Gabe McKay
Weicheng Li
Duran Suma

Homework 1

Problem 5

Algorithm:
findMin(matrix A):

$n \leftarrow A.size()$
for i in range(0, n):
isLocalMin(i, $\lfloor n/2 \rfloor$) return (i, $\lfloor n/2 \rfloor$)
isLocalMin(i, $\lceil n/2 \rceil$) return (i, $\lceil n/2 \rceil$)
minPt $\leftarrow$ min(minPt, (i, $\lfloor n/2 \rfloor$), (i, $\lceil n/2 \rceil$))
//same thing for cols...

minPtNeighbor = getSmallestNeighbor(minPt)

if minPtNeighbor is in quadrant 1:
return findMin($A[0..\lfloor n/2 \rfloor][0..\lfloor n/2 \rfloor]$)
if minPtNeighbor is in quadrant 2:
return findMin($A[\lfloor n/2 \rfloor..n][0..\lfloor n/2 \rfloor]$)
//same for quadrants 3 and 4..

Runtime:

$T(n) = T(n/4) + cn$
Work done: $n + \frac{n}{4} + \frac{n}{16} + ... \rightarrow O(n)$

Correctness:

Hypothesis: Given an n x n matrix. We return a local minimum which is stated to be smaller than each neighbor it has on each side without changing the underlying matrix.

Base Case: A.size() == 1, it returns the only point. ✓

Induction Step:
We have a matrix A, of size n x n.
Assumption:

$\forall k < n$, our algorithm outputs a local minimum for a k x k matrix while following the hypothesis.

Now consider a n x n matrix. We first survey to check if any of those points are local minimum. If that is the case, we return the point. ✓
Otherwise, we find the smallest pt among those surveyed. From there we move to the

smallest neighbor of that point which is guaranteed to be in one of the quadrants. This action guarantees that a local minimum must exist within that quadrant. This is because we have seen that the number can 'escape' because all of the edges of the quadrant are bigger so a local minimum must exist somewhere inside the quadrant.

Therefore, when we recursively call into that new matrix, our original assumption proves that the recursive call must return a local minimum as that quadrant must have one.