SC2001/ CX2101: Algorithm Design and Analysis

Part 2

Huang Shell Ying
Office: N4-02b-38
Email: assyhuang@ntu.edu.sg

Topics

- Analysis Techniques (3 hours)
- Dynamic Programming (5 hours)
- String Matching (3 hours)
- Introduction to NP Completeness (2 hours)

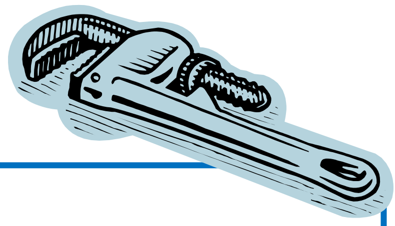
Lecture Delivery Method

1. Recorded lectures in **Course Media(Media Gallery) /Home**. Videos of one chapter in one PlayList.
2. Weekly review lectures/Q & As in **LT1** on **Mondays 12.30pm – 1.30pm**, Week 8 to Week 13. No lecture on Fridays unless notified otherwise.

Schedule

Week	Lecture materials to be studied by end of the week	Tutorials	Example classes
7	Analysis techniques (up to slide 38)	Graphs	Project 1
8	Analysis techniques (up to end), DP (up to DP slide 18)	Graphs	Project 2
9	DP (up to DP slide 40)	Analysis techniques	Project 2
10	DP (up to end)	DP	Quiz
11	String matching (up to end)	DP	Quiz
12	NP completeness (up to end)	String matching	Project 3
13		NP completeness	Project 3

Review lectures are from Week 8 to Week 13



Analysis Techniques

Huang Shell Ying

Reference: Computer Algorithms: Introduction to Design and Analysis, 3rd Ed, by Sara Basse and Allen Van Gelder.

Outline

- Review of the big oh, big omega, big theta

- Solving recurrences (1)
  1. The substitution method
  2. The iteration method
  3. The master method.

- Solving recurrences (2)
  1. Solving linear homogeneous recurrences with constant coefficients

Review of the big oh, big omega, big theta

The Big-oh notation:

Definition: Let f and g be 2 functions such that

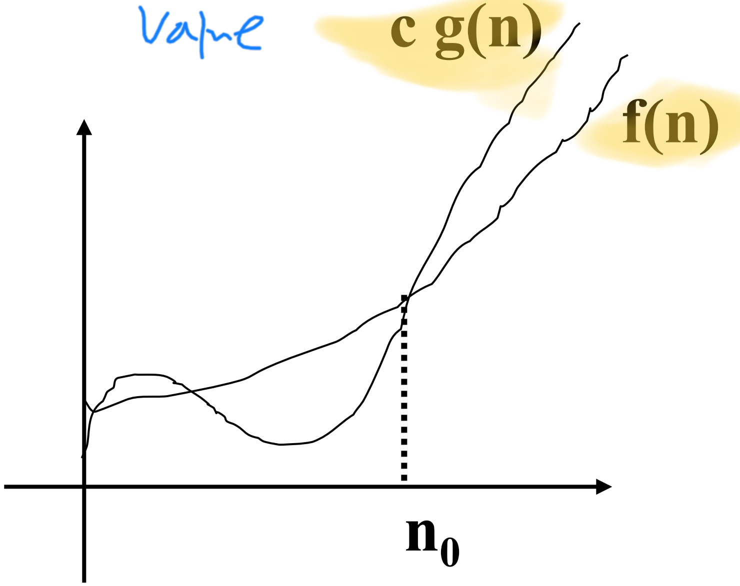
$f(n) : N \to R^+$ and $g(n) : N \to R^+$, if there exists positive

constants c and n_0 such that

$f(n) \le c * g(n)$ for all $n > n_0$

then $f(n) = O(g(n))$.

(+) value



Alternative definition: if

$$\lim_{n \to \infty} \frac{f(n)}{g(n)} = c < \infty$$

then $f(n) = O(g(n))$.

Review of the big oh, big omega, big theta

Example: $f(n)=lg(n)$, $g(n)= n$,

Let $c=1$, $n_0 = 1$, then for all $n>1$

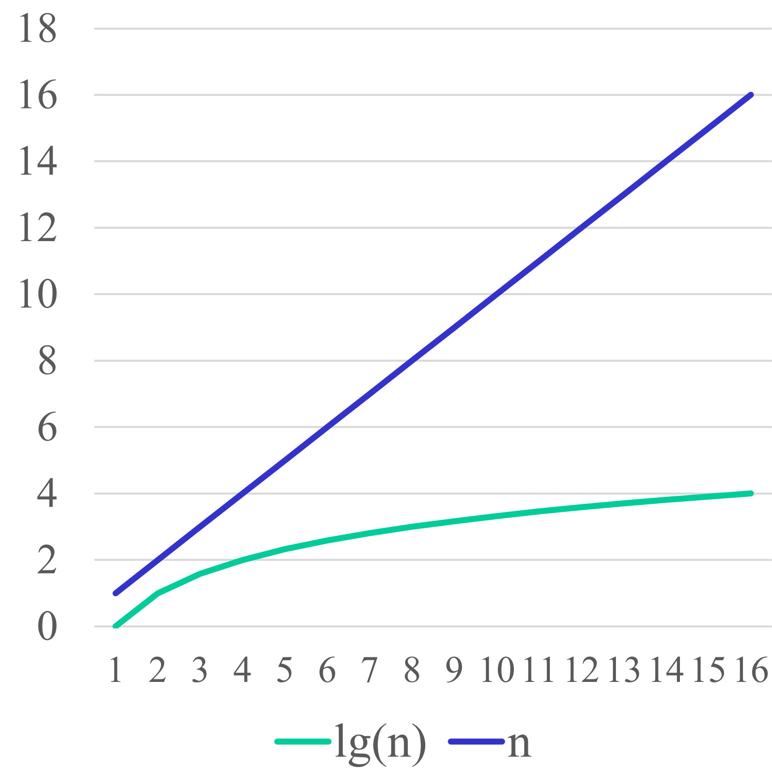
$lg(n) <= n$, i.e., $f(n) <= g(n)$

so $f(n) = O(g(n))$.

Another way: Since

$$\lim_{n\to\infty} \frac{f(x)}{g(x)} = \lim_{n\to\infty} \frac{\lg(n)}{n} = 0 < \infty$$

so $f(n) = O(g(n))$.



g(n) gives the asymptotic upper bound for f(n).

Review of the big oh, big omega, big theta

The big Omega notation

<u>Definition</u>: Let f and g be 2 functions such that

f(n) : N -> R^+ and g(n) : N -> R^+, if there exists positive

constants c and n_0 such that

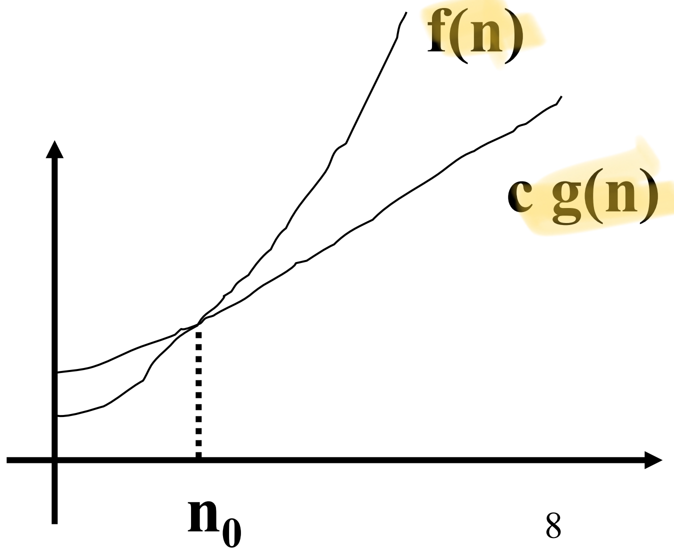
$f(n) >= c * g(n)$ for all $n > n_0$

then $f(n) = \Omega(g(n))$.

