Recurrence & Divide-and-Conquer

Prof. Zheng Zhang

Harbin Institute of Technology, Shenzhen

2024/8/27

OUTLINE

- Sort Example and Asymptotic Analysis

- Recurrence and Divide-and-Conquer
  - Three recurrence solving methods
    - Substitution method
    - Recursion-tree method
    - Master method
  - Divide-and-Conquer example
    - Big Integer Multiplication
    - Strassen Matrix Multiplication
    - Chessboard Cover
    - Order Statistic

SORT EXAMPLE AND ASYMPTOTIC ANALYSIS

Sorting Problem

Description:

Input: sequence of n numbers $< a_1, a_2, \ldots, a_n >$.

Output: A permutation $< a'_1, a'_2, \ldots, a'_n >$ such that $a'_1 \leq a'_2 \leq \ldots \leq a'_n$.

Example：

Input: **8 2 4 9 3 6**

Output: **2 3 4 6 8 9**

INSERTION SORT

Pseudocode

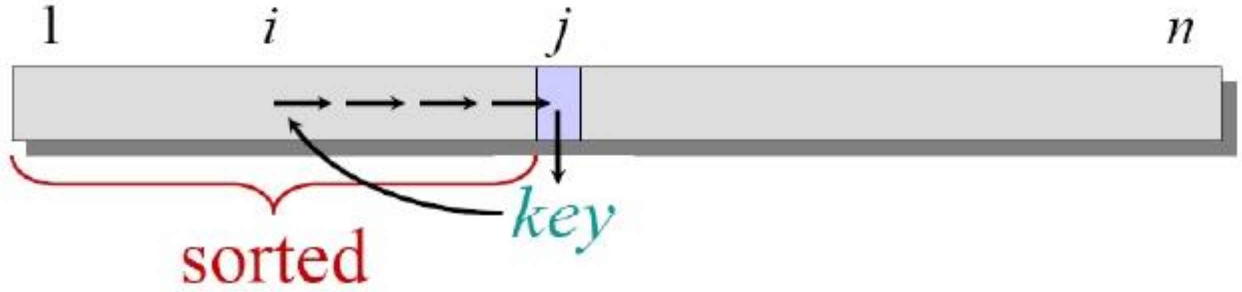
InsertionSort (A, n)

for $j \leftarrow 2$ **to** n

do

Insert $A[j]$ into the sorted sequence $A[1 .. j - 1]$.

INSERTION SORT



Pseudocode

InsertionSort (*A*, *n*)

for *j* ←2 to *n*
do
key ← *A*[*j*]
//Insert *A*[*j*] into the sorted sequence *A*[1 .. *j* - 1].
***i* ← *j* - 1**
while *i* > 0 and *A*[*i*] > key
do
A[*i* + 1] ←A[*i*]
***i* ← *i* -1**
***A*[*i* +1] ←key**

INSERTION SORT

Example:

8 2 4 9 3 6

INSERTION SORT

Example:

8 2 4 9 3 6

Insertion Sort

Example:

8 2 4 9 3 6

2 8 4 9 3 6

Insertion Sort

Example:

8	2	4	9	3	6
2	8	4	9	3	6

INSERTION SORT

Example:

8	2	4	9	3	6
2	8	4	9	3	6
2	4	8	9	3	6

Insertion Sort

Example:

8	2	4	9	3	6
2	8	4	9	3	6
2	4	8	9	3	6
2	4	8	9	3	6

Insertion Sort

Example:

8 2 4 9 3 6

2 8 4 9 3 6

2 4 8 9 3 6

2 4 8 9 3 6

INSERTION SORT

Example:

8	2	4	9	3	6
2	8	4	9	3	6
2	4	8	9	3	6
2	4	8	9	3	6
2	3	4	8	9	6

INSERTION SORT

Example:

8	2	4	9	3	6
2	8	4	9	3	6
2	4	8	9	3	6
2	4	8	9	3	6
2	3	4	8	9	6
2	3	4	6	8	9

Insertion Sort

Example:

8	2	4	9	3	6
2	8	4	9	3	6
2	4	8	9	3	6
2	4	8	9	3	6
2	3	4	8	9	6
2	3	4	6	8	9

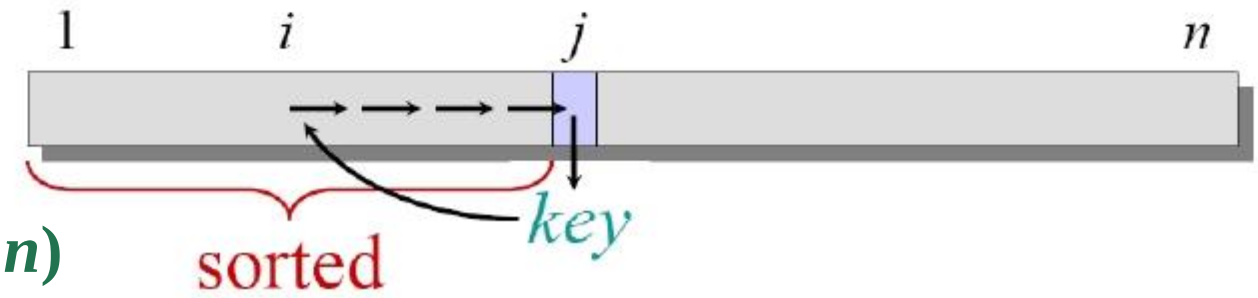
CORRECTNESS OF An ALGORITHM

For such an incremental algorithm, we can use **loop invariants** to prove the correctness of the algorithm.

Loop invariants:

- **Initialization**
  - It is true at the first loop
- **Maintenance**
  - It is true before an iteration of loop, then true before next iteration
- **Termination**
  - The invariant guarantee the correctness at last iteration.

INSERTION SORT



Pseudocode

InsertionSort (*A*, *n*)

Initialization

for *j* ←2 to *n*

do

Maintenance

key ← *A*[*j*]

//Insert *A*[*j*] into the sorted sequence *A*[1 .. *j* - 1].

***i* ← *j*- 1**

while *i* > 0 and *A*[*i*] > key

Termination

do

A[*i* + 1] ←A[*i*]

***i* ← *i* -1**

***A*[*i* +1] ←key**

Running Time

The running time depends on the input.

An already sorted sequence is easier to sort.

Parameterize the running time by the size of the input---***n***, since short sequences are easier to be sorted than the longer ones.

Generally, we seek upper bounds on the running time, because everybody likes a guarantee---**worst case**.

KINDS OF ANALYSES

Worst-case: (usually)

*T(n) = **maximum** time of algorithm on any input of size **n**.*

Average-case: (sometimes)

*T(n) = **expected** time of algorithm over all inputs of size **n**.*

Need assumption of statistical distribution of inputs.

Best-case: (bogus)

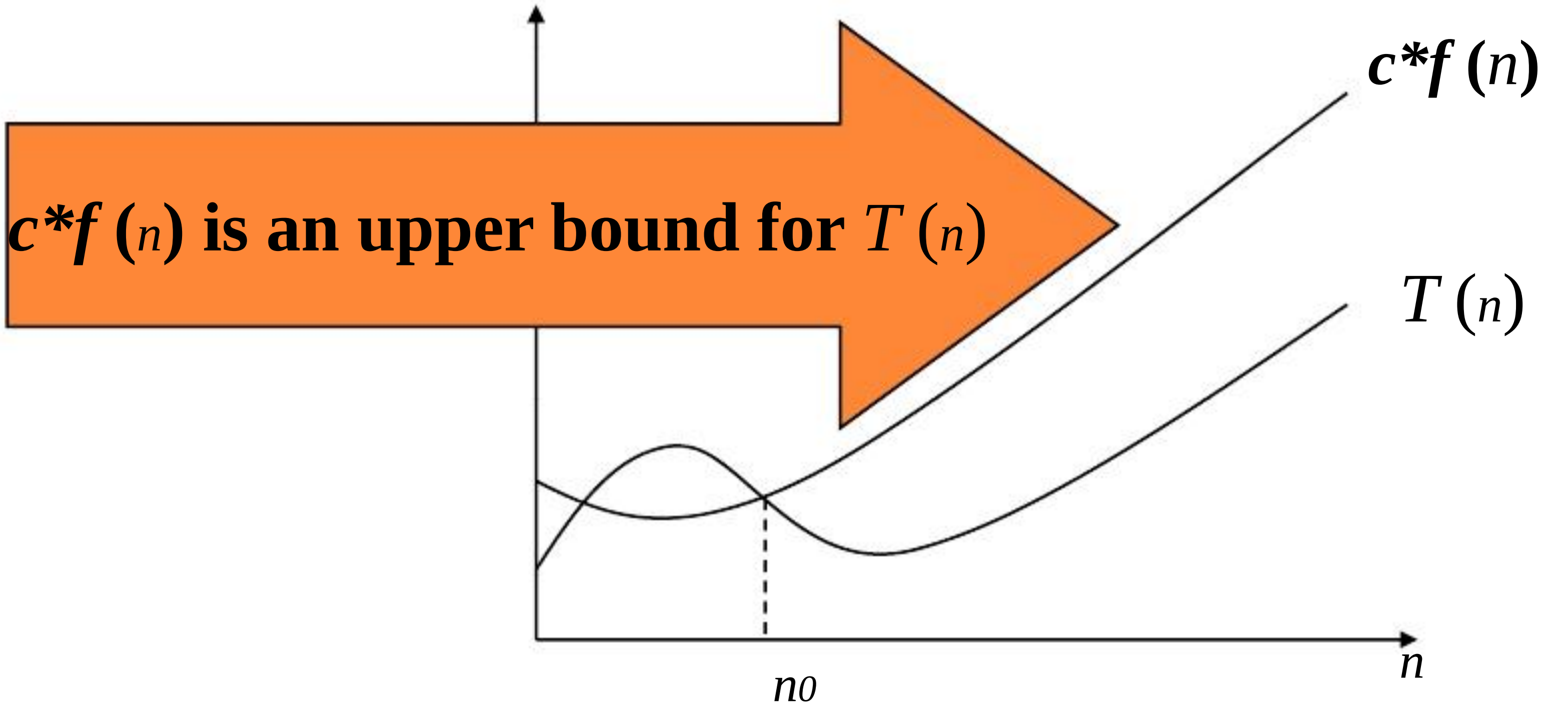
***Cheat** with a slow algorithm that works fast on some input.*

Asymptotic Time Analysis

Big *O* time complexity

$T(n) = O(f(n))$ if there exist positive constant c and n_0 such that $T(n) \le c\,f(n)$ when $n \ge n_0$.

A kind of asymptotic upper bound

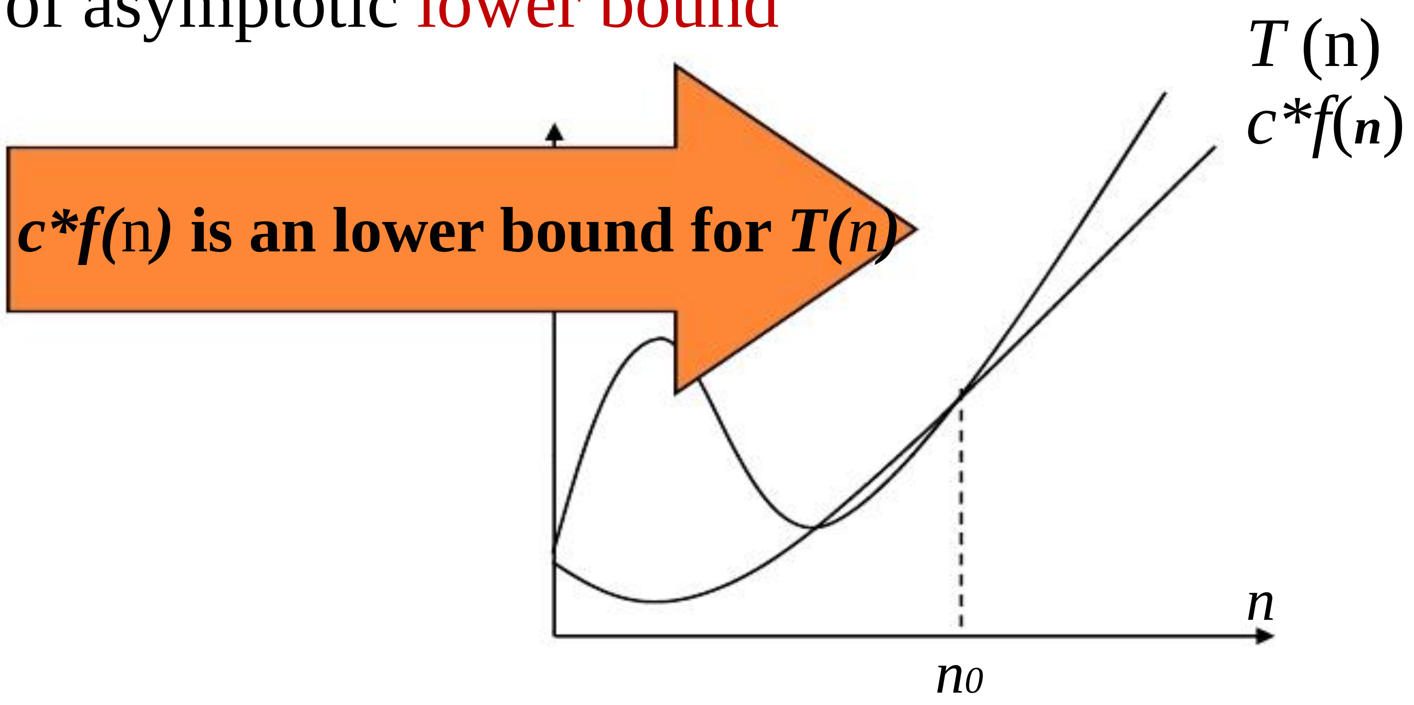


ASYMPTOTIC TIME ANALYSIS

Big Ω time complexity

$T(n) = \Omega(f(n))$ if there exist positive constant c and n_0 such that $T(n) \geq c\,f(n)$ when $n \geq n_0$.

