Sreeram Danda
CS577 Homework #2
Problem #5

a) The algorithm does eventually find a local maximum. This statement is true because the set of elements is constant and strictly decreasing with each element that is visited. When an element is selected and it is not a local max, the algorithm picks the next largest element adjacent to the current matrix. This reduces the total of possible local maximums by 2: the element that is picked and the element that is rejected. Since the total possible maximums strictly decrease, it must be the case that following the bigger adjacent element leads to an element in the matrix where all the adjacent elements are less than that element. This would be a local maximum.

Given a grid of size n x n, to show a scenario where the algorithm takes $\Omega(n^2)$ we can show an example where the algorithm takes $\Omega(0.5 * n^2)$. This means that the algorithm iterates over at least half of the elements. In the n x n grid assume the elements are ordered such that on the first row each element is greater up to the nth element. At this point, assume the next maximum element is directly below the last element on the previous row. Now the rest of this current row cannot hold a maximum value as we would have already visited it while iterating through the first row. Therefore, like the first row, assume the next row holds elements such that they strictly increase from the nth element to the 1st element. Following this pattern the algorithm would end up iterating through n/2 rows of elements. Additionally there are n/2 elements that exist as a connection between two complete rows. This means that the algorithm iterates over $n^2/2 + n/2$ elements. $\Omega(n^2/2 + n/2) = \Omega(n^2)$.

b) First we divide the matrix in four pieces. With the middle column and middle row as the boundary. Within the mid column and mid row we need to find a local minimum. This means that for the column we look for an element where the adjacent elements are less than itself. For the row we do the same. Note that this is only within the boundary of that single row and that column, so there is no reason to check horizontally as well for the column and no reason to check vertically for the row. Then, once the local max for the array is found, we check the opposite adjacent elements to see if they are less than the local max. If this is true we have found a local max and return. If not we take a look at the adjacent element that is the greatest and recursively do the above steps for the sub matrix that includes the element that was larger than our initial local max. This sub matrix is bounded by the mid column and mid row that was made initially.

c) Program Correctness:

Base Case: There is only one element in the matrix. The algorithm correctly returns this element as the maximum of the matrix.

Recursive Case: The algorithm divides the matrix into 4's with the middle column and middle rows as the boundaries. Then the algorithm searches along the column and the row to find a local maximum. Then the algorithm checks if the other adjacent elements are less than that local max. If so, the algorithm correctly returns the element as a local maximum. Otherwise, the algorithm makes a recursive call on the submatrix of the 4-sub matrices that includes the element that is greater than the initial local max. This is correct because it is exactly like following the series of largest elements to find the local max. By the IH we assume this recursive call correctly outputs the local maximum.

d) Our first step is to find the max of N+N elements. This can be done by brute force and checking all 2N elements. The next step is to check N/2+N/2 elements for the sub matrix we create. As we continue to create new sub matrices we get N/4+N4, N/8 + N/8 and on and on. The total number of steps can then be represented as a geometric series:

$$\sum_{x=1}^{n} \frac{2n}{x} \approx 4n$$

Therefore, we can conclude that the complexity is O(4n) or O(n).