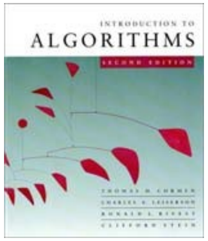
# CS60020: Foundations of Algorithm Design and Machine Learning

Sourangshu Bhattacharya

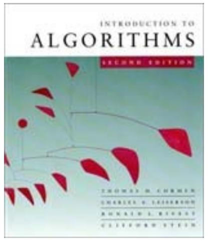
# DIVIDE AND CONQUER



# Merge sort

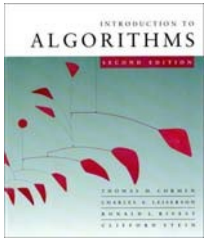
```
MERGE-SORT $A[1 \,.\,.\, n]$
    1. If $n = 1$, done.
    2. Recursively sort $A[\,1 \,.\,.\, \lceil n/2 \rceil\,]$
       and $A[\,\lceil n/2 \rceil + 1 \,.\,.\, n\,]$.
    3. "Merge" the 2 sorted lists.
```

***Key subroutine:*** **MERGE**

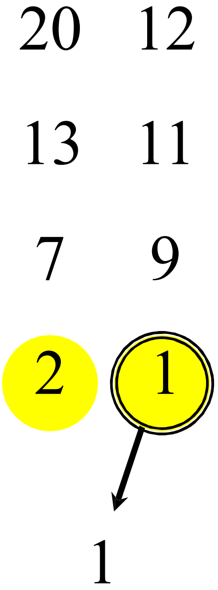


# Merging two sorted arrays

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



# Merging two sorted arrays



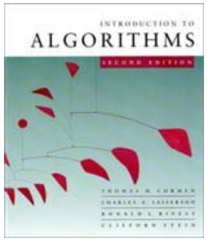
20 12

13 11

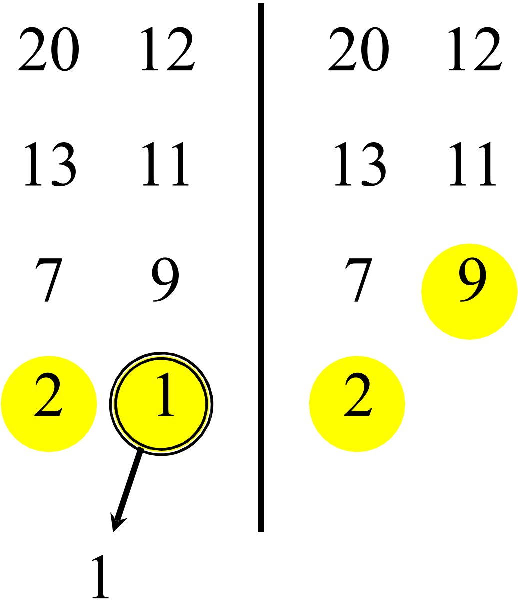
7 9

2 1

1

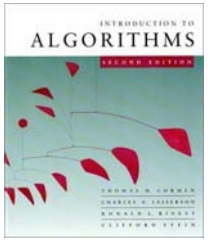


# Merging two sorted arrays

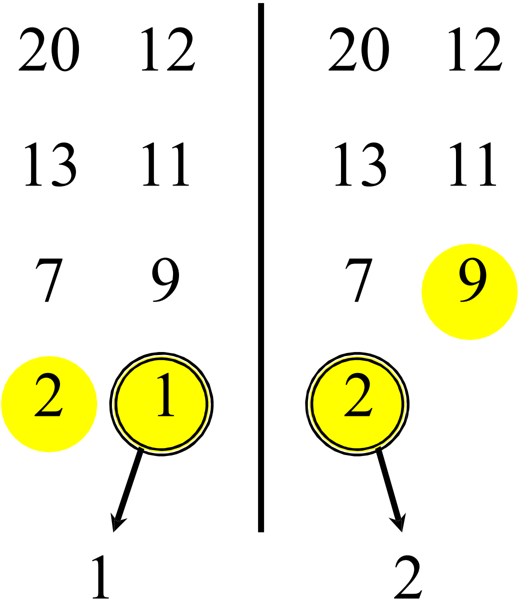


| | | | |
|---|---|---|---|
| 20 | 12 | 20 | 12 |
| 13 | 11 | 13 | 11 |
| 7 | 9 | 7 | 9 |
| 2 | 1 | 2 | |

1



# Merging two sorted arrays



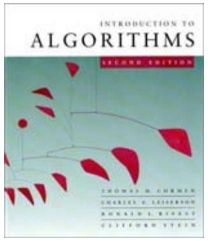
20 12 | 20 12

13 11 | 13 11

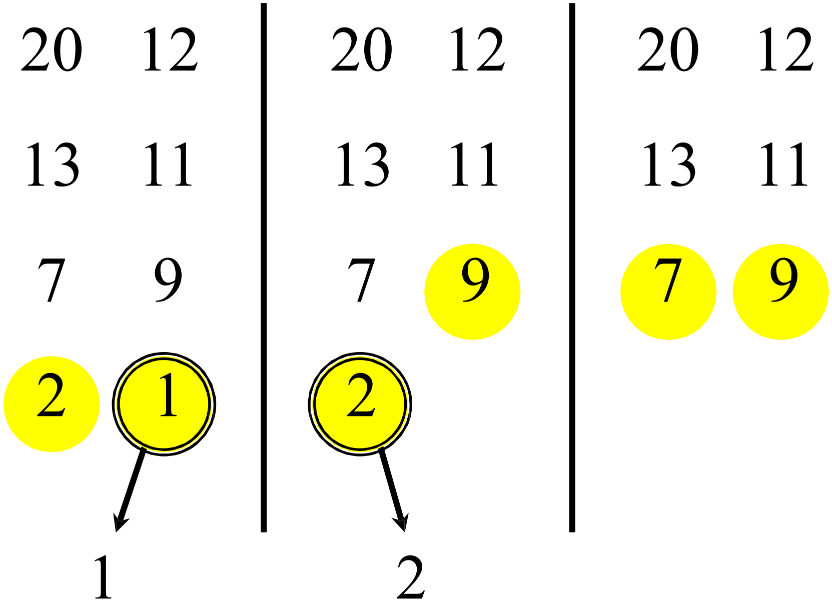
7 9 | 7 9

2 1 | 2

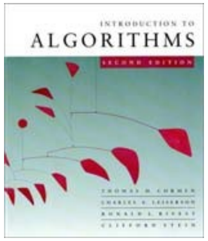
1 | 2



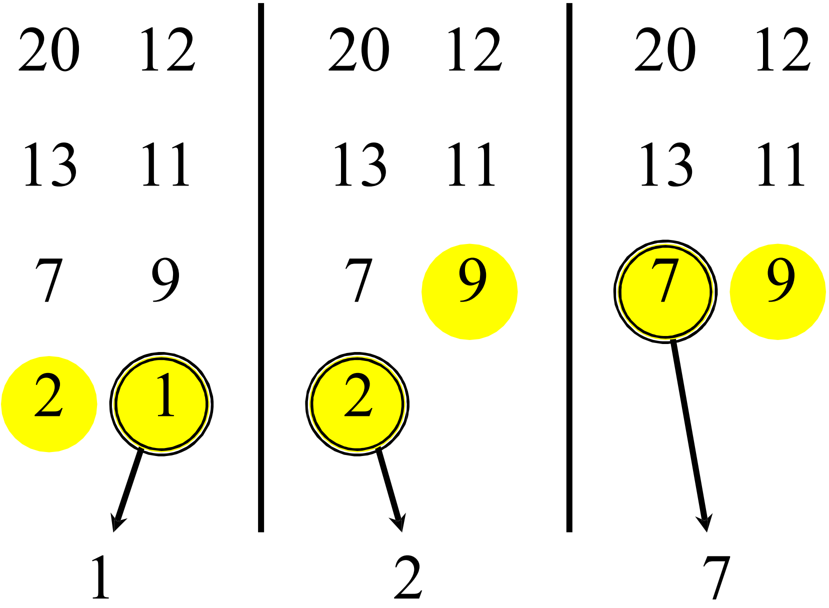
# Merging two sorted arrays



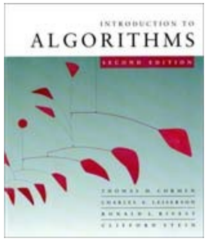
| | | | | | |
|---|---|---|---|---|---|
| 20 | 12 | 20 | 12 | 20 | 12 |
| 13 | 11 | 13 | 11 | 13 | 11 |
| 7 | 9 | 7 | 9 | 7 | 9 |
| 2 | 1 | 2 | | | |
| 1 | | 2 | | | |



