

**NAME-Sahil**

**UID-22BCS16100**

**BRANCH-BE-CSE**

**SECTION/GROUP-IOT-638-A**

**SEMESTER-6**

**DATE OF PERFORMANCE-12/02/2023**

**SUBJECT NAME-AP-LAB II**

**SUBJECT CODE-22CSP-351**

## **EXPERIMENT-4**

### **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  $0 \leq i < j < n$ , there is no index `k` with  $i < k < j$  where  $2 * \text{nums}[k] == \text{nums}[i] + \text{nums}[j]$ .

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:**

**$1 \leq n \leq 1000$**

### **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] = [lefti, righti, heighti]`:

lefti is the x coordinate of the left edge of the ith building.

righti is the x coordinate of the right edge of the ith building.

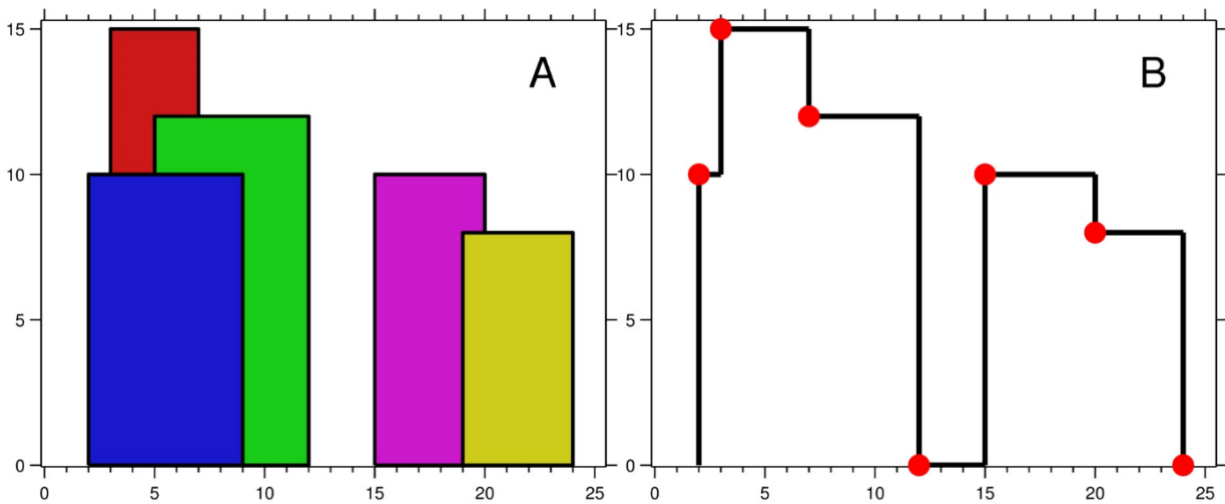
heighti is the height of the ith building.

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  $[[x_1, y_1], [x_2, y_2], \dots]$ . 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,  $[\dots, [2, 3], [4, 5], [7, 5], [11, 5], [12, 7], \dots]$  is not acceptable; the three lines of height 5 should be merged into one in the final output as such:  $[\dots, [2, 3], [4, 5], [12, 7], \dots]$

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 <= 104 0 <= lefti < righti <= 231 - 1 1 <= heighti <= 231 - 1 buildings is sorted by lefti in non-decreasing order.