<u>Alternative definition</u>: if

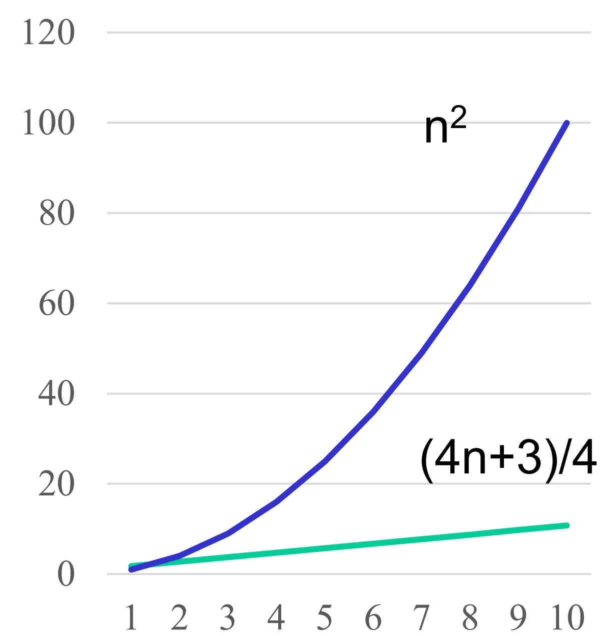
$$\lim_{n \to \infty} \frac{f(n)}{g(n)} > 0$$

then $f(n) = \Omega(g(n))$.



Review of the big oh, big omega, big theta

Example: $f(n)= n^2$, $g(n)=4n+3$



Let $c=1/4$, $n_0 = 1$, then for all $n>1$

$n^2 >= (4n+3)/4$ i.e., $f(n) >= (1/4)g(n)$

so $f(n) = \Omega(g(n))$.

Another way: Since

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = \lim_{n\to\infty} \frac{n^2}{4n+3} = \lim_{n\to\infty} \frac{n}{4+\frac{3}{n}} = \infty > 0$$

so $f(n) = \Omega(g(n))$.

g(n) gives the asymptotic lower bound for f(n).

Review of the big oh, big omega, big theta

The big Theta notation

Definition: Let f and g be 2 functions such that

$f(n) : N \to R^+$ and $g(n) : N \to R^+$, if there exists positive constants c_1, c_2 and n_0 such that

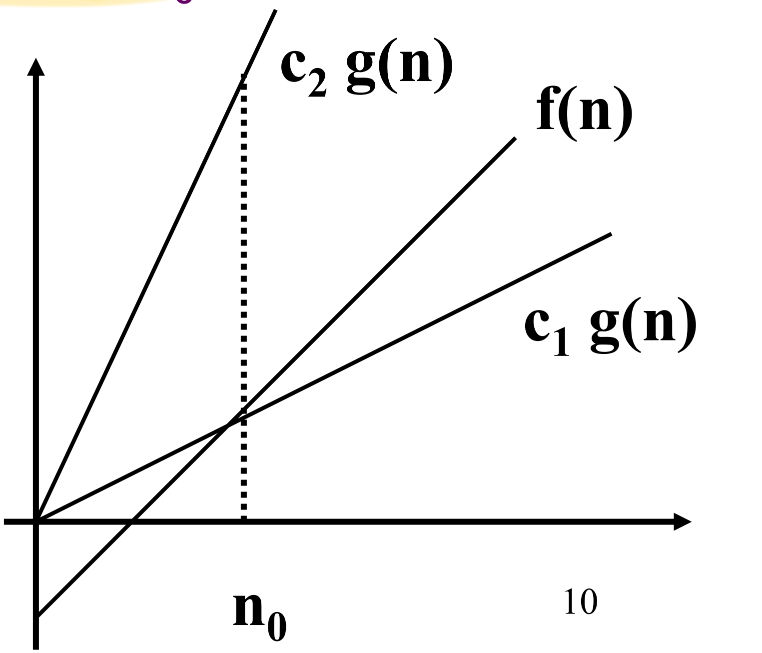
$c_1 * g(n) <= f(n) <= c_2 * g(n)$ for all $n > n_0$

then $f(n) = \theta(g(n))$.

Alternative definition: if

$$\lim_{n \to \infty} \frac{f(n)}{g(n)} = c \quad (0 < c < \infty)$$

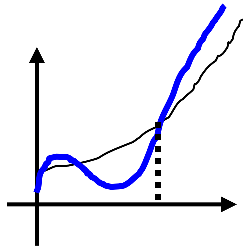
then $f(n) = \theta(g(n))$.



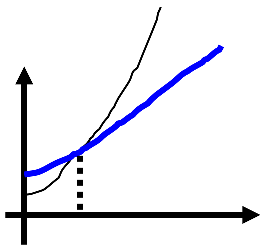
Review of the big oh, big omega, big theta

- The idea of the O, Ω and θ definitions is to establish a relative order among functions.
- We compare the **relative rates of growth**.

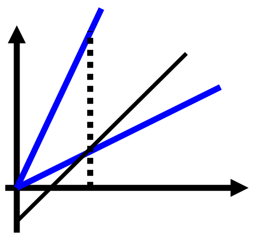
  - If f(n) = O(g(n)), g(n) gives the asymptotic upper bound



  - If f(n) = Ω(g(n)), g(n) gives the asymptotic lower bound



  - If f(n) = θ(g(n)), g(n) gives the asymptotic tight bound



Recursive algorithms and Recurrence relations

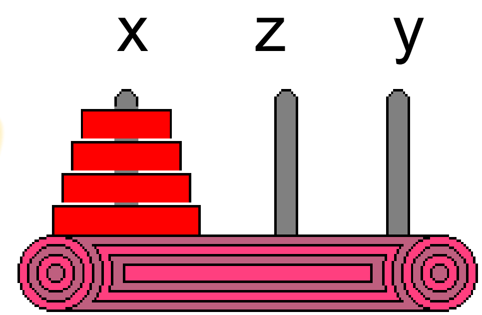
- Many problems have a recursive solution
- A common way of analysis for such solution algorithms will involve a recurrence relation that needs to be solved
- A recurrence is an equation or inequality that describe a function in terms of its value on smaller inputs, e.g.

$$M(n) = 2M(n-1) + 1$$

Example 1: Towers of Hanoi

Move all disks from the first pole to the third pole subject to the condition that only one disk can be moved at a time and that no disk is ever placed on top of a smaller one.

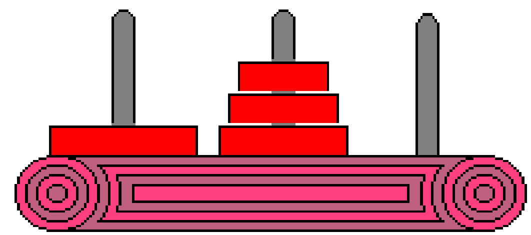
disk Pole

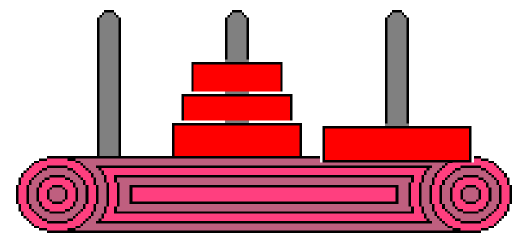


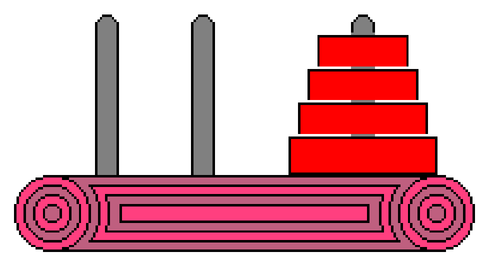

```
void TowersOfHanoi(int n, int x, int y, int z)
{ // Let M(n) be the total no. of disk moves
  if (n == 1)
     cout << "Move  disk from  " << x << " to  " << y << endl;
     // this has one disk move
  else {
    TowersOfHanoi(n-1, x, z, y);
    // this involves M(n-1) disk moves

    cout << "Move  disk from  " << x << " to  " << y << endl;
   // one disk move


    TowersOfHanoi(n-1, z, y, x);
    // another M(n-1) disk moves
  }
}
```







The number of disk moves: $M(1) = 1;$

$$M(n) = 2M(n-1) + 1$$

Example 2: Merge sort

```
void mergesort(int l, int m)
{
   int mid = (l+m)/2;
   if (m-l > 1) {
      mergesort(l, mid);
      mergesort(mid+1, m);
   }
   merge(l, m);
}
```

worst case

Let M(n) be the total no. of comparisons between array elements.