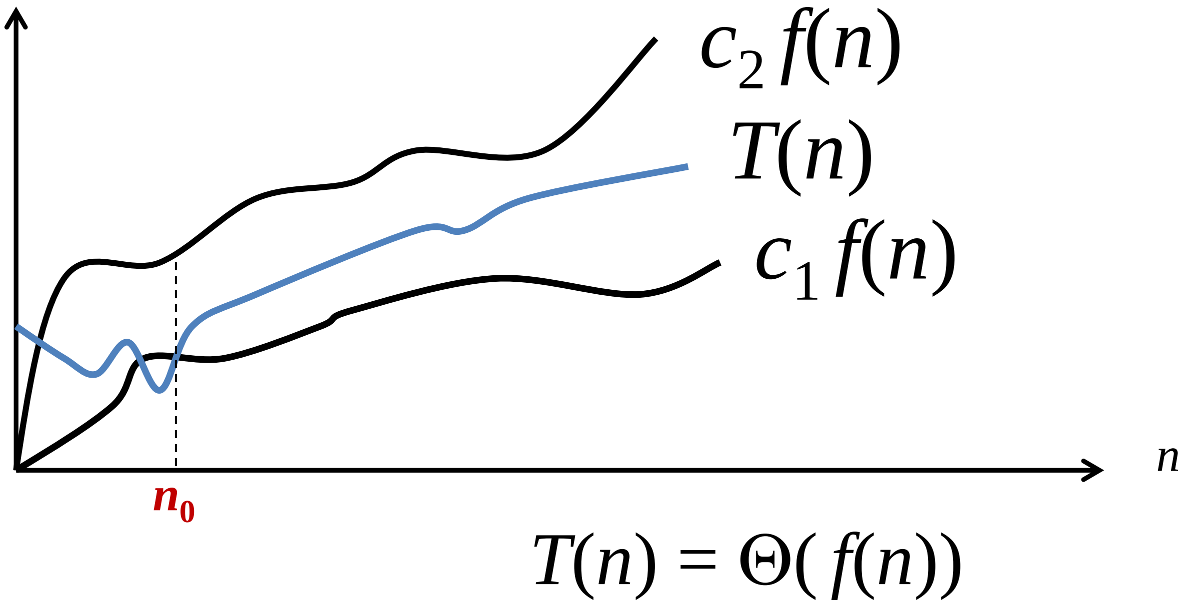
A kind of asymptotic lower bound



Asymptotic Time Analysis

Big Θ time complexity

$T(n) = \Theta(f(n))$ if there exist positive constant c_1, c_2, and n_0 such that $0 \le c_1 f(n) \le T(n) \le c_2 f(n)$ when $n \ge n_0$.



Asymptotic Time Analysis

$T(n) = \Theta(f(n))$ if and only if $T(n) = O(f(n))$ and $T(n) = \Omega(f(n))$

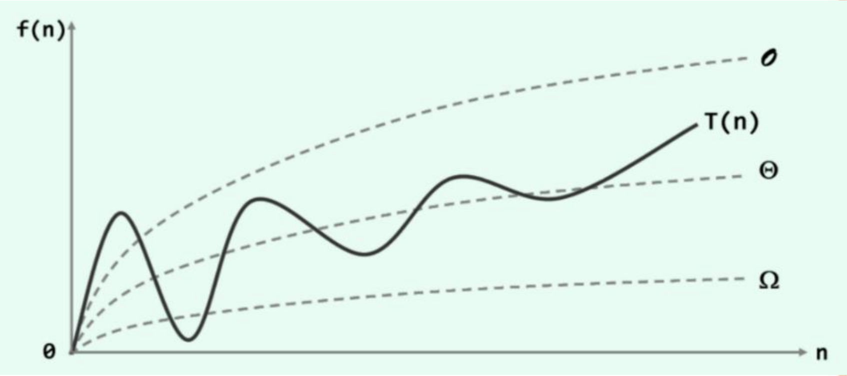
$T(n) = o(f(n))$ if $T(n) = O(f(n))$ and $T(n) \neq \Theta(f(n))$

We can determine the relative growth rates of $f(n)$ and $g(n)$ by computing $\lim_{n\text{->infinite}} f(n) / g(n)$

0: $f(n) = o(g(n)))$

A constant c: $f(n) = \Theta(g(n))$

Infinite: $g(n) = o(f(n))$

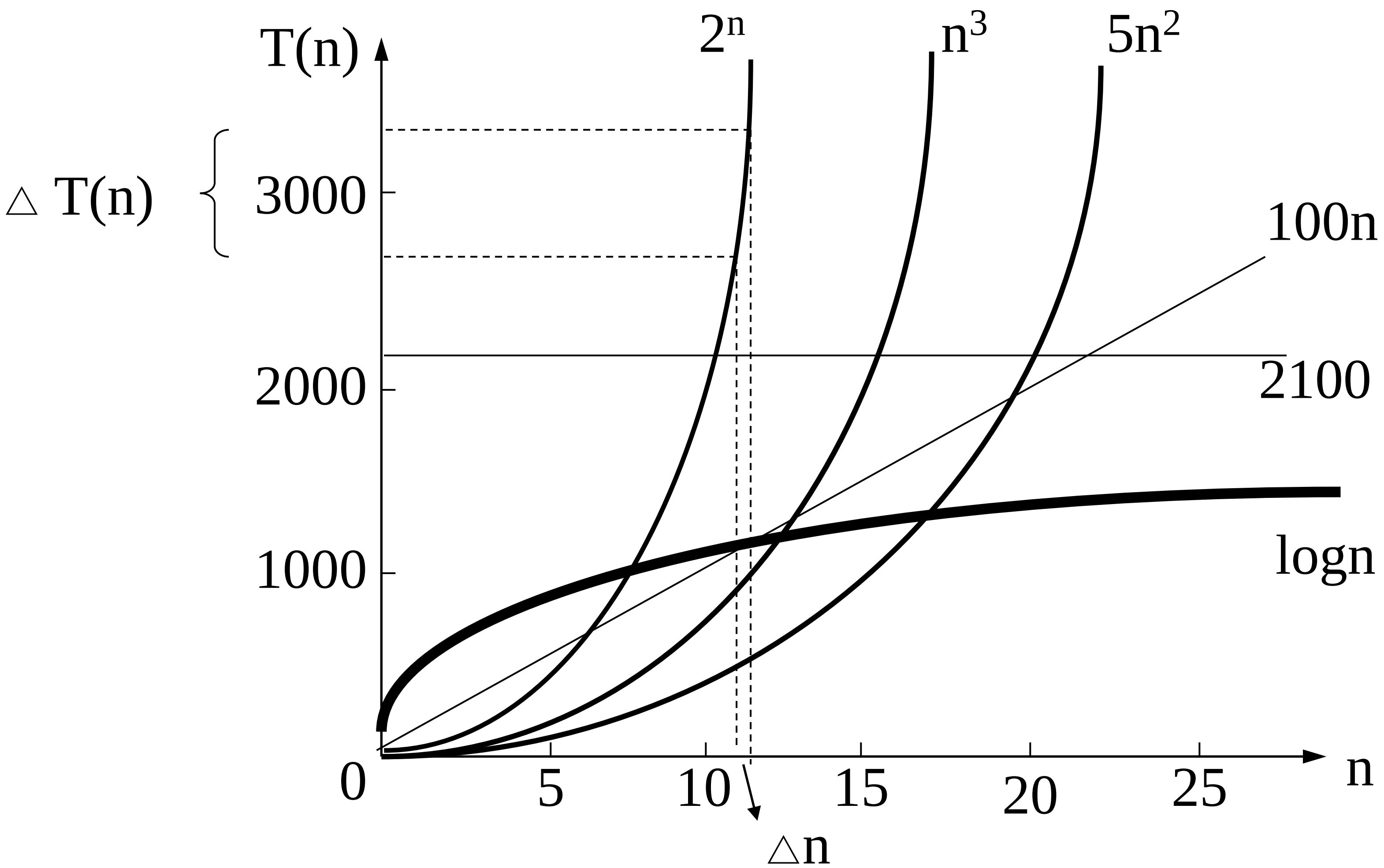


If $T_1(n) = \Theta(f(n))$ and $T_2(n) = \Theta(g(n))$

$T_1(n) + T_2(n) = \max(\Theta(f(n)), \Theta(g(n)))$

$T_1(n) * T_2(n) = \Theta(f(n)) * \Theta(g(n))$

PRACTICAL EXAMPLE



Time Complexity Comparison T (n) = O(f(n))

$O(1) < O(\log_2 n) < O(n) < O(n\log_2 n) < O(n^2) < O(n^3) < O(2^n) < O(n!) < O(n^n)$

PRACTICAL EXAMPLE

①s = 0 ;

→ $f(n) = 1$; $T_1(n) = O(f(n)) = O(1)$

②for (i=1 ; i <= n ; ++i) { ++x; s += x; }

→ $f(n) = 3n+1$; $T_2(n) = O(f(n)) = O(n)$

③for (i=1; i<=n ; ++i)

for(j=1 ; j <=n ; ++j) { ++x ; s += x; }

→ $f(n) = 3n^2+2n+1$; $T_3(n) = O(f(n)) = O(n^2)$

④for (i=1; i<=n ; ++i)

for (j=1 ; j <=n ; ++j)

{ c[i][j] = 0;

for (k=1 ; k <= n; ++k)

c[i][j] += a[i][k] * b[k][j] ; }

→ $f(n) = 2n^3+3n^2+2n+1$; $T_4(n) = O(f(n)) = O(n^3)$

PRACTICAL EXAMPLE

```
Void  BUBBLE(A)
int A[n];
{   int I,j,temp;
    for(i=0;i<n-1;i++)
        for(j=n-1;j>=i+1;j--)
            if(A[j-1]>A[j]) {
                temp=A[j-1];      O(1)
                A[j-1]=A[j];      O(1)   O(1)   O(1)   O((n-i-1) × 1)   O(∑_{i=0}^{n-2}(n-i-1))
                A[j]=temp;        O(1)                 =(n-i-1)         ≤O(n(n-1)/2)
            }                                                           =O(n^2)
}
```

$$O(1)+O(1)+O(1) = O(1), \quad O((n-i-1)\times 1) = (n-i-1), \quad O\left(\sum_{i=0}^{n-2}(n-i-1)\right) \le O(n(n-1)/2) = O(n^2)$$

PRACTICAL EXAMPLE

⑤

```
Long  fact ( int  n)
{   if ( n==0 ) || ( n ==1 )
        return( 1 );
    else
        return( n * fact( n – 1 ) );
}
```

for(p=1,i=2; i<=n; p=p*i++) ;
T(n)=O(n).

$$f(n) = \begin{cases} C & \text{当 } n=0,\ n=1 \\ G + f(n-1) & \text{当 } n > 1 \end{cases}$$

$$\begin{aligned} f(n) &= G_1 + f(n-1) \\ f(n-1) &= G_2 + f(n-2) \\ f(n-2) &= G_3 + f(n-3) \\ &\cdots\ \cdots \\ f(2) &= G_{n-1} + f(1) \\ +\quad f(1) &= C \\ \hline f(n) &= G_1 + G_2 + G_3 + \ldots\ldots + G_{n-1} + C \end{aligned}$$

$\Longrightarrow$

$$f(n) = n\,G'$$

$$\therefore T(n) = O(f(n)) = O(n)$$

INSERTION SORT ANALYSIS

Cost and times

	cost	*times*
InsertionSort **(*A*, *n*)**		
for $j \leftarrow 2$ **to** n	c_1	n
do		
key $\leftarrow A[j]$	c_2	n-1
//Insert $A[j]$ into $A[1 .. j - 1]$.		
$i \leftarrow j - 1$	c_3	n-1
while $i > 0$ **and** $A[i] >$ key	c_4	$\sum_{j=2}^{n} t_j$
do		
$A[i+1] \leftarrow A[i]$	c_5	$\sum_{j=2}^{n} (t_j - 1)$
$i \leftarrow i - 1$	c_6	
$A[i+1] \leftarrow$ key	c_7	n-1

Insertion Sort Analysis

- **Worst Case: decreasing order**
  - $t_j = j$ $\quad\quad \sum_{j=2}^{n} t_j = \frac{n(n+1)}{2} - 1 \quad\quad \sum_{j=2}^{n} (t_j - 1) = \frac{n(n-1)}{2}$

$$T(n) = c_1 n + c_2(n-1) + c_3(n-1) + c_4\left(\frac{n(n+1)}{2} - 1\right)$$

$$+ c_5\left(\frac{n(n-1)}{2}\right) + c_6\left(\frac{n(n-1)}{2}\right) + c_7(n-1)$$

$$= \left(\frac{c_4}{2} + \frac{c_5}{2} + \frac{c_6}{2}\right)n^2 + \left(c_1 + c_2 + c_3 + \frac{c_4}{2} - \frac{c_5}{2} - \frac{c_6}{2} + c_7\right)n - (c_2 + c_3 + c_4 + c_7)$$

$$T(n) = an^2 + bn + c \quad \Rightarrow \quad T(n) = \Theta(n^2)$$

Insertion Sort Analysis

Worst Case: decreasing order

InsertionSort ***(A, n)***

for ***j*** **← 2 to** ***n***

do

Insert ***A*[*j*] into the sorted sequence *A*[1 .. *j* - 1].**

$$T(n) = \sum_{j=2}^{n} \Theta(j) = \Theta(n^2)$$

How about the Best Case?

