Recurrences

COMP4500/7500

July 30, 2019

Divide and conquer algorithms

MERGE_SORT(A, p, r) sorts subarray $A[p..r]$.

MERGE(A, p, q, r) takes sorted subarrays $A[p, q]$ and $A[q+1, r]$ and merges them to produce sorted array $A[p, r]$.

Assumes MERGE(A, p, q, r) is $\Theta(n)$ where $n = r - p + 1$.

MERGE_SORT(A, p, r)

```
if p < r
    q = ⌊(p + r)/2⌋              // divide
    MERGE_SORT(A, p, q)          // solve subproblem recursively
    MERGE_SORT(A, q + 1, r)      // solve subproblem recursively
    MERGE(A, p, q, r)            // combine
```

MERGE_SORT(A,0,3)

	0	1	2	3
A =	5	2	6	4

MERGE_SORT(A,0,3)

MERGE_SORT(A,0,1)

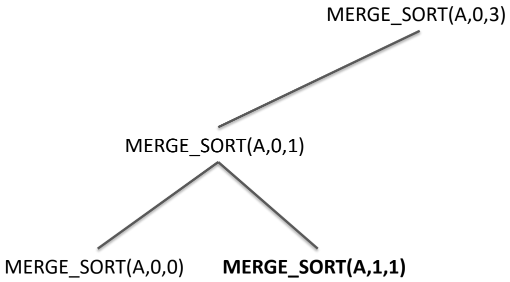
	0	**1**	**2**	**3**
A =	5	2	6	4

MERGE_SORT(A,0,3)

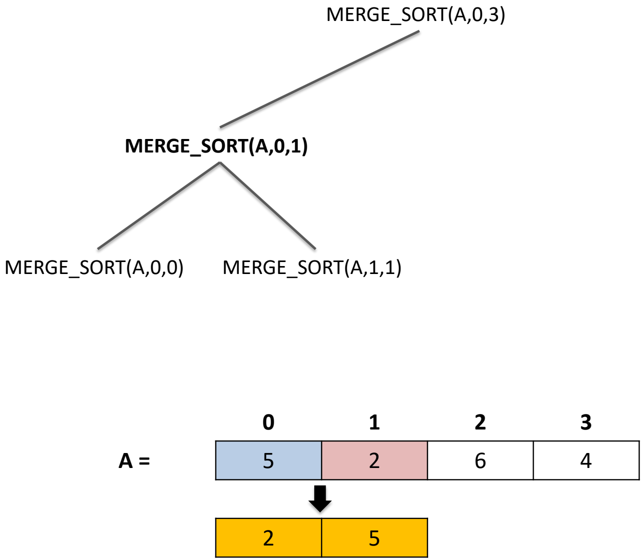
MERGE_SORT(A,0,1)

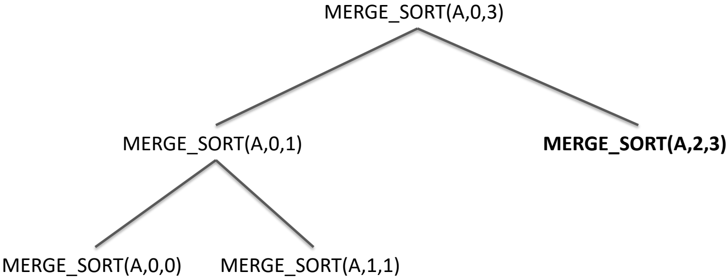
MERGE_SORT(A,0,0)

