Problem Set 1

All parts are due Thursday, February 20 at 11:59PM. Please download the .zip archive for this problem set, and refer to the `README.txt` file for instructions on preparing your solutions. Remember, your goal is to communicate. Full credit will be given only to a correct solution which is described clearly. Convoluted and obtuse descriptions might receive low marks, even when they are correct. Also, aim for concise solutions, as it will save you time spent on write-ups, and also help you conceptualize the key idea of the problem.

Your Name: Eric Klinkhammer

Collaborators: Name1, Name2

Part A

Problem 1-1.

(a) $(\log n)^{3\log} < n^{0.9} \log^3 n < 3.3n < n^{3.3} < (3.3)^n$

(b) $9^{9n} < 3^{3^n} < 3^{3^{n+1}} < 9^{9^n} < \binom{n}{n/3}$

(c) $n^9 * 3^{n/3} < 3^n < n^{n^{1/9}} < n^{n/3} < \left(\frac{n}{3}\right)^{n/3}$

Problem 1-2.

(a) $\Theta(x)$

(b) $\Theta(x \log x)$

(c) $\Theta(x \log y)$

(d) $\Theta(\log x \log y)$

(e) $\Theta(x \log y + y \log x)$

Problem 1-3. Binary Search

(a) Running time of algorithm3 It can be modeled by the following recurrance.
T(n) = T(cross(n,1)) + T(getMax(n)) + T(getBiggestNeighbor, subproblem, ect) + T(n/2)
T(n) = $n + n + c$ + T(n/2)
If you were to expand the tree, each level of the tree (of which there would be log n of them), would have order n steps. The running time would therefore be $O(n \log n)$

(b) Proof of Correctness
Assuming the returned value is both a peak of it's subproblem and the maximum value in the column (of it's subproblem), then it must be a peak of the entire problem. The only way it would not be a peak is if an adjacent value was larger. Inductively, we know that the value cannot be located within the subproblem, so we look to the parent problem. If it were the case that the adjacent element in the larger array were bigger, then the program would not have looked in the subproblem (having either found the peak or searched on the other side).

Problem 1-4. Binary Search with Peaks

(a) Running Time of Algorithm4 The running time of this (incorrect) algorithm is $O(\log x \log y)$ where x and y are the dimensions of the matrix.
The only variable time operation within each recursive call is the call to the algorithm1D find peak function, which runs in $O(\log n)$ time. The algorithm's run time can be modelled as the following relation:
T(x,y) = T(findpeak(y)) + T(x/2,y)
T(x,y) = $\log n$ + T(x/2, y)
When x = y = n,
T(n) = $O((\log n)^2) = O(\log n)$

(b) Counterexample

$$C = \begin{matrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 \\ 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ 2 & 3 & 4 & 5 & 6 & 7 & 8 \\ 3 & 4 & 5 & 8 & 9 & 8 & 9 \\ 4 & 5 & 6 & 7 & 10 & 12 & 13 \\ 5 & 4 & 3 & 5 & 6 & 8 & 14 \\ 6 & 7 & 8 & 22 & 21 & 20 & 19 \end{matrix}$$

Algorithm 4 incorrectly returns 21 as the peak. This is because 21 is the peak of the right three columns, and those columns were recursed into because the intital column (3) was only searched for a peak (8), which was less than the 9 to its right.

Problem 1-5. Binary Search, Alternating Rows and Columns

(a) Running Time
This algorithm is dominated at each recursive call by finding the divider (with the cross product function) and finding the max within each row or column. It can be modelled by the following relations:
T(x) = x + T(y/2)
T(y) = y + T(x/2)
This approach is faster than algorithm3. In order to subdivide the problem twice (thus reducing it to 1/4 its original size), this algorithm only needs x + y/2 time, as opposed to 2x time. This trend continues, but only provides a constant factor speed up. It is still asymptotically bounded by the same function, and is also $O(n \log n)$.

(b) Correctness
This algorithm is incorrect. Consider the matrix below as a counter example:

$$C = \begin{matrix} 0 & 1 & 2 & 10 & 9 & 8 & 6 \\ 1 & 2 & 3 & 11 & 8 & 6 & 7 \\ 2 & 3 & 4 & 12 & 7 & 7 & 17 \\ 3 & 4 & 5 & 13 & 14 & 15 & 16 \\ 4 & 5 & 6 & 10 & 10 & 12 & 13 \\ 5 & 4 & 3 & 9 & 6 & 8 & 14 \\ 6 & 7 & 8 & 8 & 21 & 20 & 19 \end{matrix}$$

The algorithm incorrectly says that (2,3) is a peak. The flaw in this algorithm is that when it subdivides the problem into smaller sections, it cannot guarentee that the peak of that section will be an actual peak because it does not check all of the adjacent problem. This could be fixed by including the row or column that had the maximum in the subproblem.

Problem 1-6. Greedy Algorithm

(a) Pseudocode
Below is my pseudocode implementation of the greedy algorithm. My pseudocode is a weird mix of C, English, and python.

Listing 1: Psuedocode

```
int row = 1, column = 1;
int[] Points;
while ( isNotPeak(Points, row, column) ) {
   AdjacentPoints = all of the adjacent points
 // between 2 and 4 depending on where edge is
   for point in AdjacentPoints:
       if ( point > current point ):
          current point = point;
}
return (row, column)
```

(b) Time Complexity
The worst case running time of this algorithm would happen when there was a spiral of numbers.

$$C = \begin{matrix} 0 & 0 & 22 & 21 & 20 & 19 & 18 \\ 1 & 0 & 23 & 0 & 0 & 0 & 17 \\ 2 & 0 & 24 & 0 & 31 & 0 & 16 \\ 3 & 0 & 25 & 0 & 30 & 0 & 15 \\ 4 & 0 & 26 & 28 & 29 & 0 & 14 \\ 5 & 0 & 0 & 0 & 0 & 0 & 13 \\ 6 & 7 & 8 & 9 & 10 & 11 & 12 \end{matrix}$$

In this case, the algorithm would run in $O(n^2)$ time because it has to search roughly half of the n^2 elements. It is faster than a naive approach that searches all of the points for a peak, even in the worse case (the worse case for a naive case would be the peak is in the bottom right and it searched left to right top to bottom for all of the points, requiring n^2 steps exactly) because it skips many values. If the matrix is a square, then it only needs to search $n+(n-1)+(n-1)+(n-3)+(n-3)+(n-5)+(n-5)+\cdots+1$ values. This is asymptotically identical to searching all of them, however. A greedy algorithm that just picked a larger element (as opposed to the largest element) would have the same problem with the above example.

(c) Proof of Correctness

The greedy algorithm must terminate because every matrix requires a peak (if a peak is defined as greater than or equal to all surrounding points). The algorithm's potential search space decreases by at least one point every iteration of the loop (it will never return to a previous position), unless it has found a peak. Therefore, it must terminate. The algorithm is safe because it only returns values that are peaks, otherwise it loops further.

Finally, since the algorithm only returns safe values and it must terminate, the algorithm is correct.

Part B

Submit your implemented python script.