Problem Set 1

Both theory and programming questions are due **Thursday, September 15** 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.

We will provide the solutions to the problem set 10 hours after the problem set is due, which you will use to find any errors in the proof that you submitted. You will need to submit a critique of your solutions by **Tuesday, September 20th, 11:59PM**. Your grade will be based on both your solutions and your critique of the solutions.

Collaborators: None.

Problem 1-1. [15 points] **Asymptotic Practice**

For each group of functions, sort the functions in increasing order of asymptotic (big-O) complexity:

(a) [5 points] **Group 1:**

$$\begin{aligned} f_1(n) &= n^{0.999999} \log n \\ f_2(n) &= 10000000n \\ f_3(n) &= 1.000001^n \\ f_4(n) &= n^2 \end{aligned}$$

Your Solution: In order to compare these functions, Big Oh notation and transitivity among the functions can be used. So, let's start by comparing f4 and f1.

Claim: $f_1(n) = O(f_4(n))$

Proof: $f_1(n) = O(f_4(n))$ means that:

$$\lim_{n\to\infty} \frac{f_1(n)}{f_4(n)} = 0$$

Which can also be written as:

$$\lim_{n\to\infty} \frac{n^{0.999999} \log n}{n^2} = 0$$

This simplifies to:

$$\lim_{n\to\infty} \frac{\log n}{n^{1.000001}} = 0$$

Which is true. So in terms of (big-O) complexity:

$$f_4(n) > f_1(n)$$

■

Now, lets compare f4 and f2.

Claim: $f_2(n) = O(f_4(n))$

Proof: $f_2(n) = O(f_4(n))$ means that:

$$\lim_{n\to\infty} \frac{f_2(n)}{f_4(n)} = 0$$

Which can also be written as:

$$\lim_{n\to\infty} \frac{10000000n}{n^2} = 0$$

This simplifies to:

$$\lim_{n\to\infty} \frac{10000000}{n} = 0$$

Which is true. So in terms of (big-O) complexity:

$$f_4(n) > f_2(n)$$

■

Next, comparison between f4 and f3 can be made.

Claim: $f_4(n) = O(f_3(n))$

Proof: $f_4(n) = O(f_3(n))$ means that:

$$\lim_{n\to\infty} \frac{f_4(n)}{f_3(n)} = 0$$

Which can also be written as:

$$\lim_{n\to\infty} \frac{n^2}{1.000001^n} = 0$$

To compare these functions, take log of both numerator and denominator then rearrange expression as:

$$\lim_{n\to\infty} \frac{2\log n}{n\log 1.000001} = 0$$

Lets $c = \dfrac{2}{\log 1.000001}$. In this case, above expression simplifies to:

$$\lim_{n\to\infty} \frac{c \log n}{n} = 0$$

Which is true. So in terms of (big-O) complexity:

$$f_3(n) > f_4(n)$$

■

Since $f_4(n) > f_1(n)$ and $f_4(n) > f_2(n)$ by transitivity $f_3(n) > f_1(n)$ and $f_3(n) > f_2(n)$. Lastly, we need to compare $f_1(n)$ and $f_2(n)$ in order sort the all functions in group 1.

Claim: $f_1(n) = O(f_2(n))$

Proof: $f_1(n) = O(f_2(n))$ means that:

$$\lim_{n\to\infty} \frac{f_1(n)}{f_2(n)} = 0$$

Which can also be written as:

$$\lim_{n\to\infty} \frac{n^{0.999999} \log n}{10000000n} = 0$$

This simplifies to:

$$\lim_{n\to\infty} \frac{\log n}{10000000n^{0.000001}} = 0$$

Which is true. So in terms of (big-O) complexity:

$$f_2(n) > f_1(n)$$

■

At the end, functions in Group 1 can be sorted as follows based on their (big O) complexity:

$$f_1(n) < f_2(n) < f_4(n) < f_3(n)$$

(b) [5 points] **Group 2:**

$$\begin{aligned} f_1(n) &= 2^{2^{1000000}} \\ f_2(n) &= 2^{100000n} \\ f_3(n) &= \binom{n}{2} \\ f_4(n) &= n\sqrt{n} \end{aligned}$$

Your Solution: Like in part (a), functions can be compared with each other and then by using transitivity relations they can be sorted. Lets start by comparing $f_3(n)$ and $f_4(n)$:

