Problem 1 (a). Some recursive algorithm, given any input of size n, does $\Theta(n \log n)$ work and then recurses on two subproblems of size $n/2$. The base case is when $n = 1$. Suppose the run time $C(n)$ satisfies the following recurrence relation: $C(1) = 0$ and $C(n) = 2C(n/2) + n \log_2 n$ for $n \geq 2$. Use the recursion-tree method to find a closed-form Θ value for $C(n)$, then describe your calculations as requested below. Assume n is a power of two.

Problem 1(a) answer.

In level i of the recursion tree, the number of nodes is

$$2^i$$

The size of each subproblem in level i is

$$\frac{n}{2^i}$$

The work associated with each node in level i is

$$\frac{n}{2^i}(log_2 n - i)$$

The total work in level i is

$$nlog_2 n - in$$

The number of levels is

$$log_2 n$$

By summing over the levels, $C(n)$ is equal to the following sum:

$$\sum_{i=0}^{log_2 n} nlog_2 n - in$$

The big-Θ-value of $C(n)$ is

$$\Theta(nlog^2 n)$$

Naive Upper and Lower Bounds

Problem 1 (b). Professor Bo Zo proposes a variant of Karatsuba's algorithm, BozoMult, for multiplying large integers. Given any two n-bit integers A and B (where n is a power of three) BozoMult(A, B) makes eight recursive calls, each computing the product of two $n/3$-bit integers. In addition to the recursive calls, it takes linear time, $\Theta(n)$ (just as Karatsuba's algorithm does, outside of its three recursive calls). Define $M(n)$ to be the time that BozoMult takes to multiply two n-bit integers. State a recurrence relation for $M(n)$, then use the recursion-tree method to obtain a closed-form Θ value for $M(n)$. Describe your calculations as requested below. Assume n is a power of three.

Problem 1(b) answer

Here is the recurrence relation for M. $M(1) = 1$. For $n \geq 2$,

$$M(n) = n + 8M(\frac{n}{3})$$

In level i of the recursion tree, the number of nodes is

$$8^i$$

The size of each subproblem in level i is

$$\frac{n}{3^i}$$

The work associated with each node in level i is

$$\frac{n}{3^i}$$

The total work in level i is

$$n \cdot (\frac{8}{3})^i$$

The total number of levels is

$$log_3 n$$

By summing over the levels, $M(n)$ is proportional to the following sum:

$$n \cdot \sum_{i=0}^{log_3 n} (\frac{8}{3})^i$$

The big-Θ-value of $M(n)$ is

$$\Theta(n^{log_3 8})$$

Problem 1 (c). Consider the following "lopsided" recurrence relation: $T(n) = 1$ for $n \in \{1, 2, 3, 4\}$ and $T(n) = n + T(\lfloor n/4 \rfloor) + T(\lfloor 2n/3 \rfloor)$. Choose some constant c (as small as you can easily make work) then prove by induction that $T(n)$ is at most cn for all $n \geq 1$.

Problem 1(c) answer

Lemma 1. *For all $n \geq 1$, $T(n) \leq 12n$.*

Proof (long form).

1. The proof is by induction on n.
2. Each base case $n \in \{1, 2, 3, 4\}$ holds because c $= 12 \geq Max(\frac{T(n)}{n} : n \leq 4) = 1$.
$T(1) = 1 < 12(1) = 12, T(2) = 3 < 12(2) = 24, T(3) = 5 < 3(12) = 36, T(4) = 7 < 4(12) = 48$

3.1. Consider any $n \geq 5$, and assume inductively that $T(m) \leq 12m$ for $1 \leq m < n$.

3.2. We'll show that $T(n) \leq 12n$.

3.3. Recall the recurrence relation, $T(n) = n + T(\lfloor n/4 \rfloor) + T(\lfloor 2n/3 \rfloor)$.

3.4. By the inductive assumption, Since we know for $n \geq 5$, $\frac{n}{4} = m_1 \leq n$ and $\frac{2n}{3} = m_2 \leq n$, then since for all $1 \leq m_i \leq n$, $T(m_i) \leq 12m_i$, we obtain $T(n) = n + T(\lfloor n/4 \rfloor) + T(\lfloor 2n/3 \rfloor) \leq n + 12(n/4) + 12(2n/3) = n + 3n + 8n = 12n$.

