Experiment -4

Student Name: Aditi Mourya **UID: 22BCS11624**
Branch: CSE **Section: TPP-IOT-638-A**
Semester: 6th **DOP:21/01/2025**
Subject: Advanced Programming-II **Subject Code:22CSP-351**

Problem-1 Beautiful Array

An array nums of length n is beautiful if:

- nums is a permutation of the integers in the range [1, n].
- For every 0 <= i < j < n, there is no index k with i < k < j where 2 * nums[k] == nums[i] + nums[j].

Given the integer n, return any beautiful array nums of length n. There will be at least one valid answer for the given n.

Example 1:

Input: n = 4

Output: [2,1,4,3]

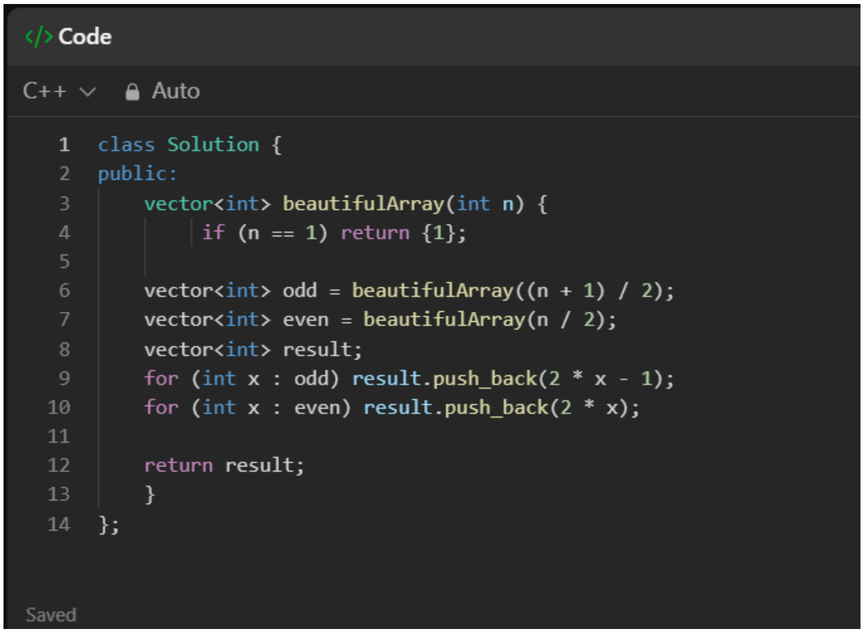
Example 2:

Input: n = 5

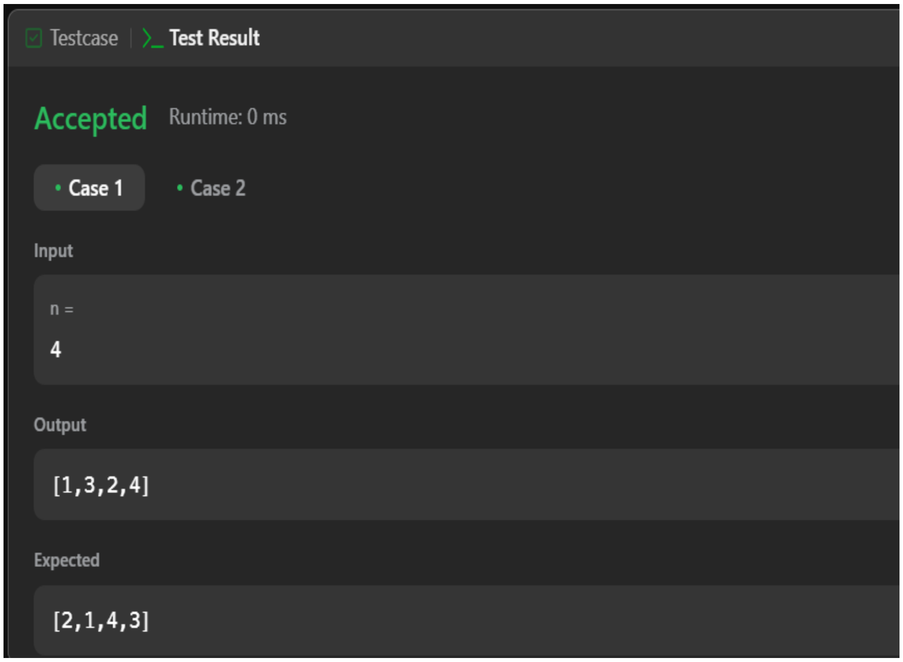
Output: [3,1,2,5,4]