Claim: $f_4(n) = O(f_3(n))$

Proof: $f_4(n) = O(f_3(n))$ means that:

$$\lim_{n\to\infty} \frac{f_4(n)}{f_3(n)} = 0$$

Which can also be written as:

$$\lim_{n\to\infty} \frac{n\sqrt{n}}{\binom{n}{2}} = 0$$

Which can also be written as:

$$\lim_{n\to\infty} \frac{n\sqrt{n}}{\frac{n(n-1)}{2!}} = 0$$

This simplifies to:

$$\lim_{n\to\infty} \frac{2\sqrt{n}}{(n-1)} = 0$$

Which is true. So in terms of (big-O) complexity:

$$f_3(n) > f_4(n)$$

■

Now, lets compare $f_3(n)$ and $f_2(n)$.

Claim: $f_3(n) = O(f_2(n))$

Proof: $f_3(n) = O(f_2(n))$ means that:

$$\lim_{n\to\infty} \frac{f_3(n)}{f_2(n)} = 0$$

Which can also be written as:

$$\lim_{n\to\infty} \frac{\frac{n(n-1)}{2!}}{2^{100000n}} = 0$$

Since n goes to infinity

$$\frac{n(n-1)}{2!} \sim n^2$$

In this case expression will be:

$$\lim_{n\to\infty} \frac{n^2}{2^{100000n}} = 0$$

To compare these functions, take log of both numerator and denominator then rearrange expression as:

$$\lim_{n\to\infty} \frac{2\log n}{100000n\log 2} = 0$$

Lets $c = \dfrac{2}{100000\log 2}$. In this case, above expression simplifies to:

$$\lim_{n\to\infty} \frac{c\log n}{n} = 0$$

Which is true. So in terms of (big-O) complexity:

$$f_2(n) > f_3(n)$$

■

Since $f_2(n) > f_3(n)$ and $f_3(n) > f_4(n)$ then by transitivity $f_2(n) > f_4(n)$.

Now, lets compare $f_4(n)$ and $f_1(n)$:

Claim: $f_1(n) = O(f_4(n))$

Proof: $f_1(n) = O(f_4(n))$ means that:

$$\lim_{n\to\infty} \frac{f_1(n)}{f_4(n)} = 0$$

Which can also be written as:

$$\lim_{n\to\infty} \frac{2^{2^{1000000}}}{n\sqrt{n}} = 0$$

Since numerator is a constant number, this expression is true. So in terms of (big-O) complexity:

$$f_4(n) > f_1(n)$$

■

At the end, functions in Group 1 can be sorted as follows based on their (big O) complexity:

$$f_1(n) < f_4(n) < f_3(n) < f_2(n)$$

(c) [5 points] **Group 3:**

$$\begin{aligned} f_1(n) &= n^{\sqrt{n}} \\ f_2(n) &= 2^n \\ f_3(n) &= n^{10} \cdot 2^{n/2} \\ f_4(n) &= \sum_{i=1}^{n}(i+1) \end{aligned}$$

Your Solution:

Like in part (b), functions can be compared with each other and then by using transitivity relations they can be sorted. Lets start by comparing $f_3(n)$ and $f_4(n)$:

Claim: $f_4(n) = O(f_3(n))$

Proof: $f_4(n) = O(f_3(n))$ means that:

$$\lim_{n\to\infty} \frac{f_4(n)}{f_3(n)} = 0$$

Which can also be written as:

$$\lim_{n\to\infty} \frac{\sum_{i=1}^{n}(i+1)}{n^{10} \cdot 2^{n/2}} = 0$$

Sum in the numerator can also be written as follows:

$$\sum_{i=1}^{n}(i+1) = \sum_{i=1}^{n} i + \sum_{i=1}^{n} 1 = \frac{n^2+n}{2} + n = \frac{n^2+3n}{2}$$

