Peak Finding In Arrays
2D:

Definition:
Let A be a matrix of size $n \times m$.
Let $A_{i-1, j}$, $A_{i, j}$, $A_{i, j-1}$, $A_{i, j+1}$, and $A_{i+1, j}$, ($i<n$, $j<m$; $(A_{i-1,j}, A_{i,j}, A_{i,j-1}, A_{i,j+1}, A_{i+1,j}) \in A$) be any five elements in A such that A_i is in the middle, and the other four elements are its neighbors.
An item A_i is a peak iff $A_{i,j} \geqslant A_{i+1,j} \wedge A_{i,j} \geqslant A_{i,j+1} \wedge A_{i,j} \geqslant A_{i,j-1} \wedge A_{i,j} \geqslant A_{i-1,j}$.
Same applies with the elements that are on the edges of the matrix, except they will have less than four neighbors.

Algorithms' asymptotic complexity:
Greedy approach: Worst case, $T(n,m)=\Theta(n \times m)$, since it might need to go through the whole size of the matrix to find the peak.

Recursive approach:
A time it takes to find the largest item in the column is $T(n)=\Theta(n)$, since the algorithm needs to walk through each of the rows at a given column.
=> Time to run the algorithm one time is: $T(n,m)=T(n,m/2)+\Theta(n)$.
As we know from before, the one dimensional peak finding will run logarithmic time.
=> $T(n,m)=\sum_{1}^{\log_2(m)} \Theta(n)=\Theta(n \times \log_2(m))$.