COMP170
Discrete Mathematical Tools for Computer Science

Solutions to Recurrences

Version without recursion trees

Version 2b.2: Last updated, November 1, 2007

Discrete Math for Computer Science
K. Bogart, C. Stein and R.L. Drysdale
Section 4.3, pp. 157-167

Growth Rates of Solutions to Recurrences

- Divide and Conquer Algorithms
- Iterating Recurrences
- Three Different Behaviors

Divide and Conquer Algorithms

In the previous section we analyzed recurrences of the form

$$T(n) = \begin{cases} a & \text{if } n = b \\ c \ \cdot T(n-1) + d & \text{if } n > b \end{cases}$$

These corresponded to the analyses of recursive algorithms in which a problem of size n is solved by recursively solving a problem(s) of size $n-1$.

We will now look at recurrences that arise from recursive algorithms in which problems of size n are solved by recursively solving problems of size n/m, for some fixed m. These recurrences will be in the form

$$T(n) = \begin{cases} \text{something given} & \text{if } n \le b \\ c \ \cdot T(n/m) + d & \text{if } n > b \end{cases}$$

Divide and Conquer Algorithms

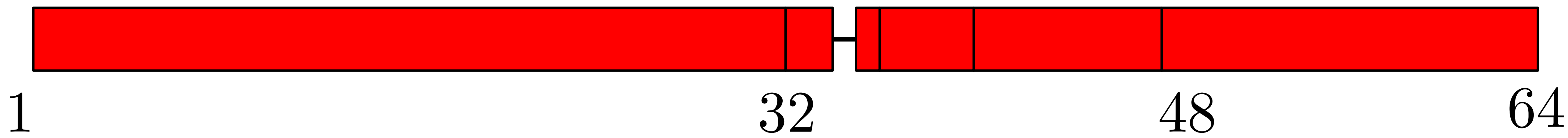
Our first example will be binary search.
Someone has chosen a number x between 1 and n.
We need to discover x.

We are only allowed to ask two types of questions:

- Is x greater than k?
- Is x equal to k?

Our strategy will be to always ask greater than questions,
at each step halving our search range,
until the range only contains one number,
when we ask a final equal to question

Binary Search Example



Is $x > 32$? Answer: Yes
Is $x > 48$? Answer: No
Is $x > 40$? Answer: No
Is $x > 36$? Answer: No
Is $x > 34$? Answer: Yes
Is $x > 35$? Answer: No
Is $x = 35$? Answer: BINGO!

Method: Each guess reduces the problem to one in which the range is only **half** as big.

This divides the original problem into one that is only half as big; we can now (recursively) conquer this smaller problem.

Note: Our derivation that, when n is a power of 2, $T(n)$, the number of questions in a binary search on $[1, n]$, satisfies

$$T(n) = \begin{cases} T(n/2) + 1 & \text{if n} \geq 2, \\ 1 & \text{if n} = 1. \end{cases}$$

was actually, implicitly, an inductive proof. This is similar to what we saw with the tower of Hanoi recurrence. We did not write out all the formal steps of the inductive proof, though.

$T(n)$: number of questions in a binary search on $[1, n]$.

Assume: n is power of 2. Give recurrence for $T(n)$.

$$T(n) = \begin{cases} T(n/2) + 1 & \text{if n} \geq 2, \\ 1 & \text{if n} = 1. \end{cases}$$

Number of questions needed for binary search on n items is:

first step

plus

time to perform binary search on the remaining $n/2$ items.

Base case (1 item): $T(1) = 1$ to ask: "Is the number k?"

(*) $$T(n) = \begin{cases} T(\lceil n/2 \rceil) + C_1 & \text{if n} \geq 2, \\ C_2 & \text{if n} = 1, \end{cases}$$

In order to avoid complications we will (usually) assume that n is a power of 2 (or sometimes 3 or 4) and also often that constants such as C_1, C_2 are 1. This will let us replace a recurrence such as (*) by one such as (**).

(**) $$T(n) = \begin{cases} T(n/2) + 1 & \text{if n} \geq 2, \\ 1 & \text{if n} = 1. \end{cases}$$

In practice, the solution of (*) will be very close to the solution of (**) (this can be proven mathematically) so, as in this class, we can restrict ourselves to (**) without losing much.