Since n goes to infinity

$$\frac{n^2 + 3n}{2} \sim n^2$$

After this rearrangement, limit expression will be as follows:

$$\lim_{n\to\infty} \frac{n^2}{n^{10} \cdot 2^{n/2}} = 0$$

This simplifies to:

$$\lim_{n\to\infty} \frac{1}{n^8 \cdot 2^{n/2}} = 0$$

Which is true. So in terms of (big-O) complexity:

$$f_3(n) > f_4(n)$$

■

Now, lets compare $f_2(n)$ and $f_3(n)$

Claim: $f_3(n) = O(f_2(n))$

Proof: $f_3(n) = O(f_2(n))$ means that:

$$\lim_{n\to\infty} \frac{f_3(n)}{f_2(n)} = 0$$

Which can also be written as:

$$\lim_{n\to\infty} \frac{n^{10} \cdot 2^{n/2}}{2^n} = 0$$

This can be simplified as:

$$\lim_{n\to\infty} \sqrt{(\frac{n^{20}}{2^n})} = 0$$

To compare these functions, take log of both numerator and denominator then rearrange expression as:

$$\lim_{n\to\infty} \sqrt{(\frac{20 \log n}{n \log 2})} = 0$$

Lets $c = \dfrac{20}{\log 2}$. In this case, above expression simplifies to:

$$\lim_{n\to\infty} \sqrt{(\frac{c \log n}{n})} = 0$$

Which is true. So in terms of (big-O) complexity:

$$f_2(n) > f_3(n)$$

■

Since $f_2(n) > f_3(n)$ and $f_3(n) > f_4(n)$ then by transitivity $f_2(n) > f_4(n)$.

Next, lets compare $f_1(n)$ and $f_4(n)$:

Claim: $f_4(n) = O(f_1(n))$

Proof: $f_4(n) = O(f_1(n))$ means that:

$$\lim_{n\to\infty} \frac{f_4(n)}{f_1(n)} = 0$$

Which can also be written as:

$$\lim_{n\to\infty} \frac{\sum_{i=1}^{n}(i+1)}{n^{\sqrt{n}}} = 0$$

Since,

$$\lim_{n\to\infty} \sum_{i=1}^{n}(i+1) = \lim_{n\to\infty} \frac{n^2+3n}{2} \sim \lim_{n\to\infty} n^2$$

Limit of functions can also be written as:

$$\lim_{n\to\infty} \frac{n^2}{n^{\sqrt{n}}} = 0$$

To compare these functions, take log of both numerator and denominator then rearrange expression as:

$$\lim_{n\to\infty} \frac{2\log n}{\sqrt{n}\log n} = 0$$

This expression equals to:

$$\lim_{n\to\infty} \frac{2}{\sqrt{n}} = 0$$

Which is true. So in terms of (big-O) complexity:

$$f_1(n) > f_4(n)$$

■

At the end, functions in Group 1 can be sorted as follows based on their (big O) complexity:

$$f_4(n) < f_1(n) < f_3(n) < f_2(n)$$

Problem 1-2. [15 points] **Recurrence Relation Resolution**

For each of the following recurrence relations, pick the correct asymptotic runtime:

(a) [5 points] Select the correct asymptotic complexity of an algorithm with runtime $T(n, n)$ where

$$\begin{array}{rcll} T(x,c) & = & \Theta(x) & \text{for } c \leq 2, \\ T(c,y) & = & \Theta(y) & \text{for } c \leq 2, \text{ and} \\ T(x,y) & = & \Theta(x+y) + T(x/2, y/2). & \end{array}$$

1. $\Theta(\log n)$.
2. $\Theta(n)$.
3. $\Theta(n \log n)$.
4. $\Theta(n \log^2 n)$.
5. $\Theta(n^2)$.
6. $\Theta(2^n)$.

Your Solution: To find correct asymptotic complexity of the given algorithm lets open it for few timesteps for x = n and y = n:

$$T(n,n) = \Theta(2n) + T(n/2, n/2)$$

$$T(n,n) = \Theta(2n) + \Theta(n) + T(n/4, n/4)$$

$$T(n,n) = \Theta(2n) + \Theta(n) + \Theta(n/2) + T(n/8, n/8)$$

$$T(n,n) = \Theta(2n) + \Theta(n) + \Theta(n/2) + \cdots + \Theta(1)$$

Since:

$$2 + 1 + 1/2 + 1/4 + \cdots = 2 + \sum_{i=0}^{n=\infty} \frac{1}{2^i} = 2 + \frac{1}{1 - 1/2} = 4$$

For this algorithm's asymptotic complexity, it can be said that for any constant $k > 4$:

$$T(n,n) = \Theta(kn) = \Theta(n)$$

(b) [5 points] Select the correct asymptotic complexity of an algorithm with runtime $T(n, n)$ where

$$\begin{array}{rcll} T(x,c) & = & \Theta(x) & \text{for } c \leq 2, \\ T(c,y) & = & \Theta(y) & \text{for } c \leq 2, \text{ and} \\ T(x,y) & = & \Theta(x) + T(x, y/2). & \end{array}$$

1. $\Theta(\log n)$.
2. $\Theta(n)$.
3. $\Theta(n \log n)$.
4. $\Theta(n \log^2 n)$.
5. $\Theta(n^2)$.
6. $\Theta(2^n)$.