	0	**1**	**2**	**3**
A =	5	2	6	4

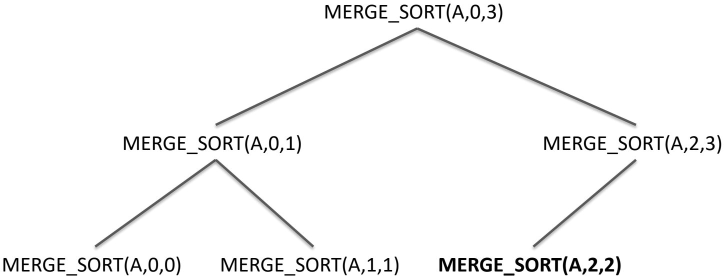


	0	**1**	**2**	**3**
A =	5	2	6	4

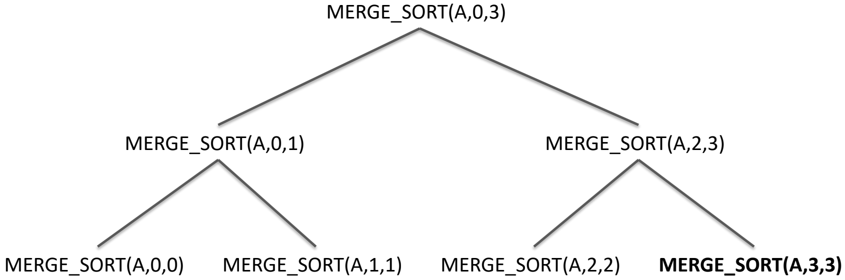

MERGE_SORT(A,0,3)
MERGE_SORT(A,0,1)
MERGE_SORT(A,0,0)
MERGE_SORT(A,1,1)
0
1
2
3
A =
5
2
6
4
2
5


MERGE_SORT(A,0,3)
MERGE_SORT(A,0,1)
MERGE_SORT(A,2,3)
MERGE_SORT(A,0,0)
MERGE_SORT(A,1,1)

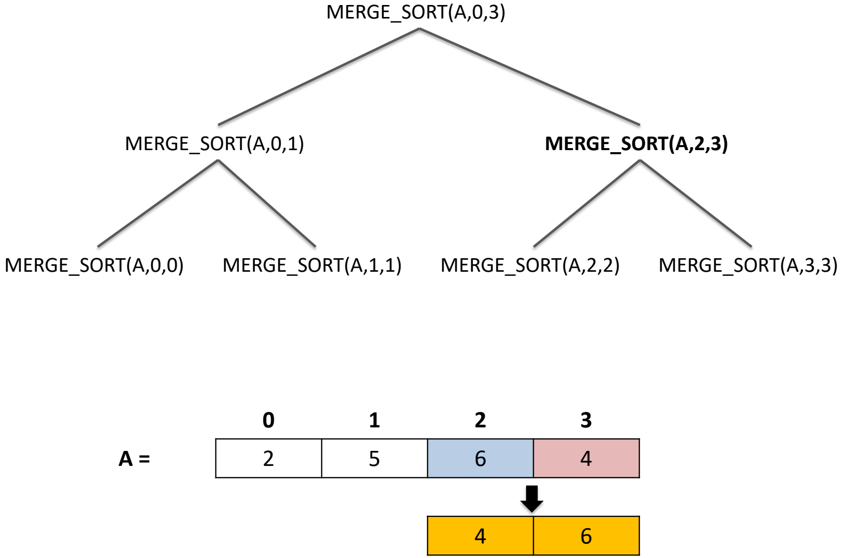
	0	**1**	**2**	**3**
A =	2	5	6	4

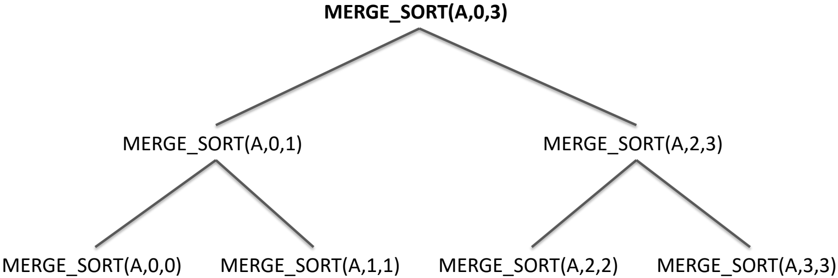

MERGE_SORT(A,0,3)
MERGE_SORT(A,0,1)
MERGE_SORT(A,2,3)
MERGE_SORT(A,0,0)
MERGE_SORT(A,1,1)
MERGE_SORT(A,2,2)

	0	**1**	**2**	**3**
A =	2	5	6	4


MERGE_SORT(A,0,3)
MERGE_SORT(A,0,1)
MERGE_SORT(A,2,3)
MERGE_SORT(A,0,0)
MERGE_SORT(A,1,1)
MERGE_SORT(A,2,2)
MERGE_SORT(A,3,3)

