TCSS 343 - Assignment 2

Jake McKenzie

July 16, 2018

(3 points) 1. Below is a self-reduction for the `MAX` problem. State a recursive algorithm using pseudocode for finding the maximum element based on this self-reduction.

Algorithm 1 Find Max integer in an Array with simple recursion

```
1: procedure FIND MAX(A)
2:     if (a == b) then
3:         return A[a]
4:     else if (a < b) then
5:         return Max(A[a],Find Max(A[a + 1]))
6:     end if
7: end procedure
8: procedure MAX(a, b) return (a < b) ? b : a
9: end procedure
```

(6 points) 2. Using the same reduction as part 1 now state a recurrence $T(n)$ that expresses the worst case run time of the recursive algorithm. Find a similar recurrence in your notes and state the tight bound on $T(n)$.

Line 3 makes 1 amount of operations while line 5 makes $T(n-1)$, this is because there are $n-1$ amount of comparisons to check for the max in the recurrence for when this list is greater than 1. **Note**: Consistency of whether the constant amount of operations is writen as 1 or $O(1)$ are inconsistent so I went with the notation I've seen the most used often.

$$T(n) = \begin{cases} 1 & \text{if } n = 1 \\ T(n-1) + 1 & \text{if } n > 1 \end{cases}$$

Claim: $\forall n > 0$, the running time of *Find Max* $\epsilon O(n)$. We consder the recurrance relation above.

1. Base Case:

$$n = 1; T(1) = 1$$

2. Inductive Hypothesis:

$$T(k) = \begin{cases} 1 & \text{if } k = 1 \\ T(k-1) + 1 & \text{if } k > 1 \end{cases}$$

Assume for an arbitrary $k, T(k) \leq k$

3. Inductive Step:

$$\text{if } k > 1$$
$$T(k+1) = T(k) + 1$$
$$T(k+1) = T(k-1) + 1 + 1$$
$$T(k+1) = k + 2$$
$$T(k+1) \epsilon O(k)$$
$$\therefore$$
$$T(k) \epsilon O(k)$$

(9 points) 3. Below is a self-reduction for the `MAX` problem. State a recursive algorithm using pseudocode for finding the maximum element based on this self-reduction.

$$M(A[a \dots b]) = \begin{cases} -\infty & \text{if } a > b \\ A[a] & \text{if } a = b \\ \max(M(A[a \dots t_1]), \max(M(A[t_1 + 1 \dots t_2]), M(A[t_2 + 1 \dots b]))) & \text{if } a < b \end{cases}$$

Algorithm 2 Find Max integer in an Array with 3-Way Split

```
1: procedure FIND MAX(A, a, t1, t2)
2:     if (a > b) then
3:         return 0x7FFFFFFF
4:     else if (a == b) then
5:         return A[a]
6:     else if (a < b) then
7:         return Max(FindMax(A,a+1,t1,t2), Max(FindMax(A, a, t1+1,t2), FindMax(A, a,
   t1,t2+1]))
8:     end if
9: end procedure
10: procedure MAX(a, b) return (a < b) ? b : a
11: end procedure
```

For what it's worth, I don't think this algorithm will find the max element if it is contained within the first third of the array(we never iterate over the first third elements, only the second and last third), but it does match the self-reduction.

(7 points) 4. Using the same reduction as part 3 now state a recurrence $T(n)$ that expresses the worst case run time of the recursive algorithm. You do not need to formally prove your recurrence, but you have to show that it is a reasonable guess by using a recursion tree or by the repeated substitution method. *Hint: assume that n is a power of 3.*

I assumed, for purposes of the problem, that we did iterate through each third of the array evenly and like you said that $n = 3^{\psi}$. Intuitively I expect this to be a worst case

linear runtime, which case each last element of the subarrays would be a local max but let propose a semi-formal argument for such. Let us assume the following recurrence reduction.

$$T(n) = \begin{cases} 1 & \text{if } n = 1 \\ 3T(\frac{n}{3}) + d & \text{if } k > 1 \end{cases}$$

if $k > 1$ and $let n = 3^{\psi}$

$$\begin{aligned} T(n) &= 3T(\frac{n}{3}) + d \\ T(n) &= 3T(3^{\psi-1}) + 3^{\psi} + d \\ T(n) &= 3T(3^{\psi-2}) + 3^{\psi-1} + 3^{\psi} + d \\ T(n) &= 3T(3^{\psi-3}) + 3^{\psi-2} + 3^{\psi-1} + 3^{\psi} + d \\ &\vdots \\ T(n) &= 3^0 + \cdots + 3^{\psi-2} + 3^{\psi-1} + 3^{\psi} + d \\ T(n) &= \sum_{\psi=0}^{k} 3^{\psi} + d \\ T(n) &= \frac{3^{k+1} - 1}{2} + d \\ T(n) &= \frac{3n - 1}{2} + d \\ T(n) &= \epsilon O(n) \end{aligned}$$

(12 points) 1. State two different self-reductions for the `SUM` problem. Use the self-reduction examples from the lectures as a guide.

