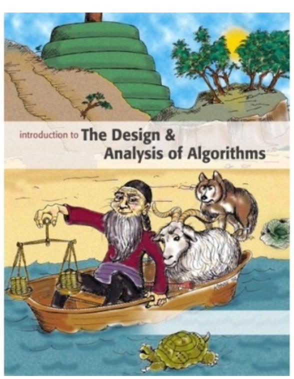
Introduction to

Algorithm Design and Analysis

[3] Recursion



Yu Huang

http://cs.nju.edu.cn/yuhuang
Institute of Computer Software
Nanjing University

In the Last Class …

- **Asymptotic growth rate**
  - o O, Ω, Θ
  - o o, ω
- **Brute force algorithms**
  - o By iteration
  - o By recursion

Recursion

- **Recursion in algorithm design**
  - The divide and conquer strategy
  - Proving the correctness of recursive procedures
- **Solving recurrence equations**
  - Some elementary techniques
  - Master theorem

Recursion in Algorithm Design

- **Computing n! with Fac(n)**
  - if n=1 then return 1 else return Fac(n-1)*n

M(1)=0 and M(n)=M(n-1)+1 for n>0
(critical operation: multiplication)

- **Hanoi Tower**
  - if n=1 then move d(1) to peg3 else
    Hanoi(n-1, peg1, peg2); move d(n) to peg3; Hanoi(n-1, peg2, peg3)

M(1)=1 and M(n)=2M(n-1)+1 for n>1
(critical operation: move)

Recursion in Algorithm Design

- **Counting the Number of Bits**
  - Input: a positive decimal integer *n*
  - Output: the number of binary digits in *n*'s binary representation

Int BitCounting (int n)
1. If(n==1) return 1; 2. Else 3. return BitCounting(n div 2) +1;

$$T(n)=\begin{cases}0 & n=1\\ T(\lfloor n/2\rfloor)+1 & n>1\end{cases}$$

Divide and Conquer

- **Divide**
  - Divide the "big" problem to smaller ones
- **Conquer**
  - Solve the "small" problems by recursion
- **Combine**
  - Combine results of small problems, and solve the original problem

Divide and Conquer

The general pattern

solve(I)

n=size(I);

if ($n \le$smallSize)

solution=directlySolve(I);

else

divide I into $I_1, \ldots I_k$;

for each $i \in \{1, \ldots, k\}$

S_i=solve(I_i);

solution=combine($S_1, \ldots, S_k$);

return solution

$T(n) = B(n)$ for $n \le smallSize$

$$T(n) = D(n) + \sum_{i=1}^{k} T(size(I_i)) + C(n)$$

for $n > smallSize$

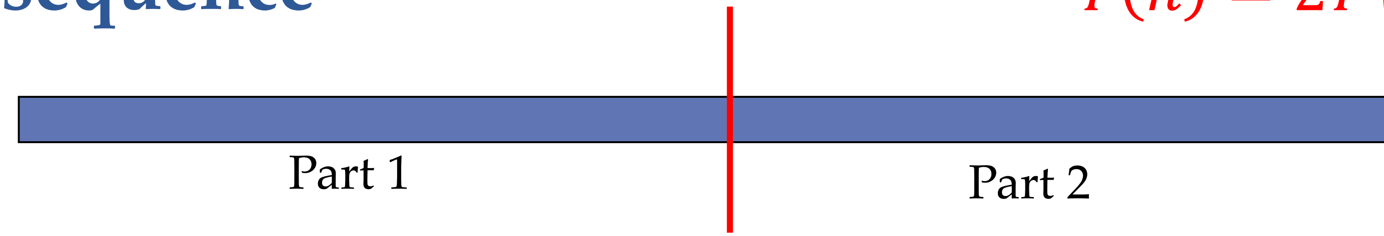
Divide Conquer

- **The BF recursion**
  - Problem size: often decreases linearly
    - "n, n-1, n-2, …"
- **The D&C recursion**
  - Problem size: often decrease exponentially
    - "n, n/2, n/4, n/8, …"