	0	**1**	**2**	**3**
A =	2	5	6	4


MERGE_SORT(A,0,3)
MERGE_SORT(A,0,1)
MERGE_SORT(A,2,3)
MERGE_SORT(A,0,0)
MERGE_SORT(A,1,1)
MERGE_SORT(A,2,2)
MERGE_SORT(A,3,3)
0 1 2 3
A =
2 5 6 4
4 6



	0	**1**	**2**	**3**
A =	2	5	4	6
⬇				
	2	4	5	6

MERGE_SORT(A, p, r): recurrence

Let $n = r - p + 1$:

$$
\begin{aligned}
T(n) &= \Theta(1) && \text{if } n \leq 1 \\
T(n) &= \Theta(1) + 2T(n/2) + \Theta(n) && \text{if } n > 1 \\
&= 2T(n/2) + \Theta(n)
\end{aligned}
$$

MERGE_SORT(A, p, r)

```
if p < r
    q = ⌊(p + r)/2⌋              // divide: Θ(1)
    MERGE_SORT(A, p, q)          // solve subproblem: T(n/2)
    MERGE_SORT(A, q + 1, r)      // solve subproblem: T(n/2)
    MERGE(A, p, q, r)            // combine: Θ(n)
```

$T(n) = 2T(n/2) + \Theta(n)$ is shorthand for $T(n) = 2T(n/2) + f(n)$ for some function function $f(n) \in \Theta(n)$. (See CLRS 34–35 [3rd])

Recurrences

To be **well-defined** a recurrence needs:

- base case, and
- recursive case(s) that converge on the base case.

Running time is usually bounded by a constant for constant-sized inputs, and so we often omit the base case and assume

$$T(n) = \Theta(1) \quad \text{for } n \leq c$$

where c is a constant.

Recurrences

Given a *divide and conquer algorithm* that:

- takes a problem of size **n** and
- breaks it into **a** parts each of size **n**/**b**,
- takes **D**(**n**) time to divide the problem, and
- takes **C**(**n**) time to combine solutions to subproblems.

we get the following recurrence:

$$
\begin{array}{rcll}
T(n) & = & aT(n/b) + D(n) + C(n) & \text{if } n > c \\
 & \in & \Theta(1) & \text{if } n \leq c
\end{array}
$$

Solving Recurrences

Ther are three general methods for solving a recurrence: [See CLRS Ch4.]

- **Substitution**:
  - guess an answer and then prove by induction
- **Iteration**:
  - expand recurrence to formulate a summation, then solve
- **Master Method**:
  - remember 3 cases for solving recurrences of the form $T(n) = aT(n/b) + f(n)$

Substitution

"Guess" a solution, prove by induction.

Substitution: example without asymptotic notation

Given the recurrence:

$$\begin{aligned} T(n) &= 2 && \text{if } n = 2 \\ &= 2T(n/2) + n && \text{if } n = 2^k \text{ for } k > 1 \end{aligned}$$

we guess that

$$T(n) = n \lg n \qquad \text{for all } n = 2^k \text{ where } k \geq 1$$

and then prove it by induction:

Base case: $T(2) = 2 = 2 \log_2 2$

Inductive step: assume $T(n/2) = n/2 \lg(n/2)$ for $n/2 = 2^{k-1}$ and prove for $n = 2^k$:

$$\begin{aligned} T(n) &= 2T(n/2) + n \\ &= 2(n/2)\lg(n/2) + n && \text{substitute inductive assumption} \\ &= n(\lg n - \lg 2) + n \\ &= n \lg n \end{aligned}$$

Substitution: example using asymptotic notation

Given the recurrence (includes floor):

$$\begin{aligned} T(n) &= 1 & \text{if } n = 1 \\ &= 2T(\lfloor n/2 \rfloor) + n & \text{if } n > 1 \end{aligned}$$

we notice that it is like the previous one, so guess that:

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

We need to prove by induction that there exist constants $n_0, c > 0$ such that

$$\forall n \geq n_0 \bullet 0 \leq T(n) \leq cn \lg n$$

We have $0 \leq T(n)$ for $n \geq 1$, so focus on $T(n) \leq cn \lg n$ part.