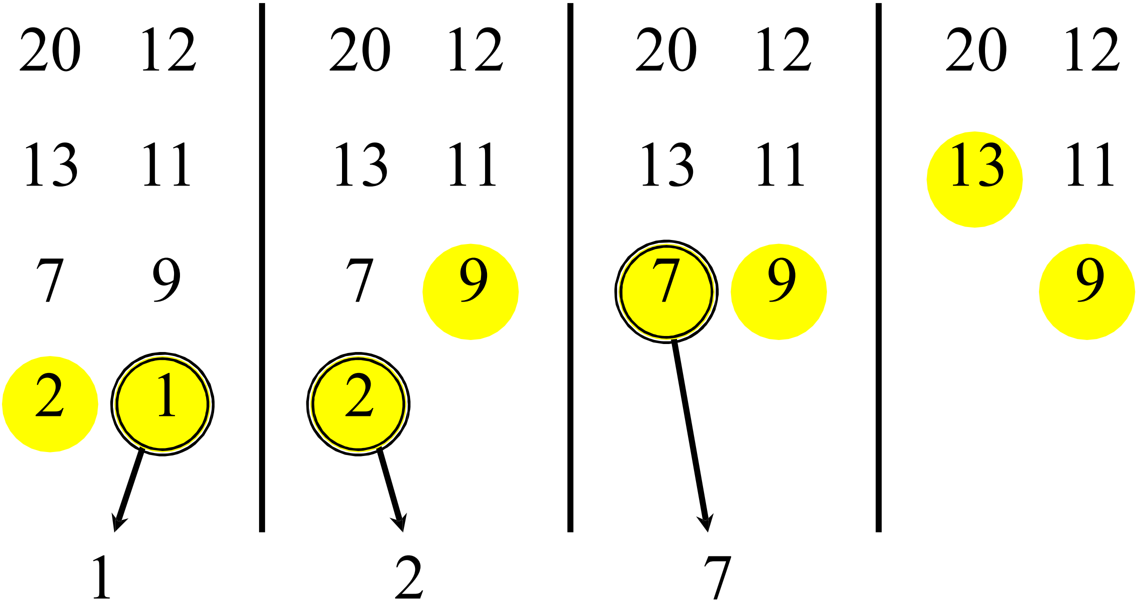
# Merging two sorted arrays

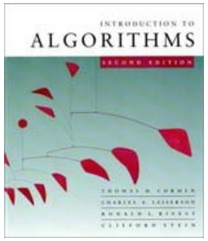


| | | | | | |
|---|---|---|---|---|---|
| 20 | 12 | 20 | 12 | 20 | 12 |
| 13 | 11 | 13 | 11 | 13 | 11 |
| 7 | 9 | 7 | 9 | 7 | 9 |
| 2 | 1 | 2 | | | |
| 1 | | 2 | | 7 | |