Your Solution: To find correct asymptotic complexity of the given algorithm lets open it for few timesteps for x = n and y = n:

$$T(n,n) = \Theta(n) + T(n, n/2)$$

$$T(n,n) = \Theta(n) + \Theta(n) + T(n, n/4)$$

$$T(n,n) = \Theta(n) + \Theta(n) + \Theta(n) + T(n, n/8)$$

$$T(n,n) = \Theta(n) + \Theta(n) + \Theta(n) + \Theta(n) + \ldots \Theta(n)$$

Which equals to:

$$T(n,n) = \log n * \Theta(n)$$

So complexity of this algorithm is $\Theta(n \log n)$.

(c) [5 points] Select the correct asymptotic complexity of an algorithm with runtime $T(n, n)$ where

$$\begin{array}{rcll} T(x,c) & = & \Theta(x) & \text{for } c \leq 2, \\ T(x,y) & = & \Theta(x) + S(x, y/2), & \\ S(c,y) & = & \Theta(y) & \text{for } c \leq 2, \text{ and} \\ S(x,y) & = & \Theta(y) + T(x/2, y). & \end{array}$$

1. $\Theta(\log n)$.
2. $\Theta(n)$.
3. $\Theta(n \log n)$.
4. $\Theta(n \log^2 n)$.

5. $\Theta(n^2)$.
6. $\Theta(2^n)$.

Your Solution: To find correct asymptotic complexity of the given algorithm lets open it for few timesteps for x = n and y = n:

$$T(n,n) = \Theta(n) + S(n, n/2)$$

$$T(n,n) = \Theta(n) + \Theta(n/2) + T(n/2, n/2)$$

$$T(n,n) = \Theta(n) + \Theta(n/2) + \Theta(n/2) + S(n/2, n/4)$$

$$T(n,n) = \Theta(n) + \Theta(n/2) + \Theta(n/2) + \Theta(n/4) + T(n/4, n/4)$$

$$T(n,n) = \Theta(n) + \Theta(n/2) + \Theta(n/2) + \Theta(n/4) + \Theta(n/4) + S(n/4, n/8)$$

$$T(n,n) = \Theta(n) + 2\Theta(n/2) + 2\Theta(n/4) + \cdots + 2\Theta(1)$$

Since:

$$1 + 2 * \sum_{i=0}^{n=\infty} \frac{1}{2^i} = 1 + 2 * \frac{1}{1 - 1/2} = 5$$

For this algorithm's asymptotic complexity, it can be said that for any constant $k > 5$:

$$T(n,n) = \Theta(kn) = \Theta(n)$$

Peak-Finding

In Lecture 1, you saw the peak-finding problem. As a reminder, a *peak* in a matrix is a location with the property that its four neighbors (north, south, east, and west) have value less than or equal to the value of the peak. We have posted Python code for solving this problem to the website in a file called `ps1.zip`. In the file `algorithms.py`, there are four different algorithms which have been written to solve the peak-finding problem, only some of which are correct. Your goal is to figure out which of these algorithms are correct and which are efficient.

Problem 1-3. [16 points] **Peak-Finding Correctness**

(a) [4 points] Is `algorithm1` correct?

1. Yes.
2. No.

Your Solution: Yes, this algorithms is correct because it uses global maximum on the dividing column to find a peak. Why global maximum? Because we know that there is no bigger number than the global maximum along the dividing column. If right and left neighbours are smaller than global maximum than it's a peak. Otherwise, algorithm continues on the side of the neighbour which contains number bigger than the global maximum along the dividing column

(b) [4 points] Is `algorithm2` correct?

1. Yes.
2. No.

Your Solution: Yes, this algorithm is correct because at the current location it checks whether there is a bigger neighbour than the number on the current location. If not, then number on the current location is a 2D peak. Otherwise, better number exists then algorithm looks for a 2D peak.

(c) [4 points] Is `algorithm3` correct?