$M(2) = 1;$

$M(n) = 2M(n/2) + n - 1$

Solving recurrences (1)

- We want to solve recurrences of the form

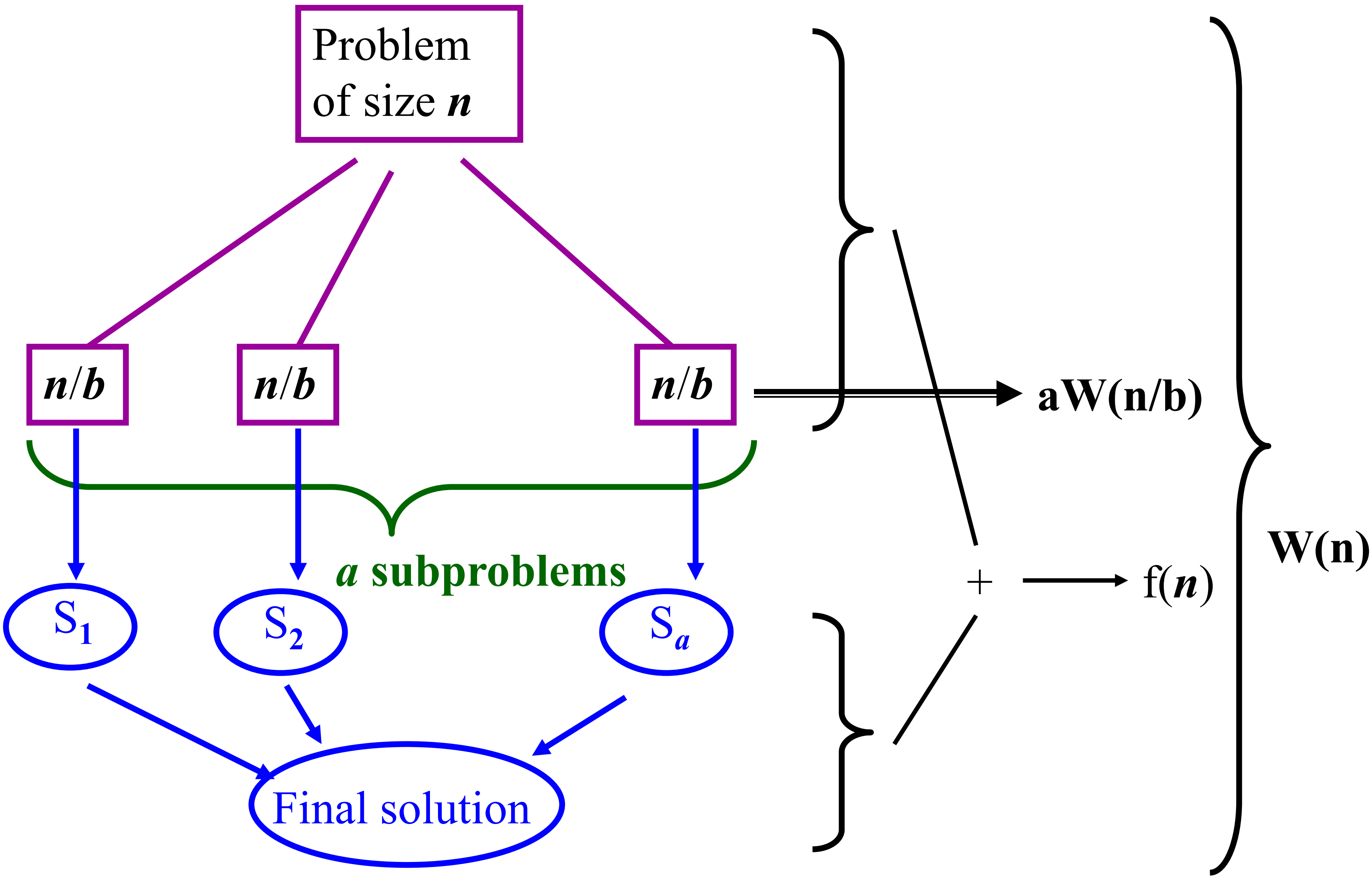
  $$W(n) = aW(n/b) + f(n)$$

  where $a \geq 1$ and $b > 1$ are constants, $f(n)$ is a function of n.

- The recurrence describes the computational cost of an algorithm that uses the "divide-and-conquer" approach.

- $f(n)$ is the cost of dividing the problem and combining the results of the subproblems.

- Usually the problem of size n is divided into subproblems of sizes either $\lceil n/b \rceil$ or $\lfloor n/b \rfloor$. However it does not change the asymptotic behaviour of the recurrence.


Problem
of size *n*
n/*b*
n/*b*
n/*b*
aW(n/b)
a subproblems
S1
S2
Sa
+
f(*n*)
W(n)
Final solution

Solving recurrences (1)

- Examples

$W(n) = 2W(n/2) + 2$	Finding the max and min from a sequence
$W(n) = W(n/2) + 2$	Binary search
$W(n) = 3W(n/2) + cn$	Multiplying two 2n-bits integers
$W(n) = 2W(n/2) + n - 1$	Merge sort
$W(n) = 7W(n/2) + 15n^2/4$	Multiplying two nxn matrices

Solving recurrences (1)

We describe three methods:

1) The substitution method
2) The iteration method
3) The master method.

1. The substitution method

- It is a "guess and check" strategy. First guess the form of the solution and then use mathematical induction to prove it.
- A powerful method because often it is easier to prove that a certain bound (in the form of the O notation) is valid than to compute the bound.

- but the method is only useful when it is easy to guess the form of the solution.
- **Mathematical Induction**: If *p*(*a*) is true and, for some integer $k \geq a$, $p(k+1)$ is true whenever $p(k)$ is true, then $p(n)$ is true for all $n \geq a$.
- Example: The worst case for merge sort ($n = 2^k$)

  $W(2) = 1$

  $W(n) = 2\,W(n/2) + n - 1$

  Guess $W(n) = O(f(n))$ then prove it.

  Show (i) $W(2) <= f(2)$ (ii) for some integer $k \geq 2$, assume $W(n) = O(f(n))$ for $n \leq 2^k$, prove $W(2n) <= f(2n)$ then $W(n) = O(f(n))$ for all $n \geq 2$.

First guess: **$W(n) = O(n^2)$**

Proof by mathematical induction that $W(n) \le cn^2$:

(1) Base case: $W(2) = 1 \le 2^2$;

(2) Inductive step: assume that $W(n) = O(n^2)$ for $n \le 2^k$. Now consider $n = 2^{k+1}$

$$
\begin{aligned}
W(2^{k+1}) &= 2W(2^k) + 2^{k+1} - 1 \\
&\le 2 * (2^k)^2 + 2^{k+1} - 1 \\
&= 2 * (2^k)^2 + 2 * 2^k - 1 \\
&\le 4 * (2^k)^2 \\
&= (2^{k+1})^2
\end{aligned}
$$

i.e. $W(2^{k+1}) \le (2^{k+1})^2$

A lot is added from step 3 to step 4

Thus $W(n) = O(n^2)$. But is this the best guess?