More EXAMPLE

循环主体中的变量参与循环条件的判断

Computing t times

Compute the complexity of the following algs.

void func(){int i=0; while(i*i*i<=n) i++;} ---------> t*t*t<=n $t<=n^{1/3}$

A.O($\log_2 n$) B.O($n^{1/2}$) C.O($n^{1/3}$) D.O($n\log_2 n$)

---

void func(){int i=1; while(i<=n) i=i*2;} ---------> $2^{t+1} <= n/2$

$=> t<=\log_2 n-2$

A.O($\log_2 n$) B.O($n^{1/2}$) C.O($n^{1/3}$) D.O($n\log_2 n$)

void func(){int j=5; while((j+1)*(j+1)<n) j=j+1;} ---------> $(t+5+1)^2< n$

$=> t<n^{1/2}-6$

A.O($\log_2 n$) B.O($n^{1/2}$) C.O(n) D.O($n\log_2 n$)

```
int i=0; k=0;
while(i<n-1)
    k=k+10*i;
    i++;
```

A.O(logn) B.O(n)

C.O($n^{1/2}$) D.O(n^2)

```
int i=0; k=0;
while(k<n-1)
    k=k+10*i;
    i++;
```

$$\Rightarrow \sum_{i=1}^{t-1} 10i = 10\sum_{i=1}^{t-1} i < n-1$$

More EXAMPLE

循环主体中的变量与循环条件是无关的

```
int fact(int n){
if(n<=1) return 1;
return n=n*fact(n-1);}
```

n*(n-1)*…*1

```
for(i=1；i<=n;i++)
  for(j=1；j<=i;j++)
    for(k=1；k<=j;k++)
      x++;
```

sum i=1→n;

sum j=1→i;

sum k=1→j;

1; 1+2; 1+2+3;…

O(1/6 n*(n+1)*(n+2))

```
for(i=n-1;i>1;i--)
 for(j=1;j<i;j++)
 if(A[j]>A[j+1]) A[j]与A[j+1]互换；
```

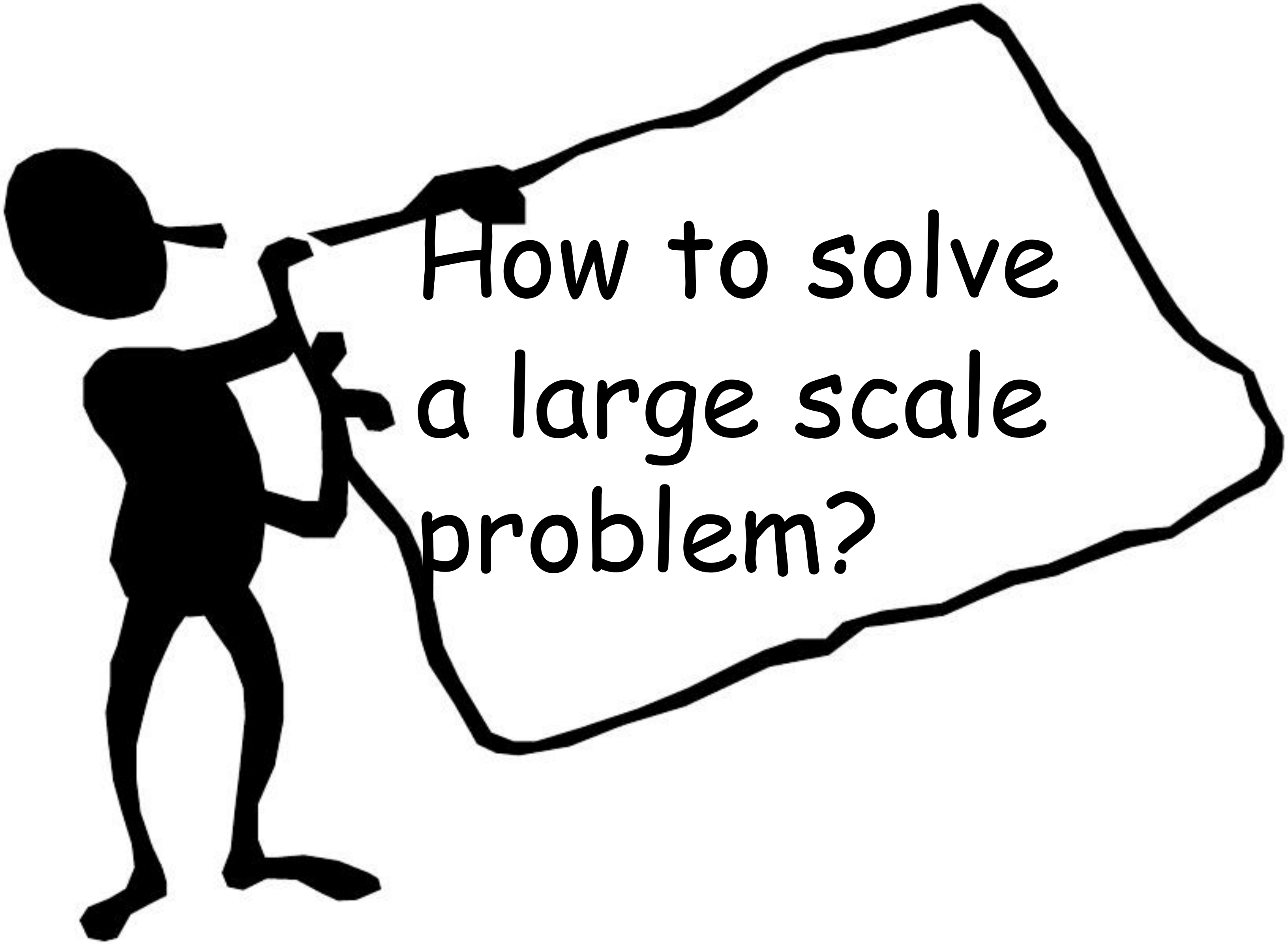
$$\sum_{i=2}^{n-1}\sum_{j=1}^{i-1} 1 = \sum_{i=2}^{n-1}(i-1) = (n-2)(n-1)/2$$

```
in m=0,i,j;
for(i=1;i<=n;i++)
 for(j=1;j<=2*i;j++) m++;
```

$$\sum_{i=1}^{n}\sum_{j=1}^{2i} 1 = \sum_{i=1}^{n} 2i = 2\sum_{i=1}^{n} i = n(n+1)$$

A.O(n^3) B.O(n) C.O($\log_2 n$) D.O(n^2)

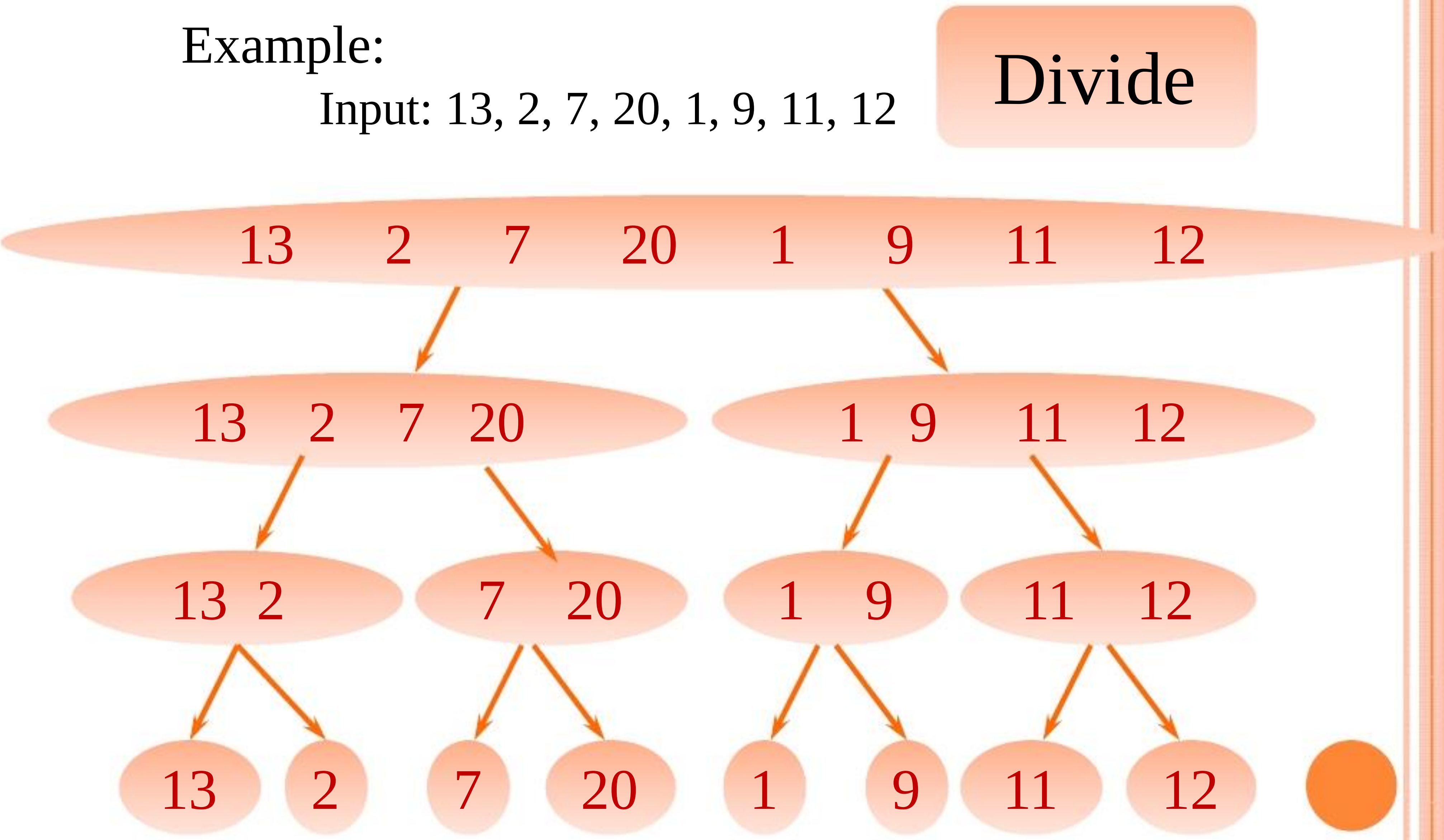
RECURRENCE & DIVIDE-AND-CONQUER


How to solve
a large scale
problem?

MERGE SORT

Example:

Input: 13, 2, 7, 20, 1, 9, 11, 12

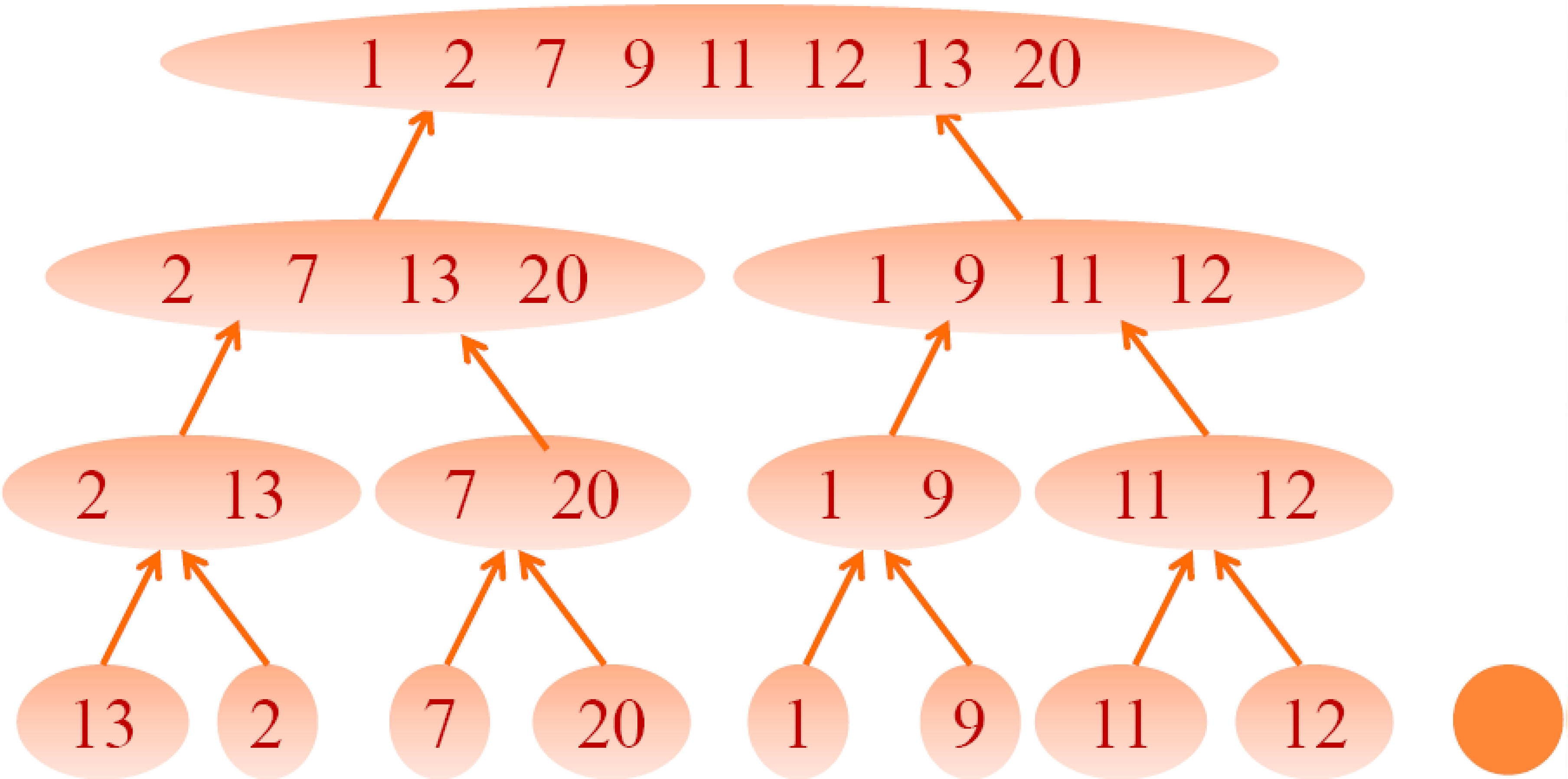


MERGE SORT

Example:

Merge

Input: 13, 2, 7, 20, 1, 9, 11, 12



MERGE SORT

Merge

{26} {5} {77} {1} {61} {11} {59} {15} {48} {19}

{ 5 26 } { 1 77 } { 11 61 } { 15 59 } { 19 48 }

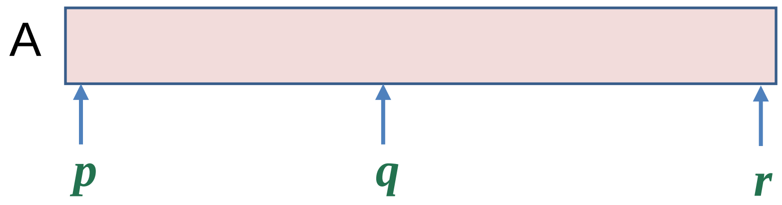
{ 1 5 26 77 } { 11 15 59 61 } { 19 48 }

{ 1 5 11 15 26 59 61 77 } { 19 48 }

{ 1 5 11 15 19 26 48 59 61 77 }

MERGE SORT

Merge Sort Algorithm



MergeSort(***A*, *p*, *r***)

if ***p*** < ***r***

then

$$q \leftarrow \lfloor (p+r)/2 \rfloor$$

MergeSort (***A*, *p*, *q***)

MergeSort (***A*, *q* + 1, *r***)

Merge (***A*, *p*, *q*, *r***)

p ≥ r? How many entries?