3.5. That is, $T(n) \leq 12n$.

4. By Block 3, for all $n \geq 5$, if $T(m) \leq 12m$ for $1 \leq m < n$, then $T(n) \leq 12n$.
5. By this, Step 2, and induction, $T(n) \leq 12n$ for all $n \geq 1$. $\square$

Problem 2. Read the definition of the 2D Local Maximum problem. Consider the following algorithm:

```
local-max($A[1..n, 1..n]$)                                   We assume $n \geq 1$.
1. Let $X$ be the set consisting of the $2n-1$ cells lying in column $(n+1)/2$ or row $(n+1)/2$ of $A$.
2. Check each cell in $X$ to see if it is a local maximum in $A$. If so, return such a cell. Otherwise:
3. Let $A[i,j]$ be any cell in $X$ with maximum value (among cells in $X$).
4. Let $A[i',j']$ be a neighbor of $A[i,j]$ in $A$ with $A[i',j'] > A[i,j]$. (It exists, as $A[i,j]$ is not a local max.)
5. Partition $A \setminus X$ (the array $A$, minus the cells in $X$) around $X$ into its four $(n-1)/2 \times (n-1)/2$
   quadrants — the upper left, upper right, lower left, and lower right — in the natural way.
6. Among those, let $A'$ be the quadrant containing the cell $A[i',j']$.
7. Recursively compute a local maximum of the subarray $A'$, and return that cell.
```

Here is an attempted proof of correctness. First we prove a utility lemma:

Lemma 2. *During any execution of the algorithm that reaches Line 7, the quadrant A' (as defined in the algorithm Line 6) contains a local maximum of A.*

Proof (long form).

1. Let i, j, i' and j' be as defined in the algorithm for that execution. Consider the following process.

```
(i)   Put a token on the cell $A[i',j']$.
(ii)  While the token's cell is not a local maximum in $A$ do:
(iii)     Move the token to a neighboring cell with larger value.
```

2. Each cell in X is no larger than $A[i,j]$ (by Line 4 of the algorithm), and $A[i,j] < A[i',j']$.
3. So each cell in X is smaller than $A[i',j']$.
4. So by inspection of the process it never moves the token into X. So the token stays in A'.
5. The process must stop, because each move increases the value of the cell with the token.
6. So the token ends on a cell in A' that is also a local maximum of A. □

Here is the rest of the purported proof:

Lemma 3. *Given any input where $n = 2^k - 1$ for some integer $k \geq 1$, the algorithm gives a correct output.*

Proof (long form).

1. The proof is by induction on k.
2. Let $P(k)$ be the predicate "for any input with $n = 2^k - 1$, the algorithm gives a correct output."
3. For any input with $n = 1 = 2^1 - 1$, the cross X is the entire array, so Line 2 returns a local max.
4. So $P(1)$ holds.

5.1. Consider any $k \geq 2$, and any execution of the algorithm on an input A with $n = 2^k - 1$. Assume $P(k-1)$.
5.2.1. <u>Case 1.</u> Suppose the algorithm returns in Line 2.
5.2.2. By inspection of Line 2, it returns a local max of A.
5.3.1. <u>Case 2.</u> Otherwise the algorithm returns in Line 7.
5.3.2. The quadrant A' contains a local max of A (by Lemma 2). Note $(n-1)/2 = (2^k - 2)/2 = 2^{k-1} - 1$.
5.3.3. By Step 5.1, $P(k-1)$ holds, so the recursive call on A' returns a local max of A'.
5.3.4. By the previous two steps, the algorithm (Line 7) returns a local max of A.
5.4. So, in either case, the algorithm returns a local max of A. That is, it is correct on input A.