Substitution example: boundary conditions

This depends on n_0. Which value for n_0 should we choose?

We can't choose $n_0 = 1$ since:

$$T(1) = 1 \quad \not\leq \quad c.1.\lg 1 = 0$$

If we choose $n_0 = 2$ then the only values of $2 \leq n$ directly dependent on $T(1)$ are $T(2)$ and $T(3)$:

$$\begin{array}{lcll} T(2) = 2T(1) + 2 = 4 & \leq & c.2.\lg 2 & \text{if } c \geq 2 \\ T(3) = 2T(1) + 3 = 5 & \leq & c.3.\lg 3 & \text{if } c \geq \frac{5}{3.\lg 3} \end{array}$$

So $\mathbf{T}(\mathbf{n}) \leq \mathbf{cn}\lg\mathbf{n}$ for base cases $\mathbf{n} = \mathbf{2}$ and $\mathbf{n} = \mathbf{3}$ if $c \geq 2$.

Substitution example: inductive step

Assume $T(n) \le cn \lg n$ for $\lfloor n/2 \rfloor$, i.e.,

$$T(\lfloor n/2 \rfloor) \le c \lfloor n/2 \rfloor \lg \lfloor n/2 \rfloor$$

then prove $T(n) \le cn \lg n$, for some suitable $c > 0$ (find c):

$$\begin{aligned}
T(n) &= 2T(\lfloor n/2 \rfloor) + n \\
&\le 2(c\lfloor n/2 \rfloor \lg \lfloor n/2 \rfloor) + n && \text{substitute inductive assumption} \\
&\le cn \lg (n/2) + n \\
&= cn \lg n - cn \lg 2 + n \\
&= cn \lg n - cn + n \\
&\le cn \lg n && \text{if } c \ge 1 \text{ and } n \ge 0
\end{aligned}$$

We must also have $c \ge 2$ to satisfy the base cases, so $n_0 = 2$ and $c = 2$ will suffice.

Substitution: more on "Guessing"

Consider

$$T(n) = 2T(\lfloor n/2 \rfloor) + \underline{17} + n$$

Difference between

$$T(\lfloor n/2 \rfloor) \qquad \text{and} \qquad T(\lfloor n/2 \rfloor) + 17$$

is small, so guess same solution.

Guess loose upper and lower bounds, then refine:

$$\begin{aligned} T(n) &= \Omega(n) \\ T(n) &= O(n^2) \end{aligned}$$

Try to lower the upper bound and raise the lower bound until convergence.

Substitution: guessing doesn't always work!

Consider

$$T(n) = T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + 1$$

We guess $T(n) = O(n)$.

Inductive step: assume $T(\lfloor n/2 \rfloor) \leq c\lfloor n/2 \rfloor$ and $T(\lceil n/2 \rceil) \leq c\lceil n/2 \rceil$ and attempt to prove $T(n) \leq cn$:

$$\begin{aligned}
T(n) &= T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + 1 \\
&\leq c\lfloor n/2 \rfloor + c\lceil n/2 \rceil + 1 && \text{substitute inductive assumptions} \\
&= cn + 1, \\
&\not\leq cn && !!!!!
\end{aligned}$$

Could prove $T(n) = O(n^2)$, but that is too weak...

Guess was *almost right*, just an annoying "1" to remove.

Substitution: strengthening the guess

Try **strengthening the guess** by subtracting a low order term. Guess that:

$$\exists c, n_0 > 0 \bullet 0 \leq T(n) \leq cn - b \qquad \text{where } b > 0$$

Inductive step: assume for $\lfloor n/2 \rfloor$ and $\lceil n/2 \rceil$ and prove for n:

$$\begin{aligned}
T(n) &= T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + 1 \\
&\leq c\lfloor n/2 \rfloor - b + c\lceil n/2 \rceil - b + 1 && \text{substitute assumptions} \\
&= cn - 2b + 1 \\
&\leq cn - b && \text{if } b \geq 1
\end{aligned}$$

Boundary conditions: now find c and n_0 to also satisfy the boundary conditions...

By making a stronger inductive assumption we can prove a stronger result!