For this problem I decided to use divide and conquer for the second self-reduction and just normal recursion for the second. I don't think these are good ways to sum over an array, there are a lot of wasted computations but it is quite readable. The first algorithm simply completes a summation on the array in reverse order. The second algorithm sums the even and odd elements at the same time.

$$T(n) = \begin{cases} 1 & \text{if } n = 1 \\ T(n-1) + 1 & \text{if } k > 1 \end{cases}$$

$$T(n) = \begin{cases} 1 & \text{if } n = 1 \\ 2T(n-2) + 1 & \text{if } k > 1 \end{cases}$$

(12 points) 2. Give recursive algorithms based on your divide-and-conquer self-reductions to solve the `SUM` problem.

Algorithm 3 Find Sum of an Array via Simple Recursion

```
1: procedure FIND SUM(A, n)
2:     if (n ≤ 0) then
3:         return 0
4:     else
5:         return FindSum(A, n - 1) + A[n - 1]
6:     end if
7: end procedure
```

Algorithm 4 Find Sum of an Array via Divide and Conquer

```
1: procedure FIND SUM(A, n)
2:     if (n ≤ 0) then
3:         return 0
4:     else
5:         return FindSum(A, n - 2) + A[n - 1] + A[n - 2]
6:     end if
7: end procedure
```

(6 points) 3. What are the worst-case runtimes of the solutions you have generated. (Just state the runtimes. You do not need to show your work.)

Both algorithm operate in linear time with a worst case runtime of $O(n)$. In fact I can make the argument that both functions are $\Theta(n)$. I came to this conclusion from reading Tim Roughgarden's Algorithms Illuminated page 56-57 where he states then proves theorem $max\{f(n), g(n)\} = \Theta(f(n) + g(n))$. Both algorithms are bounded by some constant multiple of n, while algorithm 4 operates slightly slower than 3, they both follow this fact. If we take the max of the two, which would be algorithm 4, that still operates in linear time. From this information I came to the firm conclusion that both algorithms were operating in $\Theta(n)$. I hope this makes sense to you. I know it's a roundabout way of thinking about this problem but it's what helped me come to a conclusion so I thought I would include it.

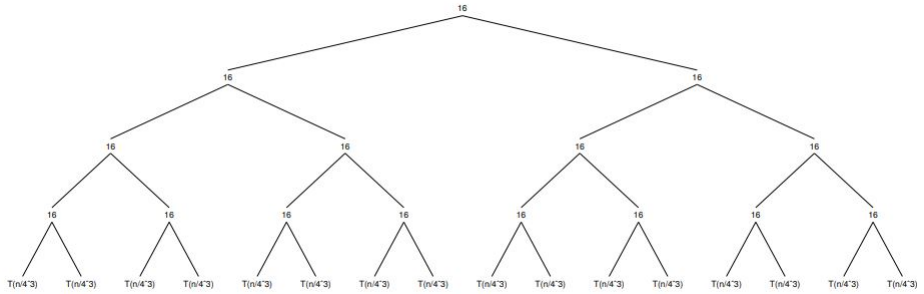
Consider the following recurrence $T(n)$:

$$T(n) = \begin{cases} c & \text{if } n = 1 \\ 2T(\lfloor \frac{n}{4} \rfloor) + 16 & \text{if } n > 1 \end{cases}$$

(10 points) 1. Use the recursion tree or repeated substitution method to come up with a good guess for a bound $g(n)$ on the recurrence $T(n)$.

Consider the following $2T(\lfloor \frac{n}{4} \rfloor) + 16 \leq 2T(\frac{n}{4}) + 16$ which is always true. From

my readings, floor functions are ignored for the conveince of proofs until they matter. I don't like doing that but I don't see any other way to prove this.