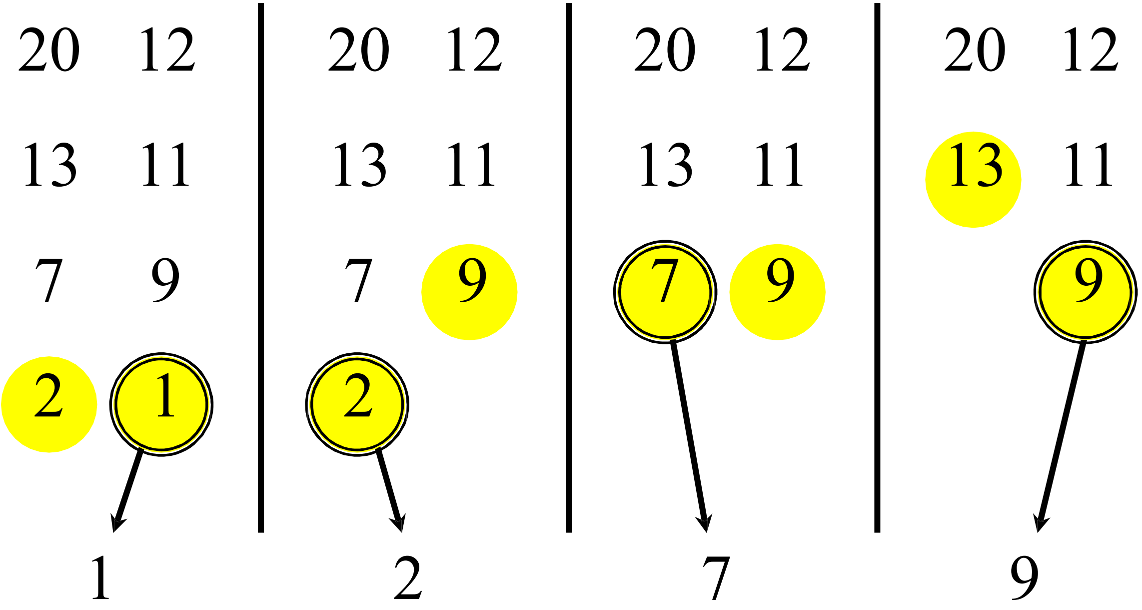


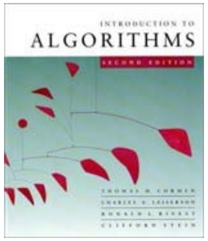
# Merging two sorted arrays



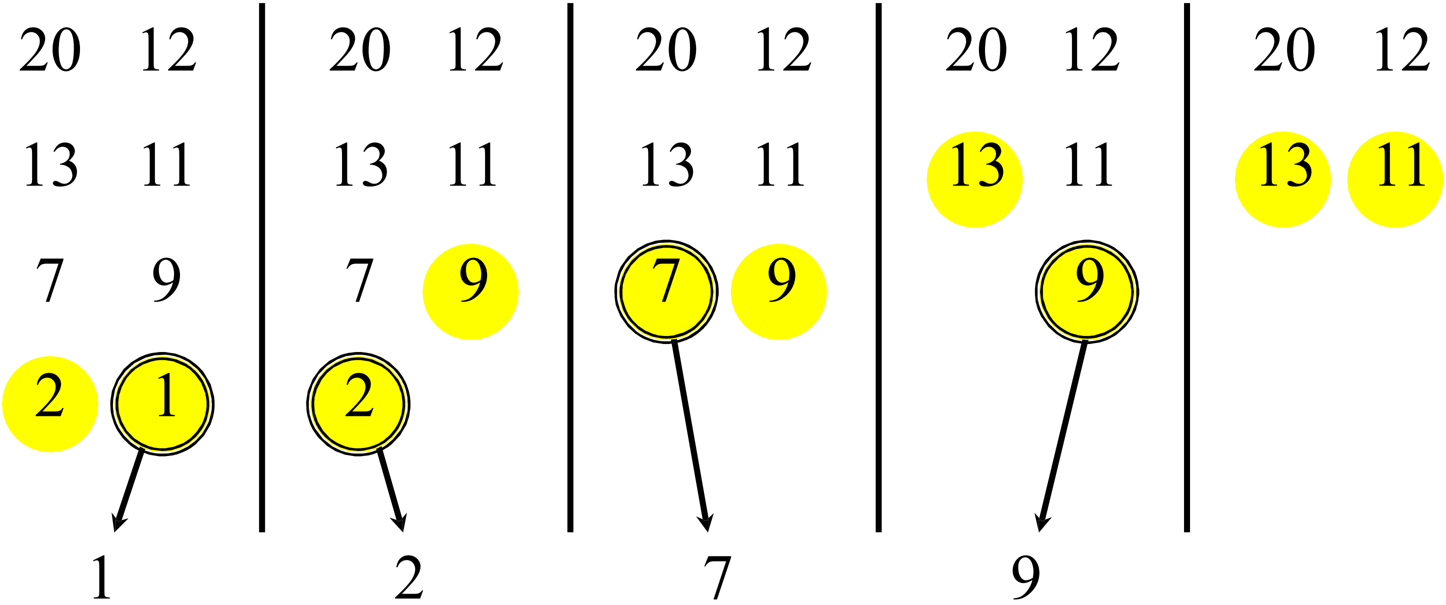


# Merging two sorted arrays





# Merging two sorted arrays



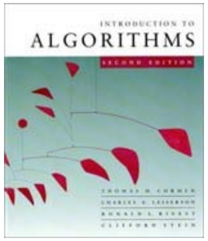
20 12
13 11
7 9
2 1
1

20 12
13 11
7 9
2
2

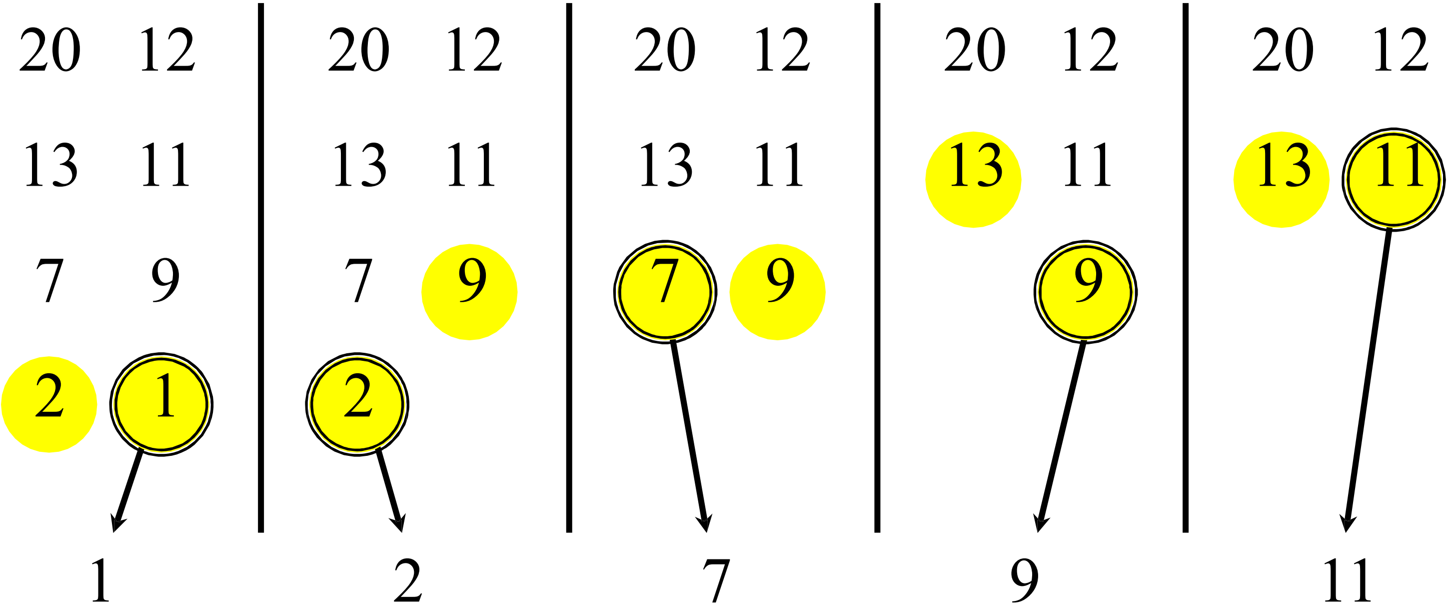
20 12
13 11
7 9
7

20 12
13 11
9
9

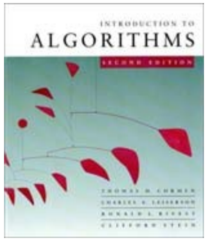
20 12
13 11



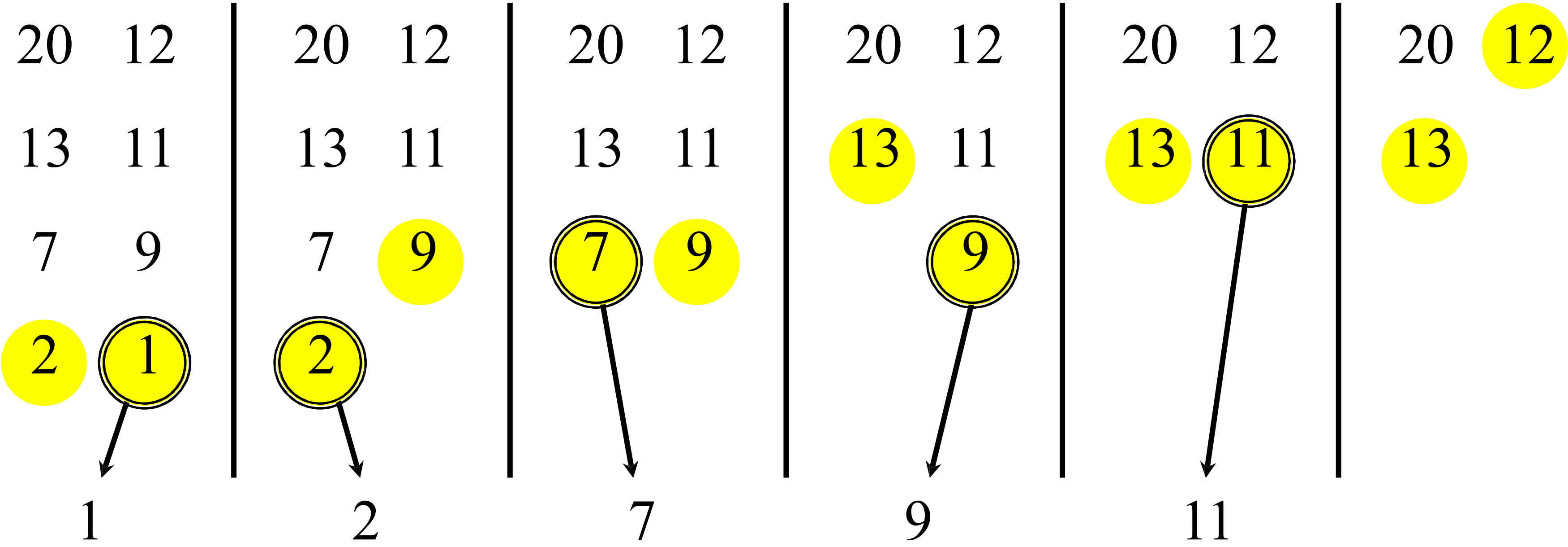
# Merging two sorted arrays



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



# Merging two sorted arrays



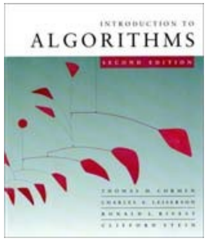
20 12 | 20 12 | 20 12 | 20 12 | 20 12 | 20 12