Substitution: guessing incorrectly + bugs in proofs

Consider again:

$$T(n) = 2T(\lfloor n/2 \rfloor) + n$$

Guess $T(n) \in O(n)$.

Incorrect inductive step:
Assume $T(\lfloor n/2 \rfloor) \leq c\lfloor n/2 \rfloor$ and prove $T(n) \leq cn$:

$$\begin{aligned} T(n) &= 2T(\lfloor n/2 \rfloor) + n \\ &\leq 2c\lfloor n/2 \rfloor + n \qquad \text{substitute inductive assumption} \\ &\leq cn + n \\ &= O(n)!!! \end{aligned}$$

But this does not prove that $T(n) \leq cn$!!

Each line is “correct”, but it is not a valid inductive proof.

Change of variables

Consider

$$T(n) = 2T(\lfloor\sqrt{n}\rfloor) + \lg n$$

Looks hard: try change of variable where $m = \lg n$ (ie, $n = 2^m$)

$$T(2^m) = 2T(2^{m/2}) + m$$

Rename: $S(m) = T(2^m)$

$$S(m) = 2S(m/2) + m$$

so $S(m) = \Theta(m \lg m)$.
Change back

$$\begin{aligned} T(n) &= T(2^m) = S(m) = \Theta(m \lg m) \\ &= \Theta(\lg n \; \lg \lg n) \end{aligned}$$

Magic???

Iteration

- Expand (iterate) the recurrence to get a summation.
- Evaluate the summation.

More maths, but no need to guess!

Iteration example

Consider

$$\begin{array}{rcll} T(n) & = & \Theta(1) & \text{if } n \leq 1 \\ T(n) & = & n + 3T(\lfloor n/4 \rfloor) & \text{if } n > 1 \end{array}$$

$$\begin{array}{rcll} T(n) & = & n + 3T(\lfloor n/4 \rfloor) & \text{if } n > 1 \\ & = & n + 3(\lfloor n/4 \rfloor + 3T(\lfloor n/16 \rfloor)) & \text{if } \lfloor \frac{n}{4} \rfloor > 1 \\ & = & n + 3(\lfloor n/4 \rfloor + 3(\lfloor n/16 \rfloor + 3T(\lfloor n/64 \rfloor))) & \text{if } \lfloor \frac{n}{16} \rfloor > 1 \\ & = & n + 3\lfloor n/4 \rfloor + 9\lfloor n/16 \rfloor + 27T(\lfloor n/64 \rfloor) & \\ & = & n + 3\lfloor n/4 \rfloor + 3^2\lfloor n/4^2 \rfloor + 3^3T(\lfloor n/4^3 \rfloor) & \\ & \ldots & & \\ & = & n + 3\lfloor n/4 \rfloor + 3^2\lfloor n/4^2 \rfloor + \ldots + 3^iT(\lfloor n/4^i \rfloor) & \text{if } \lfloor \frac{n}{4^{i-1}} \rfloor > 1 \end{array}$$

For which value of i do we stop expanding the recurrence?
For the smallest value of i such that $\lfloor n/4^i \rfloor \leq 1$.

Iteration example continued.

Assuming $T(n) = \Theta(1)$ for $n \leq 1$, we stop expanding when

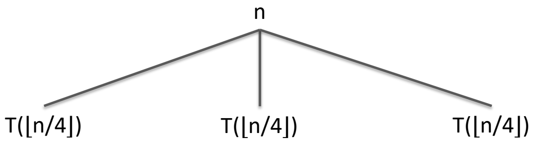
$$\lfloor n/4^i \rfloor \simeq 1 \quad \equiv \quad 4^i \simeq n \quad \equiv \quad i \simeq \log_4 n$$