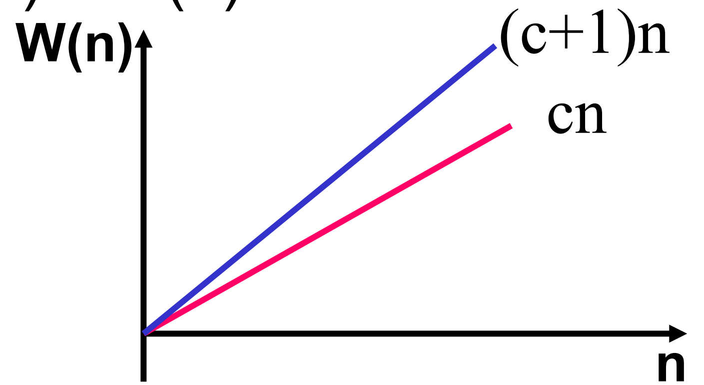
Second guess: W(n) = O(n), i.e. W(n) ≤ c * n

Proof by mathematical induction:

(1) Base case: $W(2) = 1 \le 2c$;

(2) Inductive step: assume that $W(n) = O(n)$ for $n \le 2^k$.
Now consider $n = 2^{k+1}$

$$W(2^{k+1}) = 2W(2^k) + 2^{k+1} - 1$$
$$\le 2 * c * 2^k + 2^{k+1} - 1$$
$$= c * 2^{k+1} + 2^{k+1} - 1$$



Thus $W(2^{k+1}) \le (c+1) * 2^{k+1} - 1$ but we cannot say
$W(2^{k+1}) \le c * 2^{k+1}$

Thus $W(n) \ne O(n)$.

Third guess: W(n) = O(nlgn)

Proof by mathematical induction:

(1) Base case: $W(2) = 1 \leq 2\lg 2;$

(2) Inductive step: assume that $W(n) \leq n\lg n$ for $n \leq 2^k$.
Now consider $n = 2^{k+1}$

$$
\begin{aligned}
W(2^{k+1}) &= 2W(2^k) + 2^{k+1} - 1 \\
&\leq 2 * k * 2^k + 2^{k+1} - 1 \\
&= k * 2^{k+1} + 2^{k+1} - 1 \\
&\leq (k + 1) * 2^{k+1}
\end{aligned}
$$

?

Thus W(n) = O(nlgn) is a very close upper bound.

What if the base condition does not hold?

Consider the recurrence ($n = 2^k$) :

$W(1) = 1$

$W(n) = 2\, W(n/2) + n - 1$

Prove that $W(n) = O(n\lg n)$:

(1) Base case: $W(1) = 1 > c\lg 1$;

(2) Recall the big-O notation: for $f(n) = O(g(n))$, we need $f(n) <= c * g(n)$ for all $n > n_0$.

(3) Thus to prove $W(n) = O(n\lg n)$, we may use another base case.

– We have $W(2) = 3 < c*2*\lg 2$ for any $c > 1$.

– We can assume that $W(n) \le cn\lg n$ for $n \le 2^k$ then prove $W(2^{k+1}) \le c*(k + 1) * 2^{k+1}$

Then $W(n) = O(n\lg n)$.

What can we say about the general case of n?

The worst case for merge sort :

$W(2) = 1$

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

Proof [+]:

(1) W(n) is a monotonically increasing function. So when n is not a power of 2, that is, $2^k < n < 2^{k+1}$, then $W(2^k) \leq W(n) \leq W(2^{k+1})$.

(2) We have proved that W(n) = O(nlgn) for powers of 2, so, $W(2^{k+1}) \leq c * (k+1) * 2^{k+1}$.

(3) For any $n < 2^{k+1}$ for some k, $W(n) \leq W(2^{k+1})$. Therefore

$W(n) \leq c * (k+1) * 2^{k+1} < c * \lg(2n) * (2*n) < 4cn\lg n$

Therefore W(n) = O(nlgn).

[+] **See *The design and analysis of Algorithms* by Anany Levitin (pp481-483) about Smoothness Rule.**

2. The iteration method

- The idea is to expand (iterate) the recurrence and express it as a summation of terms depending only on n and the initial condition.
- Techniques for evaluating summations can then be used to provide bounds on the solution.
- Example:

W(1) = 1, W(2) = 1, W(3) = 1,

$W(n) = 3W(\lfloor \frac{n}{4} \rfloor) + n$

we expand (iterate) it:

$W(n) = 3W(\lfloor \frac{n}{4} \rfloor) + n$

$= 3(3W(\lfloor \frac{n}{4^2} \rfloor) + \lfloor \frac{n}{4} \rfloor) + n$

$= 3^2 W(\lfloor \frac{n}{4^2} \rfloor) + 3\lfloor \frac{n}{4} \rfloor + n$

$= 3^2(3W(\lfloor \frac{n}{4^3} \rfloor) + \lfloor \frac{n}{4^2} \rfloor) + 3\lfloor \frac{n}{4} \rfloor + n$

$= 3^3 W(\lfloor \frac{n}{4^3} \rfloor) + 3^2\lfloor \frac{n}{4^2} \rfloor + 3\lfloor \frac{n}{4} \rfloor + n$

we need to iterate until we reach one of the boundary conditions, i.e $\lfloor \frac{n}{4^i} \rfloor = 1, 2$ or 3.

E.g. $n=64$, $4^3 \leq 64 < 4^4$ and $\lfloor \frac{64}{4^3} \rfloor = 1$;

$n=255$, $4^3 \leq 255 < 4^4$ and $\lfloor \frac{255}{4^3} \rfloor = 3$;

This means if $4^i \leq n < 4^{i+1}$ then $i = \lfloor \log_4 n \rfloor$. So

$W(n) = 3^i W(a) + 3^{i-1}\lfloor \frac{n}{4^{i-1}} \rfloor + \ldots + 3^2\lfloor \frac{n}{4^2} \rfloor + 3\lfloor \frac{n}{4} \rfloor + n$

a = 1,2 or 3

$$W(n) = 3^i W(a) + 3^{i-1} \left\lfloor \frac{n}{4^{i-1}} \right\rfloor + \ldots + 3^2 \left\lfloor \frac{n}{4^2} \right\rfloor + 3 \left\lfloor \frac{n}{4} \right\rfloor + n$$

$$\leq 3^{\log_4 n} W(a) + 3^{i-1} \frac{n}{4^{i-1}} + \ldots + 3^2 \frac{n}{4^2} + 3 \frac{n}{4} + n$$

Let $x = 3^{\log_4 n}$ then

$\log_4 x = \log_4 n \log_4 3$ then

$4^{\log_4 x} = 4^{\log_4 n \log_4 3}$ then

$x = n^{\log_4 3}$, i.e. $3^{\log_4 n} = n^{\log_4 3}$

$$\sum_{i=0}^{\infty} \left(\frac{3}{4}\right)^i = 4$$

$$W(n) \leq n^{\log_4 3} + n \sum_{i=0}^{\infty} \left(\frac{3}{4}\right)^i + = O(n)$$

