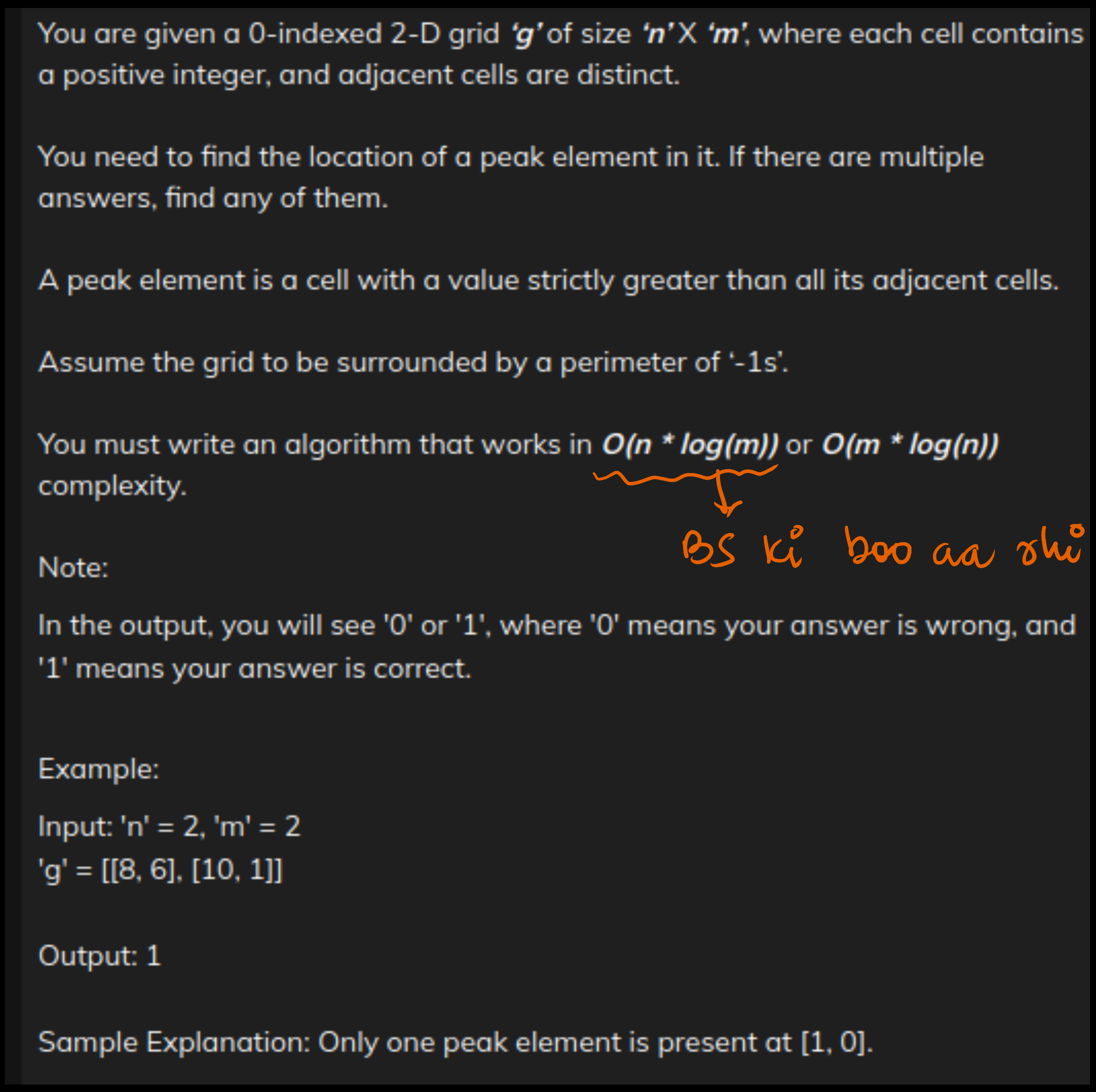
Ⓟ Find peak element



You are given a 0-indexed 2-D grid *'g'* of size *'n'* X *'m'*, where each cell contains a positive integer, and adjacent cells are distinct.

You need to find the location of a peak element in it. If there are multiple answers, find any of them.

A peak element is a cell with a value strictly greater than all its adjacent cells.

Assume the grid to be surrounded by a perimeter of '-1s'.

You must write an algorithm that works in ***O(n * log(m))*** or ***O(m * log(n))*** complexity.

BS ki boo aa rhi

Note:

In the output, you will see '0' or '1', where '0' means your answer is wrong, and '1' means your answer is correct.

Example:

Input: 'n' = 2, 'm' = 2
'g' = [[8, 6], [10, 1]]

Output: 1

Sample Explanation: Only one peak element is present at [1, 0].

→ Brute force Approach :-

↳ Just go one by one to each cell and compare all the adjacent four cells.

Time Complexity	O(4×n×m)
Space Complexity	O(1)

↳ We can make it better by just returning the largest element of the matrix.

Time Complexity	O(n×m)
Space Complexity	O(1)

→ Optimal Approach :-

low ↓ · mid ↓ · high ↓

4 (0,0)	2 (0,1)	5 (0,2)	1 (0,3)	4 (0,4)	5 (0,5)
2 (1,0)	9 (1,1)	3 (1,2)	2 (1,3)	3 (1,4)	2 (1,5)
1 (2,0)	7 (2,1)	6 (2,2)	0 (2,3)	1 (2,4)	3 (2,5)
3 (3,0)	6 (3,1)	2 (3,2)	3 (3,3)	7 (3,4)	2 (3,5)

↑ max element of the row

⇒ Definitely elements above it and elements below it will be smaller.

→

mid
low ↓ · high ↓

4 (0,0)	2 (0,1)	5 (0,2)	1 (0,3)	4 (0,4)	5 (0,5)
2 (1,0)	9 (1,1)	3 (1,2)	2 (1,3)	3 (1,4)	2 (1,5)
1 (2,0)	7 (2,1)	6 (2,2)	0 (2,3)	1 (2,4)	3 (2,5)
3 (3,0)	6 (3,1)	2 (3,2)	3 (3,3)	7 (3,4)	2 (3,5)

This is our answer.

Time Complexity	O(n* logm)
Space Complexity	O(1)

```
int findMaxIndex(vector<vector<int>> &mat, int n, int m, int col) {

  int maxValue = -1;
  int index = -1;
  for (int i = 0; i < n; i++) {
    if (mat[i][col] > maxValue) {
      maxValue = mat[i][col];
      index = i;
    }
  }
  return index;
}

vector<int> findPeakGrid(vector<vector<int>> &mat) {

  int n = mat.size();
  int m = mat[0].size();
  int low = 0;
  int high = m - 1;
  while (low <= high) {
    int mid = (low + high) / 2;
    int maxRowIndex = findMaxIndex(mat, n, m, mid);
    int left = mid - 1 >= 0 ? mat[maxRowIndex][mid - 1] : -1;
    int right = mid + 1 < m ? mat[maxRowIndex][mid + 1] : -1;
    if (mat[maxRowIndex][mid] > left && mat[maxRowIndex][mid] > right) {
      return {maxRowIndex, mid};
    }
    else if (mat[maxRowIndex][mid] < left)
      high = mid - 1;
    else
      low = mid + 1;
  }
  return {-1, -1};
}
```