We must have $i \leq \lceil \log_4 n \rceil$ since $\lfloor n/4^{\lceil \log_4 n \rceil} \rfloor \leq 1$.
It follows that:

$$\begin{aligned}
T(n) &= n + 3\lfloor n/4 \rfloor + 3^2 \lfloor n/4^2 \rfloor + \ldots + 3^i T(\lfloor n/4^i \rfloor) \quad \text{if } \lfloor \tfrac{n}{4^{i-1}} \rfloor > 1 \\
&\leq n + 3\lfloor n/4 \rfloor + 3^2 \lfloor n/4^2 \rfloor + \ldots + 3^{\lceil \log_4 n \rceil} \Theta(1) \\
&\leq n + \tfrac{3n}{4} + \tfrac{3^2 n}{4^2} + \tfrac{3^3 n}{4^3} + \ldots + \Theta(3^{\log_4 n}) \\
&\leq n \textstyle\sum_{i=0}^{\infty} (\tfrac{3}{4})^i + \Theta(3^{\log_4 n}) \\
&= 4n + O(n^{\log_4 3}) \qquad \text{using } a^{\log_b n} = n^{\log_b a} \\
&= 4n + O(n) \qquad \text{using } \log_4 3 < 1 \\
&= O(n)
\end{aligned}$$

Recursion tree for $T(n) = n + 3T(\lfloor n/4 \rfloor)$

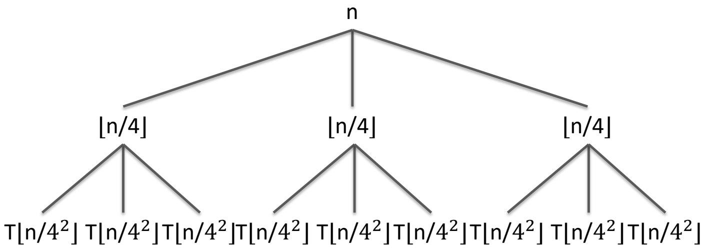
T(n) =



when n>1

Recursion tree for $T(n) = n + 3T(\lfloor n/4 \rfloor)$

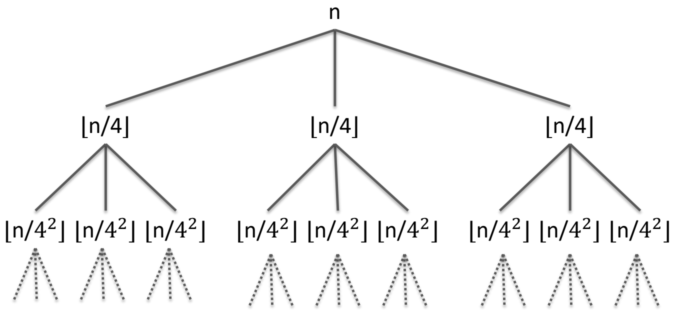
T(n) = when ⌊n/ 4⌋>1



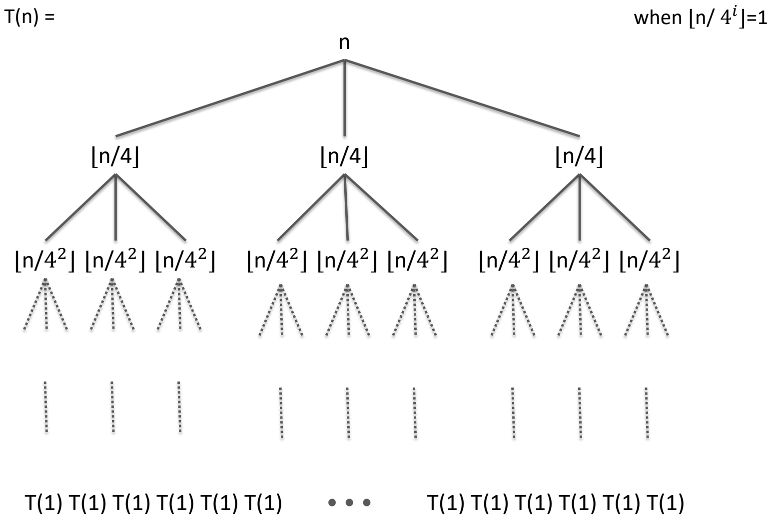
Recursion tree for $T(n) = n + 3T(\lfloor n/4 \rfloor)$