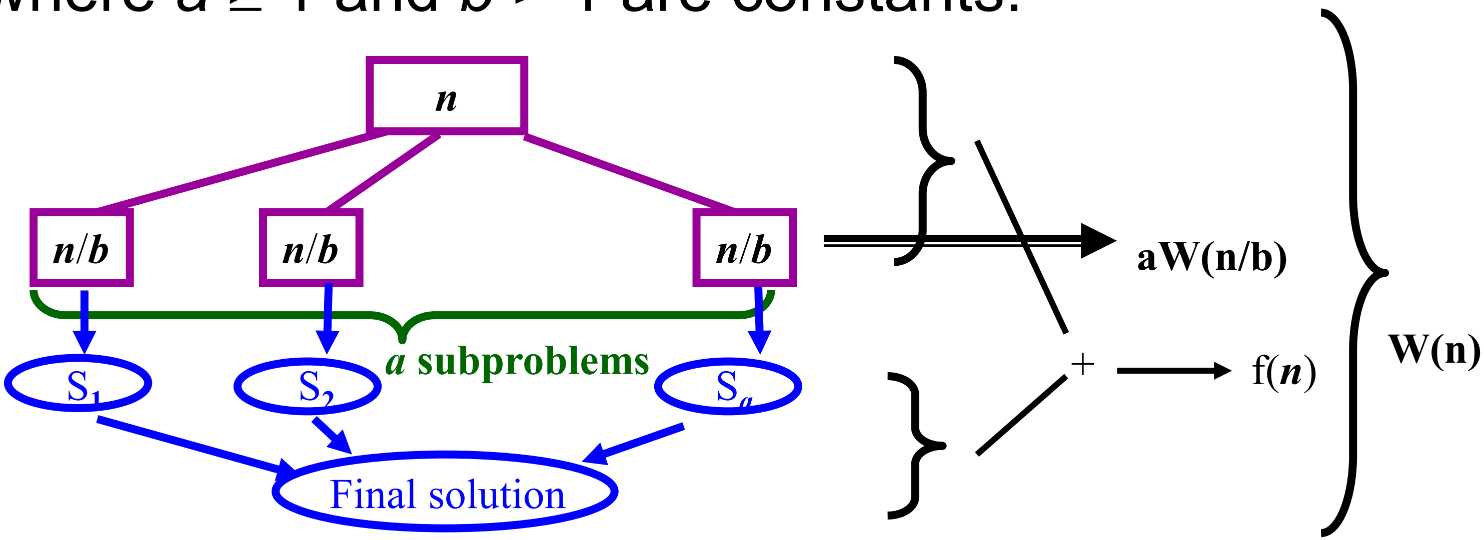
- The iteration method usually leads to lots of algebra.
- We should focus on how many times the recurrence needs to be iterated to reach the boundary condition.

3. The master method

- The master method provides a "manual" for solving recurrences of the form

$$\mathbf{W}(n) = a\mathbf{W}(n/b) + \mathbf{f}(n)$$

where $a \geq 1$ and $b > 1$ are constants.



- We are able to determine the asymptotic tight bound in the following three cases

The master theorem

For **$W(n) = aW(n/b) + f(n)$** $a \geq 1$ and $b > 1$

The manual:

1. If $f(n) = O(n^{\log_b a - \varepsilon})$ for some constant $\varepsilon > 0$, then $W(n) = \theta(n^{\log_b a})$.

2. If $f(n) = \theta(n^{\log_b a})$, then $W(n) = \theta(n^{\log_b a} \log n)$.

   If $f(n) = \theta(n^{\log_b a} \log^k n)$, $k \geq 0$,
   then $W(n) = \theta(n^{\log_b a} \log^{k+1} n)$

3. If $f(n) = \Omega(n^{\log_b a + \varepsilon})$ for some constant $\varepsilon > 0$, and if $a\, f(n/b) \leq c\, f(n)$ for some constant $c < 1$ and all sufficiently large n, then $W(n) = \theta(f(n))$.

What is $n^{\log_b a}$?