13 11 | 13 11 | 13 11 | 13 11 | 13 11 | 13

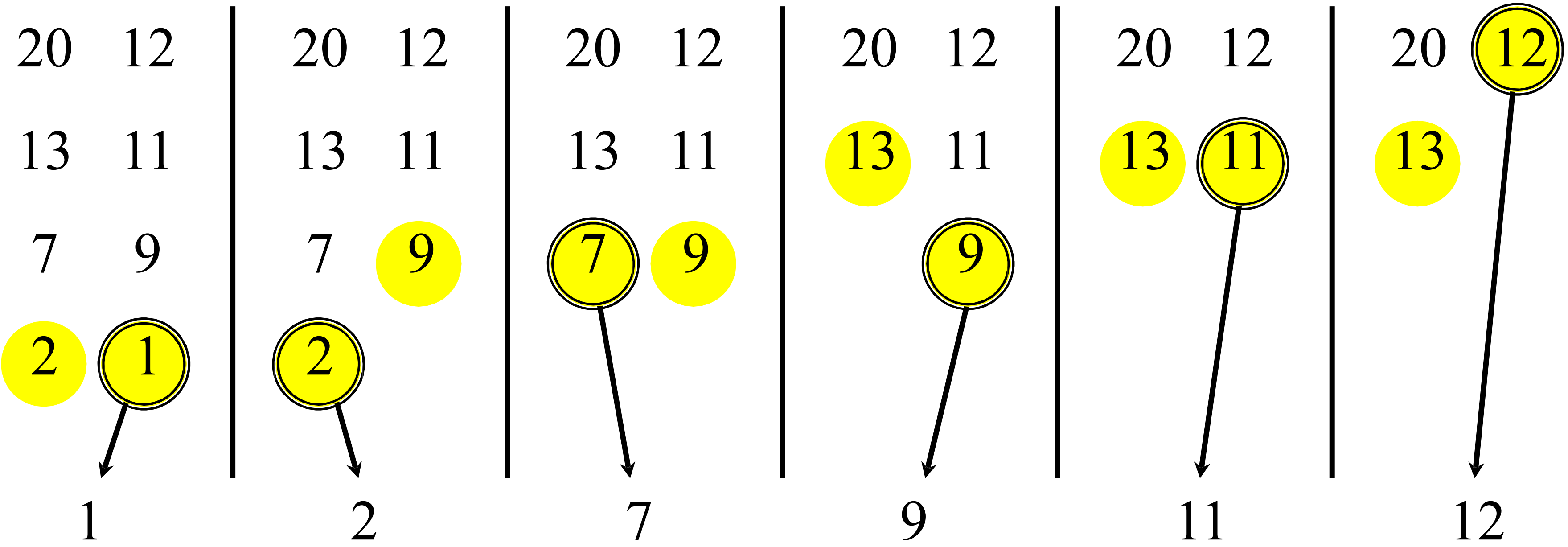
7 9 | 7 9 | 7 9 | 9

2 1 | 2

1 | 2 | 7 | 9 | 11


INTRODUCTION TO
ALGORITHMS
SECOND EDITION

# Merging two sorted arrays


20 12
13 11
7 9
2 1
1

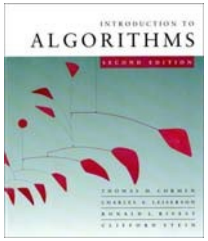
20 12
13 11
7 9
2
2

20 12
13 11
7 9
7

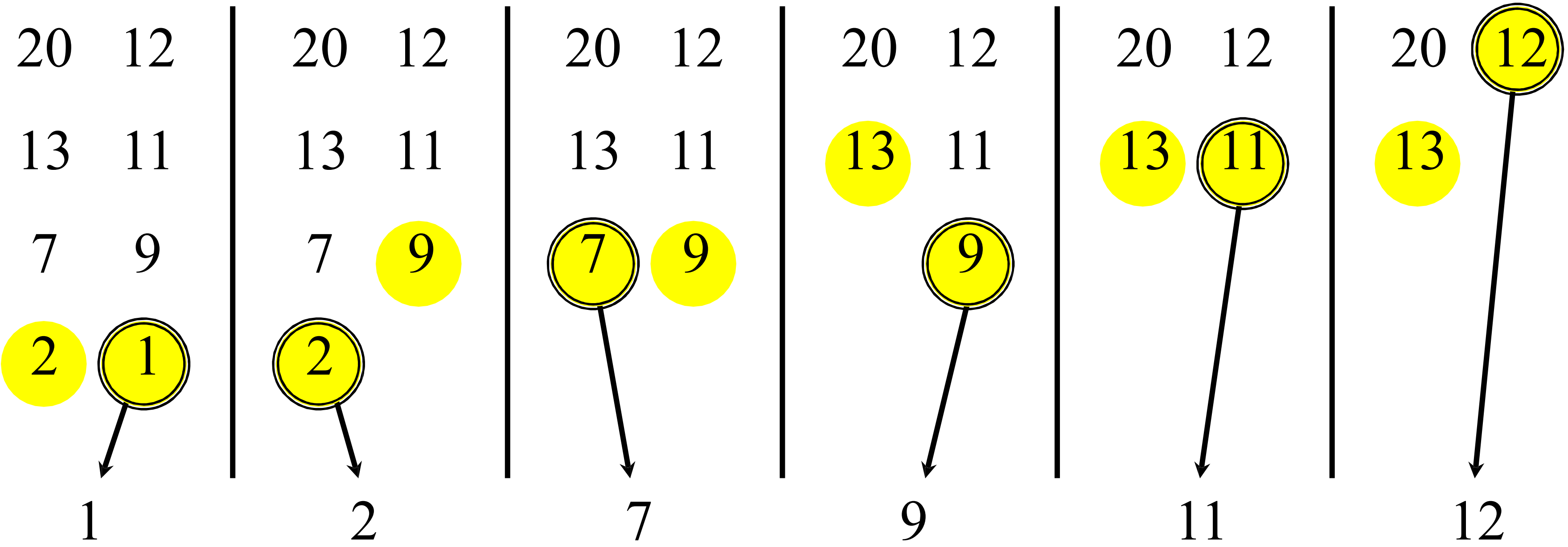
20 12
13 11
9
9

20 12
13 11
11

20 12
13
12

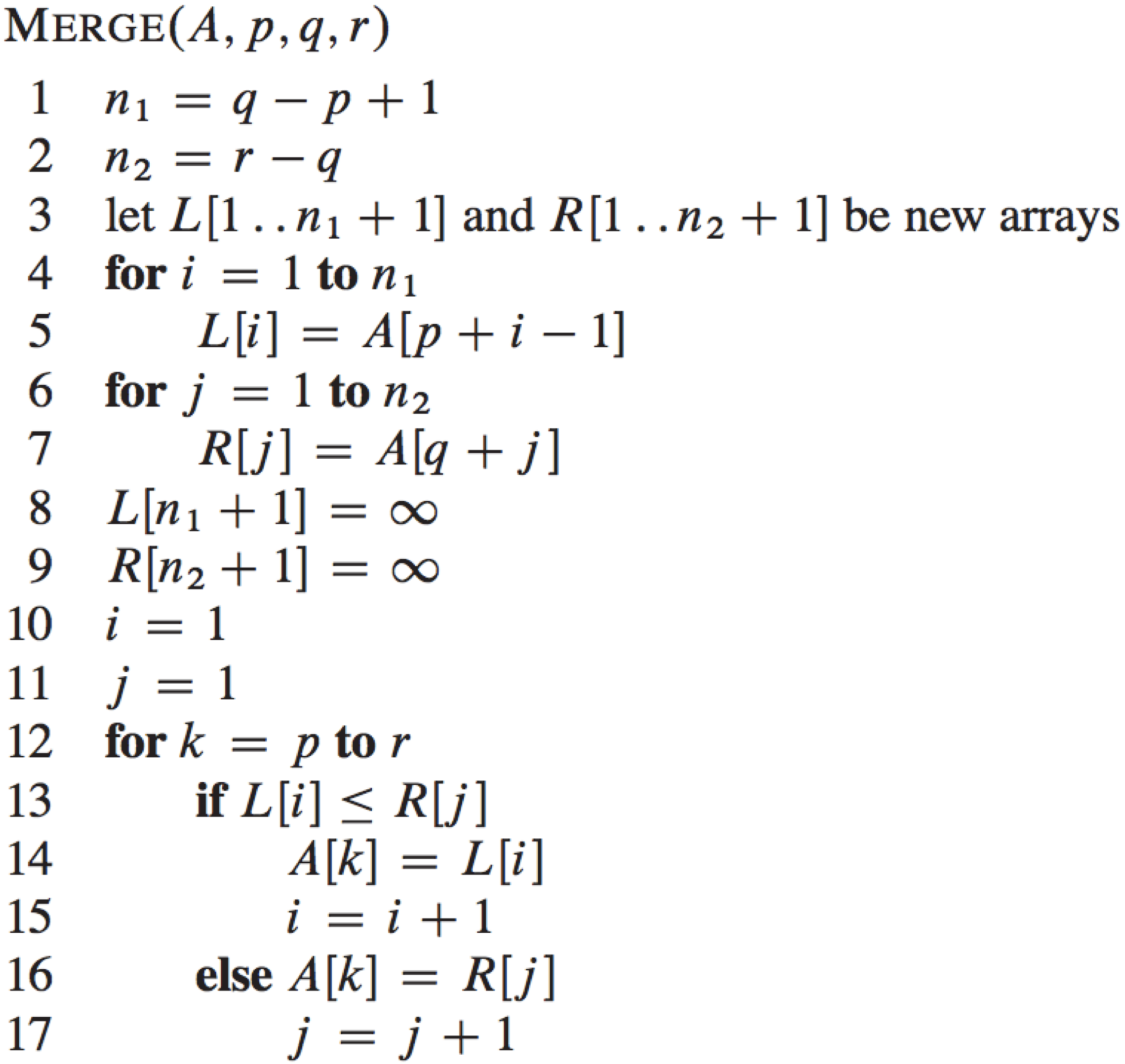


# Merging two sorted arrays

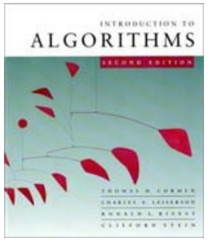


Time = $\Theta(n)$ to merge a total of $n$ elements (linear time).

# Merge



```
MERGE($A, p, q, r$)
$n_1 = q - p + 1$
$n_2 = r - q$
let $L[1 \mathinner{..} n_1 + 1]$ and $R[1 \mathinner{..} n_2 + 1]$ be new arrays
for $i = 1$ to $n_1$
    $L[i] = A[p + i - 1]$
for $j = 1$ to $n_2$
    $R[j] = A[q + j]$
$L[n_1 + 1] = \infty$
$R[n_2 + 1] = \infty$
$i = 1$
$j = 1$
for $k = p$ to $r$
    if $L[i] \leq R[j]$
        $A[k] = L[i]$
        $i = i + 1$
    else $A[k] = R[j]$
        $j = j + 1$
```

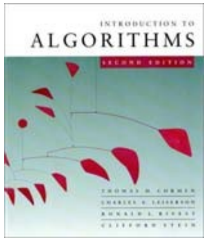


# Analyzing merge sort

| | |
|---|---|
| $T(n)$ | **MERGE-SORT** $A[1\,..\,n]$ |
| $\Theta(1)$ | 1. If $n = 1$, done. |
| $2T(n/2)$ | 2. Recursively sort $A[\,1\,..\,\lceil n/2\rceil]$ and $A[\,\lceil n/2\rceil+1\,..\,n\,]$. |
| $\Theta(n)$ | 3. ***"Merge"*** the 2 sorted lists |

***Abuse***

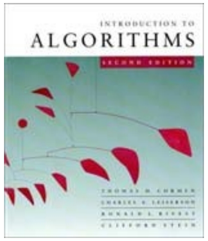
***Sloppiness:*** Should be $T(\lceil n/2\rceil) + T(\lfloor n/2\rfloor)$, but it turns out not to matter asymptotically.