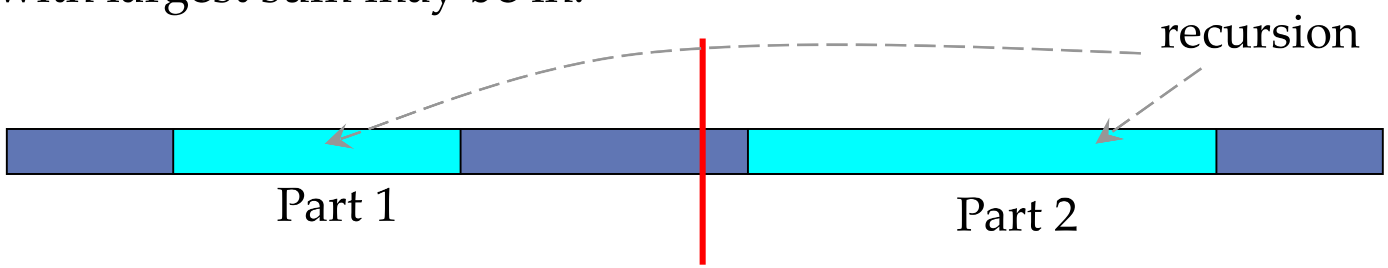
Examples

Max sum subsequence

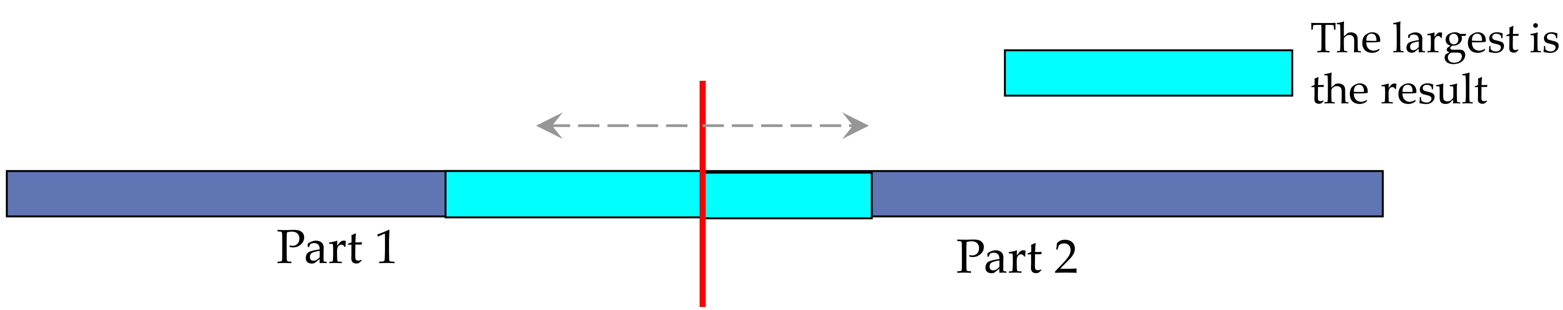
$$T(n) = 2T\left(\frac{n}{2}\right) + n$$



the sub with largest sum may be in:



or:



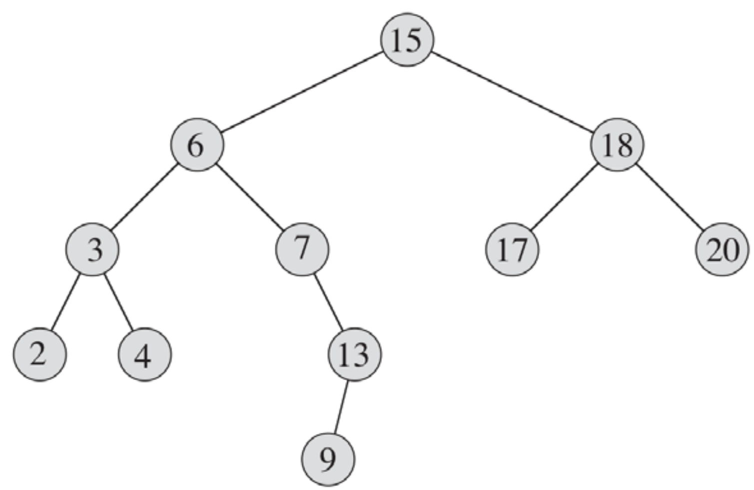
Examples

- **Maxima**
- **Frequent element**
- **Multiplication**
  - Integer
  - Matrix
- **Nearest point pair**