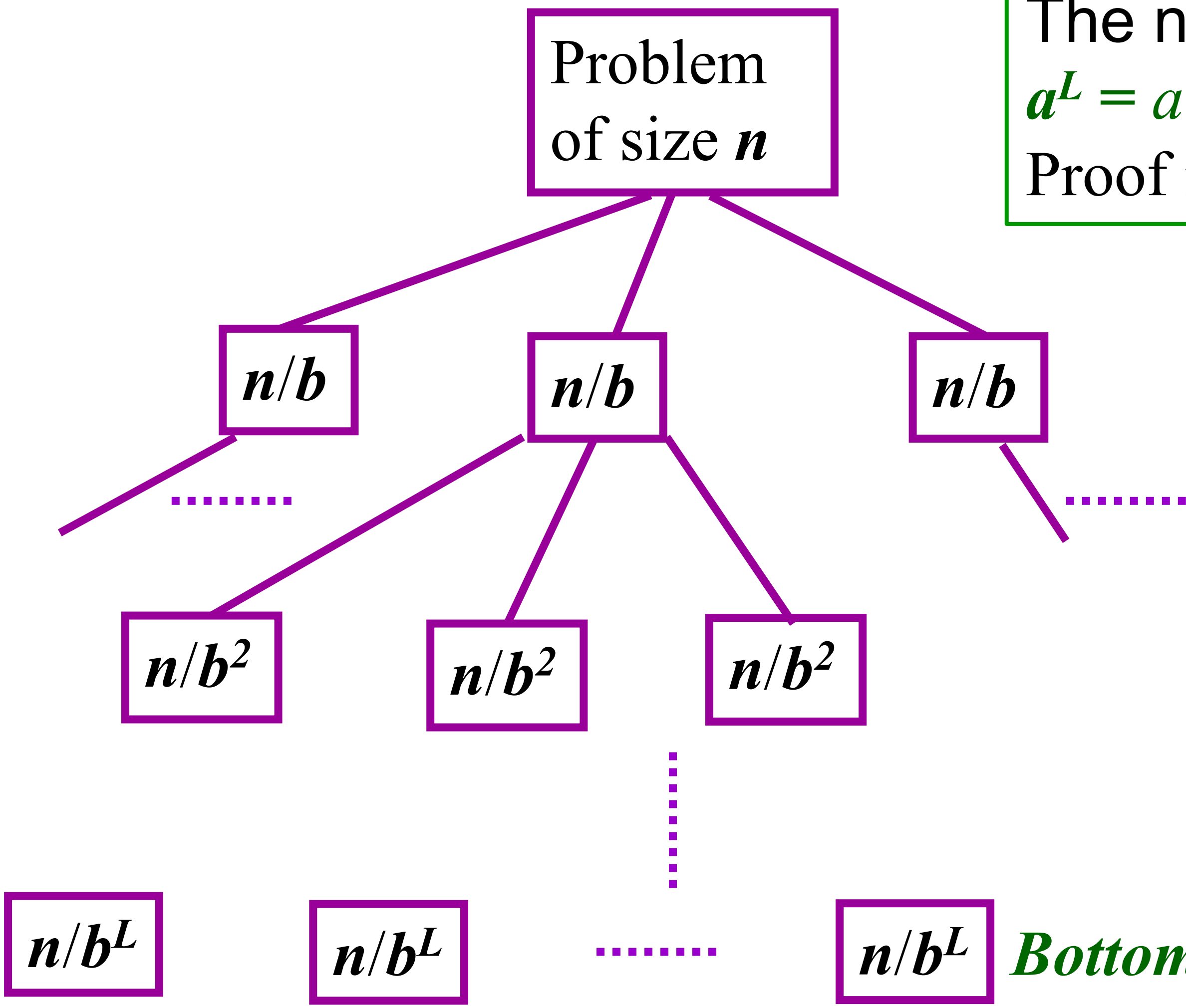
The depth of the tree

$L = \log_b n$

The number of leaves is

$a^L = a^{\log_b n} = n^{\log_b a}$

Proof in slide 54

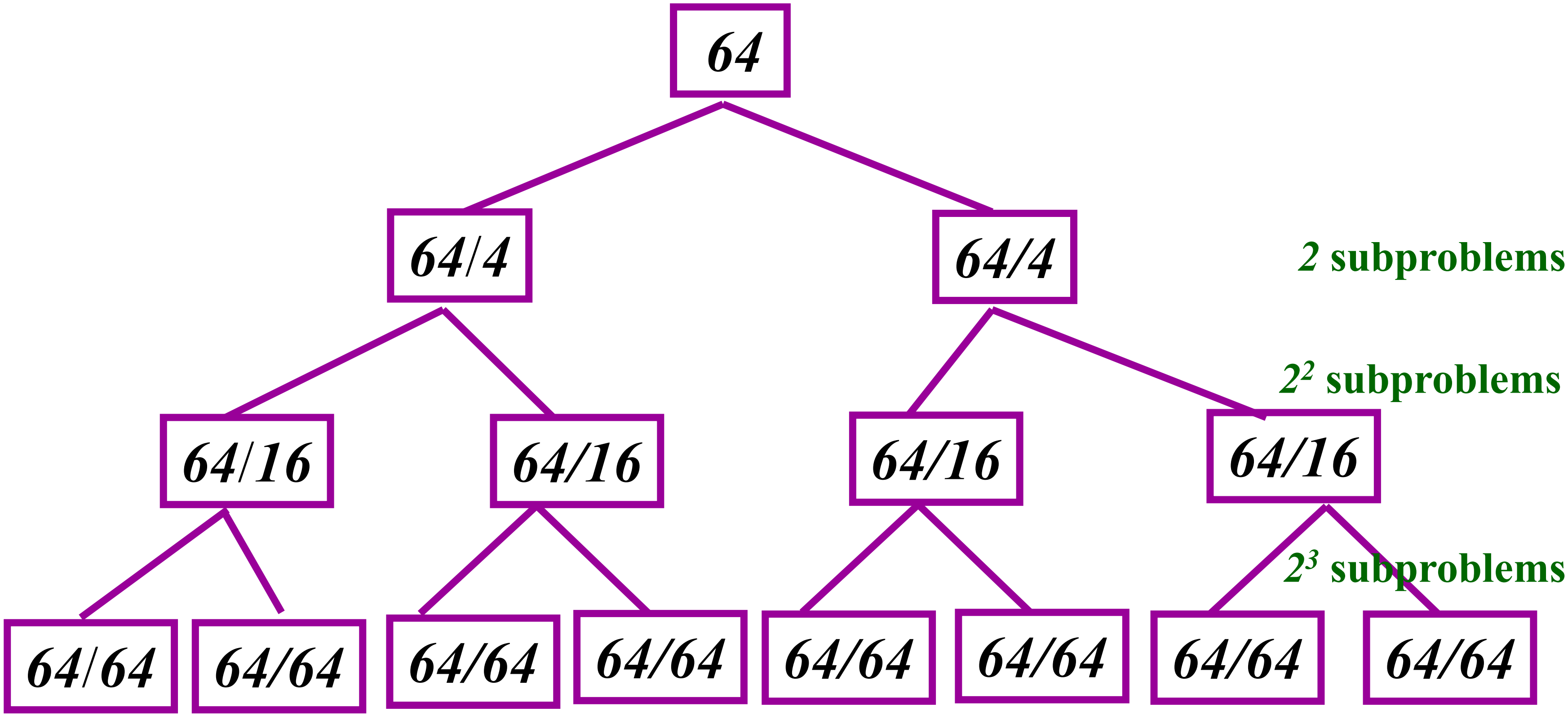


***a* subproblems**

a^2 subproblems

***Bottom level:* a^L subproblems**

E.g. $\boldsymbol{n} = 64$, $\boldsymbol{a} = 2$, $\boldsymbol{b} = 4$



Depth of tree L = $\log_4 64$, Number of leaves = $8 = 2^{\log_4 64} = 64^{\log_4 2}$

$(a^{\log_b n} = n^{\log_b a})$

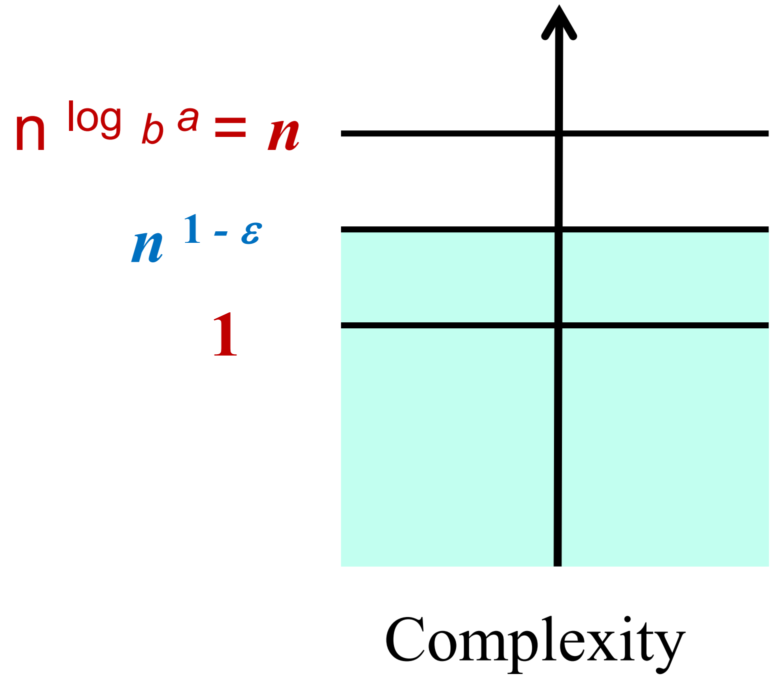
Examples

1) $W(n) = 3W(n/3) + 2$,

so $a = 3$, $b = 3$,

$n^{\log_b a} = n^1$

$f(n) = 2 = \theta(1) = O(n^1)$



We may let $\varepsilon = 0.5$ then we confirm $2 = O(n^{1-0.5})$,

i.e. $f(n) = O(n^{1-\varepsilon})$

$\Rightarrow f(n) = O(n^{\log_b a - \varepsilon})$ (case 1)

thus $W(n) = \theta(n^{\log_b a})$

$W(n) = \theta(n)$.

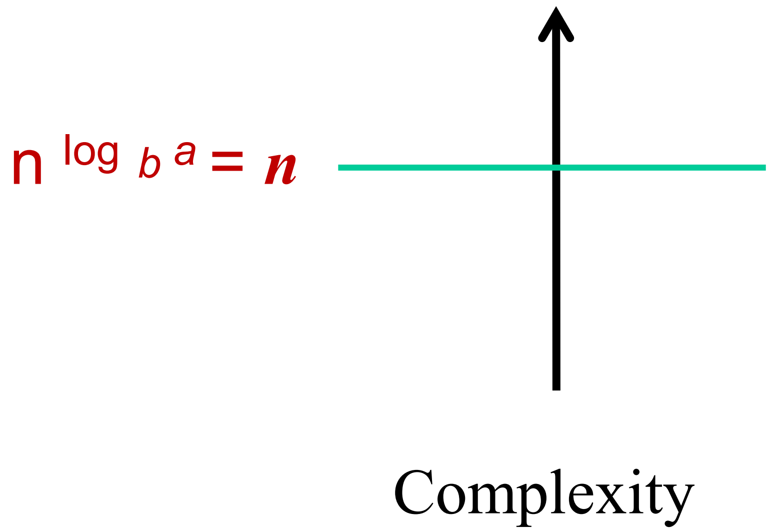
Examples

2) $W(n) = 4W(n/4) + n - 1$,

so $a = 4$, $b = 4$,

$n^{\log_b a} = n^1$

$f(n) = n - 1$



We have

$f(n) = n-1$

$= \theta(n^1)$,

$= \theta(n^{\log_b a})$, (case 2)

