Solving Recurrences

CS 234

0 Introduction

This document contains examples of good solutions for some recurrence problems. These are not the only ways to explain the solution of such recurrences.

These proofs may contain footnotes explaining different thought processes that occurred in their construction, to help show you how to think about writing proofs. Commentary may also be provided at the end about alternative approaches.

1 Proofs

1

Theorem 1. *If the work recurrence $T(n)$ is defined as follows for some $c > 0$*

$$T(n) = \begin{cases} c & n \leq 1 \\ 2 \cdot T(n/2) + c \cdot n & n > 1 \end{cases}$$

then $T(n) \in O(n \cdot log_2(n))$

Proof. Consider unrolling the call tree for $T(n)$. This tree will have:

- $log_2(n) + 1$ rows because the argument is divided by 2 until it is less than or equal to 1
- 2^r nodes in row r (0-indexed) because having 2 recursive call children from each node doubles the number of nodes each row.
- $c\frac{n}{2^r}$ work done in each node of row r, because the original argument n will have been divded by 2 a total r times before it is an argument to a row-r recursive call, and that argument scaled by c is the non-recursive work done
- c work done in each node of the last row specifically

Putting these values together, the work done outside of the last row is $\sum_{r=0}^{log_2(n)-1} 2^r \cdot c\frac{n}{2^r}$, and the work done in the last row is $2^{log_2(n)} \cdot c$. The total work can then be algebraically simplified as follows:

$$\begin{aligned} \left(\sum_{r=0}^{log_2(n)-1} 2^r \cdot c\frac{n}{2^r}\right) + 2^{log_2(n)} \cdot c &= \left(\sum_{r=0}^{log_2(n)-1} c \cdot n\right) + c \cdot n \\ &= \sum_{r=0}^{log_2(n)} c \cdot n \\ &= (log_2(n) + 1) \cdot c \cdot n \\ &= c \cdot n \cdot log_2(n) + c \cdot n \end{aligned}$$

Since the total work $c \cdot n \cdot log_2(n) + c \cdot n$ is always less than or equal to $2c \cdot n \cdot log_2(n)$ for $n \geq 1$, and c is just a constant, it follows that $T(n) = c \cdot n \cdot log_2(n) + c \cdot n \in O(n \cdot log_2(n))$. □

Note that, unlike some other recurrences in these sample proofs, this recurrence does *not* end up with a constant-bounded geometric series. Instead, this recurrence ended up with a sum of terms that was independent of the sum's index variable r, so the summation turned into multiplication. Not all recurrences work out the same way!

Here is an alternative proof using induction:

Proof. This asymptotic bound can be shown by inducting with the inductive predicate $P(n) := T(n) \leq 2c \cdot n \cdot log_2(n)$.

Base Case $n = 2$ If $n = 2$, then $T(n)$ satisfies the following chain of equalities:

$$\begin{aligned}
T(2) &= 2 \cdot T(1) + c \cdot 2 && \textit{T def} \\
&= 2 \cdot c + c \cdot 2 && \textit{T def} \\
&= 4 \cdot c && \textit{algebra} \\
&= 2c \cdot 2 \cdot log_2(2) && \textit{algebra}
\end{aligned}$$

Thus $T(2) \leq 2c \cdot 2 \cdot log_2(2)$, which satisfies $P(2)$.

Inductive Case Suppose that $P(j)$ holds for all $0 \leq j \leq k$ for an arbitrary k. We now want to show $P(k+1)$.

Consider $T(k+1)$. This value satisfies the following chain of inequalities:

$$\begin{aligned}
T(k+1) &= 2 \cdot T(\frac{k+1}{2}) + c \cdot (k+1) && \textit{T def} \\
&\leq 4c \cdot \frac{k+1}{2} \cdot log_2(\frac{k+1}{2}) + c \cdot (k+1) && \textit{IH} \\
&= 2c(k+1)(log_2(k+1) - 1) + c \cdot (k+1) && \textit{algebra} \\
&= 2c(k+1)log_2(k+1) - 2c(k+1) + c \cdot (k+1) && \textit{algebra} \\
&= 2c(k+1)log_2(k+1) - c(k+1) && \textit{algebra} \\
&\leq 2c(k+1)log_2(k+1) && c(k+1) \geq 0
\end{aligned}$$

Thus $T(k+1) \leq 2c(k+1)log_2(k+1)$, which satisfies $P(k+1)$.

Conclusion As a result of induction, we find that $P(n)$ holds for all $n \geq 2$. That is, $\forall n \geq 2. T(n) \leq 2c \cdot n \cdot log_2(n)$. By definition, it is therefore the case that $T(n) \in O(n \cdot log_2(n))$.