T(n) =

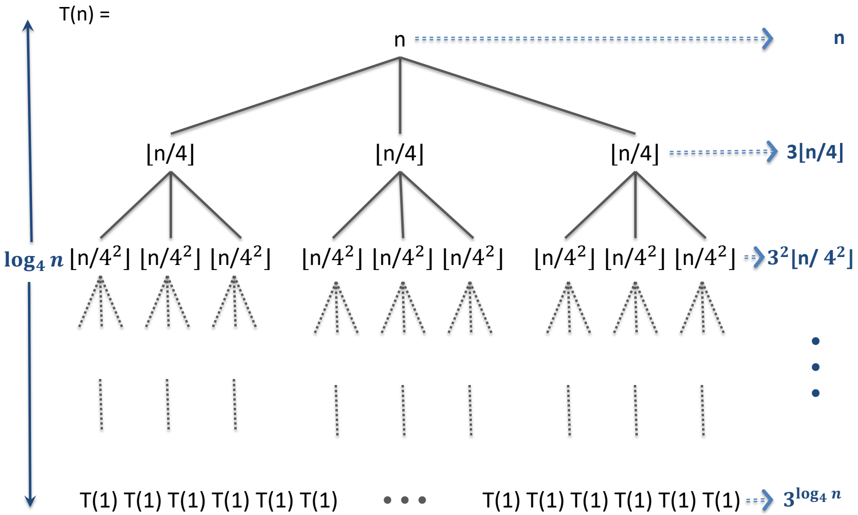
when $\lfloor n/4^2 \rfloor > 1$



Recursion tree for $T(n) = n + 3T(\lfloor n/4 \rfloor)$



Recursion tree for $T(n) = n + 3T(\lfloor n/4 \rfloor)$



Problems

- Lots of maths
- You have to work out all the terms, when to stop, and to sum what you get
- Sometimes you can start the iteration and guess the answer
- If you guess correctly you can abandon the maths and revert to substitution

Some Details

- Floors and ceilings can be troublesome.
  - Work-around: Assume n has correct form to delete them. (Previous example: $n = 4^k$).
  - Technical abuse, but often works well (OK if function "well-behaved", see problem 4.5, p74 CLR only).
- Recursion trees let you see what is going on.
  - See CLRS p89, p91 [3rd] for diagrams.

Master Method: the rough idea

Need to "memorize" 3 cases for solving (some) recurrences of the form

$$T(n) = aT(n/b) + f(n)$$

where n/b can be $\lfloor n/b \rfloor$ or $\lceil n/b \rceil$.

In each case we compare $n^{\log_b a}$ with $f(n)$:

Case 1 $n^{\log_b a}$ "polynomially larger than" $f(n)$
solution $\Theta(n^{\log_b a})$

Case 2 $n^{\log_b a}$ "same tight asymptotic bound as" $f(n)$
solution $\Theta(n^{\log_b a} \lg n)$

Case 3 $f(n)$ "polynomially larger than" $n^{\log_b a}$
and $f(n)$ is "regular"
solution $\Theta(f(n))$

Master Method: details

Function $f(n)$ is "polynomially larger than" than $g(n)$ when:

$$f(n) \in \Omega(g(n) \times n^{\epsilon}) \qquad \text{for some } \epsilon > 0$$

Equivalently:

$$\frac{f(n)}{g(n)} \in \Omega(n^{\epsilon}) \qquad \text{for some } \epsilon > 0$$

- Is $f(n) = n^2$ polynomially larger than $g(n) = n^{3/2}$?
  - Yes: $n^2 \in \Omega(n^{3/2} \times n^{1/2})$
- Is $f(n) = \log_2 n$ polynomialy larger than $g(n) = 1$?
  - No: $\log_2 n \notin \Omega(1 \times n^{\epsilon})$ for any $\epsilon > 0$

Master Method: details

Function $f(n)$ is regular when:

$$af(n/b) < cf(n) \qquad \text{for some } c < 1$$

(Most functions satisfy regularity, but still need to check.)