This is the recursion tree I obtained. Let the depth of the tree be ∇ and number of nodes be unknown for now, by looking at the tree I could discern that the level was 2^{∇} and the input for the recursive function was $\frac{n}{4^{\nabla}}$. So this is interesting, our inputs are decreasing faster than our levels are splitting. We will stop when our input becomes the identity, which means to find the number of levels we need to solve for ∇ in the equation $\frac{n}{4^{\nabla}} = 1$. The level stops at $\nabla = \log_4 n$. My guest for $g(n)$ will be that it is $\sum_{psi=0}^{\log_4 n-1} 2^{\psi} + 16\sqrt{n}$. My reasoning for this is that the cost we're adding up 16 a $\sqrt{n}$ times since that is the number of nodes and the summation comes from properties of trees representing everything else in the tree EXCEPT the leaves. I added the minus 1 in there to distinguish the leaves from everything else.

$$
\begin{aligned}
g(n) &= \sum_{\psi=0}^{\log_4 n-1} 2^{\psi} + 16\sqrt{n} \\
g(n) &= \frac{2^{\log_4 n-1+1} - 1}{2-1} + 16\sqrt{n} \\
g(n) &= 4^{\frac{1}{2}(\log_4 n-1+1)} - 1 + 16\sqrt{n} \\
g(n) &= 4^{\log_4 \sqrt{n-1}+\frac{1}{2})} - 1 + 16\sqrt{n} \\
g(n) &= \sqrt{n-1} + 2 - 1 + 16\sqrt{n} \\
g(n) &= 16\sqrt{n} + \sqrt{n-1} + 1 \\
g(n) &\epsilon \Theta(\sqrt{n})
\end{aligned}
$$

You didn't say to include this but I've been reading ahead. To check my answer I used two parts from a book I've been reading, Tim Roughgarden's Algorithms Illuminated pages 95-96 included on the next.

Standard Recurrence Format

Base case: $T(n)$ is at most a constant for all sufficiently small n.[4]

General case: for larger values of n,

$$T(n) \leq a \cdot T\left(\frac{n}{b}\right) + O(n^d).$$

Parameters:

- a = number of recursive calls
- b = input size shrinkage factor
- d = exponent in running time of the "combine step"

The base case of a standard recurrence asserts that once the input size is so small that no recursive calls are needed, the problem can be solved in $O(1)$ time. This will be the case for all the applications we consider. The general case assumes that the algorithm makes a recursive calls, each on a subproblem with size a b factor smaller than its input, and does $O(n^d)$ work outside these recursive calls. For

[3]This presentation of the master method draws inspiration from Chapter 2 of *Algorithms*, by Sanjoy Dasgupta, Christos Papadimitriou, and Umesh Vazirani (McGraw-Hill, 2006).

[4]Formally, there exist positive integers n_0 and c, independent of n, such that $T(n) \leq c$ for all $n \leq n_0$.

Theorem 4.1 (Master Method) *If $T(n)$ is defined by a standard recurrence, with parameters $a \geq 1$, $b > 1$, and $d \geq 0$, then*

$$T(n) = \begin{cases} O(n^d \log n) & \text{if } a = b^d \quad \textit{[Case 1]} \\ O(n^d) & \text{if } a < b^d \quad \textit{[Case 2]} \\ O(n^{\log_b a}) & \text{if } a > b^d \quad \textit{[Case 3].} \end{cases} \tag{4.2}$$

[5]There are also the constants suppressed in the base case and in the "$O(n^d)$" term, but the conclusion of the master method does not depend on their values.

[6]There are more general versions of the master method that accommodate a wider family of recurrences, but the simple version here is sufficient for almost any divide-and-conquer algorithm you're likely to encounter.

By the Master's Method I obtain $a = 2$, $b = 4$, $d = 0$ which gives us Case 3 since $a > b^d$. This gives me $O(n^{\log_4 2}) = O(\sqrt{n})$. I have seen in other books make a stronger statement, saying that the function is Θ not O with the master method but I trust Roughgarden more than the other sources I've found. Roughgarden has a good course on youtube in algorithmic game theory that I've taken notes from and has been teaching at Stanford and Coursera for many years.

Let us now attempt to prove, by induction, that $g(n) \epsilon \Theta(\sqrt{n})$. Let us assume that $g(n) = 16\sqrt{n} + \sqrt{n-1} + 1$.