Examples

- **Arrays**

- **Trees**

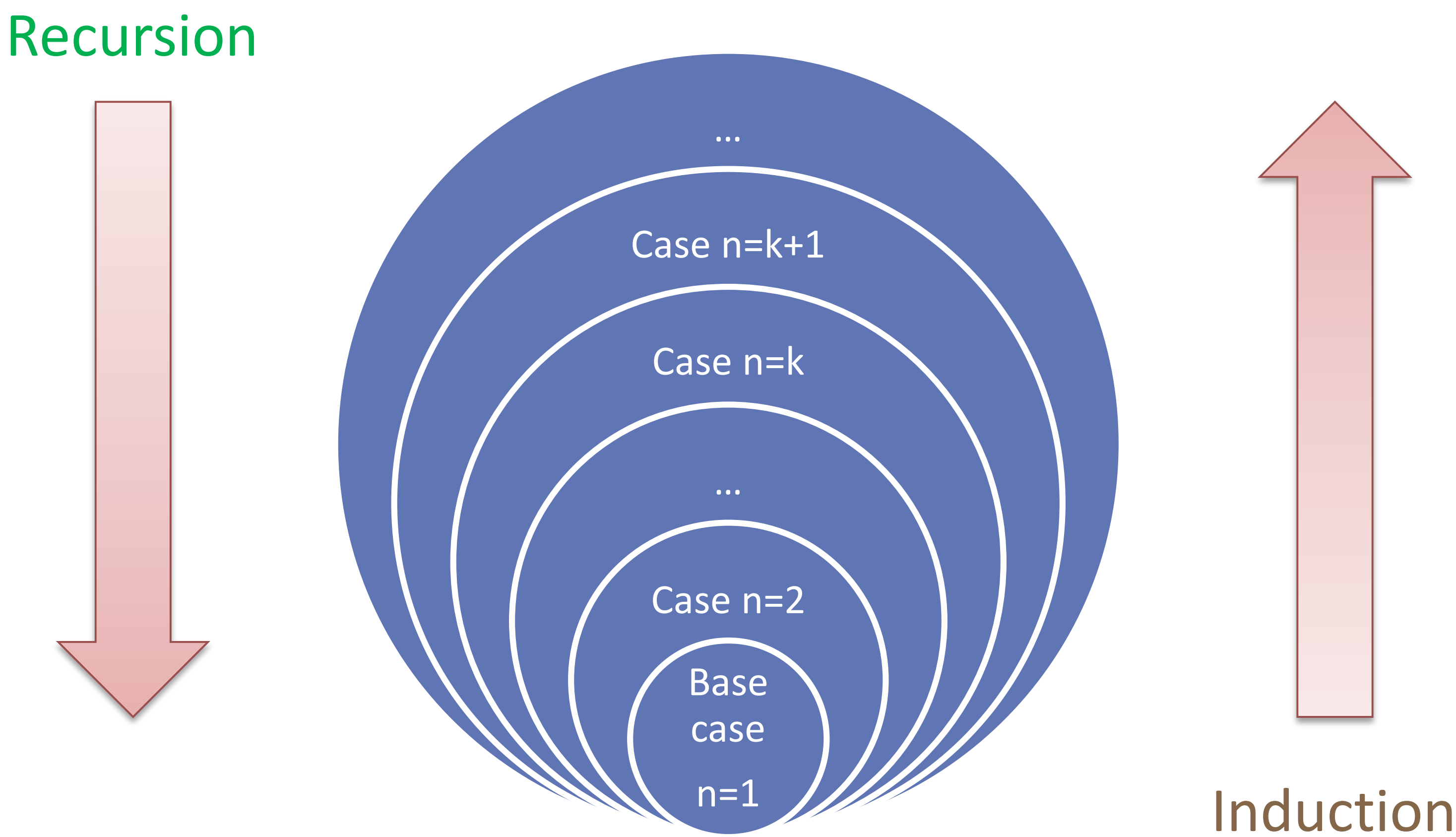


Workhorse

- “Hard division, easy combination”
- “Easy division, hard combination”

Usually, the “real work” is in one part.

Correctness of Recursion



Analysis of Recursion

- **Solving recurrence equations**

- **E.g., Bit counting**
  - Critical operation: add
  - The recurrence relation

$$T(n)=\begin{cases}0 & n=1\\ T(\lfloor n/2\rfloor)+1 & n>1\end{cases}$$

Analysis of Recursion

- **Backward substitutions**

By the recursion equation: $T(n) = T\left(\left\lfloor \frac{n}{2} \right\rfloor\right) + 1$

For simplicity, let $n = 2^k$ (k is a nonnegative integer),

that is, $k = \log n$

$$T(n) = T\left(\frac{n}{2}\right) + 1 = T\left(\frac{n}{4}\right) + 1 + 1 = T\left(\frac{n}{8}\right) + 1 + 1 + 1 = \ldots\ldots$$

$$T(n) = T\left(\frac{n}{2^k}\right) + \log n = \log n \qquad (\ T(1)=0\ )$$

Smooth Functions

- $f(n)$
  - Nonnegative eventually non-decreasing function defined on the set of natural numbers
- **$f(n)$ is called smooth**
  - If $f(2n) \in \Theta(f(n))$.

- **Examples of smooth functions**
  - $\log n$, n, $n\log n$ and n^{α} ($\alpha \geq 0$)
  - E.g., $2n\log 2n = 2n(\log n + \log 2) \in \Theta(n\log n)$

Even Smoother

- **Let $f(n)$ be a smooth function, then, for any fixed integer $b\geq 2$, $f(bn)\in\Theta(f(n))$.**
  - That is, there exist positive constants c_b and d_b and a nonnegative integer n_0 such that

$$d_b f(n) \leq f(bn) \leq c_b f(n) \quad \text{for } n\geq n_0.$$

It is easy to prove that the result holds for $b = 2^k$,
for the second inequality :
$f(2^k n) \leqslant c_2^k f(n)$ *for* $k = 1,2,3\ldots$ *and* $n \geqslant n_0$.
For an arbitrary integer $b \geqslant 2$, $2^{k-1} \leqslant b \leqslant 2^k$
Then, $f(bn) \leqslant f(2^k n) \leqslant c_2^k f(n)$, *we can use* c_2^k *as* c_b.