Constraints:

1 <= n <= 1000



```cpp
class Solution {
public:
    vector<int> beautifulArray(int n) {
        if (n == 1) return {1};

    vector<int> odd = beautifulArray((n + 1) / 2);
    vector<int> even = beautifulArray(n / 2);
    vector<int> result;
    for (int x : odd) result.push_back(2 * x - 1);
    for (int x : even) result.push_back(2 * x);

    return result;
    }
};
```



Accepted Runtime: 0 ms

Input

n =
4

Output

```
[1,3,2,4]
```

Expected

```
[2,1,4,3]
```

Problem-2: The Skyline Problem

A city's **skyline** is the outer contour of the silhouette formed by all the buildings in that city when viewed from a distance. Given the locations and heights of all the buildings, return *the **skyline** formed by these buildings collectively*.

The geometric information of each building is given in the array buildings where buildings[i] = [left_i, right_i, height_i]:

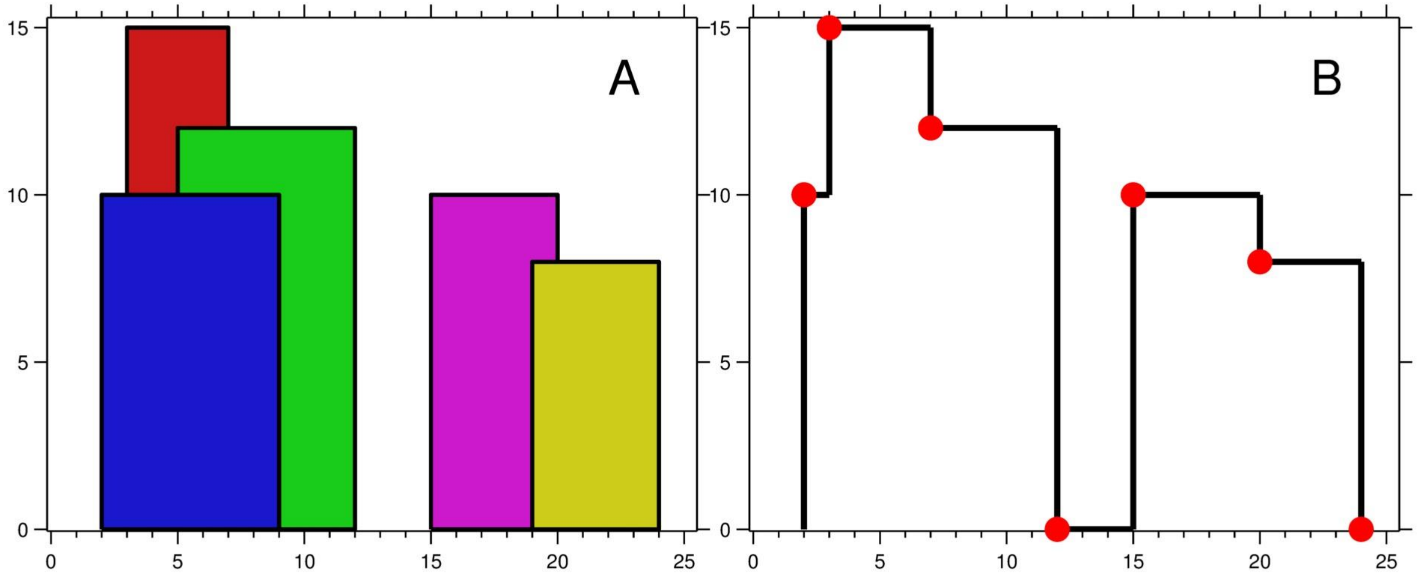
- left_i is the x coordinate of the left edge of the i^{th} building.
- right_i is the x coordinate of the right edge of the i^{th} building.
- height_i is the height of the i^{th} building.