Base Case

$$
\begin{aligned}
g(1) &= 16\sqrt{1} + \sqrt{1-1} + 1 \\
g(1) &= 16 + 0 + 1 \\
g(1) &= 17 \\
g(1) &= C
\end{aligned}
$$

For the case of argument, I don't know whether $C = 17$. No domain was given for C but it would seem plausible to me that the base case has been satisfied.

Inductive Hypothesis

$$
\begin{aligned}
T(n) &= 2^1 T(\frac{n}{4^1}) + 16 \\
T(n) &= 2^2 T(\frac{n}{4^2}) + 16 \\
T(n) &= 2^3 T(\frac{n}{4^3}) + 16 \\
T(n) &= 2^4 T(\frac{n}{4^4}) + 16 \\
&\vdots \\
T(n) &= 2^{\nabla} T(1) + 16 \\
g(n) &= 16\sqrt{n} + \sqrt{n-1} + 1 \\
g(n) &\epsilon \Theta(\sqrt{n})
\end{aligned}
$$

Inductive Step

$$
\begin{aligned}
T(n+1) &= 2^{\nabla+1} T(0) + 16 \\
T(n+1) &= 2(16\sqrt{n}) + 16 \\
T(n+1) &= 32\sqrt{n} + 16 \\
\lim_{n\to\infty} \frac{32\sqrt{n}+16}{\sqrt{n}} &\to 32 \\
&\therefore g(n) \epsilon \Theta(\sqrt{n})
\end{aligned}
$$

The series above was generated using mathematica and is an example of a puiseaux series. I don't like dealing with roots and I especially don't like taking limits of them so when I can I use mathematica to perform tranformations of them to a more usuable form. I believe by the nature of asymptotics I've answered the next two questions. This isn't how it was done in class, you proved both seperately but I look my notes and they're confusing.

I included simplifying of the root in fear of losing points due to not showing my

work, but I used the series originally to solve the problem.

(5 points) 2. State and prove by induction a theorem showing $T(n) \in O(g(n))$.

(5 points) 3. State and prove by induction a theorem showing $T(n) \in \Omega(g(n))$.

$$T(n) = \begin{cases} c & \text{if } n = 1 \\ T(\lfloor \frac{n}{2} \rfloor) + T(\lfloor \frac{n}{4} \rfloor) + 4n & \text{if } n > 1 \end{cases}$$

Before I present my recursion tree, let me for the sake of argument, state my assumptions: $2T(\frac{n}{4}) + 4n \leq T(\lfloor \frac{n}{2} \rfloor) + T(\lfloor \frac{n}{4} \rfloor) + 4n \leq 2T(\frac{n}{2}) + 4n$ since it is safe to assume that $T(n/4) \ll T(n/2)$. By the master method I expect that $g(n)$ should be sandwhiched in the following way $O(n) \ll g(n) \ll O(n \log n)$

(5 points) 1. Draw the first six levels of the recursion tree by drawing all recursive calls of the same size on the same level. Make sure on each level you indicate the size of the recursive call and the number of recursive calls.

The recursion tree is in an attached document, labeled "recursiontree.pdf". I did not include that in this document because I could not get the LaTeX to format properly.

I'll state the levels here as joining them to the image is a lot of work.
Level 0: $4n$
Level 1: $4(\frac{n}{2^1} + \frac{n}{2^2}) = 3n$
Level 2: $4(\frac{n}{2^2} + \frac{n}{2^3} + \frac{n}{2^3} + \frac{n}{2^4}) = \frac{9n}{4}$
For now on I will just state the sum result. The expressions became quite long and tedious.
Level 3: $\frac{27n}{16}$
Level 4: $\frac{81n}{64}$
Level 5: $\frac{243n}{256}$
Level 6: $\frac{729n}{1024}$
I thought I was doing something wrong but asside from algebra errors, I did follow pages 99-105 of Introduction to Algorithms (CLRS) to obtain this solution.

Apparently a property of each level being the same is only true if you have a single type of recurrence in an equation. If this had been for instance just $T(\frac{n}{4}) + n$ then you would have had equal costing levels at each recursion. You do not with this example.

According to the master method, I should expect the number of leaves (L) to be somewhere sandwhiched between $\sqrt{n} < L < n$ where n is the number of nodes.

(5 points) 2. Express the cost of all levels of the recursion tree as a sum over the cost of each level of the recursion tree.