Growth Rates of Solutions to Recurrences

- Divide and Conquer Algorithms
- Iterating Recurrences
- Three Different Behaviors

Iterating Recurrences

We will solve recurrences relations by iterating (repeating) them.

Recurrence Examples:

$$T(n) = 2T\left(\frac{n}{2}\right) + n$$

To solve some problem of size n, we
(i) solve 2 subproblems of size $n/2$ and
(ii) do n units of additional work.

$$T(n) = T\left(\frac{n}{4}\right) + n^2$$

To solve some problem of size n, we
(i) solve 1 subproblem of size $n/4$ and
(ii) do n^2 units of additional work.

$$T(n) = 3T\left(n-1\right) + n$$

To solve some problem of size n, we
(ii) solve 3 subproblems of size $n-1$ and
(ii) do n units of additional work.

We will start off by examining the recurrence

$$(*) \qquad T(n) = \begin{cases} 2T\left(\frac{n}{2}\right) + n & \text{if } n > 1 \\ T(1) & \text{if } n = 1 \end{cases}$$

This corresponds to solving a problem of size n, by
(i) solving 2 subproblems of size $n/2$ and
(ii) doing n units of additional work
or using $T(1)$ work for "bottom" case of $n = 1$

In your later "analysis of algorithms" class (COMP271), you will see that this is exactly how Mergesort, one of the most famous sorting algorithms, works.

We will now see how to "solve" (*), by algebraically iterating the recurrence.

Example 1: Algebraically iterating the recurrence

Assume n is a power of 2

$$\boxed{T(n)} = 2T\left(\frac{n}{2}\right) + n \qquad = 2\left(2T\left(\frac{n}{4}\right) + \frac{n}{2}\right) + n$$

$$= 4T\left(\frac{n}{4}\right) + 2n \quad = 4\left(2T\left(\frac{n}{8}\right) + \frac{n}{4}\right) + 2n$$

$$= 8T\left(\frac{n}{8}\right) + 3n$$

$$\vdots \qquad \vdots$$

$$= 2^i T\left(\frac{n}{2^i}\right) + in$$

End when
$i = \log_2 n$

$$\vdots \qquad \vdots$$

$$= 2^{(\log_2 n)} T\left(\frac{n}{2^{(\log_2 n)}}\right) + (\log_2 n)\, n$$

$$\boxed{= nT(1) + n \log_2 n}$$

In this class we learn how to solve recurrences by

Iterating the Recurrence

The textbook describes another method as well,

Solution by Recursion Tree.

Recursion trees are just a graphical tool for visualizing the iteration of the recurrence. You can use whichever method you are more comfortable with.

We just iterated the recurrence to derive that the solution to

$$T(n) = \begin{cases} 2T(n/2) + n & \text{if } n \geq 2, \\ T(1) & \text{if } n = 1. \end{cases}$$

is $nT(1) + n \log_2 n$.

Note: Technically, we still need to use induction to prove *that our solution is correct. Practically, we never explicitly perform this step, since it is obvious how the induction would work (the* $\boxed{\ldots}$ *in the algebraic iteration is really hiding an inductive step).*

Example 2

Assume n is a power of 2

$$T(n) = \begin{cases} T(n/2) + 1 & \text{if } n \geq 2, \\ 1 & \text{if } n = 1. \end{cases}$$

$$T(n) = T\left(\frac{n}{2}\right) + 1 \qquad = \left(T\left(\frac{n}{2^2}\right) + 1\right) + 1$$

$$= T\left(\frac{n}{2^2}\right) + 2 \qquad = \left(T\left(\frac{n}{2^3}\right) + 1\right) + 2$$

$$= T\left(\frac{n}{2^3}\right) + 3$$

$$\vdots$$

$$= T\left(\frac{n}{2^i}\right) + i$$

$$\vdots$$

$$= T\left(\frac{n}{2^{\log_2 n}}\right) + \log_2 n \qquad = \boxed{1 + \log_2 n}$$