DIVIDE-AND-CONQUER

- *Recursive problems*
  - Call themselves recursively one or more times to deal with closely related subproblems.
- Divide and Conquer
  - Break the problem into several subproblems
    - Subproblems are similar to the original problem but smaller in size
  - Conquer the subproblems by solving subproblems recursively
  - Then combine these solutions to create a solution to the original problem.

MERGE SORT

Divide

Trivial.

Conquer

Recursively sort 2 subarrays.

$2T(n/2)$

Combine

$O(n)$

Merge Sort

MergeSort (***A, p, r***)

// find p

if ***p*** < ***r***

then

$\mathbf{q \leftarrow \lfloor (p + r)/2 \rfloor}$

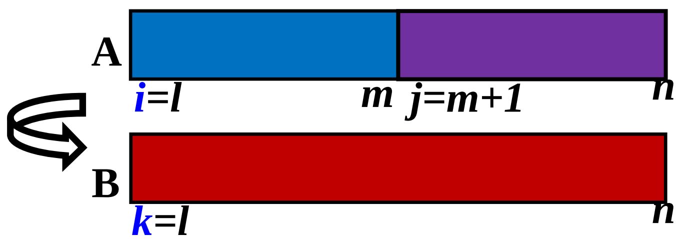
MergeSort (***A, p, q***)

MergeSort (***A, q*+1, *r***)

Merge (***A, p, q, r***)

MERGE SORT

```
Void   Merge ( ℓ, m, n, A, B ) // one possible fun
 int ℓ, m, n ;  LIST A, &B ;
{  int  i, j, k, t ;
   i = ℓ; k = ℓ; j = m+1 ;
   while ( ( i <= m ) && ( j <= n ) )
      {   if ( A[i].key <= A[j].key )
               B[k++] = A [i++] ;
             else
               B[k++] = A[j++] ;
      }
   if ( i > m )   for ( t = j ; t <= n ;  t++ )     B[k+t-j] = A[t] ;
   else           for ( t = i ; t <= m ; t++ )     B[k+t-i] = A[t] ;
}
```



算法时间复杂度

$T(n) = O(n-\ell+1)$

Merge in Linear Time

20	12
13	11
7	9
2	1

MERGE IN LINEAR TIME

20	12
13	11
7	9
2	1

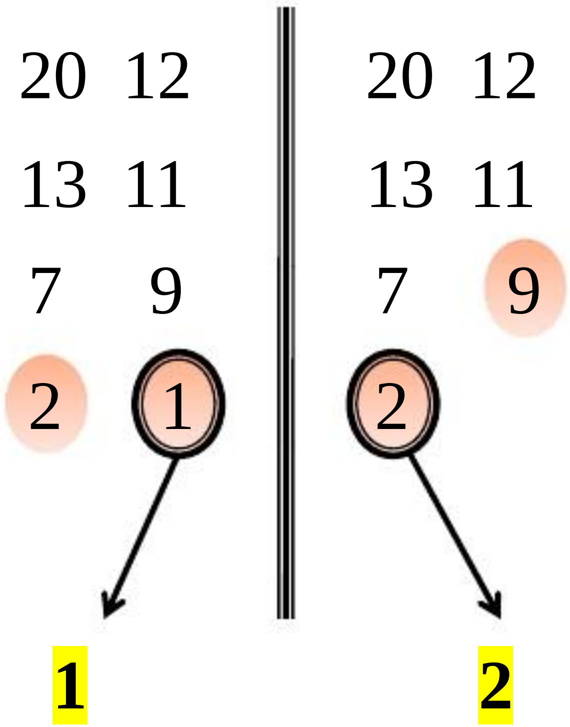
1

MERGE IN LINEAR TIME

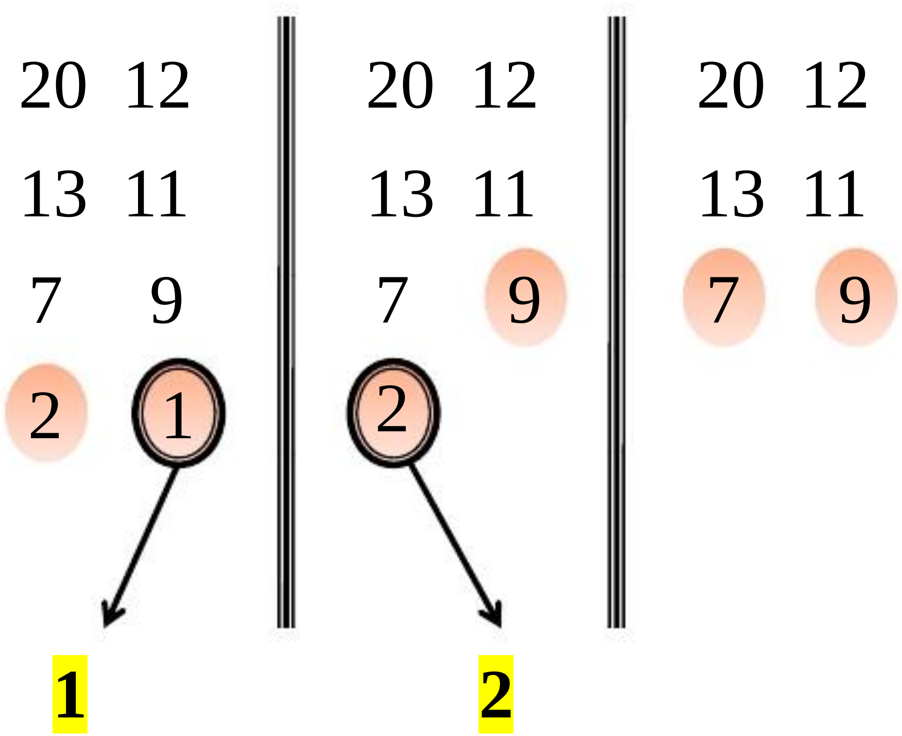
20	12		20	12
13	11		13	11
7	9		7	9
2	1		2	