Looking back at the last page, there's clearly a linear recurrance of that. Solving

for a linear recurrence can be quite difficult, so what is it that I expect the recurrance to be? I expect the recurrence to be some expression containing $\frac{1}{2} + \frac{1}{4} = \frac{2}{4} + \frac{1}{4} = \frac{3}{4}$. That appears to indeed be the case. To solve for the overall cost I will make the assumption that this tree is geometric and follow the examples from CLRS, knowing what I know from the above linear recurrence.

(5 points) 3. Give a function $g(n)$ and show it is an upper bound on the sum.

$$T(n) \leq 4n + (\frac{3}{4})^1 n + (\frac{3}{4})^2 n + (\frac{3}{4})^3 n + (\frac{3}{4})^4 n + \dots$$
$$T(n) \leq 3n + (\frac{3}{4})^0 n + (\frac{3}{4})^1 n + (\frac{3}{4})^2 n + (\frac{3}{4})^3 n + (\frac{3}{4})^4 n + \dots$$
$$T(n) \leq 3n + \sum_{\psi=0}^{\infty} n(\frac{3}{4})^{\psi}$$
$$T(n) \leq 3n + \frac{n}{1 - \frac{3}{4}}$$
$$T(n) \leq 3n + 4n$$
$$T(n) \leq 7n$$
$$T(n) \epsilon O(n)$$

This is a good upper bound but to find the exact bound I found the Akra-Bazzi method to be much more powerful, being as it seems well suited for such problems. I know that this value obtained must be smaller than the actual function for $g(n)$ because it is a geometric series that goes to infinity. The actual geometric series for this function would be finite in nature.

First we need to solve the equation $(\frac{1}{2})^p + (\frac{1}{4})^p = 1$ obtaining $p = 0.6942419136306174$.

$$T(n) \epsilon \Theta(x^p(1 + \int_1^n u^{-p} du))$$
$$T(n) \epsilon \Theta(-2.27056 n^{0.694242} + 3.27056n)$$
$$T(n) \epsilon \Theta(n)$$

Not only is the other term lower in order but it is negative! We can safely ignore it. Akra-Bazzi seems powerful in that it can be shown that all recurrences can be shown to default to the non-recurrance term if p is greater than the power of that polynomial term quickly simplifying the work needed to be done to find the asymptotic boundaries. I learned of this method from watching Lecture 14 in MIT 6.02J Mathematics for Computer Science Fall of 2018 timestamp 1 : 02 : 45

(5 points) 4. State and prove by induction a theorem showing $T(n) \in O(g(n))$.

(5 points) 5. State and prove by induction a theorem showing $T(n) \in \Omega(g(n))$.

Let us assume that $T(n) \epsilon \Theta(n)$ where $g(n) = -2.27056 n^{0.694242} + 3.27056n$.

$$T(n) = \begin{cases} c & \text{if } n = 1 \\ T(\lfloor \frac{n}{2} \rfloor) + T(\lfloor \frac{n}{4} \rfloor) + 4n & \text{if } n > 1 \end{cases}$$

Base Case

$$\begin{aligned} T(1) &= C \\ T(1) &= -2.27056(1)^{0.694242} + 3.27056(1) \\ T(1) &= -2.27056 + 3.27056 \\ T(1) &= 1 \end{aligned}$$

Again no domain or value was given for C but assuming that $C = 1$ we can safely assume that the base case was satisfied.

Inductive Hypothesis

$$\begin{aligned} T(n) &= 2^{\nabla} T(1) + 0 + 4n \\ g(n) &= -2.27056n^{0.694242} + 3.27056n \forall n > 1 \\ g(n) &\epsilon \Theta(n) \end{aligned}$$

Inductive Step

If we look at our tree in the adjacent pdf, the leaves do not terminate in a uniform manner. This makes the problem illsuited for induction, but alas we will try.

$$\begin{aligned} T(n+1) &= 2^{\nabla+1} T(0) + 0 + 4(n+1) \\ T(n+1) &= 2(2^{\log n} T(1)) + 4n + 4 \\ T(n+1) &= 2(n * 1) + 4n + 4 \\ T(n+1) &= 6n + 4 \\ T(n+1) &\epsilon \Theta(n) \end{aligned}$$

To assess my work let's recap. I've shown now that $3.27056n \leq g(n) \leq 6n < 7n$. Clearly it is a function of n and grows linearly. From Akra-Bazzi I know that if $p <$ power of the additional term, which in this case is true, $0.6942419136306174 < 1$ then we know that the n term will dominate in the expression for $T(n)$. The reccurence terms fall away for large n.