Smoothness Rule

- **Let $T(n)$ be an eventually non-decreasing function and $f(n)$ be a smooth function.**
  - If $T(n) \in \Theta(f(n))$ for values of n that are powers of $b (b \geq 2)$, then $T(n) \in \Theta(f(n))$.

Just proving the big-Oh part :

By the hypothsis : $T(b^k) \leqslant cf(b^k)$ *for* $b^k \geqslant n_0$.

By the prior result : $f(bn) \leqslant c_b f(n)$ *for* $n \geqslant n_0$.

Let $n_0 \leqslant b^k \leqslant n \leqslant b^{k+1}$,

$$T(n) \leqslant T(b^{k+1}) \leqslant cf(b^{k+1}) = cf(bb^k) \leqslant cc_b f(b^k) \leqslant cc_b f(n)$$

Non-decreasing hypothesis Prior result Non-decreasing

Computing the Fibonacci Number

$T(0)=0$

$T(1)=1$

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

0, 1, 1, 2, 3, 5, 8, 13, 21, 34, ...

$$a_n = r_1 a_{n-1} + r_2 a_{n-2} + \cdots + r_k a_{n-k}$$

is called linear homogeneous relation of degree k.

For the special case of Fibonacci: $a_n=a_{n-1}+a_{n-2}$, $r_1=r_2=1$

Computing the Fibonacci Number

$f_0=0$

$f_1=1$

$f_n=f_{n-1}+f_{n-2}$

0, 1, 1, 2, 3, 5, 8, 13, 21, 34, ...

$$a_n = r_1 a_{n-1} + r_2 a_{n-2} + \cdots + r_k a_{n-k}$$

is called linear homogeneous relation of degree *k*.

For the special case of Fibonacci: $a_n=a_{n-1}+a_{n-2}$, $r_1=r_2=1$

Characteristic Equation

- **For a linear homogeneous recurrence relation of degree *k***

$$a_n = r_1 a_{n-1} + r_2 a_{n-2} + \cdots + r_k a_{n-k}$$

the polynomial of degree *k*

$$x^k = r_1 x^{k-1} + r_2 x^{k-2} + \cdots + r_k$$

is called its characteristic equation.

- **The characteristic equation of linear homogeneous recurrence relation of degree 2 is:**

$$\boxed{x^2 - r_1 x - r_2 = 0}$$

Solution of Recurrence Relation

- If the characteristic equation $x^2 - r_1 x - r_2 = 0$ of the recurrence relation $a_n = r_1 a_{n-1} + r_2 a_{n-2}$ has two distinct roots s_1 and s_2, then

$$a_n = u s_1^n + v s_2^n$$

where u and v depend on the initial conditions, is the explicit formula for the sequence.

- If the equation has a single root s, then, both s_1 and s_2 in the formula above are replaced by s

Proof of the Solution

$$Remember \quad equation: \; x^2 - r_1 x - r_2 = 0$$

$$We\ need\ to\ prove\ that: us_1^n + vs_2^n = r_1 a_{n-1} + r_2 a_{n-2}$$

$$us_1^n + vs_2^n = us_1^{n-2} s_1^2 + vs_2^{n-2} s_2^2$$

$$= us_1^{n-2}(r_1 s_1 + r_2) + vs_2^{n-2}(r_1 s_2 + r_2)$$

$$= r_1 us_1^{n-1} + r_2 us_1^{n-2} + r_1 vs_2^{n-1} + r_2 vs_2^{n-2}$$

$$= r_1(us_1^{n-1} + vs_2^{n-1}) + r_2(us_1^{n-2} + vs_2^{n-2})$$

$$= r_1 a_{n-1} + r_2 a_{n-2}$$

Back to Fibonacci Sequence

$f_0=0$

$f_1=1$

$f_n=f_{n-1}+f_{n-2}$

$\Rightarrow$ **0, 1, 1, 2, 3, 5, 8, 13, 21, 34, ……**

Explicit formula for Fibonacci Sequence

The characteristic equation is x^2-x-1=0, which has roots:

$$s_1=\frac{1+\sqrt{5}}{2} \quad and \quad s_2=\frac{1-\sqrt{5}}{2}$$

Note: (by initial conditions) $f_1=us_1+vs_2=1 \ \ and \ \ f_2=us_1{}^2+vs_2{}^2=1$

which means:

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

Guess and Prove

- **Example: $T(n)=2T(\lfloor n/2 \rfloor) + n$**
- **Guess**
  - o $T(n) \in O(n)$?
    - $T(n) \le cn$, to be pr
  - o $T(n) \in O(n^2)$?
    - $T(n) \le cn^2$, to be prove
  - o **Or maybe**, $T(n) \in O(n\log$
    - $T(n) \le cn\log n$, to be prove
- **Prove**
  - o by substitution

