CS 341: ALGORITHMS (S18) — LECTURE 3

SOLVING RECURRENCES

ERIC BLAIS

Many of the algorithms that we will have to analyze in this course have a recursive structure. To complete this analysis, we need to develop tools for solving recurrences. This is what we do today.

1. Recurrence trees

Recall that the MergeSort algorithm is defined as follows.

Algorithm 1: MergeSort $(A = (A[1], \ldots, A[n]))$

if $n = 1$ **return**;
MergeSort $(A[1, \ldots, \lfloor n/2 \rfloor])$;
MergeSort $(A[\lfloor n/2 \rfloor + 1, \ldots, n])$;
Merge $(A[1, \ldots, \lfloor n/2 \rfloor], A[\lfloor n/2 \rfloor + 1, \ldots, n])$;

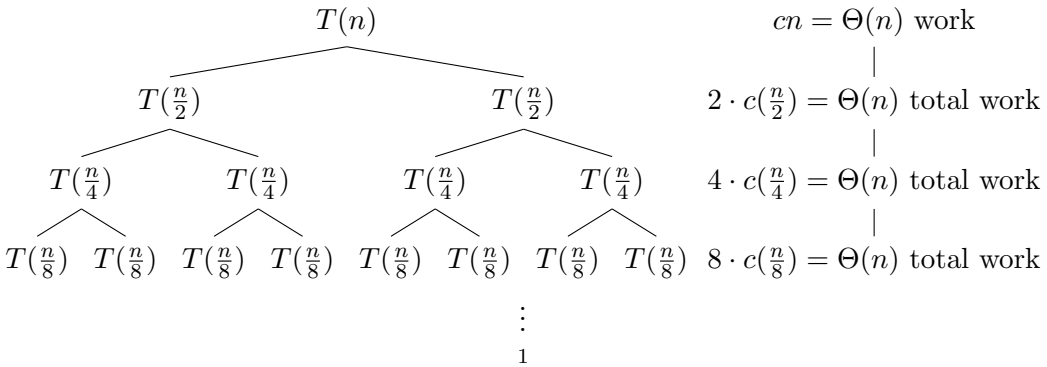
When Merge is an algorithm that merges the two input arrays in time $\Theta(n)$, what is the time complexity of MergeSort? If we let $T(n)$ denote the time complexity of the algorithm on arrays with n entries, we find that

$$T(n) = T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + \Theta(n).$$

To simplify the analysis, let's assume that n is a power of 2. Then we have

$$T(n) = 2\,T(\tfrac{n}{2}) + \Theta(n).$$

To determine the solution to this recursion, the most natural approach is to draw the recursion tree and write down how much time is spent on each level of the tree:



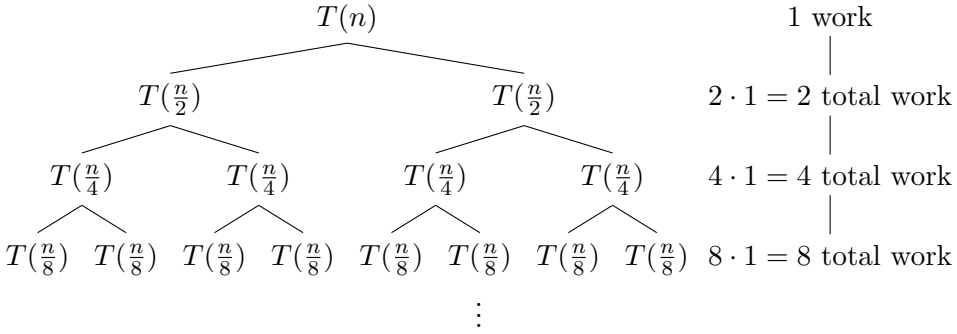
The recursion tree has depth $\log n$ and $\Theta(n)$ time is spent at each level of the tree on completing the merges. The total time complexity of the MergeSort algorithm is therefore

$$T(n) = \Theta(n \log n).^1$$

Many other recursions can also be solved using the same recursion tree approach. For example, what if we (magically) allowed the Merge algorithm to run in a single unit of time? Then the recursion defining the time complexity of the MergeSort algorithm would be

$$T(n) = 2\,T(\tfrac{n}{2}) + 1$$

and the recursion tree would now look like this:



This tree again has depth $\log n$, so the total time complexity in this case is

$$T(n) = 1 + 2 + 4 + 8 + \cdots + \tfrac{n}{2} + n = 2n - 1 = \Theta(n).$$

One more variant: what if we proceed recursively but now divide the array in three instead of two pieces?