1

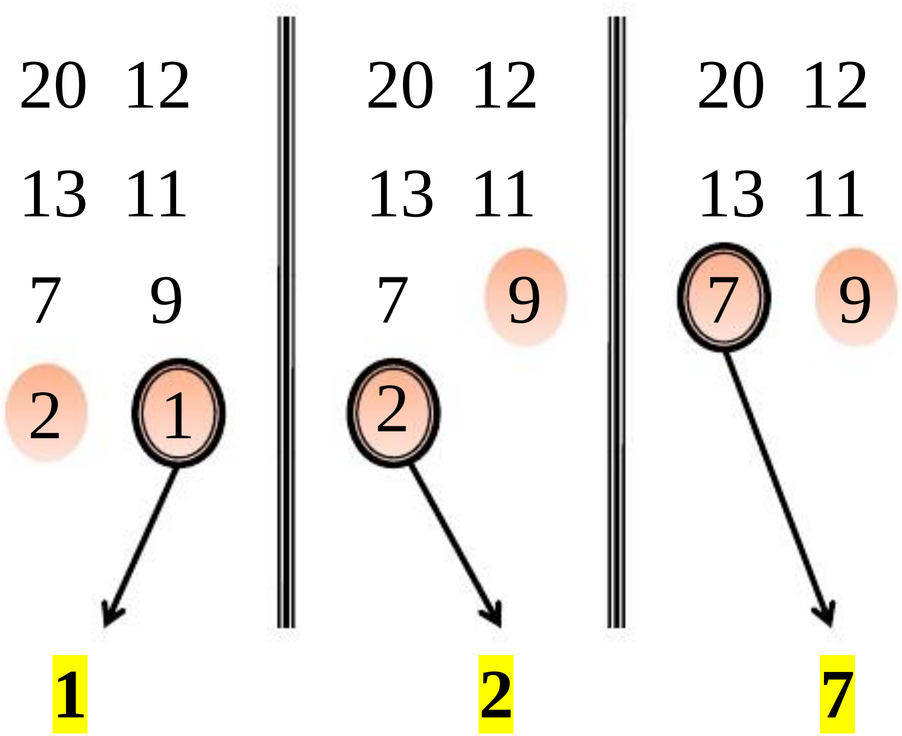
MERGE IN LINEAR TIME



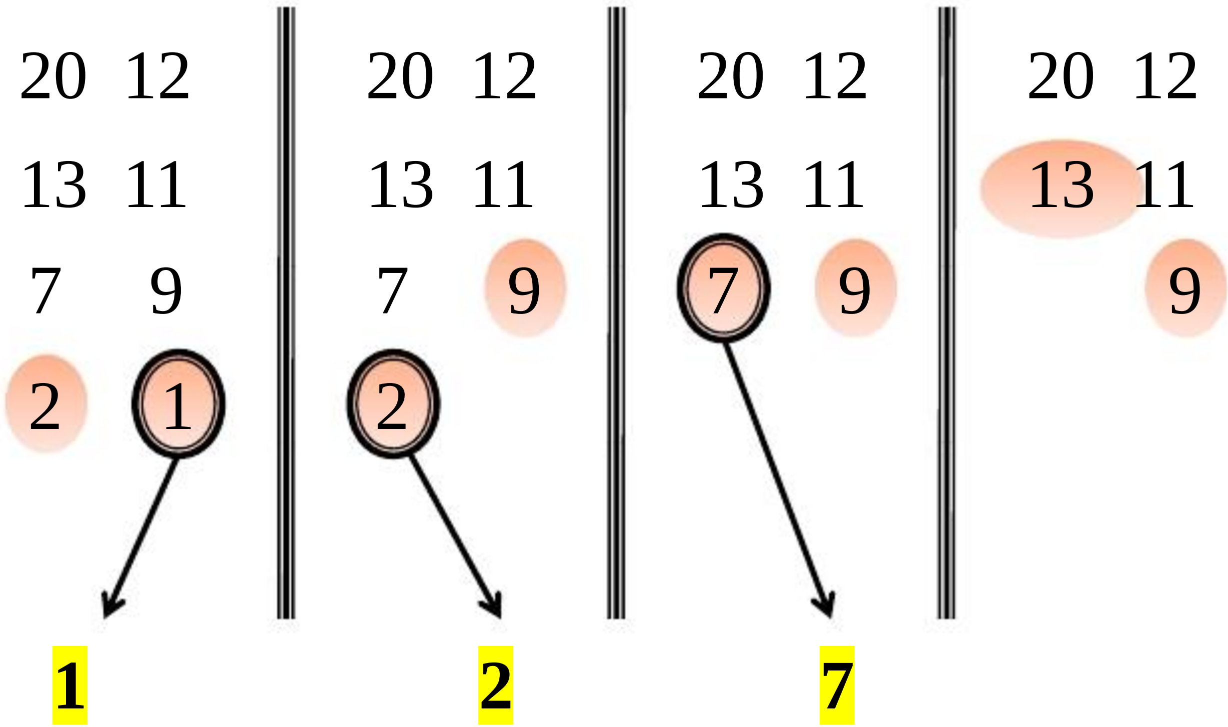
Merge in Linear Time



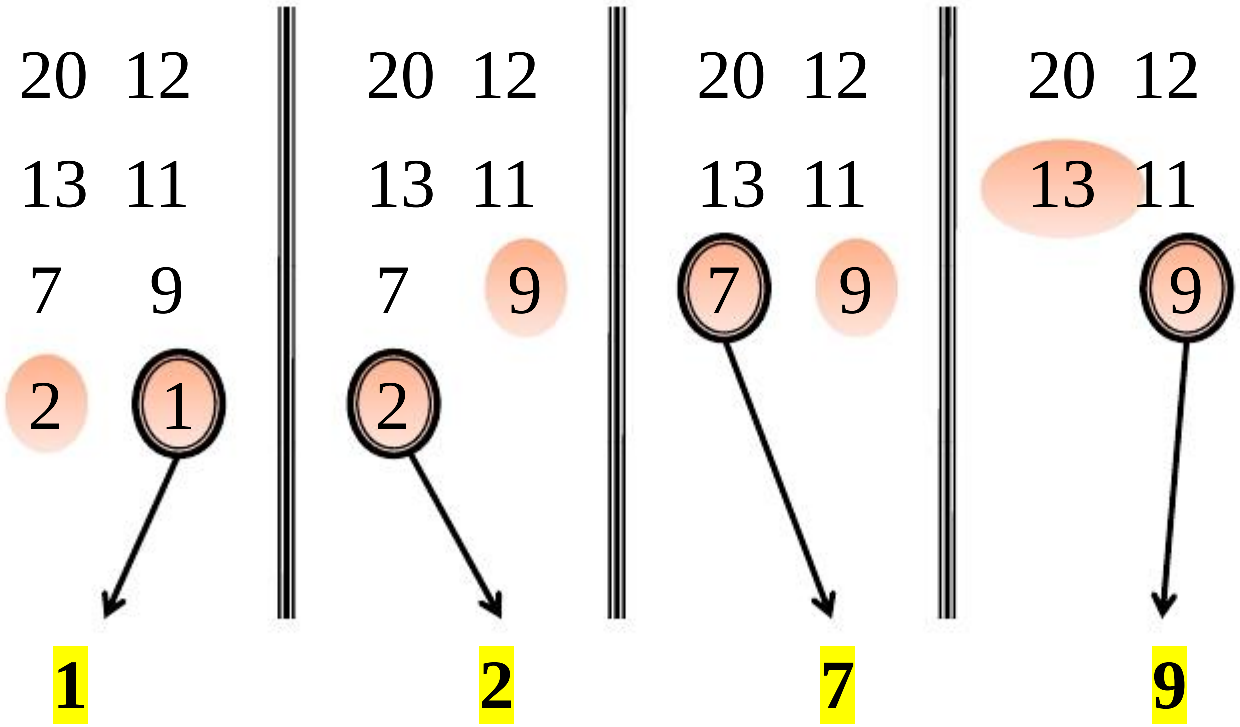
Merge in Linear Time



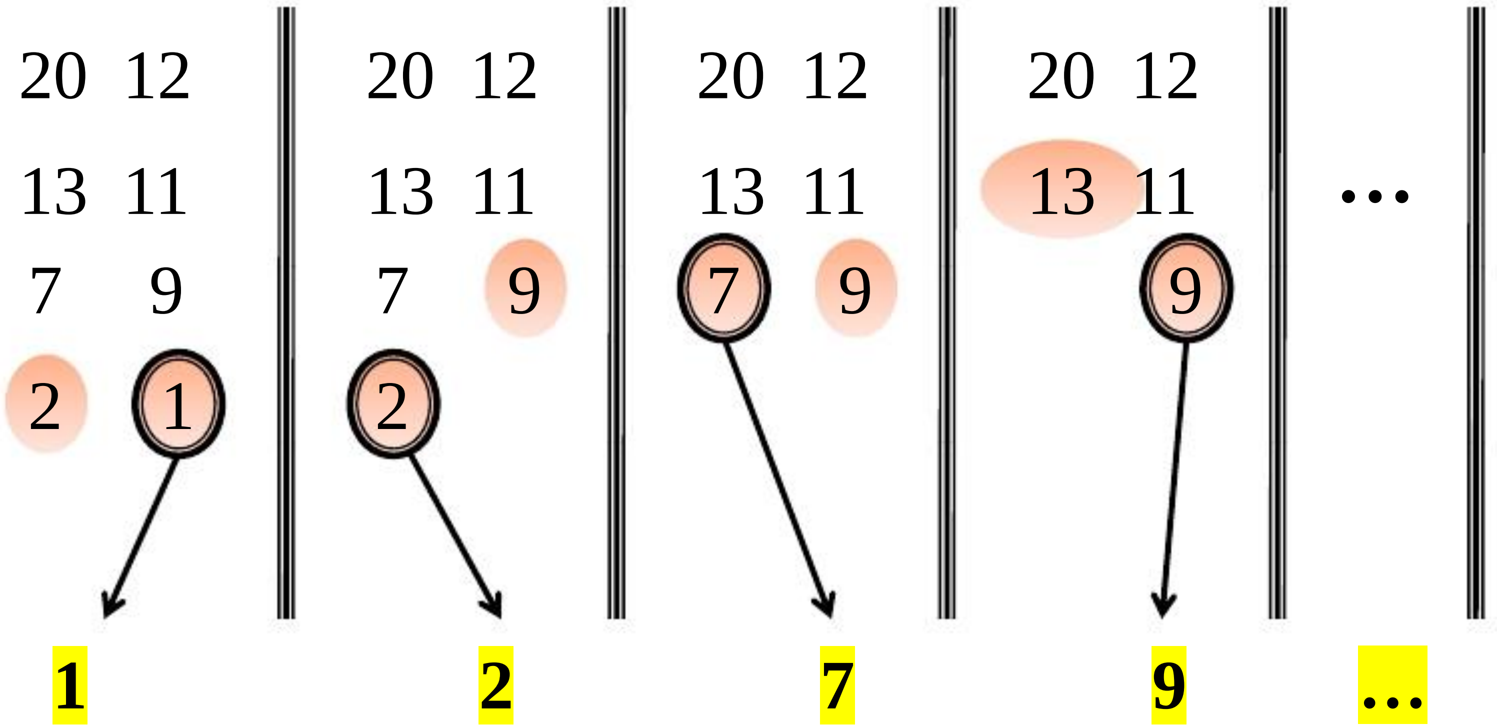
MERGE IN LINEAR TIME



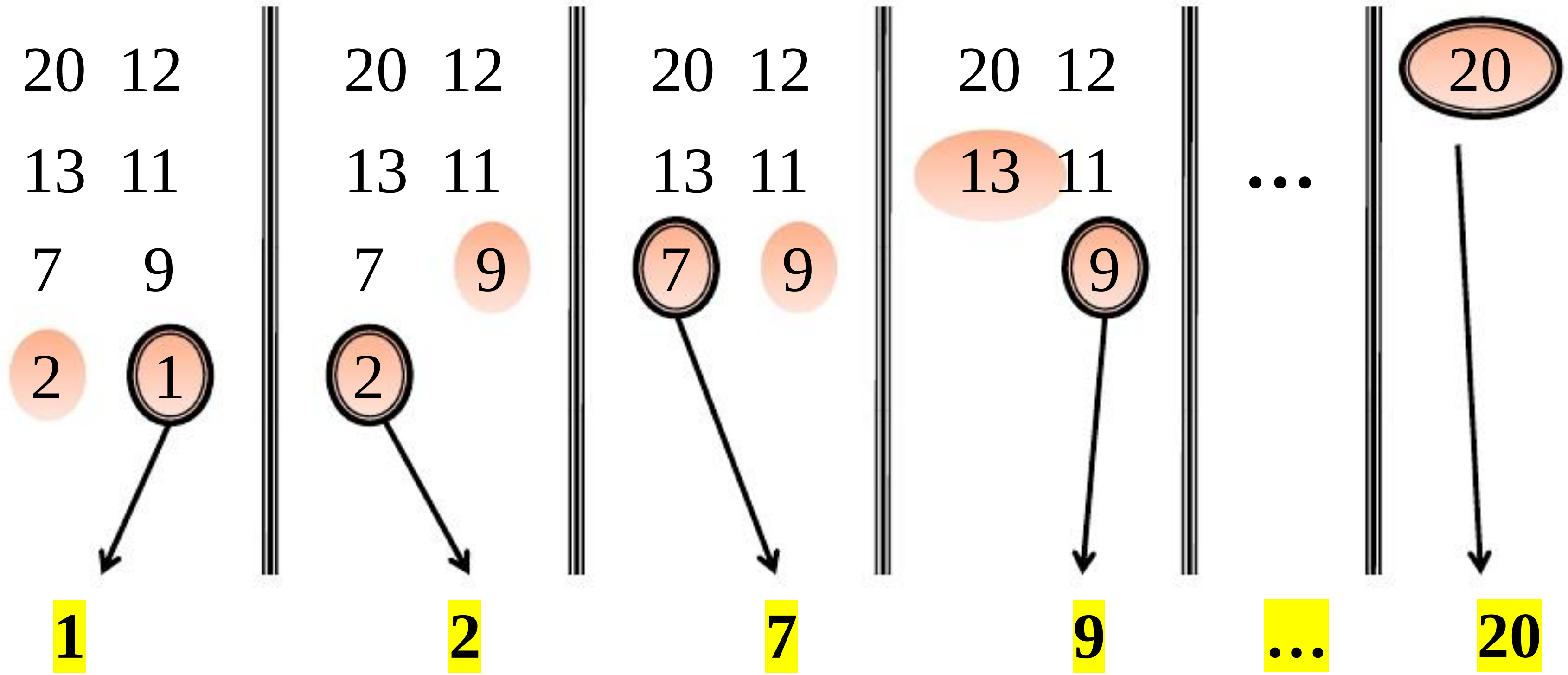
Merge in Linear Time



Merge in Linear Time



Merge in Linear Time



Merge Sort

Divide

Trivial.

Conquer

Recursively sort 2 subarrays.

$2T(n/2)$

Combine

Merge in linear time

$$T(n) = \begin{cases} \Theta(1) & \text{if } n = 1 \\ 2T(n/2) + \Theta(n) & \text{if } n \neq 1 \end{cases}$$

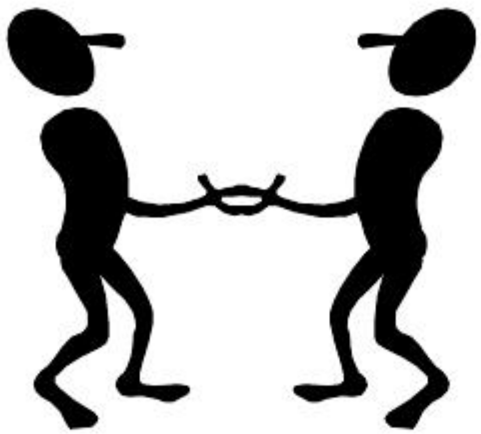
Recurrences And Divide-and-conquer

Recurrence

A recurrence is an equation or inequality that describes a function in terms of its value on smaller inputs.

Recurrence and Divide and Conquer

Twins



Inequal problem?

???How to resolve a Recurrence?

Recurrences

Three methods:

- Substitution method
- Recursion tree method
- Master method

SUBSTITUTION METHOD

The most general method:

1. *Guess* the form of the solution.
2. *Verify* by induction.
3. *Solve* for constants.

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

- [Assume that $T(1) = \Theta(1)$.]
- Guess $O(n^3)$. (Prove O and Ω separately.)
- Assume that $T(k) \leq ck^3$ for $k < n$.
- Prove $T(n) \leq cn^3$ by induction.

Example of Substitution

- We must also handle the initial conditions, that is, ground the induction with base cases.
- ***Base:*** $T(n) = \Theta(1)$ for all $n < n_0$, where n_0 is a suitable constant.
- For $1 \le n < n_0$, we have "$\Theta(1)$" $\le cn^3$, if we pick c big enough.

- Guess $O(n^3)$

Example (continued)

$$
\begin{aligned}
T(n) &= 4T(n/2) + n \\
&\le 4c(n/2)^3 + n \\
&= (c/2)n^3 + n \\
&= cn^3 - ((c/2)n^3 - n) \longleftarrow \textit{desired - residual} \\
&\le cn^3 \longleftarrow \textit{desired}
\end{aligned}
$$

Whenever $(c/2)n^3 - n \ge 0$, for example,
if $c \ge 2$ and $n \ge 1$.

residual

This bound is not tight!

A tighter upper bound?

We shall prove that $T(n) = O(n^2)$.

Assume that $T(k) \le ck^2$ for $k < n$:

$$
\begin{aligned}
T(n) &= 4T(n/2) + n \\
&\le cn^2 + n \\
&= cn^2 - (-n) \\
&\le cn^2
\end{aligned}
$$

for no choice n when $c > 0$. Lose!

A tighter upper bound!

IDEA: Strengthen the inductive hypothesis.

- ***Subtract*** a low-order term.

Inductive hypothesis: $T(k) \le c_1 k^2 - c_2 k$ for $k < n$.

$$
\begin{aligned}
T(n) &= 4T(n/2) + n \\
&\le 4(c_1 (n/2)^2 - c_2(n/2)) + n \\
&= c_1 n^2 - 2c_2 n + n \\
&= c_1 n^2 - c_2 n - (c_2 n - n) \\
&\le c_1 n^2 - c_2 n \quad \text{if } c_2 > 1.
\end{aligned}
$$

Pick c_1 big enough to handle the initial conditions.

Example of Substitution

- Prove the solution of $T(\mathrm{n})=T(\left\lceil \frac{n}{2} \right\rceil)+1$ is $O(\lg \mathrm{n})$.
  - ***Base:*** $n_0 = 2$, $\mathrm{T}(2) = c_0 \le c_0 \lg 2 = c_0$ ***n_0 = 1 ?***
  - Assume for $2 \le n < k$, and $c \ge c_0$, we have

$$\mathrm{T(n)} \le c\ lg\ n.$$

When n = k, we also

$$\mathrm{T(k)} = \mathrm{T}(\left\lceil \frac{k}{2} \right\rceil)+1$$

$$\le c\ lg \left\lceil \frac{k}{2} \right\rceil +1 \le c\ lg \left\lceil \frac{k}{\sqrt{2}} \right\rceil +1$$

$$\le \mathrm{c \lg k} + 1 - \mathrm{c}/2 \le \mathrm{c \lg k}$$

$c_0 \ge 2$

Example of Substitution

- Prove the solution of $T(\mathrm{n})=2T(\lfloor \frac{n}{2} \rfloor)+\mathrm{n}$ is $O(n\lg n)$.
  - ***Base:*** $n_0 = 2$, $\mathrm{T}(2) = c_0 \leq c_0\,(2 \lg 2 + 2)$
  - Assume for $2 \leq n < k$, and $c \geq c_0$, we have

$$\mathrm{T}(n) \leq c\, n\lg n.$$

When $n = k$, we also

$$\mathrm{T}(k) \leq 2\,(c \left\lfloor \frac{k}{2} \right\rfloor lg \left\lfloor \frac{k}{2} \right\rfloor) + k \leq c\, k\, lg\, \frac{k}{2} + k$$

$$= ck \lg k - ck \lg 2 + k \leq c\, k\lg k$$

$c \geq 1$

Recursion-tree method

- A recursion tree **models** the costs (time) of a recursive execution of an algorithm.
- The **recursion tree method** is good for generating guesses for the **substitution method**.