1. Yes.
2. No.

Your Solution: No, this algorithm is not correct because it recurse on the quarter which contains number bigger than global maximum of the cross and try to find a peak on that quarter. Since algorithm tries to find global maximum on the cross of the current matrix and use it to go to a smaller quarter there may be cases where the returned number by algorithm is smaller than one of it's neighbour because neighbour cell belongs to neighbour quarter which doesn't checked by algorithm.

(d) [4 points] Is `algorithm4` correct?

1. Yes.
2. No.

Your Solution: Yes, this algorithm is correct because every time it splits a row or column it finds global maximum along the dividing column or row and then checks for better neighbours.

Problem 1-4. [16 points] **Peak-Finding Efficiency**

(a) [4 points] What is the worst-case runtime of `algorithm1` on a problem of size $n \times n$?

1. $\Theta(\log n)$.
2. $\Theta(n)$.
3. $\Theta(n \log n)$.
4. $\Theta(n \log^2 n)$.

5. $\Theta(n^2)$.
6. $\Theta(2^n)$.

Your Solution:

Each time algorithm divides columns into two parts => n/2

Then finds the global maximum along the dividing column => n

If T(n,n) denotes the work required to solve problem with n rows and n columns, then:

$$T(n,n) = T(n,n/2) + \Theta(n)$$

$$T(n,n) = T(n,n/4) + \Theta(n) + \Theta(n)$$

$$T(n,n) = \Theta(n) + ... + \Theta(n)$$

$$T(n,n) = \log n * \Theta(n)$$

$$T(n,n) = \Theta(n \log n)$$

(b) [4 points] What is the worst-case runtime of `algorithm2` on a problem of size $n \times n$?

1. $\Theta(\log n)$.
2. $\Theta(n)$.
3. $\Theta(n \log n)$.
4. $\Theta(n \log^2 n)$.
5. $\Theta(n^2)$.
6. $\Theta(2^n)$.

Your Solution: This algorithm compares the numbers in the neighbour cells with the number in the current location. In worst case, algorithm checks every cell in the matrix so:

$$T(n,n) = n^2 * \Theta(1) = \Theta(n^2)$$

(c) [4 points] What is the worst-case runtime of `algorithm3` on a problem of size $n \times n$?

1. $\Theta(\log n)$.
2. $\Theta(n)$.
3. $\Theta(n \log n)$.
4. $\Theta(n \log^2 n)$.
5. $\Theta(n^2)$.
6. $\Theta(2^n)$.

Your Solution:
Each time algorithm divides columns and rows into two parts => n/2, n/2

Then finds the global maximum along the dividing column and row => 2n

If T(n,n) denotes the work required to solve problem with n rows and n columns, then:

$$T(n,n) = T(n/2, n/2) + \Theta(2n)$$

$$T(n,n) = T(n/4, n/4) + \Theta(n) + \Theta(2n)$$

$$T(n,n) = \Theta(1) + ... + \Theta(n/2) + \Theta(n) + \Theta(2n)$$

Since:

$$2 + 1 + 1/2 + 1/4 + \cdots = 2 + \sum_{i=0}^{n=\infty} \frac{1}{2^i} = 2 + \frac{1}{1 - 1/2} = 4$$

For this algorithm's asymptotic complexity, it can be said that for any constant $k > 4$:

$$T(n,n) = \Theta(kn) = \Theta(n)$$

(d) [4 points] What is the worst-case runtime of `algorithm4` on a problem of size $n \times n$?

1. $\Theta(\log n)$.
2. $\Theta(n)$.
3. $\Theta(n \log n)$.
4. $\Theta(n \log^2 n)$.
5. $\Theta(n^2)$.
6. $\Theta(2^n)$.

Your Solution:
Each time algorithm divides columns or rows into two parts => n/2

Then finds the global maximum along the dividing column or row => n

If T(n,n) denotes the work required to solve problem with n rows and n columns, then:

$$T(n,n) = T(n/2, n) + \Theta(n)$$

$$T(n,n) = T(n/4, n/2) + \Theta(n/2) + \Theta(n)$$

$$T(n,n) = \Theta(1) + ... + \Theta(n/4) + \Theta(n/2) + \Theta(n)$$

Since:

$$1 + 1/2 + 1/4 + \cdots = \sum_{i=0}^{n=\infty} \frac{1}{2^i} = \frac{1}{1 - 1/2} = 2$$

For this algorithm's asymptotic complexity, it can be said that for any constant $k > 2$:

$$T(n, n) = \Theta(kn) = \Theta(n)$$

Problem 1-5. [19 points] **Peak-Finding Proof**

Please modify the proof below to construct a proof of correctness for the *most efficient correct algorithm* among `algorithm2`, `algorithm3`, and `algorithm4`.