# Recurrence for merge sort

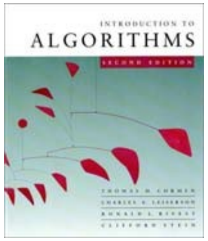
$$T(n) = \begin{cases} \Theta(1) \text{ if } n = 1; \\ 2T(n/2) + \Theta(n) \text{ if } n > 1. \end{cases}$$

- We shall usually omit stating the base case when $T(n) = \Theta(1)$ for sufficiently small $n$, but only when it has no effect on the asymptotic solution to the recurrence.
- CLRS and Lecture 2 provide several ways to find a good upper bound on $T(n)$.



# Recursion tree

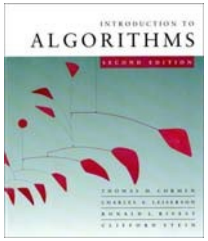
Solve $T(n) = 2T(n/2) + cn$, where $c > 0$ is constant.



# Recursion tree

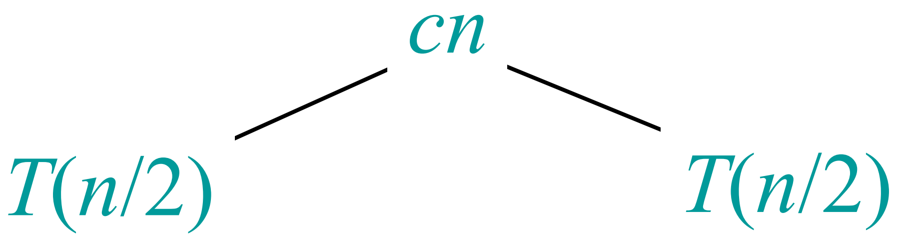
Solve $T(n) = 2T(n/2) + cn$, where $c > 0$ is constant.

$$T(n)$$



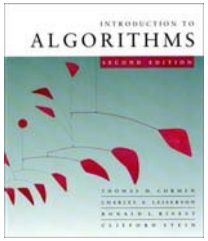
# Recursion tree

Solve $T(n) = 2T(n/2) + cn$, where $c > 0$ is constant.



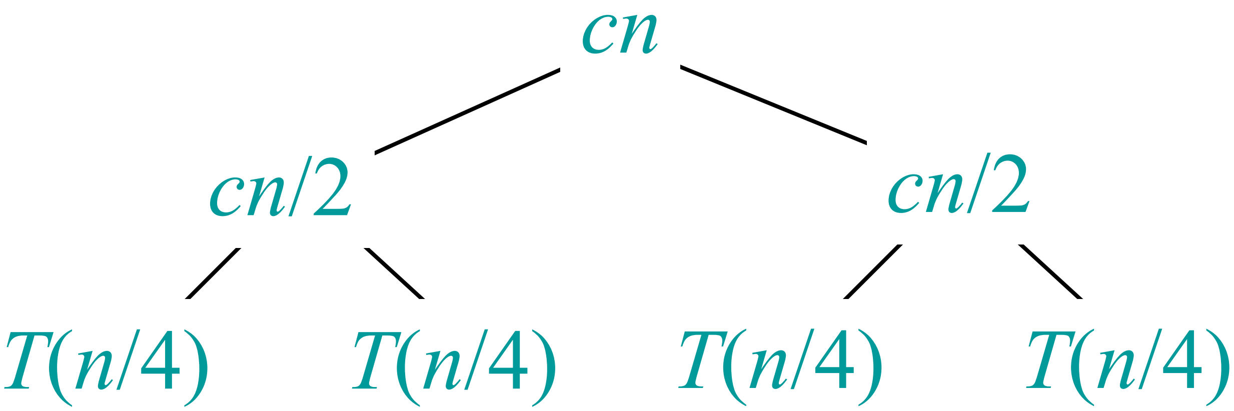
$cn$

$T(n/2)$ $T(n/2)$



# Recursion tree

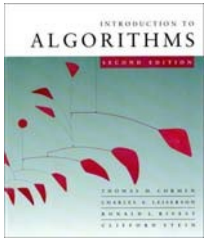
Solve $T(n) = 2T(n/2) + cn$, where $c > 0$ is constant.



$cn$

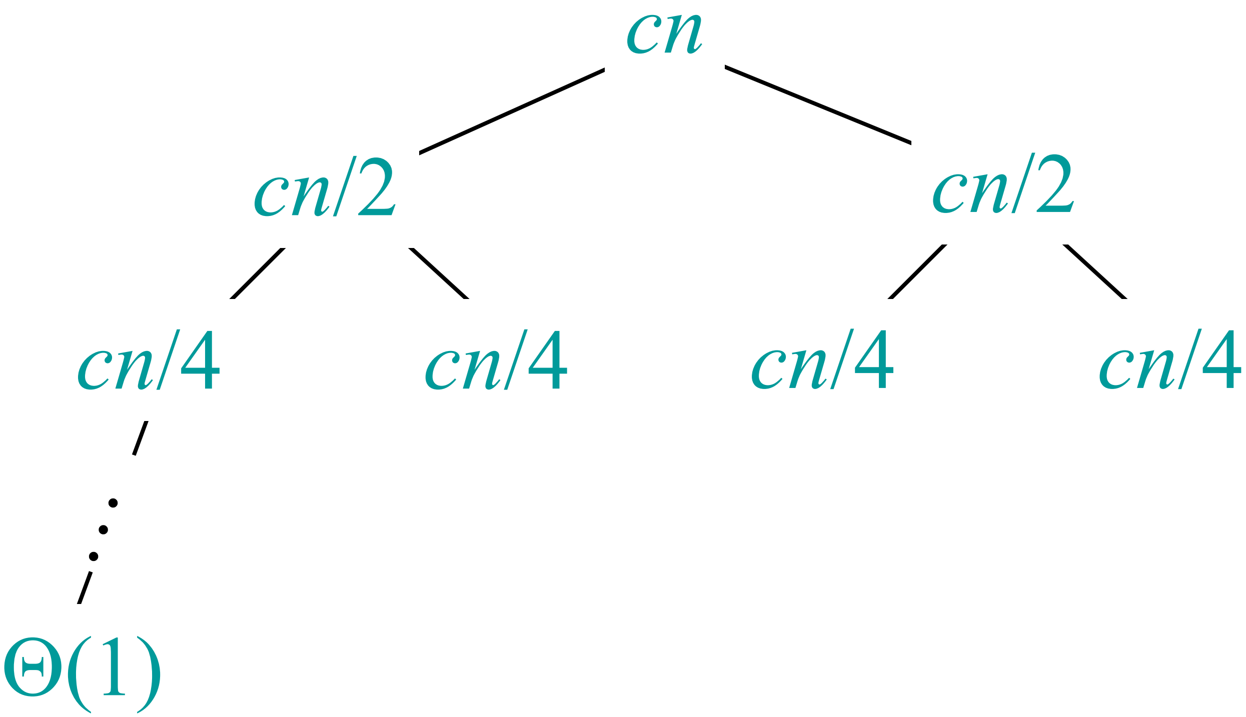
$cn/2$ $cn/2$

$T(n/4)$ $T(n/4)$ $T(n/4)$ $T(n/4)$



# Recursion tree

Solve $T(n) = 2T(n/2) + cn$, where $c > 0$ is constant.