Try to prove $T(n) \le cn$:

However:

$T($

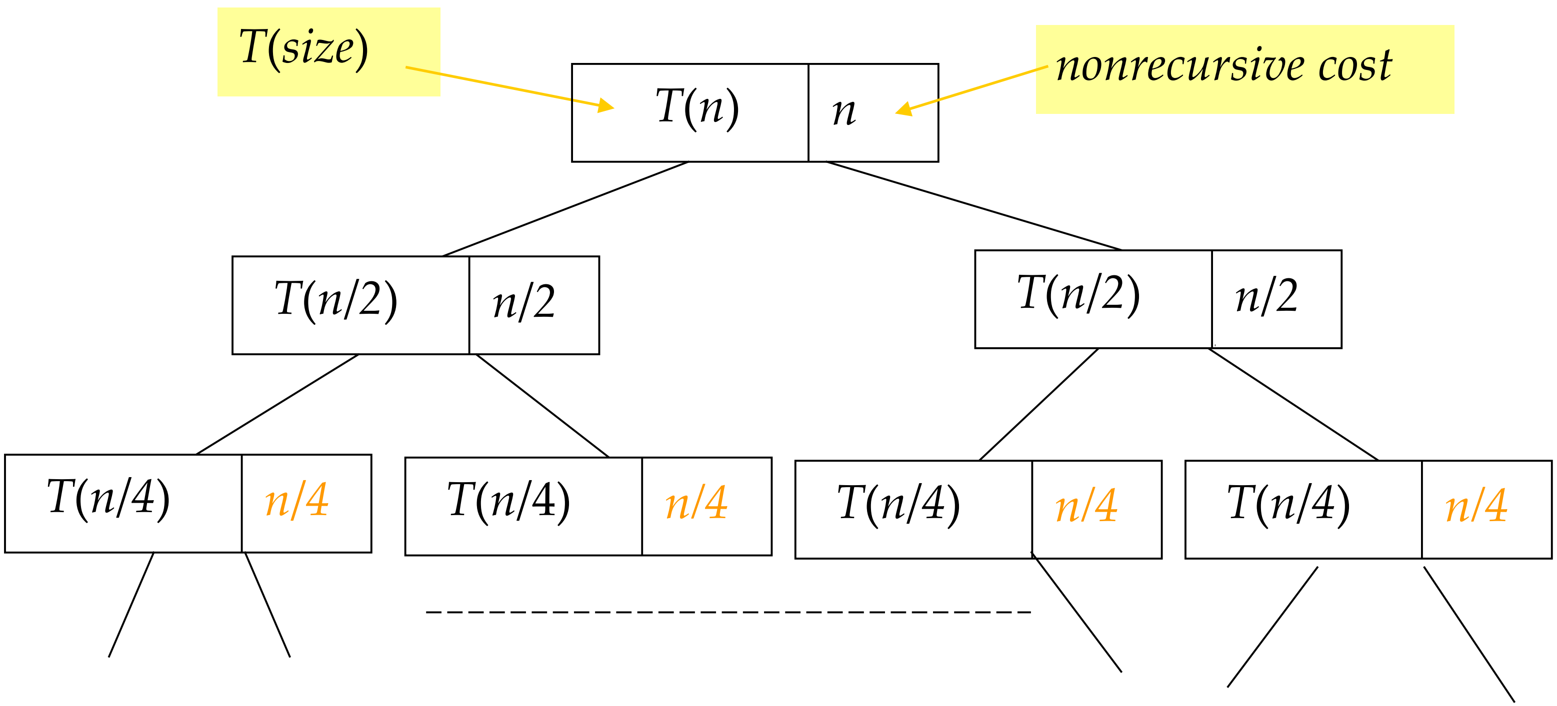
$$
\begin{aligned}
T(n) &= 2T(\lfloor n/2 \rfloor)+n \\
&\le 2(c\lfloor n/2 \rfloor \log(\lfloor n/2 \rfloor))+n \\
&\le cn\log(n/2)+n \\
&= cn\log n - cn\log 2 + n \\
&= cn\log n - cn + n \\
&\le cn\log n \quad \text{for } c \ge 1
\end{aligned}
$$

Divide and Conquer Recursions

- **Divide and conquer**
  - Divide the "big" problem to smaller ones
  - Solve the "small" problems by recursion
  - Combine results of small problems, and solve the original problem
- **Divide and conquer recursion**

$$T(n) = \underbrace{b}_{\text{divide}}\ \underbrace{T}_{\text{conquer}}(n/\underbrace{c}_{\text{divide}}) + \underbrace{f(n)}_{\text{combine}}$$