Example 3

Assume n is a power of 2

$$T(n) = \begin{cases} T(n/2) + n & \text{if } n \geq 2, \\ 1 & \text{if } n = 1. \end{cases}$$

$$\begin{aligned}
T(n) &= T\left(\frac{n}{2}\right) + n \\
&= T\left(\frac{n}{2^2}\right) + \frac{n}{2} + n \\
&= T\left(\frac{n}{2^3}\right) + \frac{n}{2^2} + \frac{n}{2} + n \\
&\vdots \\
&= T\left(\frac{n}{2^i}\right) + \frac{n}{2^{i-1}} + \ldots + \frac{n}{2^2} + \frac{n}{2} + n \\
&\vdots \\
&= T\left(\frac{n}{2^{\log_2 n}}\right) + \frac{n}{2^{\log_2 n - 1}} + \ldots + \frac{n}{2^2} + \frac{n}{2} + n \\
&= \boxed{1 + 2 + 2^2 + \ldots + \frac{n}{2^2} + \frac{n}{2} + n}
\end{aligned}$$

Total amount of work:

$$
\begin{aligned}
n + \frac{n}{2} + \frac{n}{4} + \cdots + 2 + 1 \\
&= n\left(1 + \frac{1}{2} + \frac{1}{4} + \cdots + \left(\frac{1}{2}\right)^{\log n}\right) \\
&= n \sum_{i=0}^{\log_2 n} \left(\frac{1}{2}\right)^{i}
\end{aligned}
$$

Theorem 4.4 tells us that the value of the geometric series is $O(1)$ (in fact it is ≤ 2) so, the total amount of work done is $O(n)$.

Example 4

$$T(n) = \begin{cases} 3T(n/3) + n & \text{if } n \geq 3, \\ 1 & \text{if } n < 3. \end{cases}$$

assume n *is power of* 3

$$T(n) = 3T\left(\frac{n}{3}\right) + n \qquad = 3\left(3T\left(\frac{n}{3^2}\right) + \frac{n}{3}\right) + n$$

$$= 3^2T\left(\frac{n}{3^2}\right) + 2n \quad = 3^2\left(3T\left(\frac{n}{3^3}\right) + \frac{n}{3^2}\right) + 2n$$

$$= 3^3T\left(\frac{n}{3^3}\right) + 3n$$

$$\vdots$$

$$= 3^iT\left(\frac{n}{3^i}\right) + in$$

$$\vdots$$

$$= 3^{\log_3 n}T\left(\frac{n}{3^{\log_3 n}}\right) + n\log_3 n$$

$$= \boxed{n + n\log_3 n}$$

Example 5

assume n is a power of 2

$$T(n) = \begin{cases} 4T(n/2) + n & \text{if } n \geq 2, \\ 1 & \text{if } n = 1. \end{cases}$$

$$T(n) = 4T\left(\frac{n}{2}\right) + n \qquad = 4\left(4T\left(\frac{n}{2^2}\right) + \frac{n}{2}\right) + n$$

$$= 4^2T\left(\frac{n}{2^2}\right) + \frac{4}{2}n + n \qquad = 4^2\left(4T\left(\frac{n}{2^3}\right) + \frac{n}{2^2}\right) + \frac{4}{2}n + n$$

$$= 4^3T\left(\frac{n}{2^3}\right) + \frac{4^2}{2^2}n + \frac{4}{2}n + n$$

$$\vdots$$

$$= 4^iT\left(\frac{n}{2^i}\right) + \frac{4^{i-1}}{2^{i-1}}n + \ldots + \frac{4^2}{2^2}n + n$$

$$\vdots$$

$$= 4^{\log_2 n}T\left(\frac{n}{2^{\log_2 n}}\right) + \frac{4^{\log_2 n-1}}{2^{\log_2 n-1}}n + \ldots + \frac{4}{2}n + n$$