Algorithm 2: TriMergeSort $(A = (A[1], \ldots, A[n]))$

if $n = 1$ **return**;
TriMergeSort $(A\big[1, \ldots, \lfloor n/3 \rfloor\big])$;
TriMergeSort $(A\big[\lfloor n/3 \rfloor + 1, \ldots, \lfloor 2n/3 \rfloor\big])$;
TriMergeSort $(A\big[\lfloor 2n/3 \rfloor + 1, \ldots, n\big])$;
Merge ($A\big[1, \ldots, \lfloor n/3 \rfloor\big]$, $A\big[\lfloor n/3 \rfloor + 1, \ldots, \lfloor 2n/3 \rfloor\big]$, $A\big[\lfloor 2n/3 \rfloor + 1, \ldots, n\big]$);

Let Merge again be an algorithm with time complexity $\Theta(n)$. When n is a power of 3, the time complexity of the TriMergeSort algorithm is

$$T(n) = 3\,T(\tfrac{n}{3}) + \Theta(n).$$

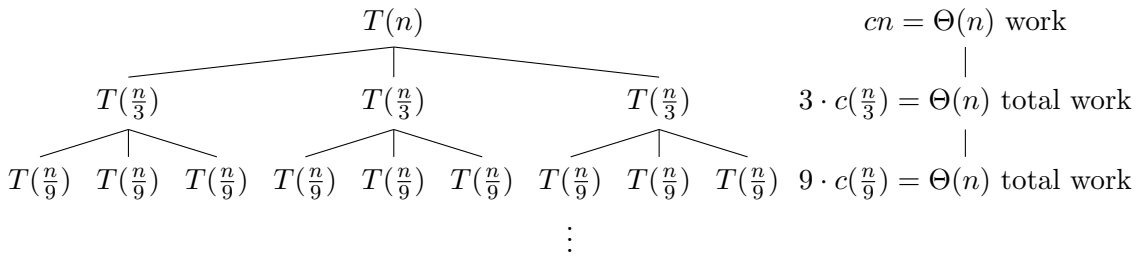
The recursion tree for this recurrence is:

The tree has depth $\log_3 n$, so the time complexity of TriMergeSort is

$$T(n) = \Theta(n \log_3 n) = \Theta(n \log n),$$

the same (asymptotically) as MergeSort!

[1] Here and throughout this course, logarithms without subscripts are over base 2.



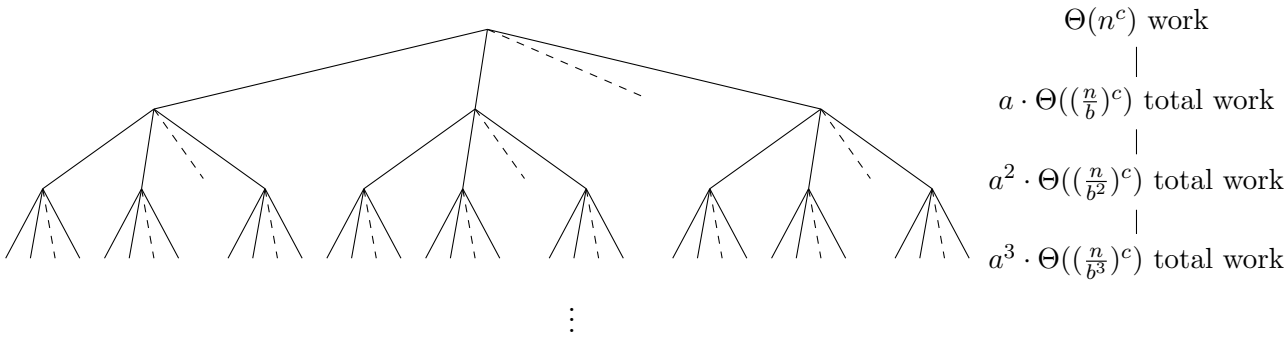
2. Master theorem

We could continue exploring various different examples using the recursion tree method. But let's be a bit more ambitious and try to consider a general scenario that captures many individual examples at once. Let T be defined by the recurrence

$$T(n) = aT(\tfrac{n}{b}) + \Theta(n^c)$$

for some constants $a > 0$, $b > 1$, and $c \geq 0$.

We can again use the recursion tree approach to solve for T in this case.



We can therefore express T as

$$T(n) = \left(1 + \frac{a}{b^c} + \left(\frac{a}{b^c}\right)^2 + \cdots + \left(\frac{a}{b^c}\right)^{\log_b n}\right) \cdot \Theta(n^c).$$

To determine T, we must simply understand the geometric sequence with ratio $\frac{a}{b^c}$. There are three cases to consider.