You may assume all buildings are perfect rectangles grounded on an absolutely flat surface at height 0.

The **skyline** should be represented as a list of "key points" **sorted by their x-coordinate** in the form $[[x_1,y_1],[x_2,y_2],...]$. Each key point is the left endpoint of some horizontal segment in the skyline except the last point in the list, which always has a y-coordinate 0 and is used to mark the skyline's termination where the rightmost building ends. Any ground between the leftmost and rightmost buildings should be part of the skyline's contour.

Note: There must be no consecutive horizontal lines of equal height in the output skyline. For instance, [...,[2 3],[4 5],[7 5],[11 5],[12 7],...] is not acceptable; the three lines of height 5 should be merged into one in the final output as such: [...,[2 3],[4 5],[12 7],...]

Example 1:



Input: buildings = [[2,9,10],[3,7,15],[5,12,12],[15,20,10],[19,24,8]]

Output: [[2,10],[3,15],[7,12],[12,0],[15,10],[20,8],[24,0]]

Explanation:

Figure A shows the buildings of the input.

Figure B shows the skyline formed by those buildings. The red points in figure B represent the key points in the output list.

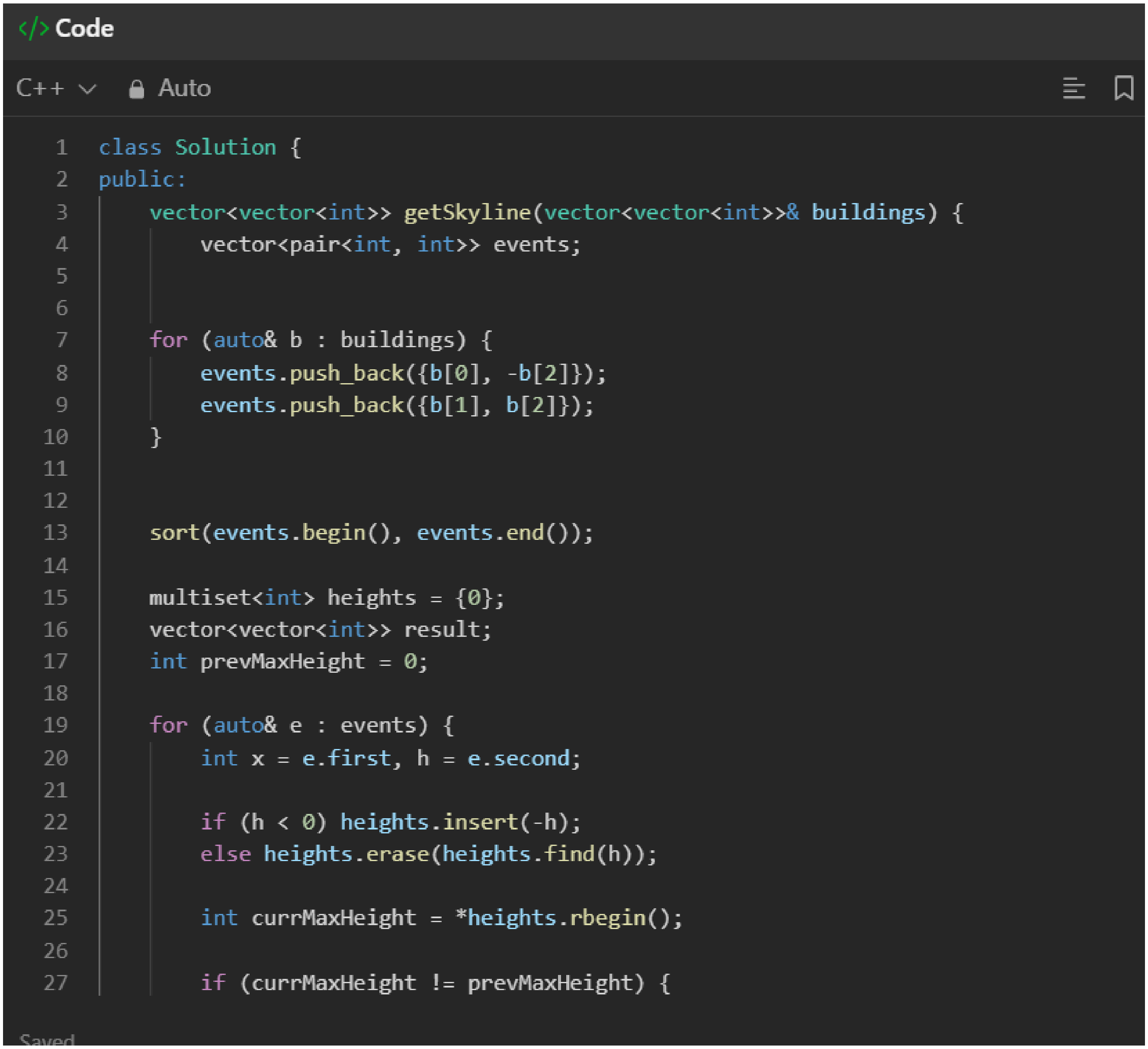
Example 2:

