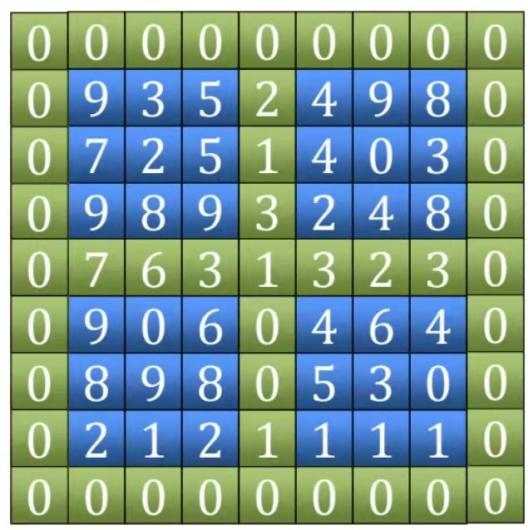
**Algorithm Description**

The basic idea is Divide&Conquer.

Firstly,for arbitrary m by n array,mark the element on the boundary,middle row and the middle column as the window element.

For example,the green elements in the figure 1 are window element,blue elements are not window element.

Figure 1



Quadrant 1

Quadrant 4

Quadrant 2

Quadrant 3

Second step is finding the maximum element in the window(green element).

Observing the Figure 1,the whole two dimensional array is divided into four parts,which are refer as four quadrant.

The blue elements block and its green boundary on the top right corner is refer as quadrant 1.The similar blocks on the top left corner is quadrant 2.And the similar block on the left bottom is the quadrant 3.The similar block on the right bottom is the quadrant 4.

After finding the maximum element in the window,check that whether it is a peak element.If it is,return the row index and the column index and complete the mission.Otherwise,continue to execute the rest part.

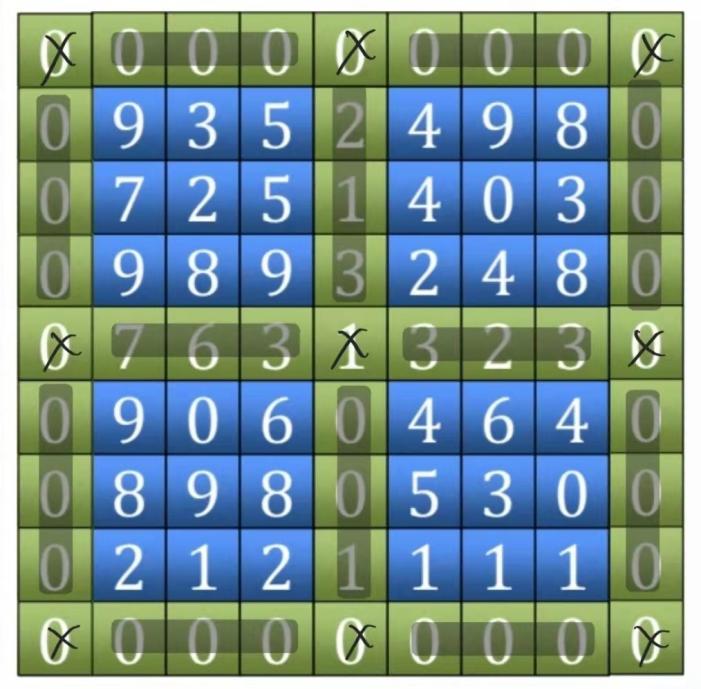
Since the maximum number in the window is not a peak element,the next problem is finding the peak element in one of the quadrant to reduce the problem scale.

**How to select the appropriate quadrant?**

Now we knows that the maximum number in the window isn’t a peak,so the position of the maximum number isn’t on the crossing of the window.Because if the maximum number is on the crossing,it must be a peak number since its neighbors are also in the window.The maximum number must be on the boundary of the whole block, the middle row or the middle column(except the crossing position).

If the maximum element on the boundary,there must be only one neighbor in one of four quadrant larger than the maximum element in the window.So program will continue to search the peak element in that quadrant.

Otherwise,the element must on the middle row or column.There will be two neighbors.Continue to search the peak element in the larger neighbor quadrant.



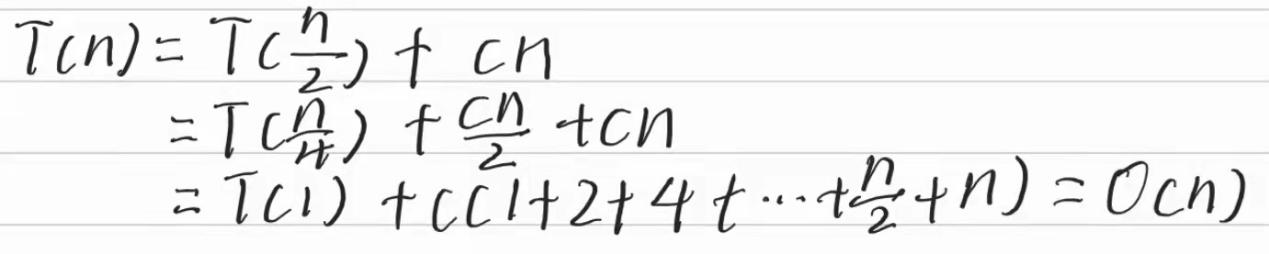
The crossed positions can not be the position of maximum value.

The shaded blocks are the possible position for maximum value.

We can see that calling the peak finder function recursively will get the peak element and the time complexity is O(m+n).

**Time Complexity Analysis**

For a n by n matrix(array),each peak finder function call will reduce the problem scale to the n/2 by n/2 sub matrix with the cost of searching in the window.The time complexity of searching is O(n).

****