Example of recursion tree

Solve $T(n) = T(n/4) + T(n/2) + n^2$:

Example of recursion tree

Solve $T(n) = T(n/4) + T(n/2) + n^2$:

$$T(n)$$

Example of recursion tree

Solve $T(n) = T(n/4) + T(n/2) + n^2$:

$$n^2$$

$$T(n/4) \qquad\qquad T(n/2)$$

Example of recursion tree

Solve $T(n) = T(n/4) + T(n/2) + n^2$:

$$n^2$$

$$(n/4)^2 \qquad\qquad (n/2)^2$$

$$T(n/16) \quad T(n/8) \qquad T(n/8) \quad T(n/4)$$

Example of recursion tree

Solve $T(n) = T(n/4) + T(n/2) + n^2$:

$$n^2$$

$$(n/4)^2 \qquad (n/2)^2$$

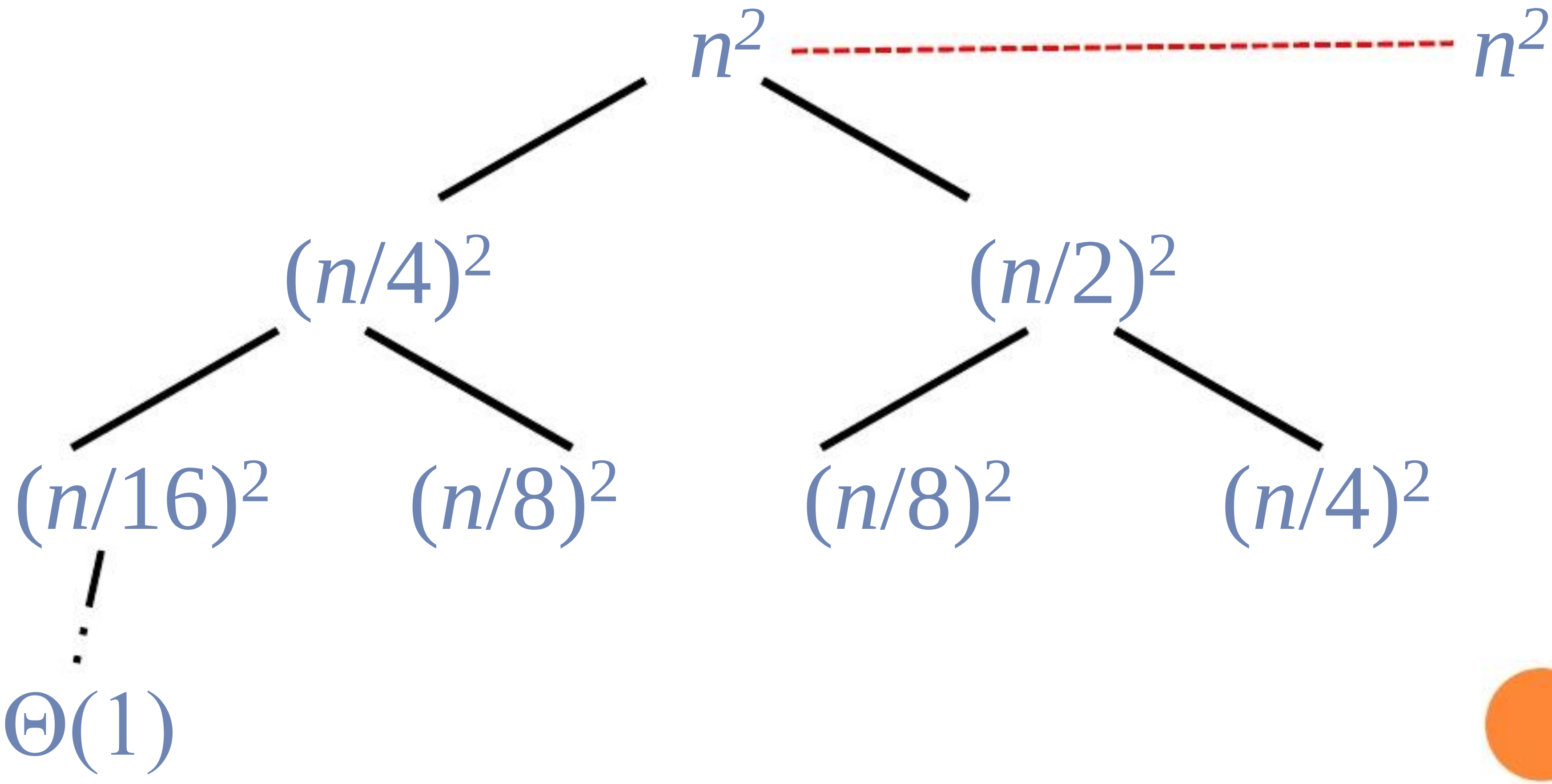
$$T(n/16) \qquad T(n/8) \qquad T(n/8) \qquad T(n/4)$$

$$\vdots$$

$$\Theta(1)$$

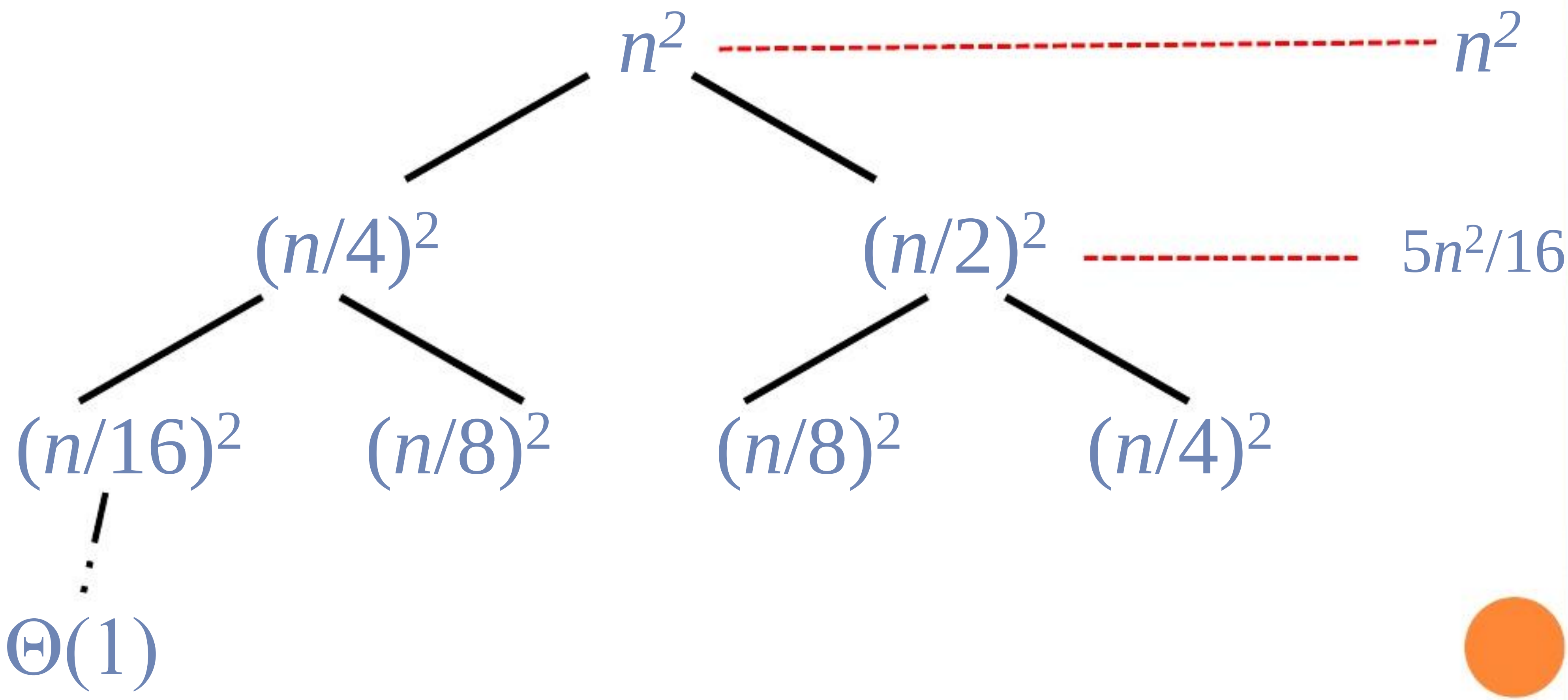
Example of recursion tree

Solve $T(n) = T(n/4) + T(n/2) + n^2$:



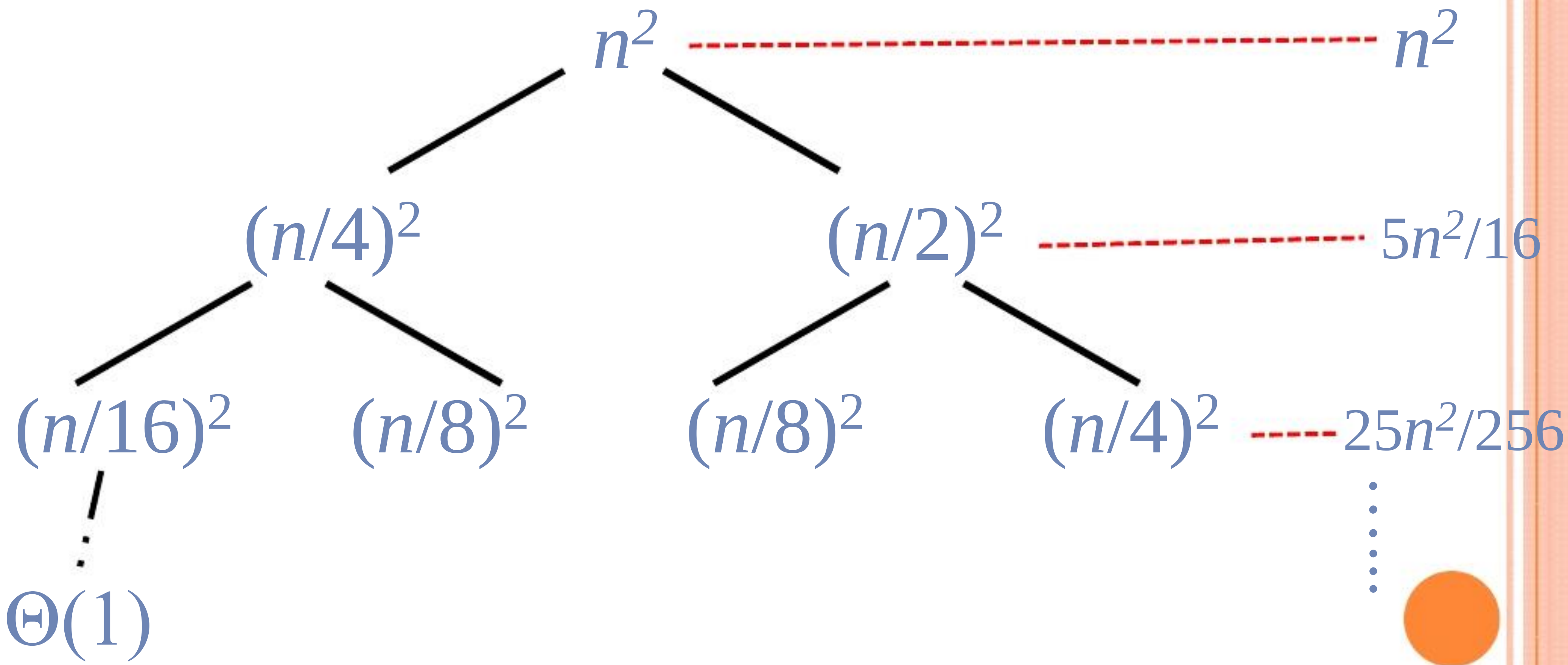
Example of recursion tree

Solve $T(n) = T(n/4) + T(n/2) + n^2$:



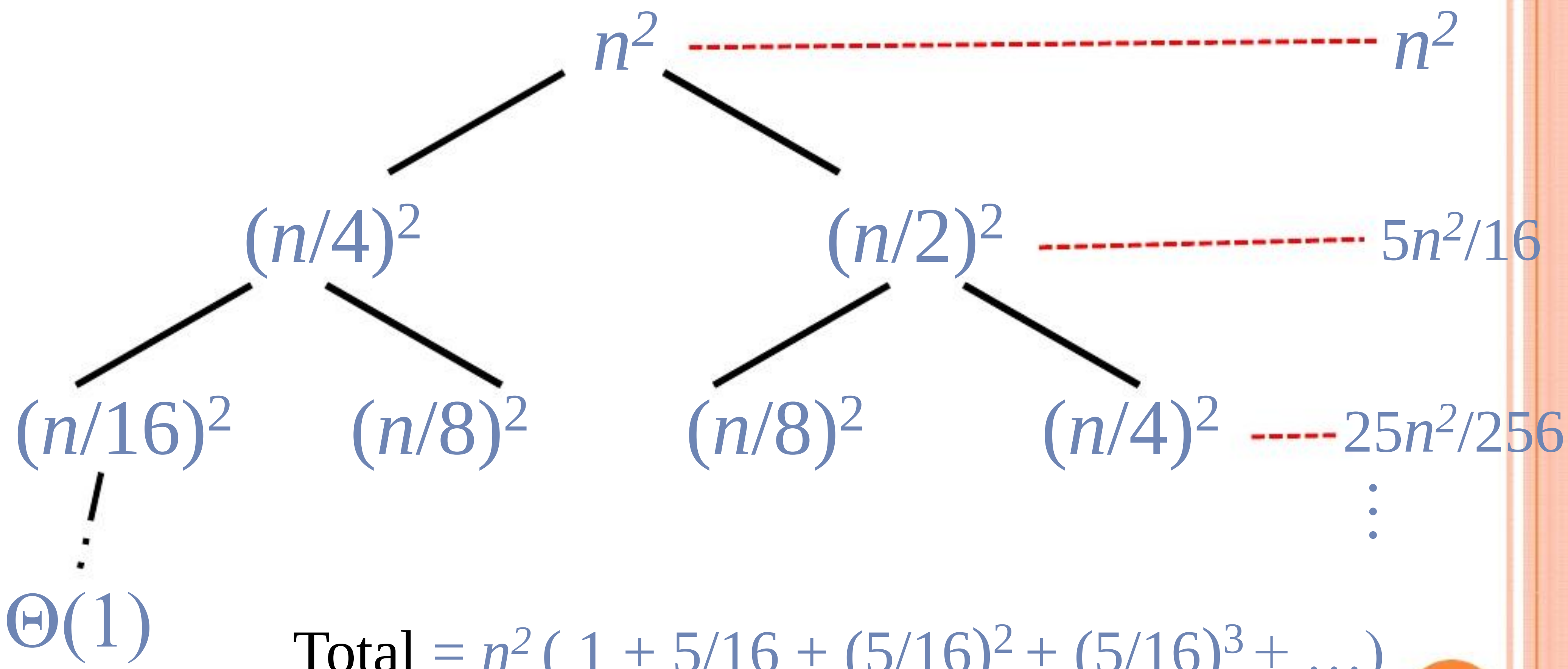
Example of recursion tree

Solve $T(n) = T(n/4) + T(n/2) + n^2$:



Example of recursion tree

Solve $T(n) = T(n/4) + T(n/2) + n^2$:



Total $= n^2 \left(1 + 5/16 + (5/16)^2 + (5/16)^3 + \ldots \right)$

$= \Theta(n^2)$ *geometric series*

The Master Method

The master method applies to recurrences of the form

$$T(n) = a\,T(n/b) + f(n)\ ,$$

where $a \geq 1$, $b > 1$, and $f(n)$ is an asymptotically non-negative function.

'$\frac{n}{b}$' is explained by $\left\lceil \frac{n}{b} \right\rceil$ or $\left\lfloor \frac{n}{b} \right\rfloor$.