The following is the proof of correctness for `algorithm1`, which was sketched in Lecture 1.

> We wish to show that `algorithm1` will always return a peak, as long as the problem is not empty. To that end, we wish to prove the following two statements:
>
> **1. If the peak problem is not empty, then `algorithm1` will always return a location.** Say that we start with a problem of size $m \times n$. The recursive subproblem examined by `algorithm1` will have dimensions $m \times \lfloor n/2 \rfloor$ or $m \times (n - \lfloor n/2 \rfloor - 1)$. Therefore, the number of columns in the problem strictly decreases with each recursive call as long as $n > 0$. So `algorithm1` either returns a location at some point, or eventually examines a subproblem with a non-positive number of columns. The only way for the number of columns to become strictly negative, according to the formulas that determine the size of the subproblem, is to have $n = 0$ at some point. So if `algorithm1` doesn't return a location, it must eventually examine an empty subproblem.
>
> We wish to show that there is no way that this can occur. Assume, to the contrary, that `algorithm1` does examine an empty subproblem. Just prior to this, it must examine a subproblem of size $m \times 1$ or $m \times 2$. If the problem is of size $m \times 1$, then calculating the maximum of the central column is equivalent to calculating the maximum of the entire problem. Hence, the maximum that the algorithm finds must be a peak, and it will halt and return the location. If the problem has dimensions $m \times 2$, then there are two possibilities: either the maximum of the central column is a peak (in which case the algorithm will halt and return the location), or it has a strictly better neighbor in the other column (in which case the algorithm will recurse on the non-empty subproblem with dimensions $m \times 1$, thus reducing to the previous case). So `algorithm1` can never recurse into an empty subproblem, and therefore `algorithm1` must eventually return a location.
>
> **2. If `algorithm1` returns a location, it will be a peak in the original problem.** If `algorithm1` returns a location (r_1, c_1), then that location must have the best value in column c_1, and must have been a peak within some recursive subproblem. Assume, for the sake of contradiction, that (r_1, c_1) is not also a peak within the original problem. Then as the location (r_1, c_1) is passed up the chain of recursive calls, it must eventually reach a level where it stops being a peak. At that level, the location (r_1, c_1) must be

adjacent to the dividing column c_2 (where $|c_1 - c_2| = 1$), and the values must satisfy the inequality $val(r_1, c_1) < val(r_1, c_2)$.

Let (r_2, c_2) be the location of the maximum value found by `algorithm1` in the dividing column. As a result, it must be that $val(r_1, c_2) \leq val(r_2, c_2)$. Because the algorithm chose to recurse on the half containing (r_1, c_1), we know that $val(r_2, c_2) < val(r_2, c_1)$. Hence, we have the following chain of inequalities:

$$val(r_1, c_1) < val(r_1, c_2) \leq val(r_2, c_2) < val(r_2, c_1)$$

But in order for `algorithm1` to return (r_1, c_1) as a peak, the value at (r_1, c_1) must have been the greatest in its column, making $val(r_1, c_1) \geq val(r_2, c_1)$. Hence, we have a contradiction.

Your Solution:

We wish to show that `algorithm4` will always return a peak, as long as the problem is not empty. To that end, we wish to prove the following two statements:

1. If the peak problem is not empty, then `algorithm4` will always return a location. Say that we start with a problem of size $m \times n$. The recursive subproblem examined by `algorithm4` will have dimensions $\lfloor m/2 \rfloor \times n$ or $(m - \lfloor m/2 \rfloor - 1) \times n$ or $m \times \lfloor n/2 \rfloor$ or $m \times (n - \lfloor n/2 \rfloor - 1)$. Therefore, the number of columns or rows in the problem strictly decreases with each recursive call as long as $m > 0$ and $n > 0$. So `algorithm4` either returns a location at some point, or eventually examines a subproblem with a non-positive number of columns and number of rows. The only way for the number of columns or number of rows to become strictly negative, according to the formulas that determine the size of the subproblem, is to have $n = 0$ or $m = 0$ at some point. So if `algorithm4` doesn't return a location, it must eventually examine an empty subproblem.