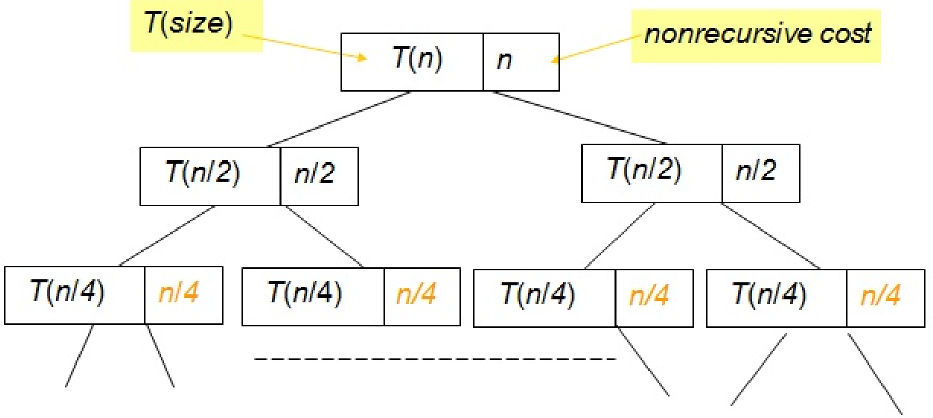
Recursion Tree



The recursion tree for $T(n) = 2T(n/2) + n$

Recursion Tree

- **Node**
  - Non-leaf
    - Non-recursive cost
    - Recursive cost
  - Leaf
    - Base case
- **Edge**
  - Recursion



The recursion tree for T(n)=T(n/2)+T(n/2)+n

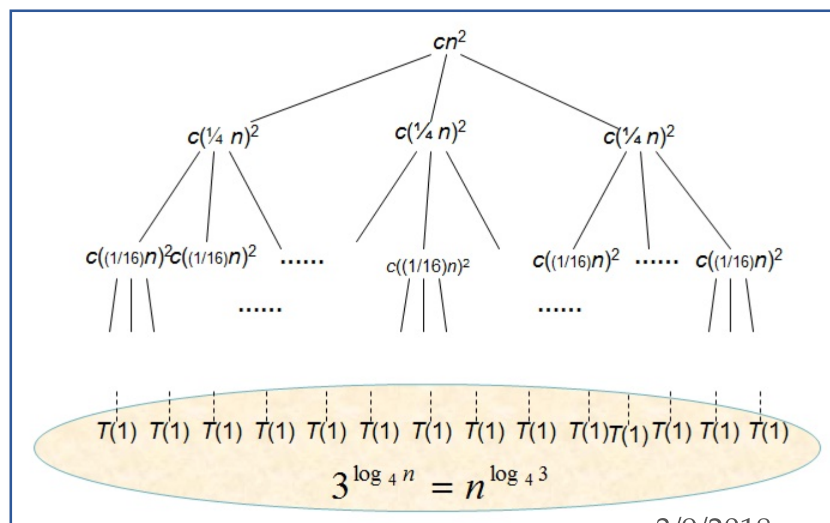
Recursion Tree

- $T(n) = \overbrace{\underbrace{3}_{\text{\# of sub-problems}}T(n/\underbrace{4}_{\text{size of sub-problems}})}^{\text{Recursive cost}} + \overbrace{\Theta\,(n^2)}^{\text{Non-recursive cost}}$