6. By Block 5, for any $k \geq 2$, if $P(k-1)$ holds, then $P(k)$ holds.
7. By Step 4, $P(1)$ holds.
8. By the previous two steps and induction, $P(k)$ holds for all $k \geq 1$.
9. That is, the algorithm gives the correct output on any input where $n = 2^k - 1$ for some integer $k \geq 1$. □

In the proof of correctness (the two lemmas), what is the *first* step in the long-form proof of either lemma that asserts a statement that is not necessarily true? What, precisely, is the mistake in the reasoning in that line? Give an input with $n = 7$ on which the 2D Local Max algorithm given above fails. Describe the execution of the algorithm on that input. We encourage you to verify your answer using the Python implementation at this URL: `https://repl.it/repls/ParchedAttachedBase`.

Problem 2 answer

(a) Where is the first step that makes an assertion that is not necessarily true?
In Lemma 3 , Step 5.3.4 .

(b) In one or two sentences, what precisely is wrong with the reasoning given in that step?
The algorithm finds the maximum value in X to move to quadrant A', however, because this calls the algorithm recursively on the quadrant, the values in X are no longer considered to be neighbors of the edge values of A. As a result, if the algorithm moved into a quadrant and the recursive call found a local max in that quadrant on the new cross which is adjacent the original X (thus it's a local max of A'), then it isn't necessarily a local max of A, as A includes X that's adjacent to the local max found of A' (and A' doesn't consider the adjacent X as a neighbor due to the recursive call only being on A'), which could be larger that this local max found in A'.

(c) Here is an input with $n = 7$ on which the algorithm fails to return a local maximum:

$$\begin{array}{ccc|c|ccc}
2 & 2 & 2 & 1 & 2 & 2 & 2 \\
2 & 2 & 2 & 1 & 2 & 2 & 2 \\
2 & 2 & 2 & 1 & 2 & 4 & 2 \\
\hline
1 & 1 & 1 & 0 & 1 & 3 & 1 \\
\hline
2 & 2 & 2 & 1 & 1 & 2 & 1 \\
2 & 2 & 2 & 1 & 1 & 1 & 1 \\
2 & 2 & 2 & 4 & 5 & 1 & 1
\end{array}$$

Here is what the algorithm does when given that input:

1. The algorithm checks each value in the middle row or column to see if it's a local max.
2. Since none of them are a local max, it picks the largest value, 4 to be A[i,j] and then since it isn't a local max, sets A[i',j'] to be the 5 next to it, as $5 > 4$.
3. The algorithm then recurs on the 3x3 quadrant A' which includes the 5.
4. The cross of A' is checked to see if any of the values are a local max, and because the 2 is (at row 5, col 6 or A[4,5]), the algorithm returns this 2 at A[4,5] as the local max.
5. This is incorrect, because in A the top neighbor of that 2, at A[3,5], is 3, and since $3 > 2$, the 2 is not a local max of A, and the algorithm has returned an incorrect value for the local max. ⋮

Problem 3. You have been commissioned to design a new bus system that will run along Huntington Avenue. The bus system must provide service to N stops (numbered $1, 2, \ldots, N$) along the eastbound route. Commuters may begin their trip at any stop i and end at any stop j where $1 \leq i < j \leq N$. Here are two naive ways to design the system:

- You could have a bus run from stop 1 to stop N, stopping at every stop in between. The system would be cheap because it would only require $N-1$ route segments, $\{1 \to 2, 2 \to 3, \ldots, (N-1) \to N\}$, for the entire system. However, a person traveling from stop 1 to stop N would have to wait while the bus makes $N-1$ stops.

- You could have, for every pair of stops i, j with $i < j$, a special express bus from i to j. No commuter would ever have to make more than one stop. However, this system would be expensive as it requires $\binom{N}{2} = \Theta(N^2)$ route segments, $\{i \to j : 1 \leq i < j \leq N\}$.

Using divide and conquer, you will find a compromise — a set of at most $N \log_2 N$ route segments, with the property that for any stops i and $j > i$, the *hop length* from i to j in the set is at most 2. That is, either the (direct) route segment $i \to j$ is in the set (so the hop length is 1), or, for some m with $i < m < j$, the set contains the two segments $i \to m$ and $m \to j$ (so the hop length is 2).