## 2.CODE

```
#include <bits/stdc++.h>
```

```
using namespace std;
```

```
// Function to generate a beautiful array
```

```
vector<int> beautifulArray(int n) {
```

```
    vector<int> res = {1};
```

```
    while (res.size() < n) {
```

```
        vector<int> temp;
```

```
        for (int x : res) if (2 * x - 1 <= n) temp.push_back(2 * x - 1);
```

```
        for (int x : res) if (2 * x <= n) temp.push_back(2 * x);
```

```
        res = temp;
```

```
    }
```

```
    return res;
```

```
}
```

```
// Function to get the skyline
```

```
vector<vector<int>> getSkyline(vector<vector<int>>& buildings) {
```

```
    vector<pair<int, int>> events;
```

```
    for (auto& b : buildings) {
```

```
        events.emplace_back(b[0], -b[2]); // start of building, negative height
```

```

        events.emplace_back(b[1], b[2]); // end of building, positive height
    }
    sort(events.begin(), events.end());

    multiset<int> heights = {0};
    vector<vector<int>> res;
    int prevHeight = 0;

    for (auto& [x, h] : events) {
        if (h < 0) heights.insert(-h);
        else heights.erase(heights.find(h));

        int currHeight = *heights.rbegin();
        if (currHeight != prevHeight) {
            res.push_back({x, currHeight});
            prevHeight = currHeight;
        }
    }
    return res;
}

```

```

int main() {
    int n = 5; // Example input for beautiful array
    vector<int> beautiful = beautifulArray(n);
    cout << "Beautiful Array: ";
    for (int num : beautiful) cout << num << " ";
    cout << endl;
}

```

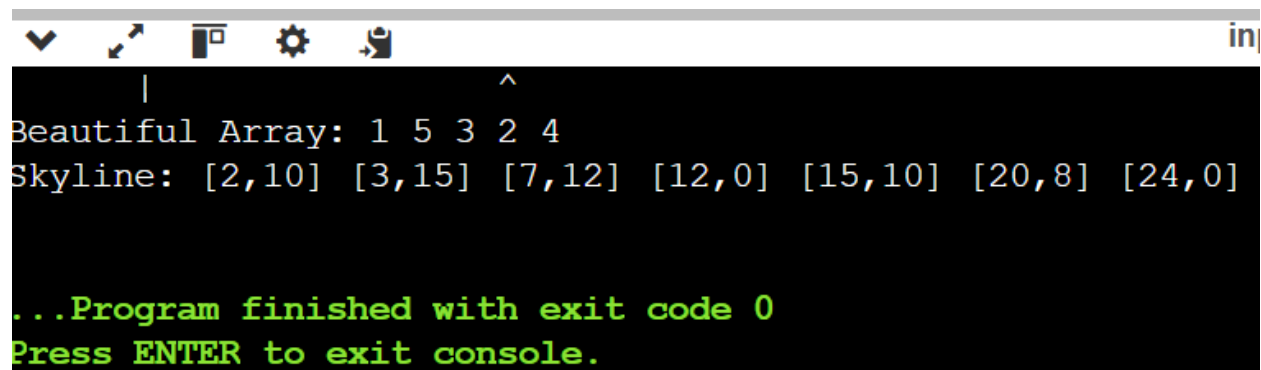
```

vector<vector<int>> buildings = {{2,9,10}, {3,7,15}, {5,12,12}, {15,20,10}, {19,24,8}};
vector<vector<int>> skyline = getSkyline(buildings);

cout << "Skyline: ";
for (auto& point : skyline) {
    cout << "[" << point[0] << ", " << point[1] << "]" ";
}
cout << endl;
return 0;
}

```

## OUTPUT:



```

Beautiful Array: 1 5 3 2 4
Skyline: [2,10] [3,15] [7,12] [12,0] [15,10] [20,8] [24,0]

...Program finished with exit code 0
Press ENTER to exit console.

```

## LEARNING OUTCOMES:

- 🔗 **Understanding Divide and Conquer** – The **Beautiful Array** problem demonstrates how **divide and conquer** can be used to construct a permutation satisfying given constraints.
- 🔗 **Sweep Line Algorithm & Data Structures** – The **Skyline Problem** teaches the **sweep line algorithm** and the use of a **multiset (balanced BST)** to efficiently track active building heights.
- 🔗 **Sorting and Event-based Processing** – Both problems highlight the importance of **sorting** and handling events in a structured manner to optimize performance.

🔗 **Algorithm Optimization & Complexity Analysis** – The solutions showcase **efficient approaches ( $O(n \log n)$  complexity)** for handling large constraints, reinforcing skills in algorithmic **design and analysis**.