Master method

Recurrence:

$$T(n) = aT\left(\frac{n}{b}\right) + f(n)$$

Case 1

$T(n) \in \Theta(n^{\log_b a})$ if $f(n) \in O(n^{\log_b a-\epsilon})$ for some $\epsilon > 0$

Case 2

$T(n) \in \Theta(n^{\log_b a} \lg n)$ if $f(n) \in \Theta(n^{\log_b a})$

Case 3

$T(n) \in \Theta(f(n))$ if $f(n) \in \Omega(n^{\log_b a+\epsilon})$ for some $\epsilon > 0$
and $af\left(\frac{n}{b}\right) \leq cf(n)$ for some $c < 1$
(i.e. regularity)

Master method

Recurrence:

$$T(n) = aT\left(\frac{n}{b}\right) + f(n)$$

Case 1

$T(n) \in \Theta(n^{\log_b a})$ if $\frac{f(n)}{n^{\log_b a}} \in O(n^{-\epsilon})$ for some $\epsilon > 0$

Case 2

$T(n) \in \Theta(n^{\log_b a} \lg n)$ if $\frac{f(n)}{n^{\log_b a}} \in \Theta(1)$

Case 3

$T(n) \in \Theta(f(n))$ if $\frac{f(n)}{n^{\log_b a}} \in \Omega(n^{\epsilon})$ for some $\epsilon > 0$

and $af\left(\frac{n}{b}\right) \leq cf(n)$ for some $c < 1$

Master method: Merge sort

Recurrence: $T(n) = 2T\left(\frac{n}{2}\right) + f(n)$ where $f(n) \in \Theta(n)$

$$\frac{f(n)}{n^{\log_b a}} \in \Theta\left(\frac{n}{n^{\log_2 2}}\right) = \Theta\left(\frac{n}{n}\right) = \Theta(1)$$

Hence **case 2** and $T(n) \in \Theta(n^1 \lg n) = \Theta(n \lg n)$

Master method: Binary search

Recurrence: $T(n) = 1\,T\left(\frac{n}{2}\right) + f(n)$ where $f(n) \in \Theta(1)$

$$\frac{f(n)}{n^{\log_b a}} = \Theta\left(\frac{1}{n^{\log_2 1}}\right) = \Theta\left(\frac{1}{1}\right) = \Theta(1)$$

Hence **case 2** and $T(n) \in \Theta(n^0 \lg n) = \Theta(\lg n)$

Master method: Strassen's matrix multiply

Recurrence: $T(n) = 7T\left(\frac{n}{2}\right) + f(n)$ where $f(n) \in \Theta(n^2)$

$$\frac{f(n)}{n^{\log_b a}} \in \Theta\left(\frac{n^2}{n^{\log_2 7}}\right) = \Theta\left(\frac{1}{n^{\log_2 7 - 2}}\right)$$

where $\log_2 7 - 2 > 0$.

Hence **case 1** and $T(n) \in \Theta(n^{\log_b a}) = \Theta(n^{\log_2 7}) \approx \Theta(n^{2.81})$

Master method: Example

- Recurrence: $T(n) = 4T\left(\frac{n}{2}\right) + f(n)$ where $f(n) \in \Theta(n^3)$

$$\frac{f(n)}{n^{\log_b a}} \in \Theta\left(\frac{n^3}{n^{\log_2 4}}\right) = \Theta\left(\frac{n^3}{n^2}\right) = \Theta(n^1)$$

Hence **case 3** and $T(n) \in \Theta(f(n)) = \Theta(n^3)$ provided

$$af(n/b) \leq cf(n) \quad \text{for some } c < 1$$

that is

$$af\left(\frac{n}{b}\right) = 4\left(\frac{n}{2}\right)^3 = 4\frac{n^3}{8} = \frac{1}{2}n^3 \leq cn^3$$

for $c = \frac{1}{2}$.

Master Method leaves gaps

Master Method does not apply if

- functions “larger” but not “polynomially larger”.
- In case 3 it is not regular

See CLRS p99 [3rd] p77 [2nd]; CLR p67 [1st] for a picture of the proof of the master theorem.

FYI: Extension to Master Theorem

Fills a gap in the Master Theorem.

- If $f(n) \in \Theta(n^{\log_b a} \lg^k n)$
- $f(n)$ is larger, but **not** polynomially larger
- Solution: $\Theta(n^{\log_b a} \lg^{k+1} n)$

Case 2 (above) is a special case of this when $k = 0$.
See CLRS Ex 4.6-2 p106 [3rd]; Ex 4.4.2, p84 [2nd]; CLR Ex 4.4.4, p72 [1st]