Idea of master theorem

Recursion tree:

$h = \log_b n$

$f(n)$ -- $f(n)$

a

$f(n/b)$ $f(n/b)$ … $f(n/b)$ -------- $af(n/b)$

a

$f(n/b^2)$ $f(n/b^2)$ … $f(n/b^2)$ -------------------- $a^2 f(n/b^2)$

$\vdots$

$T(1)$ $\qquad n^{\log_b a} T(1)$

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

$$\Theta\left(n^{\log_b a}\right) + \sum_{j=0}^{\log_b (n-1)} a^j f(n/bj)$$

Three common cases

Compare $f(n)$ with $n^{\log_b a}$:

1. $f(n) = O(n^{\log_b a - \varepsilon})$ for some constant $\varepsilon > 0$.
   - $f(n)$ grows polynomially slower than $n^{\log_b a}$ (by an n^{ε} factor). $f(n) = O(n^{\log_b a} / n^{\varepsilon})$

   Solution: $T(n) = \Theta(n^{\log_b a})$.

2. $f(n) = \Theta(n^{\log_b a})$.
   - $f(n)$ and $n^{\log_b a}$ grow at similar rates.

   Solution: $T(n) = \Theta(n^{\log_b a} \lg n) = \Theta(f(n) \lg n)$.

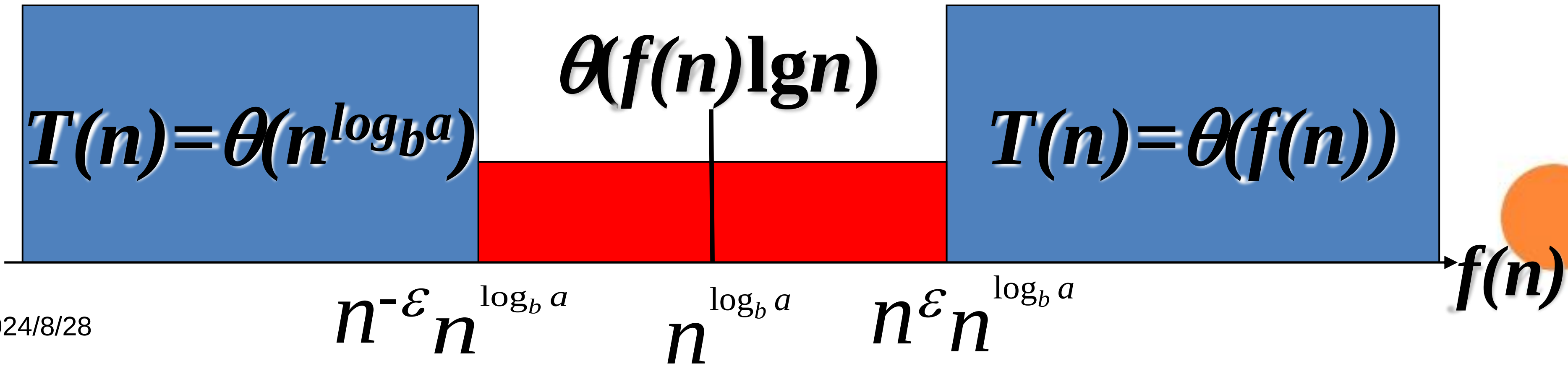
Three common cases

Compare $f(n)$ with $n^{\log_b a}$:

3. $f(n) = \Omega(n^{\log_b a + \varepsilon})$ for some constant $\varepsilon > 0$.
   - $f(n)$ grows polynomially faster than $n^{\log_b a}$ (by an n^{ε} factor), $f(n) = O(n^{\log_b a} * n^{\varepsilon})$

   and $f(n)$ satisfies the *regularity condition* that $a f(n/b) \le c f(n)$ for some constant $c < 1$ and large n.

   Solution: $T(n) = \Theta(f(n))$.



Examples

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

$a = 4, b = 2 \Rightarrow n^{\log_b a} = n^2; \; f(n) = n.$

CASE 1: $f(n) = O(n^{2-\varepsilon})$ for $\varepsilon = 1$.

$\therefore T(n) = \Theta(n^2).$

Ex. $T(n) = 4T(n/2) + n^2$

$a = 4, b = 2 \Rightarrow n^{\log_b a} = n^2; f(n) = n^2.$

CASE 2: $f(n) = \Theta(n^{\log_b a}) = \Theta(n^2).$

$\therefore T(n) = \Theta(n^2 \lg n).$

Examples

Ex. $T(n) = 4T(n/2) + n^3$

$a = 4,\ b = 2 \Rightarrow n^{\log_b a} = n^2;\ f(n) = n^3.$

CASE 3: $f(n) = \Omega(n^{2+\varepsilon})$ for $\varepsilon = 1$

and $4(cn/2)^3 \le cn^3$ (reg. cond.) for $c = 1/2$.

$\therefore\ T(n) = \Theta(n^3).$

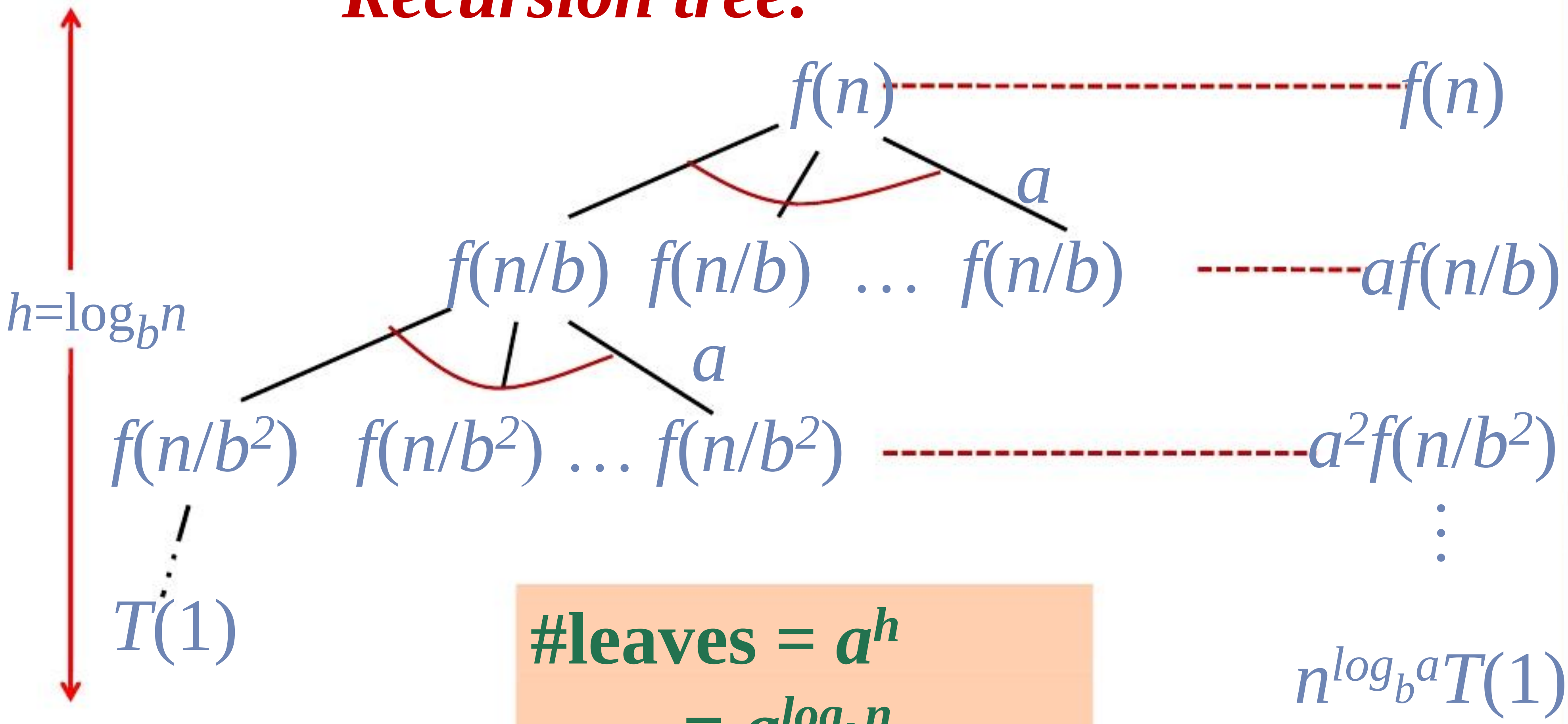
Ex. $T(n) = 4T(n/2) + n^2/\lg n$

$a = 4,\ b = 2 \Rightarrow n^{\log_b a} = n^2;\ f(n) = n^2/\lg n.$

Master method does not apply.

Idea of master theorem

Recursion tree:



#leaves $= a^h$

$= a^{\log_b n}$

$= n^{\log_b a}$

Idea of master theorem

Recursion tree:

$f(n)$ ------------------------------ $f(n)$

a

$f(n/b)$ $f(n/b)$ … $f(n/b)$ -------- $af(n/b)$

$h = \log_b n$

a

$f(n/b^2)$ $f(n/b^2)$ … $f(n/b^2)$ -------------------- $a^2 f(n/b^2)$

$\vdots$

$T(1)$

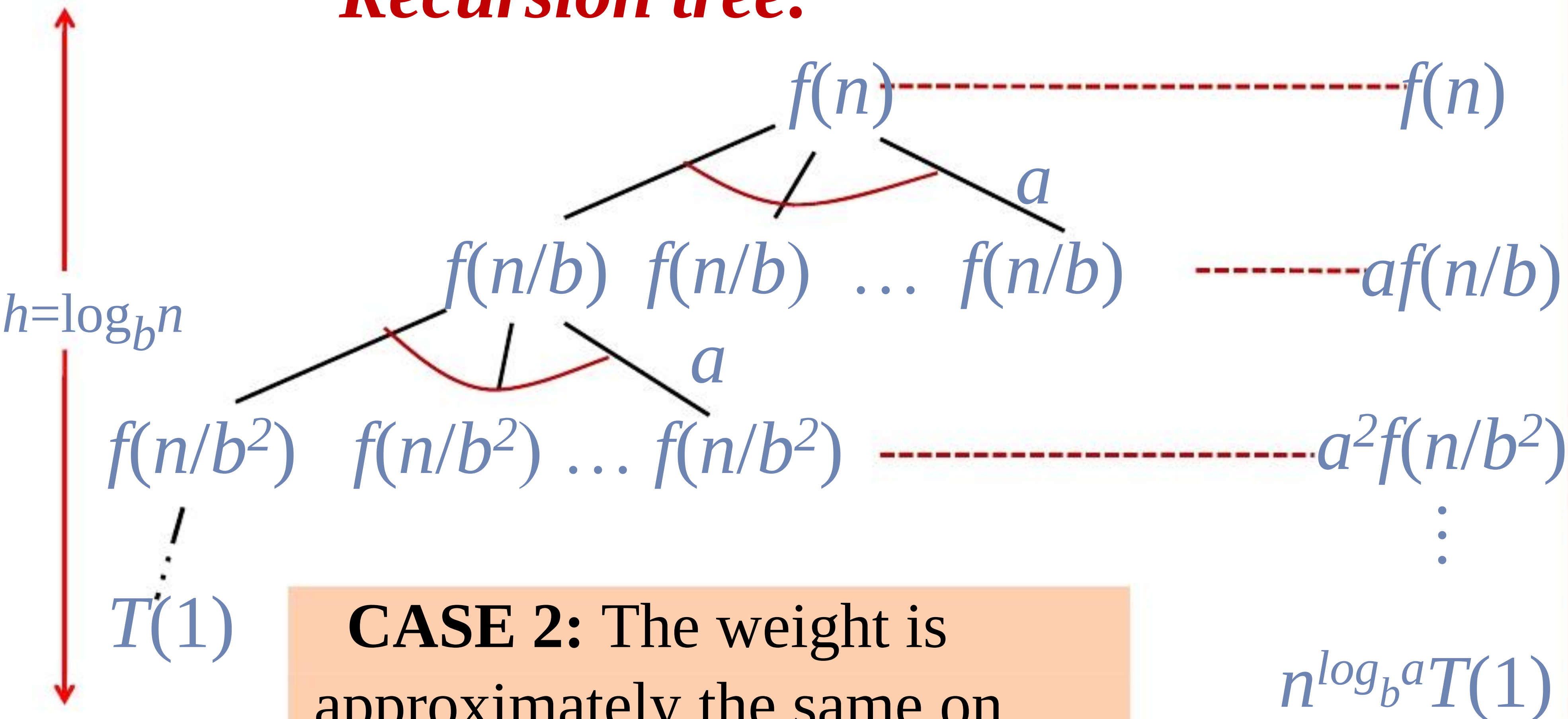
$n^{\log_b a} T(1)$

CASE 1: The weight increases geometrically from the root to the leaves. The leaves hold a constant fraction of the total weight

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

Idea of master theorem

Recursion tree:



CASE 2: The weight is approximately the same on each of the $\log_b n$ levels

$\Theta(n^{\log_b a} \lg n)$

Idea of master theorem

Recursion tree:

$f(n)$ ——————— $f(n)$

a

$f(n/b)$ $f(n/b)$ … $f(n/b)$ ——— $af(n/b)$

$h = \log_b n$

a

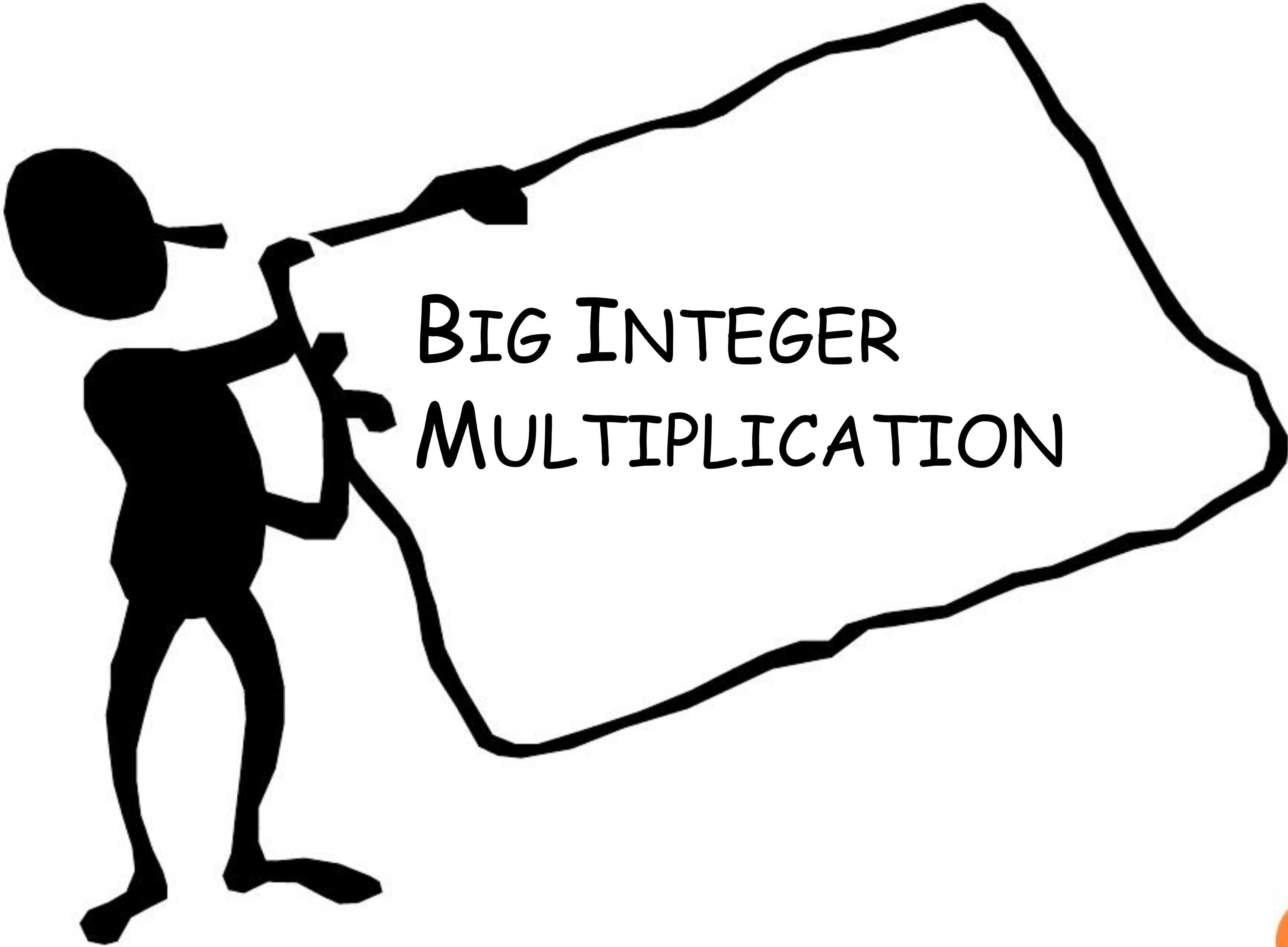
$f(n/b^2)$ $f(n/b^2)$ … $f(n/b^2)$ ——————— $a^2 f(n/b^2)$

$\vdots$

$T(1)$ $n^{\log_b a} T(1)$

CASE 3: The weight decreases geometrically from the root to the leaves. The root holds a constant fraction of the total weight.

$\boldsymbol{\Theta}(f(n))$



Big Integer Multiplication

BIG INTEGER MULTIPLICATION

Input: two n-bit integer X and Y

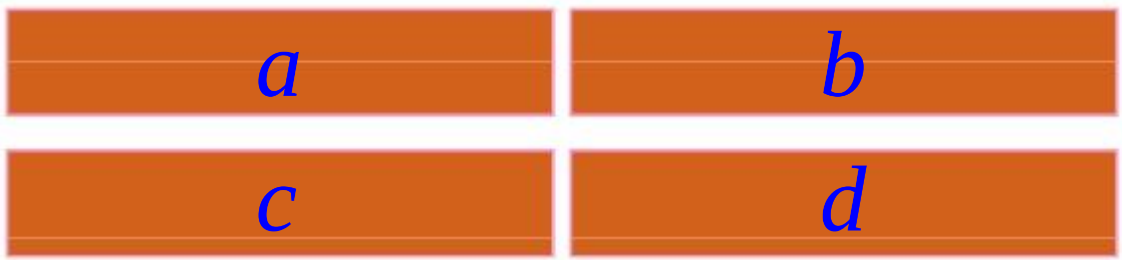
Output: the product of X and Y

Traditional method： $O(n^2)$ low efficiency

Divide-and-conquer:

$X =$

$Y =$



$X = a\,2^{n/2} + b \qquad Y = c\,2^{n/2} + d$

$XY = ac\,2^{n} + (ad + bc)\,2^{n/2} + bd$

BIG INTEGER MULTIPLICATION

Input: two n-bit integer X and Y

Output: the product of X and Y

Traditional method： O(n^2) low efficiency

Divide-a

Complexity

$$T(n)=\begin{cases}\Theta(1) & n=1\\ 4T(n/2)+\Theta(n) & n>1\end{cases}$$

$$T(n)=\Theta(n^2)$$ **no improvement**

X=

Y=

X =

$XY = ac\,2^n + (ad + bc)\,2^{n/2} + bd$

BIG INTEGER MULTIPLICATION

$$XY = ac\ 2^n + (ad + bc)\ 2^{n/2} + bd$$

Reduce the times of multiplication

1. $XY = ac\ 2^n + ((a - b)(d - c) + ac + bd)\ 2^{n/2} + bd$
2. $XY = ac\ 2^n + ((a + b)(d + c) - ac - bd)\ 2^{n/2} + bd$

Notice: we do not use euqation 2, for summation may conduct n+1 bits number.

Big integer multiplication

$XY = ac\, 2^n + (ad + bc)\, 2^{n/2} + bd$

Reduce the times of multiplication

1. $XY = ac\, 2^n + ((a - b)(d - c) + ac + bd)\, 2^{n/2} + bd$
2. $XY = ac\, 2^n + ((a + b)(c + d) - ac - bd)\, 2^{n/2} + bd$

Not
may

Complexity

$$T(n) = \begin{cases} \Theta(1) & n = 1 \\ 3T(n/2) + \Theta(n) & n > 1 \end{cases}$$

$$T(n) = \Theta\left(n^{log3}\right) = \Theta(n^{1.59})$$

improved

BIG INTEGER MULTIPLICATION

Even faster algorithm:

•Divide into more pieces, and use the complex methods to merge, may leads to more optimal algorithm.

•This idea conduct Fast Fourier Transform (FFT). FFT can be seen as a complex Divide-and-Conquer method. For Multiple it solve in $\Theta(n\log n)$ 。

Strassen Matrix Multiplication

STRASSEN MATRIX MULTIPLICATION

$$C = AB \text{ where } C[i][j] = \sum_{k=1}^{n} A[i][k]B[k][j]$$

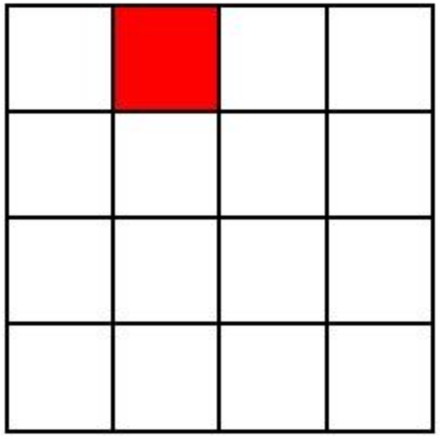
Traditional algorithm: $T(n) = \Theta(n^3)$
Divide and Conquer: $T(n) = \Theta(n^{\log 7}) = \Theta(n^{2.81})$

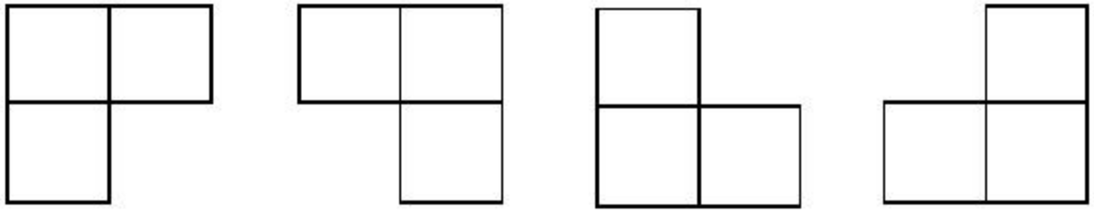
Now the best performance is $\boldsymbol{\Theta(n^{2.376})}$

Chessboard Cover

CHESS BOARD COVER

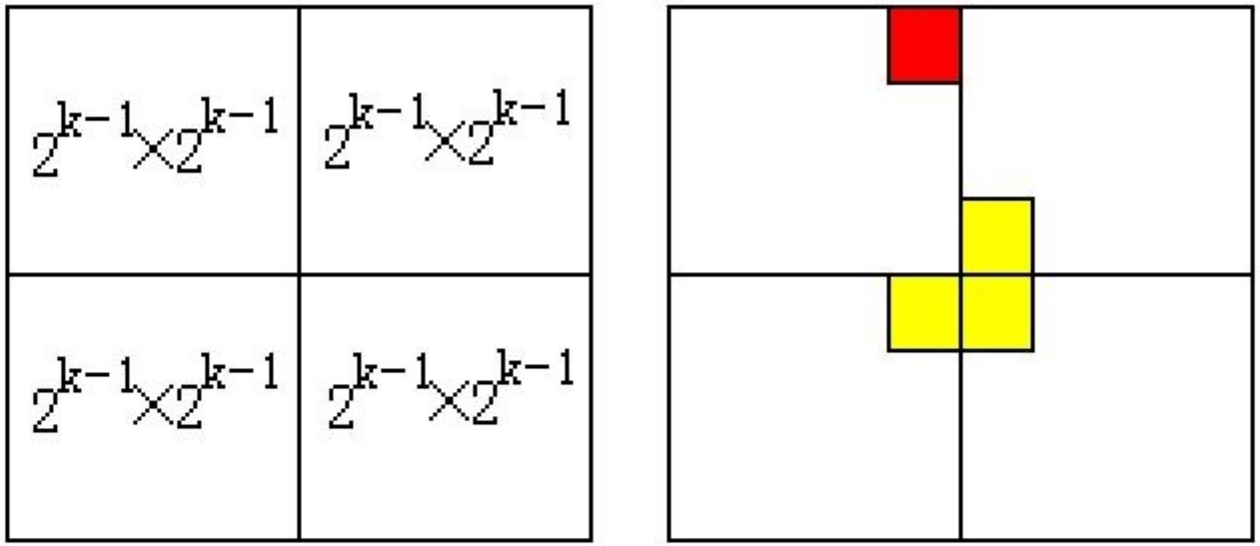
On a $2^k \times 2^k$ chessboard， only one square is different, called *specific*. In the chessboard cover problem, we use the following four kinds of *L*-shape cards to cover the whole chessboard squares except the specific, and request that there is no overlapping.





Chessboard Cover

- When $k>0$， partition $2^k \times 2^k$ chessboard into four $2^{k-1} \times 2^{k-1}$ sub-chessboard
  - The *specific* must be in one of the four sub-chessboard, and the other three have no specific.
- Now lay a *L*-shaped cards on the joint of the three sub-chessboard.
  - Then we get four smaller chessboard cover problem ($2^{(k-1)} * 2^{(k-1)}$).
- Do recursively until we get 1×1 chessboard。



(a) (b)

Chessboard Cover

•When k>0， partition $2^k \times 2^k$ chessboard into four $2^{k-1} \times 2^{k-1}$ sub-chessboard

•The *specific* must be in one of the four sub-chessboard, and the other three hav

•Now lay

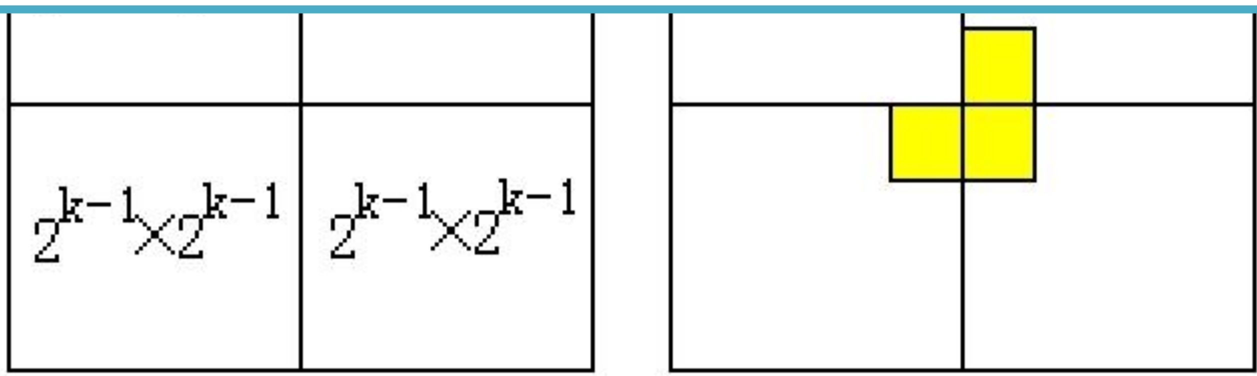
•Then

• Do recur

Complexity

$$T(k)=\begin{cases}\Theta(1) & k=0\\ 4T(k-1)+\Theta(1) & k>0\end{cases}$$

$$T(k)=\Theta(4^k)$$



(a) (b)

Binary Searching

Given n elements arranged in ascending order, find a particular element K.

Compare the middle element with the particular look up element X:

- if X is equal to the middle element, then the searching is successful and this algorithm is terminated;
- If X is less than the middle element, continue the searching in the first half of the sequence;
- otherwise, continue the searching in the second half of the sequence.

Searching 17 in the sequence [5,8,15,17,25,30,34,39,45,52,60] . Here, variables "low" and "high" stands for the searching scope, "mid" stands for the middle of the searching scope. In fact, mid=(low+high)/2)

0	1	2	3	4	5	6	7	8	9	10
5	8	15	17	25	30	34	39	45	52	60

low=0 mid=5 high=10

0	1	2	3	4	5	6	7	8	9	10
5	8	15	17	25	30	34	39	45	52	60

low=0 mid=5 high=10

0	1	2	3	4	5	6	7	8	9	10
5	8	15	17	25	30	34	39	45	52	60

low=0 mid=2 high=4

0	1	2	3	4	5	6	7	8	9	10
5	8	15	17	25	30	34	39	45	52	60

mid=3 low=3 high=4

Successfully complete the searching，
terminate the searching algorithm.