# 1901. Find a Peak Element II

Leetcode Link

## Problem Description

Given a 2D grid represented as a 0-indexed m x n matrix `mat`, where each element `mat[i][j]` is strictly greater than all of its adjacent neighbors to the left, right, top, and bottom, the goal is to find any peak element in the grid. A peak element is surrounded by an outer perimeter with the value -1 in each cell. The task is to return the length 2 array `[i, j]` for any peak element `mat[i][j]`, and the algorithm must have a time complexity of O(m log(n)) or O(n log(m)).

*Example:*

Consider the following input matrix `mat`:

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

The peak element in this matrix is `mat[0][1]` with the value 4. Hence, the correct output would be `[0, 1]`.

## Approach

The solution is based on a binary search algorithm where we perform the search across rows. We first find the middle row and compare the maximum element of the middle row with the maximum element of the adjacent row. We then adjust the search space based on the comparison result and continue the search until we find the row containing the peak element. Finally, we find the column index of the peak element in that row and return the coordinates as a pair `[i, j]`.

### Example

Let's walk through an example to understand how the algorithm works:

Suppose we have the following grid:

```
mat = [
    [1, 2, 3],
    [4, 5, 1],
    [5, 4, 6]
]
```

1. First, we initialize the left (l) and right (r) boundaries as `l = 0` and `r = mat.size() - 1` (which is 2).
2. Now we start a while loop until `l < r`.
3. In each iteration, we find the middle row `m` as `(l + r) / 2`.
    - For the first iteration, `m = (0 + 2) / 2 = 1`.
4. We compare the maximum element of row `m` with the maximum element of row `m + 1`.
    - For the first iteration, the maximum element of row 1 is 5 and the maximum element of row 2 is 6. Since `5 < 6`, we update `l` to `m + 1` (which is 2).
5. The loop continues until `l >= r`. When this happens, we found the row containing the peak element.
    - In this case, the loop ends when `l = 2`.
6. To get the column index of the peak element, we call the helper function `getMaxIndex()` with the row containing the peak element.
    - Here, we pass row `[5, 4, 6]` to the function, which returns the index 2.
7. The algorithm returns `[2, 2]`, which is the correct location of the peak element `mat[2][2]` with the value 6.

## Solution using C++

```cpp
#include <vector>
#include <algorithm>

using namespace std;

class Solution {
public:
  vector<int> findPeakGrid(vector<vector<int>>& mat) {
    int l = 0;
    int r = mat.size() - 1;

    while (l < r) {
      const int m = (l + r) / 2;
      if (*max_element(begin(mat[m]), end(mat[m])) >=
          *max_element(begin(mat[m + 1]), end(mat[m + 1])))
        r = m;
      else
        l = m + 1;
    }

    return {l, getMaxIndex(mat[l])};
  }

private:
  int getMaxIndex(const vector<int>& A) {
    pair<int, int> res{0, A[0]};
    for (int i = 1; i < A.size(); ++i)
      if (A[i] > res.second)
        res = {i, A[i]};
    return res.first;
  }
};
```

## Solution using Java

```java
import java.util.ArrayList;
import java.util.Arrays;

class Solution {
    public int[] findPeakGrid(int[][] mat) {
        int l = 0;
        int r = mat.length - 1;

        while (l < r) {
            int m = (l + r) / 2;
            if (Arrays.stream(mat[m]).max().getAsInt() >=
                Arrays.stream(mat[m + 1]).max().getAsInt()) {
                r = m;
            } else {
                l = m + 1;
            }
        }

        return new int[]{l, getMaxIndex(mat[l])};
    }

    private int getMaxIndex(int[] A) {
        int resIndex = 0;
        int resValue = A[0];
        for (int i = 1; i < A.length; ++i) {
            if (A[i] > resValue) {
                resIndex = i;
                resValue = A[i];
            }
        }

        return resIndex;
    }
}
```

## Solution using Python

```python
from typing import List

class Solution:
    def findPeakGrid(self, mat: List[List[int]]) -> List[int]:
        l = 0
        r = len(mat) - 1

        while l < r:
            m = (l + r) // 2
            if max(mat[m]) >= max(mat[m + 1]):
                r = m
            else:
                l = m + 1

        return [l, mat[l].index(max(mat[l]))]
```

## Solution using JavaScript

```javascript
/**
 * @param {number[][]} mat
 * @return {number[]}
 */
var findPeakGrid = function(mat) {
    let l = 0;
    let r = mat.length - 1;

    while (l < r) {
        const m = Math.floor((l + r) / 2);
        if (Math.max(...mat[m]) >= Math.max(...mat[m + 1])) {
            r = m;
        } else {
            l = m + 1;
        }
    }

    return [l, mat[l].indexOf(Math.max(...mat[l]))];
};
```

## Solution using C#

```csharp
using System;
using System.Linq;

public class Solution {
    public int[] findPeakGrid(int[][] mat) {
        int l = 0;
        int r = mat.Length - 1;

        while (l < r) {
            int m = (l + r) / 2;
            if (mat[m].Max() >= mat[m + 1].Max()) {
                r = m;
            } else {
                l = m + 1;
            }
        }

        return new int[]{l, Array.IndexOf(mat[l], mat[l].Max())};
    }
}
```

By using binary search algorithm based on the rows of the grid, we were able to create solutions to find the peak element in a grid for multiple programming languages including C++, Java, Python, JavaScript, and C#. The time complexities of these solutions are O(m log(n)) or O(n log(m)), which meet the desired criteria.