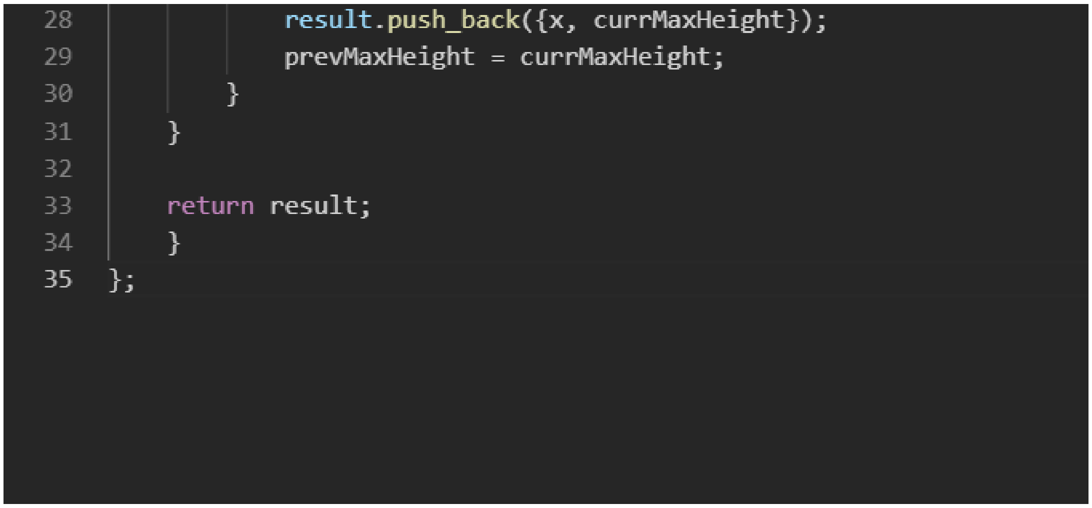
Input: buildings = [[0,2,3],[2,5,3]]

Output: [[0,3],[5,0]]