Case 1. When $\frac{a}{b^c} = 1$, then each term in the sequence is 1 and so

$$T(n) = \log_b(n) \cdot \Theta(n^c) = \Theta(n^c \log n).$$

Case 2. When $\frac{a}{b^c} < 1$, then the geometric series $1 + \frac{a}{b^c} + \left(\frac{a}{b^c}\right)^2 + \cdots + \left(\frac{a}{b^c}\right)^{\log_b n} = \Theta(1)$ is a constant and so

$$T(n) = \Theta(n^c).$$

Case 3. When $\frac{a}{b^c} > 1$, then we have an increasing geometric series that is dominated by the last term, so that

$$T(n) = \Theta\left(n^c \cdot \left(\frac{a}{b^c}\right)^{\log_b n}\right) = \Theta(a^{\log_b n}) = \Theta(n^{\log_b a}).$$

This result establishes what is known as the *Master Theorem*.

Theorem 3.1 (Master theorem)**.** *If $T(n) = aT(\frac{n}{b}) + \Theta(n^c)$ for some constants $a > 0$, $b > 1$, and $c \geq 0$, then*

$$T(n) = \begin{cases} \Theta(n^c) & \text{if } c > \log_b a \\ \Theta(n^c \log n) & \text{if } c = \log_b a \\ \Theta(n^{\log_b a}) & \text{if } c < \log_b a. \end{cases}$$

What is most important to remember of this theorem is not the statement of the theorem itself but rather the method that we used to obtain it: with the recursion tree method, you can easily recover the Master Theorem itself and also solve recursions that are not of the form covered by the theorem.

3. Geometric series

The different cases of the Master Theorem were handled rather quickly in the section above. It's worth slowing down a bit to see exactly how the asymptotic results about geometric series are obtained. The starting point for those results is the fundamental identity for geometric series.[2]

Fact 3.2. *For any $r \neq 1$ and any $n \geq 1$,*

$$1 + r + r^2 + r^3 + \cdots + r^n = \frac{1 - r^{n+1}}{1 - r}.$$

We can now use this lemma to give the big-Θ closed form expressions for geometric series when $r \neq 1$. Let's start with the case where $r < 1$.

Theorem 3.3. *For any $0 < r < 1$, the function $f : n \mapsto 1 + r + r^2 + \cdots + r^n$ satisfies $f = \Theta(1)$.*

Proof. From the geometric series identity,

$$1 + r + r^2 + r^3 + \cdots + r^n = \frac{1 - r^{n+1}}{1 - r}$$

we observe that for every $n \geq 1$, $f(n) \leq \frac{1}{1-r}$ so, setting $n_0 = 1$ and $c = \frac{1}{1-r}$ we have that for all $n \geq n_0$, $f(n) \leq c \cdot 1$ and, therefore, $f = O(1)$.

Similarly, if we let n_0 be the smallest integer for which $r^{n_0} \leq \frac{1}{2}$ (which is the value $n_0 = \lceil \log_{1/r}(2) \rceil$), then for every $n \geq n_0$ we have $\frac{1-r^{n+1}}{1-r} \geq \frac{1}{2(1-r)}$ and, taking the constant $c = \frac{1}{2(1-r)}$, $f \geq c \cdot 1$ so $f = \Omega(1)$.

Since $f = O(1)$ and $f = \Omega(1)$, then also $f = \Theta(1)$. □

The analysis of geometric series when $r > 1$ is similar.

Theorem 3.4. *For any $r > 1$, the function $f : n \mapsto 1 + r + r^2 + \cdots + r^n$ satisfies $f = \Theta(r^n)$.*

Proof. The geometric series identity can also be expressed as

$$1 + r + r^2 + r^3 + \cdots + r^n = \frac{r^{n+1} - 1}{r - 1}.$$

With $n_0 = 1$ and $c = \frac{r}{r-1}$, this identity implies that for all $n \geq n_0$, $f(n) \leq \frac{r^{n+1}}{r-1} = c \cdot r^n$ so $f = O(r^n)$.

[2] Can you see how to prove this identity? *Hint:* Multiply the left-hand side of the identity by $1 - r$ and remove the terms that cancel out.

And when we choose n_0 to be the smallest integer that satisfies $r^{n_0} \geq 2$ (or, equivalently, $n_0 = \lceil \log_r(2) \rceil$) and $c = \frac{r}{2(r-1)}$ then we have that for all $n \geq n_0$, $f(n) = \frac{r^{n+1}-1}{r-1} \geq \frac{r^{n+1}-\frac{1}{2}r^{n+1}}{r-1} = c \cdot r^n$ so $f = \Omega(r^n)$.

Since $f = O(1)$ and $f = \Omega(1)$, then also $f = \Theta(1)$. □