals (xstart xend):\n";
    for (int i = 0; i < n; i++) {
        cin >> points[i].start >> points[i].end;
    }

    cout << "Minimum arrows needed: " << findMinArrows(points, n) << endl;
    return 0;
```

}

OUTPUT :

```
cd "c:\Users\DHRUVI\Desktop\CP\" ; if ($?) { g++ cie2.cpp -o cie2 } ; if ($?) { .\cie2 }
Enter number of balloons: 4
Enter the balloon intervals (xstart xend):
1 2
2 3
3 4
4 5
Minimum arrows needed: 2
```

```
PS C:\Ucd "c:\Users\DHRUVI\Desktop\CP\" ; if ($?) { g++ cie2.cpp -o cie2 } ; if ($?) { .\cie2 }
Enter number of balloons: 4
Enter the balloon intervals (xstart xend):
1 2
3 4
5 6
7 8
Minimum arrows needed: 4
```

```
PS C:\Ucd "c:\Users\DHRUVI\Desktop\CP\" ; if ($?) { g++ cie2.cpp -o cie2 } ; if ($?) { .\cie2 }
Enter number of balloons: 4
Enter the balloon intervals (xstart xend):
10 16
2 8
1 6
7 12
Minimum arrows needed: 2
```