Total work is

$$= 4^{\log_2 n} T\left(\frac{n}{2^{\log_2 n}}\right) + \frac{4^{\log_2 n - 1}}{2^{\log_2 n - 1}} n + \ldots + \frac{4}{2} n + n$$

$$= 4^{\log_2 n} T(1) + n \sum_{i=0}^{\log_2 n - 1} \left(\frac{4}{2}\right)^i$$

$$= 2^{2\log_2 n} + n \sum_{i=0}^{\log_2 n - 1} 2^i$$

$$= n^2 + n \frac{2^{\log_2 n} - 1}{2 - 1}$$

$$= n^2 + n(n-1) \qquad = \boxed{2n^2 - n}$$

Growth Rates of Solutions to Recurrences

- Divide and Conquer Algorithms
- Iterating Recurrences
- Three Different Behaviors

Three Different Behaviors

Compare the iteration for the recurrences

$$T(n) = 2T(n/2) + n$$
$$T(n) = T(n/2) + n$$
$$T(n) = 4T(n/2) + n$$

- all three recurrences iterate $\log_2 n$ times

- in each case, size of subproblem in next iteration is half the size in the preceeding iteration level

Lemma 4.7:
Suppose that we have a recurrence of the form
$$T(n) = aT\left(\frac{n}{2}\right) + n,$$
where a is a positive integer and $T(1)$ is nonnegative.
Then we have the following big Θ bounds on the solution:

1. If $a < 2$, then $T(n) = \Theta(n)$.
2. If $a = 2$, then $T(n) = \Theta(n \log n)$.
3. If $a > 2$, then $T(n) = \Theta(n^{\log_2 a})$.

Proof:

We already proved Case 1 when $a = 1$ in Example 2.
(will not prove it for $1 < a < 2$)

We already proved Case 2 in Example 1.

We will now prove Case 3.

$T(n) = aT\left(\frac{n}{2}\right) + n$ where $a > 2$. Assume n is a power of 2.

Iterating as in Example 5 gives

$$T(n) = a^i T\left(\frac{n}{2^i}\right) + \left(\frac{a^{i-1}}{2^{i-1}} + \frac{a^{i-2}}{2^{i-2}} + \ldots + \frac{a}{2} + 1\right) n$$

$$\Rightarrow \quad T(n) = \underbrace{a^{\log_2 n} T(1)}_{\text{Work at "bottom"}} \quad + \quad n \underbrace{\sum_{i=0}^{(\log_2 n)-1} \left(\frac{a}{2}\right)^i}_{\text{Iterated Work}}.$$

Total work is

$$a^{\log_2 n} T(1) + n \sum_{i=0}^{(\log_2 n)-1} \left(\frac{a}{2}\right)^i .$$

This sum is a geometric series.

Because $a/2 \neq 1$, Theorem 4.4 tells us that the sum will be big Θ of the largest term.

Because $a > 2$, the largest term in this case is clearly the last one, namely, $(a/2)^{(\log_2 n)-1}$.

n times the largest term in the geometric series is

$$n\left(\frac{a}{2}\right)^{(\log_2 n)-1} = \frac{2}{a}\cdot\frac{n\cdot a^{\log_2 n}}{2^{\log_2 n}} = \frac{2}{a}\cdot\frac{n\cdot a^{\log_2 n}}{n} = \frac{2}{a}\cdot a^{\log_2 n}$$

Now notice that

$$a^{\log_2 n} = \left(2^{\log_2 a}\right)^{\log_2 n} = \left(2^{\log_2 n}\right)^{\log_2 a} = n^{\log_2 a}$$

so the total work done is

$$\underbrace{a^{\log_2 n}T(1)}_{\Theta\left(n^{\log_2 a}\right)} + \underbrace{n\sum_{i=0}^{(\log_2 n)-1}\left(\frac{a}{2}\right)^i}_{\Theta\left(n^{\log_2 a}\right)} = \Theta\left(n^{\log_2 a}\right)$$

and we are done!

As an example of Case 3 consider

$$T(n) = \begin{cases} 4T(n/2) + n & \text{if n} \geq 2, \\ 1 & \text{if n} = 1. \end{cases}$$

$a = 4$ so the Theorem says that

$$T(n) = \Theta\left(n^{\log_2 a}\right) = \Theta\left(n^{\log_2 4}\right) = \Theta(n^2)$$

This matches with the exact answer of $2n^2 - n$, which we already derived in Example 5.