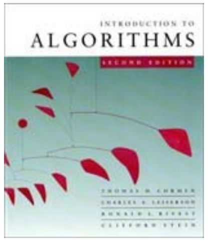


$cn$

$cn/2$ $cn/2$

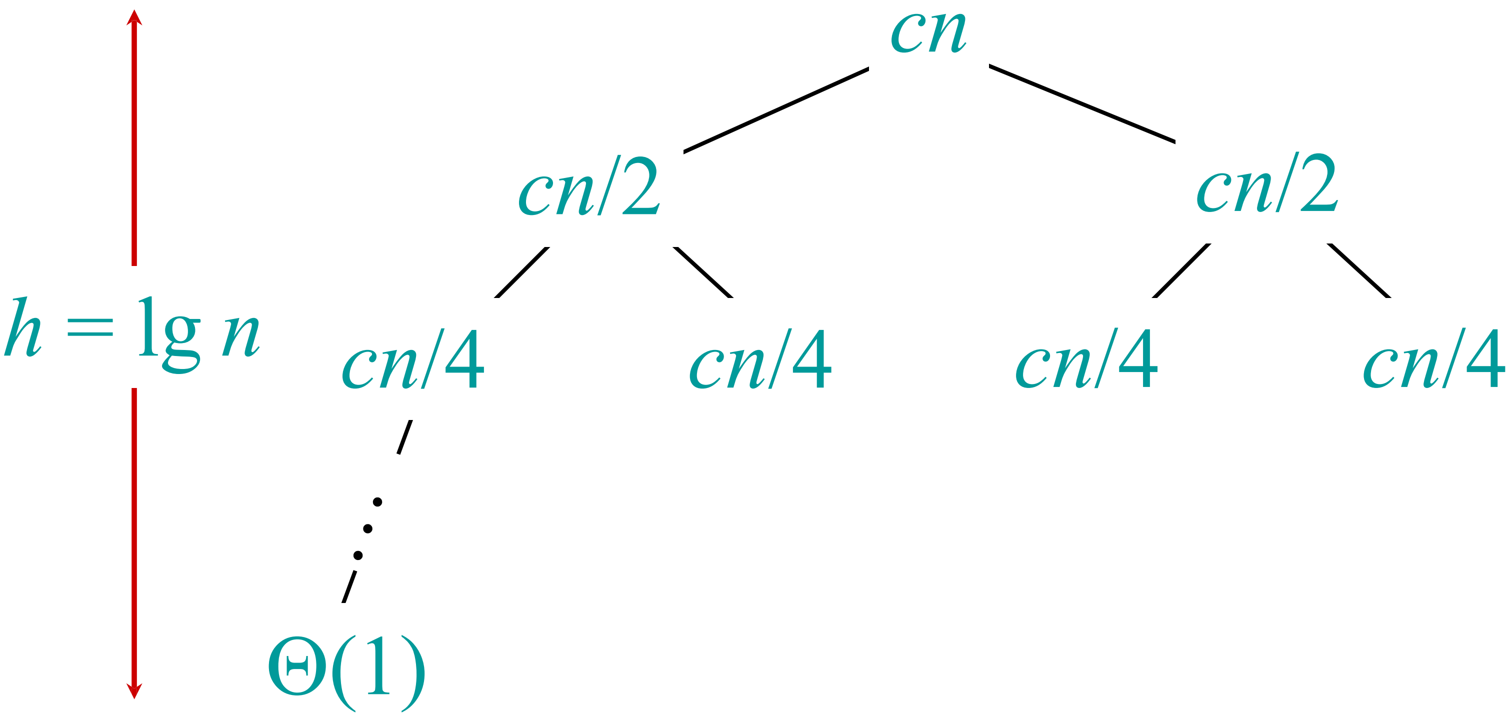
$cn/4$ $cn/4$ $cn/4$ $cn/4$

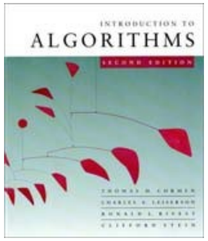
$\Theta(1)$



# Recursion tree

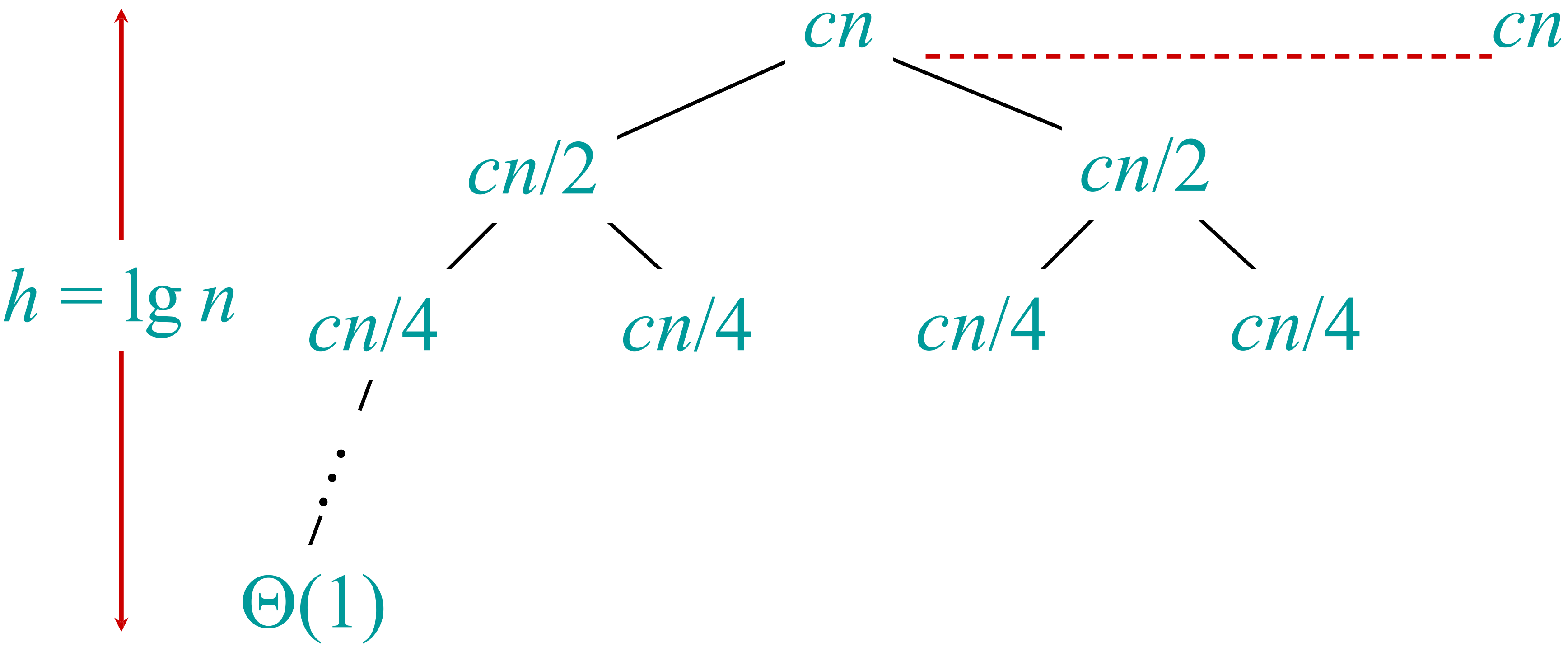
Solve $T(n) = 2T(n/2) + cn$, where $c > 0$ is constant.

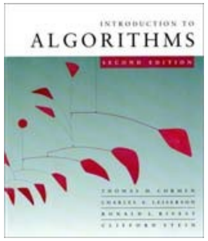




# Recursion tree

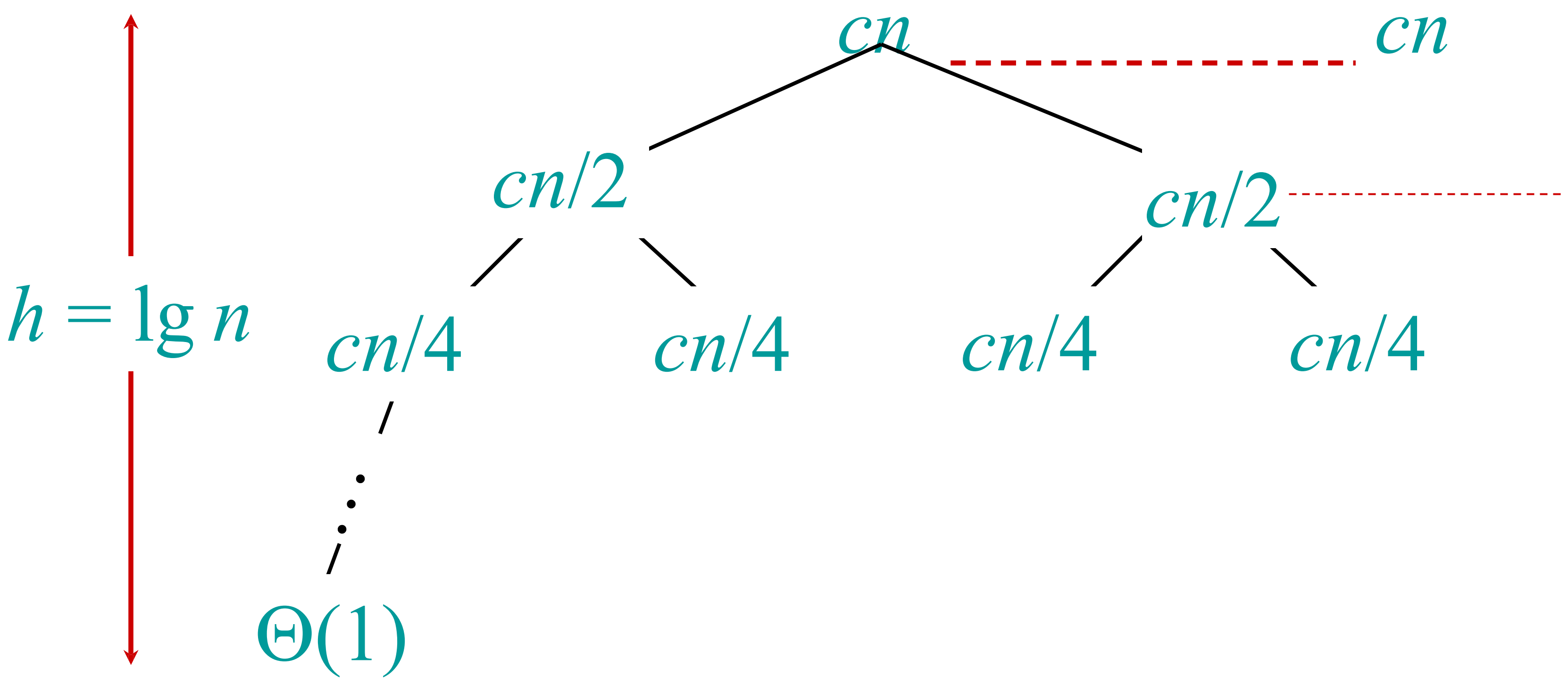
Solve $T(n) = 2T(n/2) + cn$, where $c > 0$ is constant.