Consider any sequence $(\ell, \ell+1, \ldots, u)$ of consecutive stops (where $1 \leq \ell \leq u \leq N$). Let $n = u - \ell + 1$ be the number of stops. We will recursively define a set $S(\ell, u)$ of segments that handles trips from any stop i to any stop j such that $\ell \leq i < j \leq u$.

For the base case $n = 1$, for every stop ℓ, define $S(\ell, \ell)$ to be the empty set. For the base case $n = 2$, for every stop ℓ, define $S(\ell, \ell+1)$ to be $\{\ell \to \ell+1\}$. (Then any commuter can get from stop ℓ to $\ell+1$ using just the one segment in $S(\ell, \ell+1)$, so the hop length from ℓ to $\ell+1$ in $S(\ell, \ell+1)$ is 1.)

Now consider any sequence $(\ell, \ell+1, \ldots, u)$ of stops where the number of stops $n = u - \ell + 1$ is more than two. Let $m = \lfloor (\ell + u)/2 \rfloor$ be the middle stop.

(a) Give a definition for a set $Q(\ell, u)$ of segments such that, for every pair of stops (i, j) where i is in the first half of the stops ($\ell \leq i \leq m$) and j is in the second half of the stops ($m < j \leq u$), the hop length from i to j in $Q(\ell, u)$ is at most 2. Your set $Q(\ell, u)$ should contain at most $n - 1$ segments (the number of stops minus one). This definition should not be recursive.

(b) Now give a recursive definition for a set $S(\ell, u)$ of segments such that

(P1) For *any* pair of stops (i, j) with $\ell \leq i < j \leq u$, the hop length from i to j in $S(\ell, u)$ is at most two.

(P2) $S(\ell, u)$ contains at most $n \log_2 n$ segments.

Hint: Divide the stops into the first and second halves, recurse on each half, then add $Q(\ell, u)$.

(c) Let $N(n)$ be the maximum number of segments in any set $S(\ell, u)$ such that $u - \ell + 1 = n$ (the set covers n consecutive stops). State a recurrence relation for $N(n)$. Use the recurrence to prove by induction that $N(n) \leq n \log_2 n$ for every $n \geq 1$.

(d) Prove by induction that, for all $n \geq 1$, for any sequence $(\ell, \ell+1, \ldots, u)$ of n consecutive stops (so $u - \ell + 1 = n$), the set $S(\ell, u)$ as you defined it has property (P1) above.

(continued)

Problem 3 answer Consider any sequence $(\ell, \ell+1, \ldots, u)$ of more than two stops, with middle stop $m = \lfloor(\ell+u)/2\rfloor$.

(a) Define $Q(\ell, u)$ as follows:

$$Q(\ell, u) = \{i \to m : \ell \leq i \leq m = \lfloor\frac{\ell+u}{2}\rfloor\} \cup \{j \to m : m = \lfloor\frac{\ell+u}{2}\rfloor < j \leq u\}$$

Then $Q(\ell, u)$ has at most $n-1 = \ell - u$ segments, and for any (i, j) with $\ell \leq i \leq m \leq j \leq u$ the hop length from i to j in $Q(\ell, u)$ is at most two.

(b) Define $S(\ell, u)$ as follows:

$$S(\ell, u) = Q(\ell, u) + S(\ell, m-1) + S(m+1, u)$$

(continued)

Problem 3(c) answer Here is the recurrence: $N(1) = 0$, $N(2) = 1$, and, for any $n \geq 2$,

$$N(n) = (n-1) + N(\lfloor \frac{n-1}{2} \rfloor) + N(\lceil \frac{n-1}{2} \rceil)$$

Lemma 4. *$N(n) \leq n \log_2 n$ for every $n \geq 1$.*

Proof (long form).

1. The proof is by induction on n.
2. The base case $n = 1$ holds because each set $S(\ell, \ell)$ has no segments, and $n \log_2 n$ is zero.
3. The base case $n = 2$ holds because each set $S(\ell, \ell + 1)$ has one segment, and $n \log_2 n$ is one.