□

2

Theorem 2. *If the work recurrence $T(n)$ is defined as follows for some $c > 0$*

$$T(n) = \begin{cases} c & n \leq 1 \\ 3 \cdot T(n/2) + c \cdot n & n > 1 \end{cases}$$

then $T(n) \in O(n^{log_2(3)})$

Proof. Consider unrolling the call tree for $T(n)$. This tree will have:

- $log_2(n) + 1$ rows because the argument is divided by 2 until it is less than or equal to 1
- 3^r nodes in row r (0-indexed) because having 3 recursive call children from each node triples the number of nodes each row.
- $c\frac{n}{2^r}$ work done in each node of row r, because the original argument n will have been divded by 2 a total r times before it is an argument to a row-r recursive call, and that argument scaled by c is the non-recursive work done
- c work done in each node of the last row specifically

Putting these values together, the work done outside of the last row is $\sum_{r=0}^{log_2(n)-1} 3^r \cdot c\frac{n}{2^r}$, and the work done in the last row is $3^{log_2(n)} \cdot c$. The total work can then be algebraically simplified as follows:

$$\begin{aligned}
\left(\sum_{r=0}^{log_2(n)-1} 3^r \cdot c\frac{n}{2^r}\right) + 3^{log_2(n)} \cdot c &= \left(\sum_{r=0}^{log_2(n)-1} c \cdot n \cdot \left(\frac{3}{2}\right)^r\right) + c \cdot n^{log_2(3)} \\
&= c \cdot n \cdot \left(\sum_{r=0}^{log_2(n)-1} \left(\frac{3}{2}\right)^r\right) + c \cdot n^{log_2(3)} \\
&= c \cdot n \left(\frac{(\frac{3}{2})^{log_2(n)} - 1}{\frac{3}{2} - 1}\right) + c \cdot n^{log_2(3)} \\
&= 2c \cdot n \left(\frac{3^{log_2(n)}}{n} - 1\right) + c \cdot n^{log_2(3)} \\
&= 2c \cdot 3^{log_2(n)} - 2c \cdot n + c \cdot n^{log_2(3)} \\
&= 3c \cdot n^{log_2(3)} - 2c \cdot n
\end{aligned}$$

Since the total work $3c \cdot n^{log_2(3)} - 2c \cdot n$ is always less than or equal to $3c \cdot n^{log_2(3)}$ [1] for $n \geq 1$, and c is just a constant, it follows that $T(n) = 3c \cdot n^{log_2(3)} - 2c \cdot n \in O(n^{log_2(3)})$.

□

[1] Note that $log_2(3)$ is approximately 1.585.

3

Theorem 3. *If the work recurrence $T(n)$ is defined as follows:*

$$T(n) = \begin{cases} 0 & n \leq 1 \\ 2 \cdot T(n/4) + n & n > 1 \end{cases}$$

then $T(n) \in O(n)$

Proof. Consider unrolling the call tree for $T(n)$. This tree will have:

- $log_4(n) + 1$ rows because the argument is divided by 4 until it is less than or equal to 1
- 2^r nodes in row r (0-indexed) because having 2 recursive call children from each node doubles the number of nodes each row.
- $\frac{n}{4^r}$ work done in each node of row r, because the original argument n will have been divded by 4 a total r times before it is an argument to a row-r recursive call, and that argument is precisely the non-recursive work done

Because 0 work is done in the last row, we can ignore the last row and consider there to only be $log_4(n)$ rows.

Putting these values together, the work done is $\sum_{r=0}^{log_4(n)-1} 2^r \cdot \frac{n}{4^r}$, which can be algebraically simplified as follows:[2]

$$\begin{aligned}
\sum_{r=0}^{log_4(n)-1} 2^r \cdot \frac{n}{4^r} &= \sum_{r=0}^{log_4(n)-1} \frac{n}{2^r} \\
&= n \cdot \sum_{r=0}^{log_4(n)-1} \frac{1}{2^r} \\
&= n \frac{(\frac{1}{2})^{log_4(n)} - 1}{\frac{1}{2} - 1} \\
&= 2n \left(1 - \left(\frac{1}{2}\right)^{log_4(n)}\right) \\
&= 2n(1 - 4^{-\frac{1}{2} log_4(n)}) \\
&= 2n(1 - \frac{1}{\sqrt{n}}) \\
&= 2n - 2\sqrt{n}
\end{aligned}$$

Since the total work $2n - 2\sqrt{n}$ is always less than or equal to $2n$ for $n \geq 1$, it follows that $T(n) = 2n - 2\sqrt{n} \in O(n)$. □

[2]This simplification makes use of the geometric series rule: $\sum_{i=0}^{n} p^i = \frac{p^{n+1}-1}{p-1}$. Note that if $0 < |p| < 1$, then this sum is always in $\Theta(1)$.

Technically $T(n) \in \Theta(n)$, but I only care that you verify the upper bound for each asymptotic, and that those asymptotics are tight are chosen to be tight (here n as opposed to, e.g., n^2). This proof verifies $O(n)$, but not $\Omega(n)$, and that is sufficient for what I will ask you.