# Recursion tree

Solve $T(n) = 2T(n/2) + cn$, where $c > 0$ is constant.



$cn$

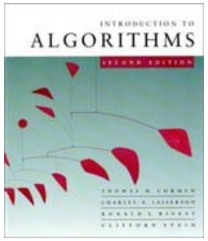
$cn/2$ $cn/2$

$cn/4$ $cn/4$ $cn/4$ $cn/4$

$\Theta(1)$

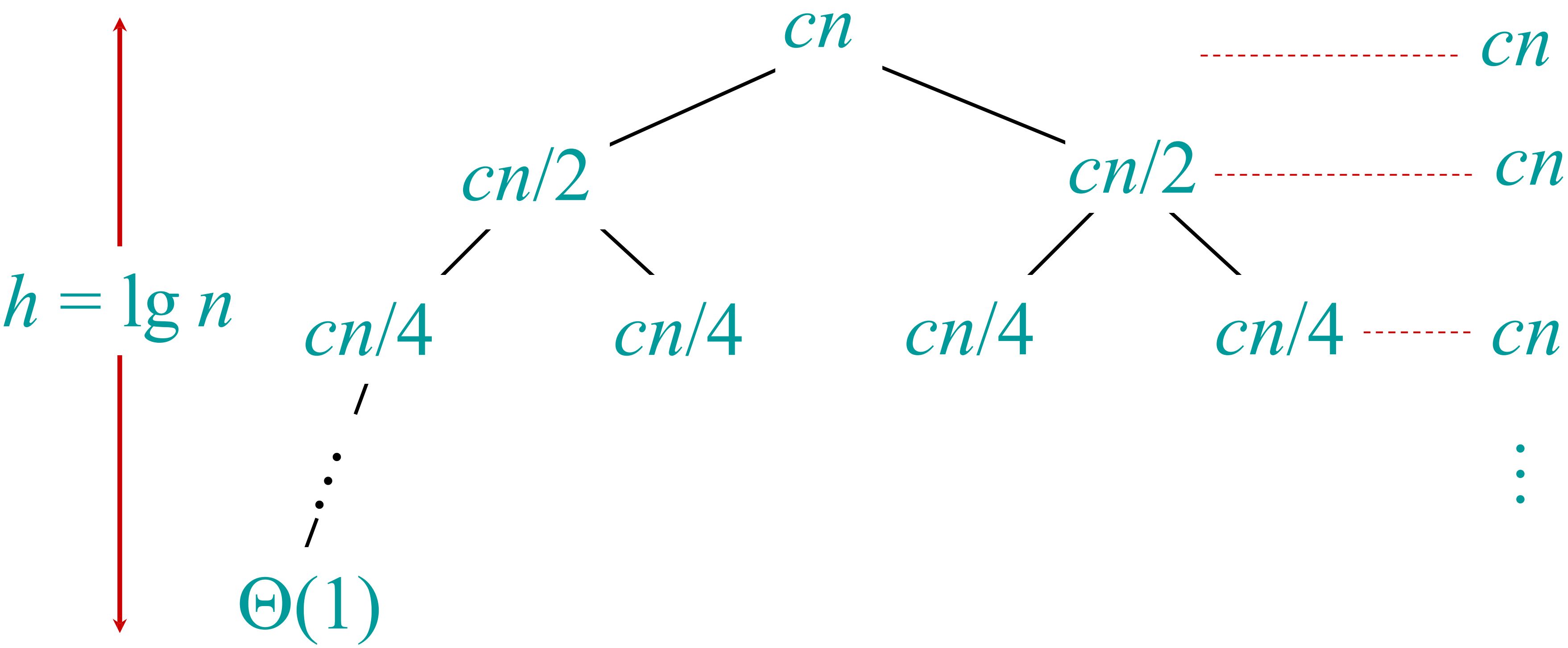
$h = \lg n$

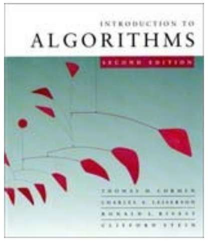
$cn$



# Recursion tree

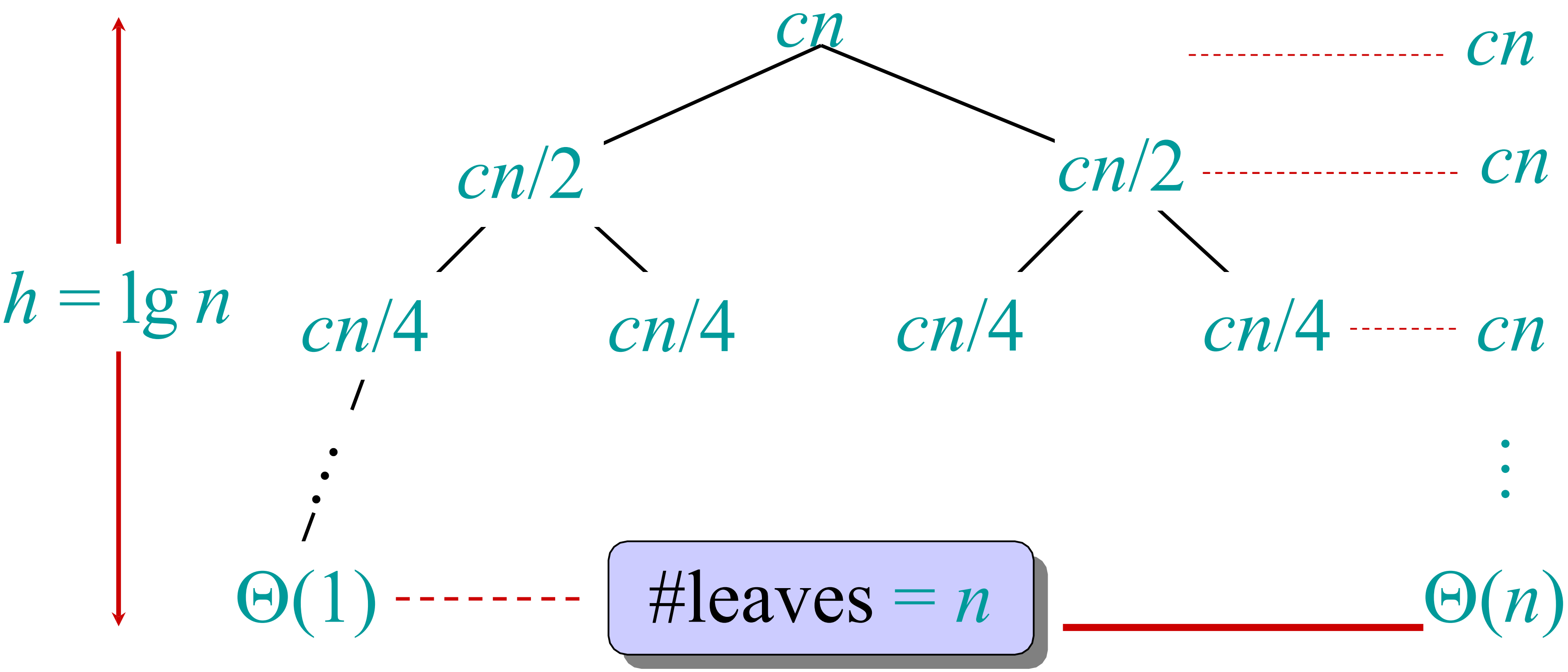
Solve $T(n) = 2T(n/2) + cn$, where $c > 0$ is constant.



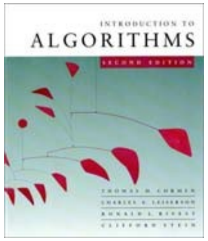


# Recursion tree

Solve $T(n) = 2T(n/2) + cn$, where $c > 0$ is constant.