We wish to show that there is no way that this can occur. Assume, to the contrary, that `algorithm4` does examine an empty subproblem. Just prior to this, it must examine a subproblem of size $1 \times n$ or $2 \times n$ or $m \times 1$ or $m \times 2$. If the problem is of size $n \times 1$ or $m \times 1$, then calculating the maximum of the central column or row is equivalent to calculating the maximum of the entire problem. Hence, the maximum that the algorithm finds must be a peak, and it will halt and return the location. If the problem has dimensions $n \times 2$ or $m \times 2$, then there are two possibilities: either the maximum of the central row or column is a peak (in which case the algorithm will halt and return the location), or it has a strictly better neighbor in the other row or column (in which case the algorithm will recurse on the non-empty subproblem with dimensions $n \times 1$ or $m \times 1$, thus reducing to the previous case). So `algorithm4` can never recurse into an empty subproblem, and therefore `algorithm4` must eventually return a location.

2. If `algorithm4` returns a location, it will be a peak in the original problem. If `algorithm4` returns a location (r_1, c_1), then that location must have the best value in row r_1 or column c_1 (depending on current step), and must have been a peak within

some recursive subproblem. Assume, for the sake of contradiction, that (r_1, c_1) is not also a peak within the original problem. Then as the location (r_1, c_1) is passed up the chain of recursive calls, it must eventually reach a level where it stops being a peak.

There are 2 cases to consider:

Case 1: (r1, c1) is on dividing column: At that level, the location (r_1, c_1) must be adjacent to the dividing column c_2 (where $|c_1 - c_2| = 1$), and the values must satisfy the inequality $val(r_1, c_1) < val(r_1, c_2)$.
Let (r_2, c_2) be the location of the maximum value found by `algorithm4` in the dividing column. As a result, it must be that $val(r_1, c_2) \leq val(r_2, c_2)$. Because the algorithm chose to recurse on the half containing (r_1, c_1), we know that $val(r_2, c_2) < val(r_2, c_1)$. Hence, we have the following chain of inequalities:

$$val(r_1, c_1) < val(r_1, c_2) \leq val(r_2, c_2) < val(r_2, c_1)$$

But in order for `algorithm4` to return (r_1, c_1) as a peak, the value at (r_1, c_1) must have been the greatest in its column, making $val(r_1, c_1) \geq val(r_2, c_1)$. Hence, we have a contradiction.

Case 1: (r1, c1) is on dividing row: At that level, the location (r_1, c_1) must be adjacent to the dividing row r_2 (where $|r_1 - r_2| = 1$), and the values must satisfy the inequality $val(r_1, c_1) < val(r_2, c_1)$
Let (r_2, c_2) be the location of the maximum value found by `algorithm4` in the dividing row. As a result, it must be that $val(r_2, c_1) \leq val(r_2, c_2)$. Because the algorithm chose to recurse on the half containing (r_1, c_1), we know that $val(r_2, c_2) < val(r_1, c_2)$. Hence, we have the following chain of inequalities:

$$val(r_1, c_1) < val(r_2, c_1) \leq val(r_2, c_2) < val(r_1, c_2)$$

But in order for `algorithm4` to return (r_1, c_1) as a peak, the value at (r_1, c_1) must have been the greatest in its column, making $val(r_1, c_1) \geq val(r_1, c_2)$. Hence, we have a contradiction.

Problem 1-6. [19 points] **Peak-Finding Counterexamples**

For each incorrect algorithm, upload a Python file giving a counterexample (i.e. a matrix for which the algorithm returns a location that is not a peak).

Your Solution: The only incorrect algorithm is the 3rd and a counterexample matrix is provided below. In this counter example algorithm3 firstly found 4 as the global maximum of the cross and then goes to the upper left quarter because $5 > 4$. After that algorithm tries to find global maximum of the cross of this quarter, in this example it is 2. Algorithm checks neighbours of the 2 in the quarter and returns 2 as peak. But notice that $3 > 2$ but algorithm didn't check it because 3 doesn't belong to upper left quarter and therefore algorithm didn't consider it as neighbour of 2.

```
problemMatrix = [
    [0, 0, 5, 4, 0, 0],
    [0, 0, 0, 0, 0, 0],
    [0, 2, 0, 0, 0, 0],
    [0, 3, 0, 0, 0, 0],
    [0, 0, 0, 0, 0, 0],
    [0, 0, 0, 0, 0, 0]
]
```