4

Theorem 4. *If the work recurrence $T(n)$ is defined as follows:*

$$T(n) = \begin{cases} 0 & n \leq 1 \\ 2 \cdot T(n/2) + n^2 & n > 1 \end{cases}$$

then $T(n) \in O(n^2)$

Proof. Consider unrolling the call tree for $T(n)$. This tree will have:

- $log_2(n) + 1$ rows because the argument is divided by 2 until it is less than or equal to 1
- 2^r nodes in row r (0-indexed) because having 2 recursive call children from each node doubles the number of nodes each row.
- $\frac{n^2}{4^r}$ work done in each node of row r, because the original argument n will have been divded by 2 a total r times before it is an argument to a row-r recursive call, yielding $\frac{n}{2^r}$ and the square of that argument is the non-recursive work done

Because 0 work is done in the last row, we can ignore the last row and consider there to only be $log_2(n)$ rows.

Putting these values together, the work done is $\sum_{r=0}^{log_2(n)-1} 2^r \cdot \frac{n^2}{4^r}$, which can be algebraically simplified as follows:

$$\begin{aligned}
\sum_{r=0}^{log_2(n)-1} 2^r \cdot \frac{n^2}{4^r} &= \sum_{r=0}^{log_2(n)-1} \frac{n^2}{2^r} \\
&= n^2 \cdot \sum_{r=0}^{log_2(n)-1} \frac{1}{2^r} \\
&= n^2 \frac{(\frac{1}{2})^{log_2(n)} - 1}{\frac{1}{2} - 1} \\
&= 2n^2 \left(1 - \left(\frac{1}{2}\right)^{log_2(n)}\right) \\
&= 2n^2(1 - 2^{-log_2(n)}) \\
&= 2n^2 \left(1 - \frac{1}{n}\right) \\
&= 2n^2 - 2n
\end{aligned}$$

Since the total work $2n^2 - 2n$ is always less than or equal to $2n^2$ for $n \geq 1$, it follows that $T(n) = 2n^2 - 2n \in O(n^2)$. □

Again, technically this upper bound is tight. It holds that $T(n) \in \Theta(n^2)$. But this proof only verifies the upper bound.

5

Theorem 5. *If the work recurrence $T(n)$ is defined as follows:*

$$T(n) = \begin{cases} 0 & n \leq 1 \\ 5 \cdot T(n/8) + n^3 & n > 1 \end{cases}$$

then $T(n) \in O(n^3)$

Proof. Consider unrolling the call tree for $T(n)$. This tree will have

- $log_8(n) + 1$ rows because the argument is divided by 8 until it is less than or equal to 1
- 5^r nodes in row r (0-indexed) because having 5 recursive call children from each node multiplies the number of nodes each row by 5.
- $\frac{n^3}{8^{3r}}$ work done in each node of row r, because the original argument n will have been divded by 8 a total r times before it is an argument to a row-r recursive call, yielding $\frac{n}{8^r}$ and the cube of that argument is the non-recursive work done

Because 0 work is done in the last row, we can ignore the last row and consider there to only be $log_8(n)$ rows.

Putting these values together, the work done is $\sum_{r=0}^{log_2(n)-1} 5^r \cdot \frac{n^2}{8^{3r}}$, which can be algebraically simplified as follows:

$$\begin{aligned}
\sum_{r=0}^{log_8(n)-1} 5^r \cdot \frac{n^3}{8^{3r}} &= \sum_{r=0}^{log_8(n)-1} n^3 \cdot \left(\frac{5}{8^3}\right)^r \\
&= n^3 \sum_{r=0}^{log_8(n)-1} \left(\frac{5}{8^3}\right)^r \\
&= n^3 \frac{(\frac{5}{8^3})^{log_8(n)} - 1}{\frac{5}{8^3} - 1} \\
&= \frac{8^3}{8^3 - 5} n^3 \left(1 - \left(\frac{5}{8^3}\right)^{log_8(n)}\right) \\
&= \frac{512}{507} n^3 \left(1 - \frac{5^{log_8(n)}}{n^3}\right) \\
&= \frac{512}{507} n^3 \left(1 - \frac{n^{log_8(5)}}{n^3}\right) \\
&= \frac{512}{507} n^3 - \frac{512}{507} n^{log_8(5)}
\end{aligned}$$

Since the total work $\frac{512}{507}n^3 - \frac{512}{507}n^{log_8(5)3}$ is always less than or equal to $\frac{512}{507}n^3$ for $n \geq 1$, it follows that $T(n) = \frac{512}{507}n^3 - \frac{512}{507}n^{log_8(5)} \in O(n^3)$. □

[3]Note that $log_8(5)$ is approximately 0.774.