- **Total cost**

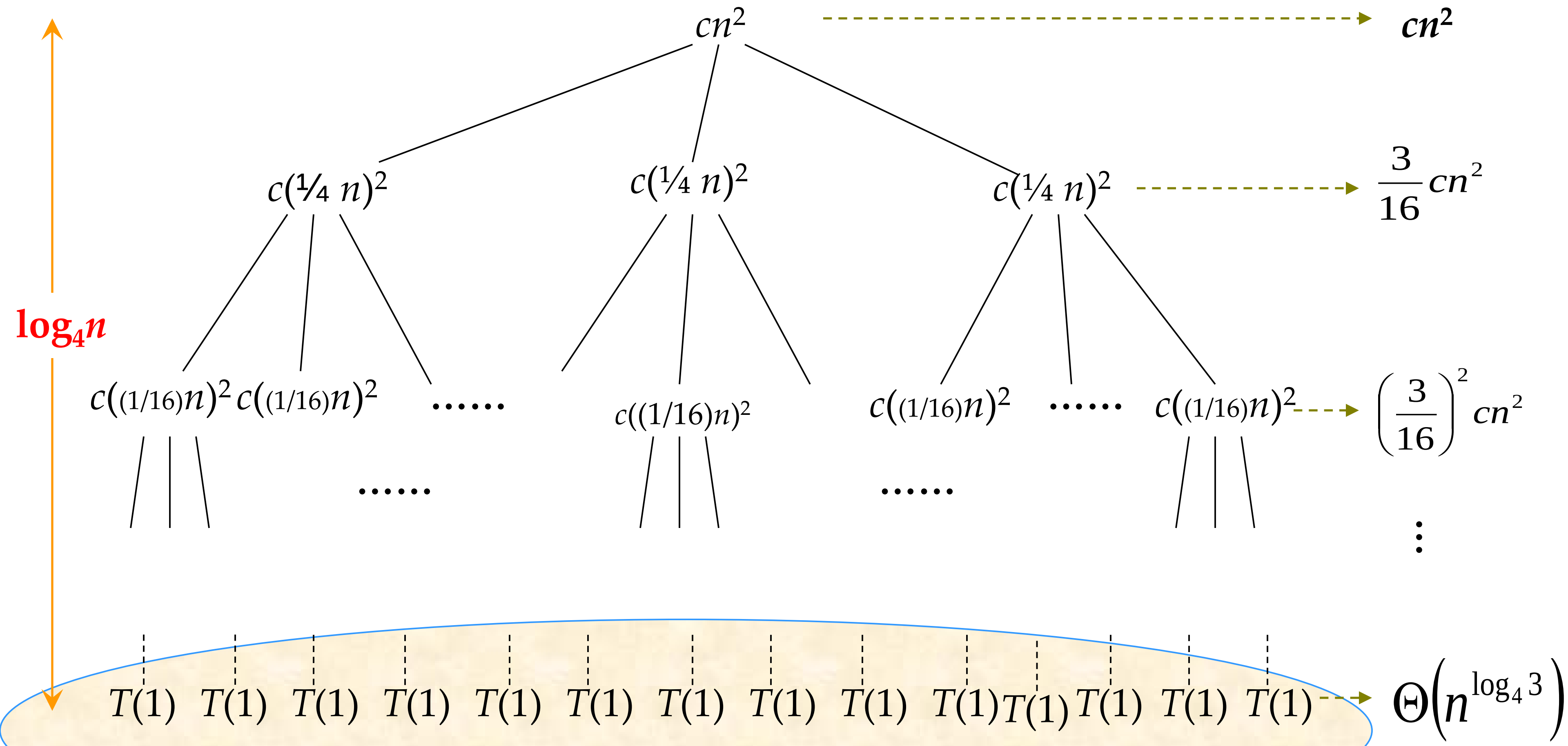
Σ

Sum of row sums



Sum of Row-sums

$T(n) = 3T(\lfloor n/4 \rfloor) + \Theta(n^2)$



Note: $3^{\log_4 n} = n^{\log_4 3}$

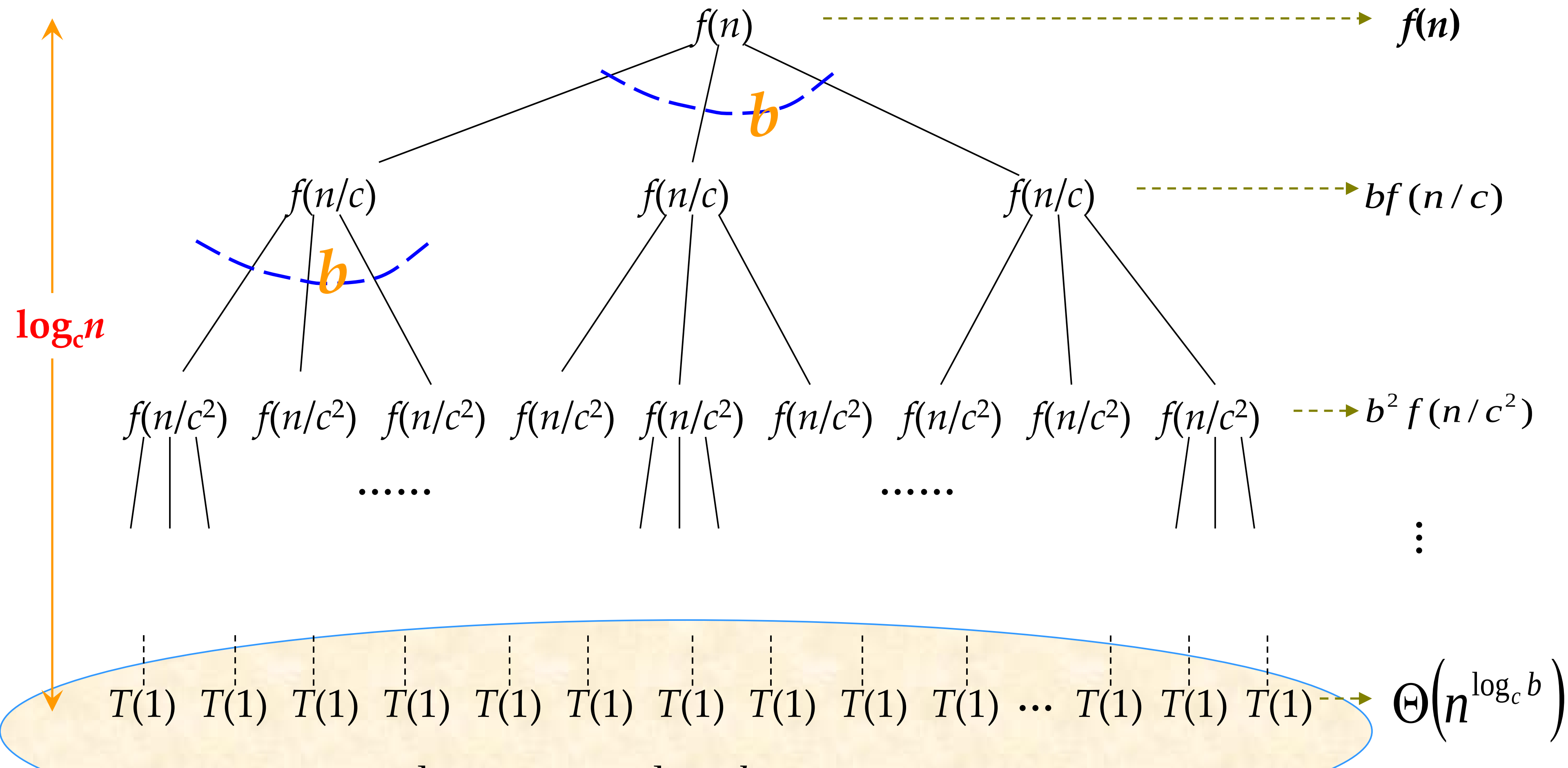
Total: $\Theta(n^2)$

Solving the Divide-and-Conquer Recurrence

- **The recursion equation for divide-and-conquer, the general case: $T(n)=bT(n/c)+f(n)$**
- **Observations:**
    - Let base-cases occur at depth D(leaf), then $n/c^D=1$, that is $D=\log(n)/\log(c)$
    - Let the number of leaves of the tree be L, then $L=b^D$, that is $L=b^{(\log(n)/\log(c))}$.
    - By a little algebra: $L=n^E$, where $E=\log(b)/\log(c)$, called *critical exponent*.

Recursion Tree for

$T(n)=bT(n/c)+f(n)$



Note: $b^{\log_c n} = n^{\log_c b}$

Total ?

Divide-and-Conquer - the Solution

- **The solution of divide-and-conquer equation is the non-recursive costs of all nodes in the tree, which is the sum of the row-sums**
  - The recursion tree has depth D=log(n)/ log(c), so there are about that many row-sums.

- **The 0^{th} row-sum**
  - is $f(n)$, the nonrecursive cost of the root.
- **The D^{th} row-sum**
  - is n^E, assuming base cases cost 1, or $\Theta(n^E)$ in any event.

Solution by Row-sums

- **[Little Master Theorem] Row-sums decide the solution of the equation for divide-and-conquer:**
  - Increasing geometric series: $T(n) \in \Theta(n^E)$
  - Constant: $T(n) \in \Theta(f(n) \log n)$
  - Decreasing geometric series: $T(n) \in \Theta(f(n))$