thus

$W(n) = \theta(n^{\log_b a} \log n)$

$= \theta(n \log n)$

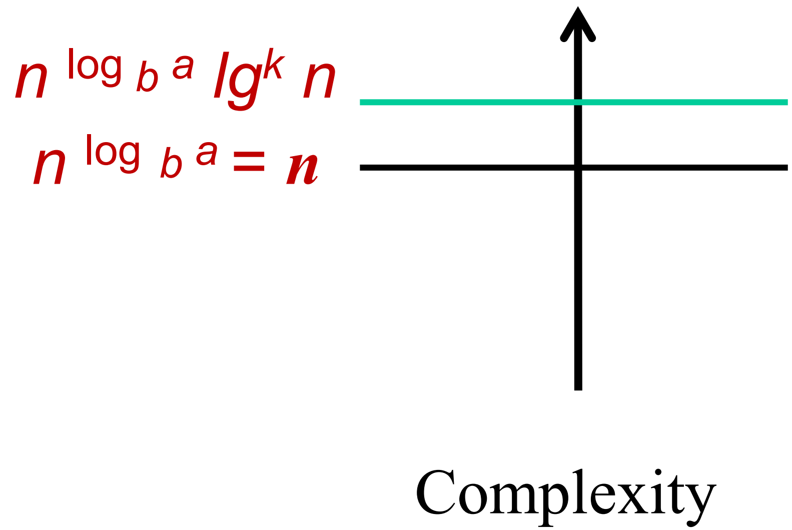
Examples

3) W(n) = 2W(n/2) + n lg n,

so a = 2, b = 2,

f(n) = n lg n

$n^{\log_b a} = n^1$



We have

f(n) = $\theta(n^1\, lg\, n)$,

= $\theta(n^{\log_b a}\, lg^k\, n)$, (case 2: k = 1)

thus

W(n) = $\theta(n^{\log_b a}\, lg^2\, n)$

= $\theta(n\, (lg\, n)^2)$

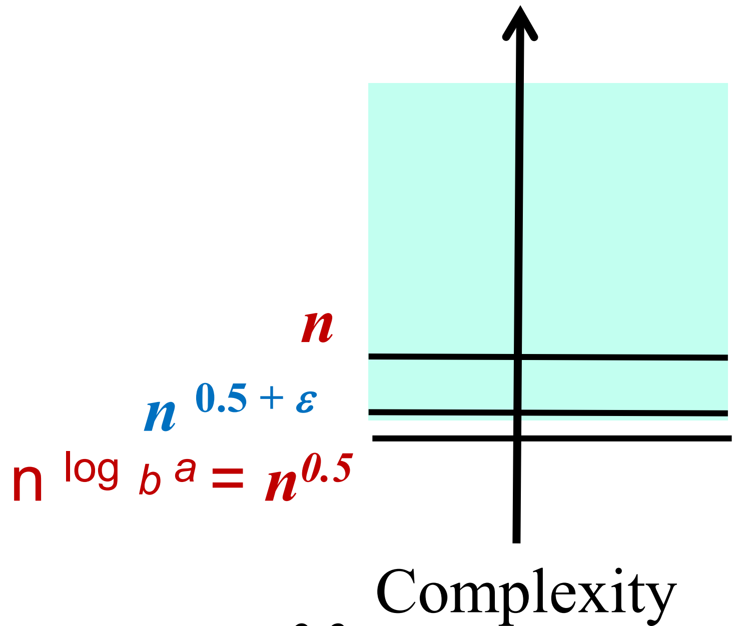
Examples

4) W(n) = 2W(n/4) + n,

so a = 2, b = 4,

$n^{\log_b a} = n^{\log_4 2} = n^{0.5}$

$f(n) = n = \theta(n)$



We may let $\varepsilon = 0.1$ then we have $n = \Omega(n^{0.6})$

i.e. $f(n) = \Omega(n^{\log_b a + \varepsilon})$, and

for all sufficiently large n, we can find a value for c, say, c = ¾, to show that $a\, f(n/b) \le c\, f(n)$. (case 3)

$a^*f(n/b) = 2^*f(n/4) = n/2 \le c^*n$

thus $W(n) = \theta(n)$.

Sometimes the master method cannot apply

Example 1: W(n) = 3W(n/3) + n/lgn, $n^{\log_b a} = n^1$

f(n) = n/lg$n$ = O(n^1) because $\lim_{n\to\infty} \frac{n/lgn}{n^1} = \lim_{n\to\infty} \frac{1}{lgn} = 0$

f(n) = O($n^{1-\varepsilon}$) ? (L'Hôpital's rule, slide 55)

i.e. n/lg$n$ = O($n^{1-\varepsilon}$) ?

No, because asymptotically, n/lg n > $n^{1-\varepsilon}$ for any ε > 0

$$\lim_{n\to\infty} \frac{n/lgn}{n^{1-\varepsilon}} = \lim_{n\to\infty} \frac{n^{\varepsilon}}{lgn} = \infty$$

This recurrence falls into the gap between case 2 and case 3.

So the Master Theorem cannot apply.

Sometimes the master method cannot apply

Example 2: W(n) = W(n/3) + f(n)

where f(n) = $\begin{cases} 3n + 2^{3n} & for\ n = 2^i \\ 3n & otherwise \end{cases}$

so a = 1, b = 3 then $n^{\log_b a} = n^0$

let $\varepsilon = 1$ then f(n) = $\Omega(n^{0+1})$, case 3?

a f(n/b) ≤ c f(n) for all sufficiently large n?

When $n = 3 * 2^i$, a f(n/b) = f(2^i) = $n + 2^n$, but cf(n) = c(3n)

i.e. a f(n/b) > c f(n) . E.g. for n = 6 or greater

So the Master Theorem cannot apply.

- Notice that when we want to find the order of a recurrence, the initial conditions are not important. This is because the running costs of the terminating conditions are small constants that do not affect the order.

Solving recurrences (2)

- Definition: A *linear homogeneous recurrence relation of degree k with constant coefficients* is a recurrence relation of the form

  $$a_n = c_1 a_{n-l} + c_2 a_{n-2} + \ldots + c_k a_{n-k},$$

  where $c_1, c_2, \ldots, c_k$ are real constants and $c_k \neq 0$.

The two different notations: A(n) and a_n

- When using A(n), we mean the function value with parameter n
- When using a_n, we mean the nth term in a sequence $a_1, a_2, \ldots, a_n$.
- If we list A(1), A(2), …, A(n) in a sequence, we can write them as $a_1, a_2, \ldots, a_n$. They are equivalent.

Solving recurrences (2)

- Definition: A *linear homogeneous recurrence relation of degree k with constant coefficients* is a recurrence relation of the form

  $$a_n = c_1 a_{n-1} + c_2 a_{n-2} + \dots + c_k a_{n-k},$$

  where $c_1, c_2, \dots, c_k$ are real constants and $c_k \neq 0$.

  - **Linear**: $a_{n-1}, a_{n-2}, \dots, a_{n-k}$ appear in separate terms and to the first power
  - **Homogeneous**: the total degree of each term is the same, e.g. no constant term
  - **Constant coefficients**: $c_1, c_2, \dots, c_k$ are fixed real constants that do not depend on n
  - ***Degree k***: the expression for a_n contains the previous k terms $a_{n-1}, a_{n-2}, \dots, a_{n-k}$, ($c_k \neq 0$)

- Examples
  - A *linear homogeneous recurrence relation of degree 2:* $a_n = a_{n-1} + a_{n-2}$
  - A *linear homogeneous recurrence relation of degree 1:* $a_n = 1.04a_{n-1}$
  - A *linear homogeneous recurrence relation of degree 3 :* $a_n = a_{n-3}$
- Non-examples
  - $a_n = a_{n-1} + a_{n-2} + 1$: non-homogeneous
  - $a_n = a_{n-1}a_{n-2}$: not linear
  - $a_n = na_{n-1}$: coefficient not constant

- A linear homogeneous recurrence relation of degree *k* can be systematically solved, i.e. find the explicit expression for a_n

- The basic approach is to look for solutions of the form $a_n = t^n$ where t is a constant