4.1. Consider any $n \geq 3$ and assume the lemma holds for smaller values. We show it holds for n.

4.2. Recall the recurrence relation, $N(n)$ is $(n-1) + N(\lfloor \frac{n-1}{2} \rfloor) + N(\lceil \frac{n-1}{2} \rceil)$

4.3. Applying the inductive assumption in Step 4.1

$$N(n) \leq (n-1) + \frac{n-1}{2} log_2(\frac{n-1}{2}) + \frac{n-1}{2} log_2(\frac{n-1}{2})$$

4.4. Algebra: $(n-1) + \frac{n-1}{2} log_2(\frac{n-1}{2}) + \frac{n-1}{2} log_2(\frac{n-1}{2}) = (n-1)(1 + (\frac{1}{2})(log_2(\frac{n-1}{2}) + log_2(\frac{n-1}{2})) = (n-1)(1+(\frac{1}{2})(log_2(\frac{n-1}{2})^2) = (n-1)(1+log_2(\frac{n-1}{2})) = (n-1)(log_2 2+log_2(\frac{n-1}{2})) = (n-1)log_2(n-1) \leq nlog_2 n$.

5. By Block 4, for any $n \geq 3$, the lemma holds for that n if it holds for smaller values.
6. By this, Steps 2 and 3, and induction, the lemma holds for all $n \geq 1$. $\square$

(continued)

Problem 3(d) answer

Lemma 5. *For any $n \geq 1$, for any n consecutive stops $(\ell, \ell+1, \dots, u)$, for every pair of stops (i, j) with $\ell \leq i < j \leq u$, the hop length from i to j in $S(\ell, u)$ is at most 2.*

Proof (long form).

1. The proof is by induction on n.
2. For the base case $n = 1$, there is no such pair (i, j) so the lemma holds trivially.
3. For the base case $n = 2$, there is one such pair $(\ell, \ell+1)$. The lemma holds because the hop length from ℓ to $\ell + 1$ in $S(\ell, \ell + 1)$ is one (consecutive stops).

4.1. Consider any $n \geq 3$ and (ℓ, u) as in the lemma, and assume the lemma holds for smaller values.
4.2. Let $m = \lfloor (\ell + u)/2 \rfloor$ be the middle stop.
4.3. Recall that by definition $S(\ell, u)$ is Q$(\ell, u) + S(\ell, m - 1) + S(m + 1, u)$

4.4.1. Consider any pair of stops (i, j) with $\ell \leq i < j \leq u$.
4.4.2.1. Case 1. First consider the case that $\ell \leq i \leq m = \lfloor \frac{\ell+u}{2} \rfloor < j \leq u$
4.4.2.2. By the definition of $Q(\ell, u)$, the hop length from i to j is at most two in $Q(\ell, u)$.
4.4.2.3. So the hop length from i to j is at most two in $S(\ell, u)$.
4.4.3.1. Case 2. Next consider the case that $\ell \leq i < j \leq m = \lfloor \frac{\ell+u}{2} \rfloor < u$
4.4.3.2. If j = m, there is already a segment connecting i and j by definition of $Q(\ell, u)$. Else, the algorithm gets called recursively : $S(\ell, m - 1)$ Applying the inductive assumption, we know there is a maximum hop length of 2 in this subsection of the consecutive stops.
4.4.3.3. So the hop length from i to j is at most two in $S(\ell, u)$.
4.4.4.1. Case 3. In the remaining case $\ell \leq m = \lfloor \frac{\ell+u}{2} \rfloor < i < j \leq u$
4.4.4.2. The algorithm gets called recursively : $S(m + 1, u)$ Applying the inductive assumption, we know there is a maximum hop length of 2 in this subsection of the consecutive stops.
4.4.4.3. So the hop length from i to j is at most two in $S(\ell, u)$.
4.4.5. In any case, the hop length from i to j in $S(\ell, u)$ is at most two.

4.5. By Block 4.4, for all (i, j) with $\ell \leq i < j \leq u$, the i-to-j hop length in $S(\ell, u)$ is at most two.
5. By Block 4, for any $n \geq 3$, the lemma holds for n if it holds for smaller values.
6. By this, Steps 2 and 3, and induction, the lemma holds for all $n \geq 1$. $\square$