This can be generalized to get a result not using explicitly row-sums.

Master Theorem

- **Loosening the restrictions on $f(n)$**
  - Case 1: $f(n) \in O(n^{E-\varepsilon})$, ($\varepsilon>0$), then:

    $$T(n) \in \Theta(n^E)$$

  - Case 2: $f(n) \in \Theta(n^E)$, as all node depth contribute about equally:

    $$T(n) \in \Theta(f(n)\log(n))$$

  - case 3: $f(n) \in \Omega(n^{E+\varepsilon})$, ($\varepsilon>0$), and if $bf(n/c) \leq \theta f(n)$ for some constant $\theta < 1$ and all sufficiently large n, then:

    $$T(n) \in \Theta(f(n))$$

The positive ε is critical, resulting gaps between cases as well

Using Master Theorem

- Example 1: $T(n) = 9T(\frac{n}{3}) + n$
  $b = 9, c = 3, E = 2, f(n) = n = O(n^{E-1})$
  Case 1 applies: $T(n) = \Theta(n^2)$
- Example 2: $T(n) = T(\frac{2}{3}n) + 1$
  $b = 1, c = \frac{3}{2}, E = 0, f(n) = 1 = \Theta(n^E)$
  Case 2 applies: $T(n) = \Theta(\log n)$
- Example 3: $T(n) = 3T(\frac{n}{4}) + n \log n$
  $b = 3, c = 4, E = \log_4 3, f(n) = \Omega(n^{E+\epsilon})$
  $bf(\frac{n}{4}) = \frac{3}{4}n \log n - \frac{3}{2}n$
  Case 3 applies: $T(n) = \Theta(n \log n)$

Using Master Theorem

- **T(n) = 2T(n/2) + nlogn**
  - Does Case 3 apply? Why?

- **T(n)=$\sqrt{n}$ T($\sqrt{n}$) + n**

- **The gap between the 3 cases**
  - Often, none of the 3 cases apply
  - Your task: design more non-solvable recursions

Thank you!

Q & A

Yu Huang
yuhuang@nju.edu.cn
http://cs.nju.edu.cn/yuhuang