- If $a_n = t^n$ is a solution for

$$a_n = c_1 a_{n-1} + c_2 a_{n-2} + \ldots + c_k a_{n-k},$$

Then

$$t^n = c_1 t^{n-1} + c_2 t^{n-2} + \ldots + c_k t^{n-k}$$

$$\Rightarrow t^k = c_1 t^{k-1} + c_2 t^{k-2} + \ldots + c_k \qquad \text{(divide both side by } t^{n-k}\text{)}$$

$$\Rightarrow t^k - c_1 t^{k-1} - c_2 t^{k-2} - \ldots - c_k = 0$$

- This means if we can solve the equation

$$t^k - c_1 t^{k-1} - c_2 t^{k-2} - \ldots - c_k = 0$$

to find t, then $a_n = t^n$ is a solution for

$$a_n = c_1 a_{n-1} + c_2 a_{n-2} + \ldots + c_k a_{n-k}$$

We call

$$t^k - c_1 t^{k-1} - c_2 t^{k-2} - \ldots - c_k = 0$$

the **characteristic equation** of

$$a_n = c_1 a_{n-1} + c_2 a_{n-2} + \ldots + c_k a_{n-k}$$

The solutions to the characteristic equation are called the **characteristic roots**

- We consider a linear homogeneous recurrence relation of degree 2

  $a_n = Aa_{n-1} + Ba_{n-2}$ *for all* $n \geq 2$

  where A and B are real constants

- The **characteristic equation**

  $t^2 - At - B = 0$

  may have

  1) two distinct roots

  2) a single root

- **Theorem 1 (Distinct Roots Theorem)**

  Suppose a sequence $a_0, a_1, a_2, \ldots$. satisfies a recurrence relation

  $$a_n = Aa_{n-l} + Ba_{n-2} \text{ for all } n \geq 2$$

  where A and B are real constants and $B \neq 0$. If the characteristic equation

  $$t^2 - At - B = 0$$

  has two distinct roots r and s, then $a_0, a_1, a_2, \ldots$. is given by the explicit formula

  $$\boldsymbol{a_n = Cr^n + Ds^n}$$

  where C and D are determined by the values of a_0 and a_1.

- Example 1:

$$F_n = F_{n-l} + F_{n-2} \text{ for all } n \geq 2, \quad \text{and } F_0 = F_1 = 1$$

The characteristic equation is

$$t^2 - t - 1 = 0$$

For $ax2 + bx + c = 0$

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

The roots are

$$t = \frac{1 \pm \sqrt{1 - 4(-1)}}{2} = \begin{cases} \frac{1+\sqrt{5}}{2} \\ \frac{1-\sqrt{5}}{2} \end{cases}$$

$$F_n = C\left(\frac{1+\sqrt{5}}{2}\right)^n + D\left(\frac{1-\sqrt{5}}{2}\right)^n$$

- To find C and D, we have

$$F_0 = 1 = C\left(\frac{1+\sqrt{5}}{2}\right)^0 + D\left(\frac{1-\sqrt{5}}{2}\right)^0 = C\cdot 1 + D\cdot 1 = C + D$$

$$F_1 = 1 = C\left(\frac{1+\sqrt{5}}{2}\right)^1 + D\left(\frac{1-\sqrt{5}}{2}\right)^1 = C\left(\frac{1+\sqrt{5}}{2}\right) + D\left(\frac{1-\sqrt{5}}{2}\right)$$

To solve this system of 2 equations with 2 unknowns, from

$C + D = 1$

$\Rightarrow \left(\frac{1+\sqrt{5}}{2}\right)C + \left(\frac{1+\sqrt{5}}{2}\right)D = \left(\frac{1+\sqrt{5}}{2}\right)$

Then

$$D\left(\left(\frac{1+\sqrt{5}}{2}\right) - \left(\frac{1-\sqrt{5}}{2}\right)\right) = \left(\frac{1+\sqrt{5}}{2}\right) - 1$$

$$\Rightarrow \quad D\sqrt{5} = \left(\frac{1+\sqrt{5}}{2}\right) - 1$$

$$\Rightarrow D = \left(\frac{-1+\sqrt{5}}{2\sqrt{5}}\right)$$

Then $C = 1 - D = 1 - \left(\frac{-1+\sqrt{5}}{2\sqrt{5}}\right)$

$$\Rightarrow C = \frac{1+\sqrt{5}}{2\sqrt{5}}$$

We can write

$$D = \left(\frac{-(1-\sqrt{5})}{2\sqrt{5}}\right)$$

- So

$$F_n = \left(\frac{1+\sqrt{5}}{2\sqrt{5}}\right)\left(\frac{1+\sqrt{5}}{2}\right)^n + \left(\frac{-(1-\sqrt{5})}{2\sqrt{5}}\right)\left(\frac{1-\sqrt{5}}{2}\right)^n$$

After simplifying it, we get

$$F_n = \frac{1}{\sqrt{5}}\left(\frac{1+\sqrt{5}}{2}\right)^{n+1} - \frac{1}{\sqrt{5}}\left(\frac{1-\sqrt{5}}{2}\right)^{n+1}$$

for all $n \geq 0$.

- Example 2:

$$a_n = 5a_{n-l} - 6a_{n-2}\ ,\ a_0 = 9,\ a_1 = 20$$

The characteristic equation is

$$t^2 - 5t + 6 = 0$$

$\Rightarrow \quad (t-2)(t-3) = 0 \quad \Rightarrow$ two roots: $t = 2,\ t = 3$

$a_n = C2^n + D3^n$ for all n $\geq$ 0.

To find *C* and *D*:

$9 = C + D, \Rightarrow 18 = 2C + 2D$

$20 = 2C + 3D$

Thus $D = 2$, $C = 7$ So $a_n = 7*2^n + 2*3^n$ for all n $\geq$ 0

- **Theorem 2 (Single-Root Theorem)**

  Suppose a sequence $a_0, a_1, a_2, \ldots.$ satisfies a recurrence relation

  $$a_n = Aa_{n-l} + Ba_{n-2} \text{ for all } n \geq 2$$

  where A and B are real constants and $B \neq 0$. If the characteristic equation

  $$t^2 - At - B = 0$$

  has a single (real) root, then $a_0, a_1, a_2, \ldots.$ is given by the explicit formula

  $$\boldsymbol{a_n = Cr^n + Dnr^n}$$

  where C and D are determined by the values of a_0 and any other known value of the sequence.

- Example

  $b_n = 4b_{n-l} - 4b_{n-2}$ *for all* $n \geq 2$

  with $b_0 = 1$, $b_1 = 3$.

  The characteristic equation is

  $t^2 - 4t + 4 = 0$

  $\Rightarrow (t-2)^2 = 0 \quad \Rightarrow$ single root $t = 2$

  The explicit formula is

  $b_n = C2^n + Dn2^n$

  where C and D are determined by the values of b_0 and b_1.

  We have $1 = C$ and $3 = 2C + 2D$, so $D = ½$ and $C = 1$.

- Therefore

$$b_n = 2^n + (½)n2^n = (1 + n/2)\, 2^n$$

Theorem 1 can be generalised to the recurrence relation

$$a_n = c_1 a_{n-1} + c_2 a_{n-2} + \dots + c_k a_{n-k}$$

with characteristic equation

$$t^k - c_1\, t^{k-1} - c_2\, t^{k-2} - \dots - c_k = 0$$

having *k* distinct roots.

Theorem 2 can be generalised to less than *k* distinct roots.

Proof of $a^{\log_b n} = n^{\log_b a}$

- Let $L = \log_b n$, i.e. $b^L = n$

$$\Rightarrow (b^L)^{\log_b a} = n^{\log_b a}$$

$$\Rightarrow (b^{\log_b a})^L = n^{\log_b a}$$

$$\Rightarrow a^L = n^{\log_b a}$$

$$\Rightarrow a^{\log_b n} = n^{\log_b a}$$

L'Hôpital's rule

L'Hôpital's rule states that for functions $f(x)$ and $g(x)$, if:

$$\lim_{n\to\infty} f(x) = \lim_{n\to\infty} g(x) = \pm\infty$$

then:

$$\lim_{n \to \infty} \frac{f(n)}{g(n)} = \lim_{n \to \infty} \frac{f'(n)}{g'(n)}$$

where the prime (') denotes the derivative.