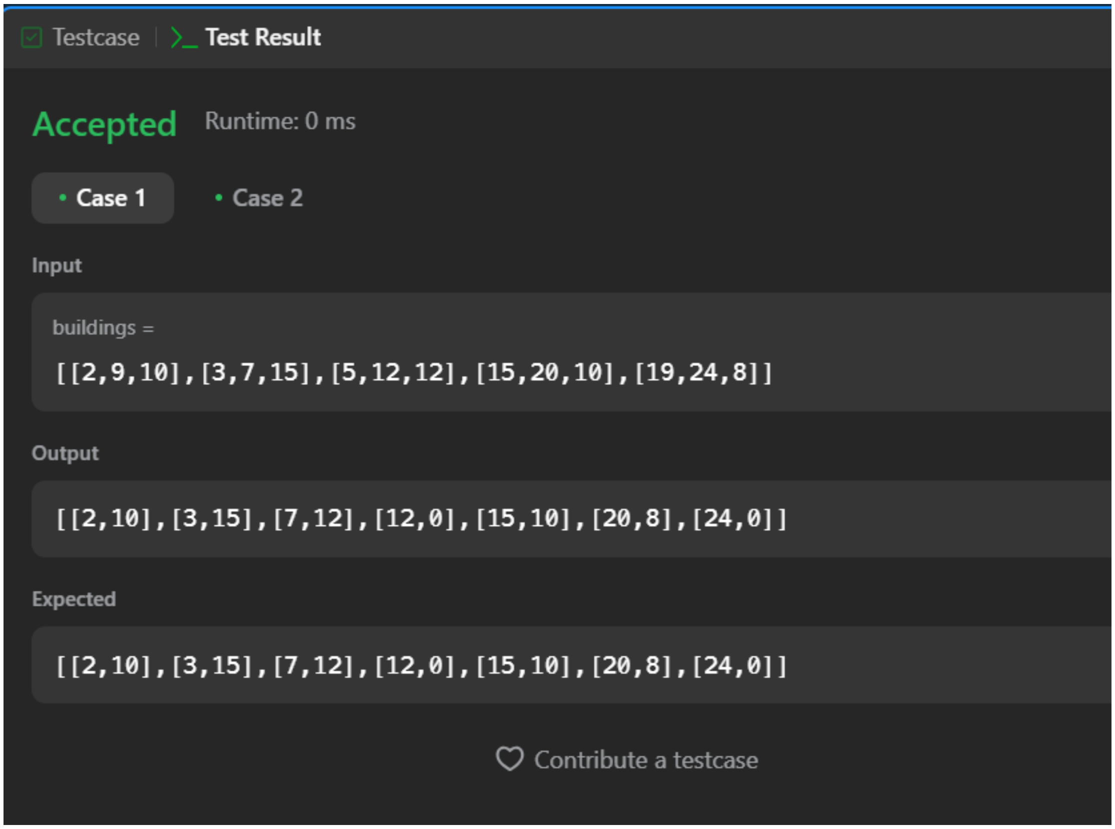
Constraints:

- 1 <= buildings.length <= 10^4
- 0 <= left_i < right_i <= 2^{31} - 1
- 1 <= height_i <= 2^{31} - 1
- buildings is sorted by left_i in non-decreasing order.





```cpp
class Solution {
public:
    vector<vector<int>> getSkyline(vector<vector<int>>& buildings) {
        vector<pair<int, int>> events;


    for (auto& b : buildings) {
        events.push_back({b[0], -b[2]});
        events.push_back({b[1], b[2]});
    }


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

    multiset<int> heights = {0};
    vector<vector<int>> result;
    int prevMaxHeight = 0;

    for (auto& e : events) {
        int x = e.first, h = e.second;

        if (h < 0) heights.insert(-h);
        else heights.erase(heights.find(h));

        int currMaxHeight = *heights.rbegin();

        if (currMaxHeight != prevMaxHeight) {
            result.push_back({x, currMaxHeight});
            prevMaxHeight = currMaxHeight;
        }
    }

    return result;
    }
};
```



Accepted Runtime: 0 ms

Input

buildings =

```
[[2,9,10],[3,7,15],[5,12,12],[15,20,10],[19,24,8]]
```

Output

```
[[2,10],[3,15],[7,12],[12,0],[15,10],[20,8],[24,0]]
```

Expected

```
[[2,10],[3,15],[7,12],[12,0],[15,10],[20,8],[24,0]]
```