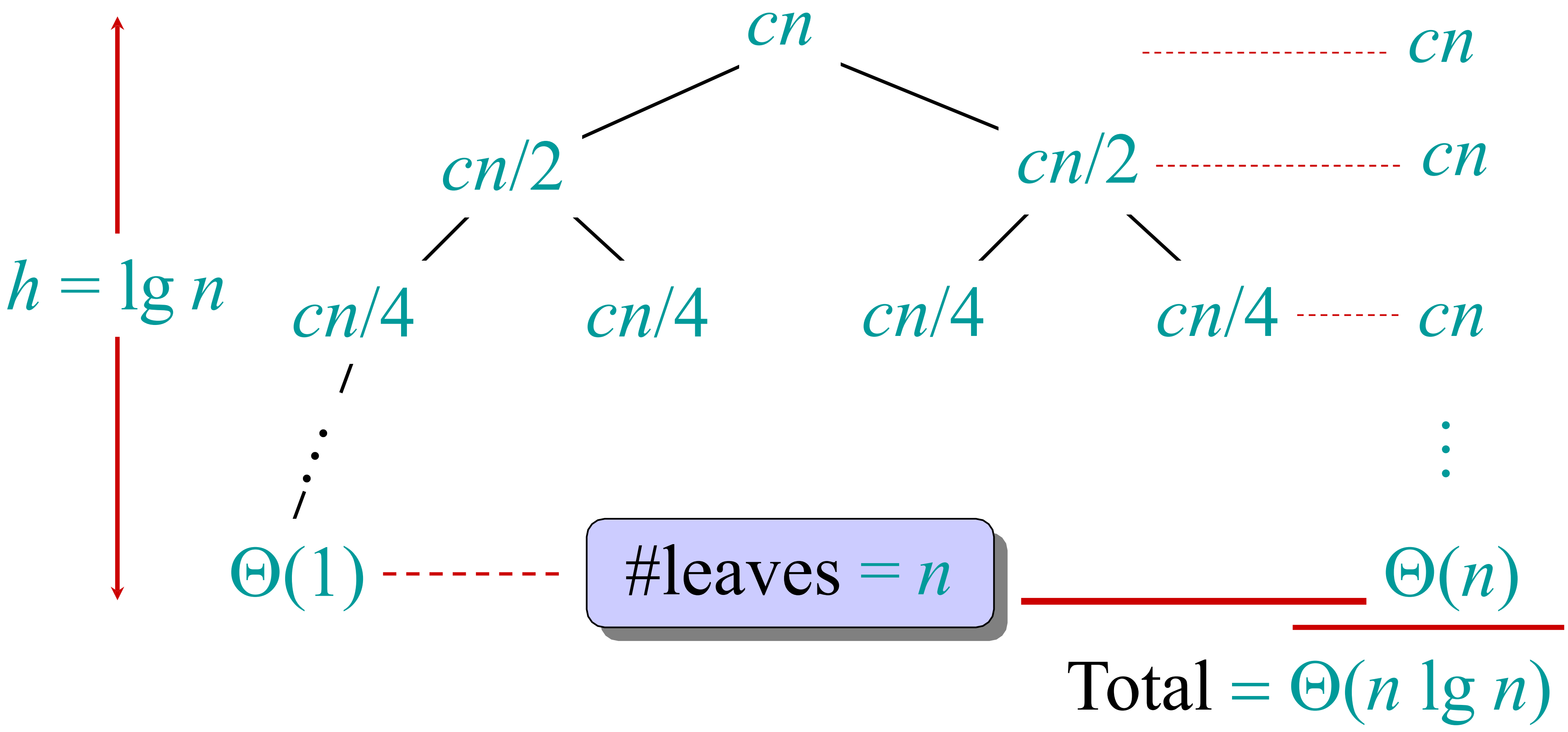


Copyright © 2001-5 Erik D. Demaine and Charles E. Leiserson



# Recursion tree

Solve $T(n) = 2T(n/2) + cn$, where $c > 0$ is constant.



$cn$

$cn/2$ $cn/2$

$cn/4$ $cn/4$ $cn/4$ $cn/4$

$\Theta(1)$

$h = \lg n$

$cn$

$cn$

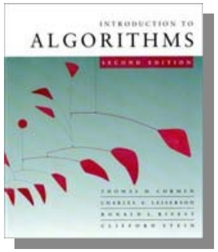
$cn$

$\vdots$

#leaves $= n$

$\Theta(n)$

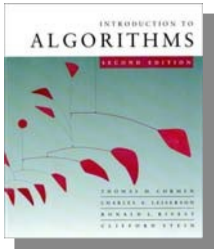
Total $= \Theta(n \lg n)$



# Merge sort

1. ***Divide:*** Trivial.
2. ***Conquer:*** Recursively sort 2 subarrays.
3. ***Combine:*** Linear-time merge.

$$T(n) = \underbrace{2}_{\text{\# subproblems}}\, T(\underbrace{n/2}_{\text{subproblem size}}) + \underbrace{\Theta(n)}_{\text{work dividing and combining}}$$



# Master theorem

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

**CASE 1:** $f(n) = O(n^{\log_b a - \varepsilon})$, constant $\varepsilon > 0$

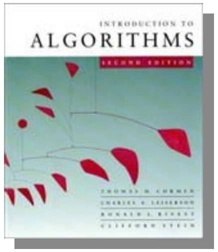
$\Rightarrow T(n) = \Theta(n^{\log_b a})$ .

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

$\Rightarrow T(n) = \Theta(n^{\log_b a} \lg n)$ .

**CASE 3:** $f(n) = \Omega(n^{\log_b a + \varepsilon})$, constant $\varepsilon > 0$, and regularity condition

$\Rightarrow T(n) = \Theta(f(n))$ .



# Master theorem

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

**CASE 1:** $f(n) = O(n^{\log_b a - \varepsilon})$, constant $\varepsilon > 0$

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

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

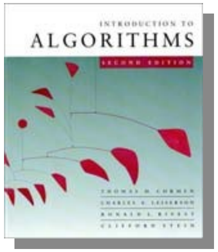
$\Rightarrow T(n) = \Theta(n^{\log_b a} \lg n)$ .

**CASE 3:** $f(n) = \Omega(n^{\log_b a + \varepsilon})$, constant $\varepsilon > 0$, and regularity condition

$\Rightarrow T(n) = \Theta(f(n))$ .

***Merge sort:*** $a = 2$, $b = 2$ $\Rightarrow$ $n^{\log_b a} = n^{\log_2 2} = n$

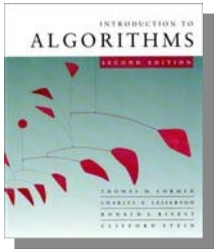
$\Rightarrow$ CASE 2 $\Rightarrow$ $T(n) = \Theta(n \lg n)$ .



# Binary search

- Find an element in a sorted array:

1. ***Divide:*** Check middle element.
2. ***Conquer:*** Recursively search 1 subarray.
3. ***Combine:*** Trivial.



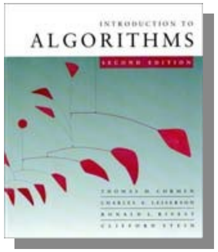
# Binary search

- Find an element in a sorted array:

1. ***Divide:*** Check middle element.
2. ***Conquer:*** Recursively search 1 subarray.
3. ***Combine:*** Trivial.

- ***Example:*** Find 9

| | | | | | | |
|---|---|---|---|---|---|---|
| 3 | 5 | 7 | 8 | 9 | 12 | 15 |

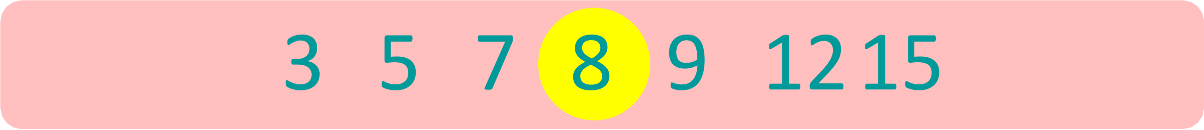


# Binary search

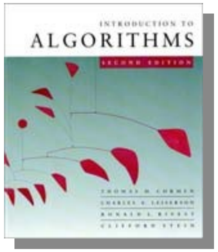
- Find an element in a sorted array:

1. ***Divide:*** Check middle element.
2. ***Conquer:*** Recursively search 1 subarray.
3. ***Combine:*** Trivial.

- ***Example:*** Find 9



3 5 7 8 9 12 15



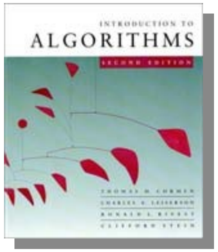
# Binary search

- Find an element in a sorted array:

1. ***Divide:*** Check middle element.
2. ***Conquer:*** Recursively search 1 subarray.
3. ***Combine:*** Trivial.

- ***Example:*** Find 9

3 5 7 8 9 12 15

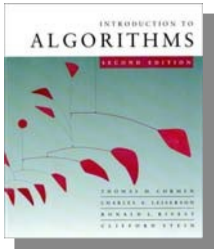


# Binary search

- Find an element in a sorted array:

1. ***Divide:*** Check middle element.
2. ***Conquer:*** Recursively search 1 subarray.
3. ***Combine:*** Trivial.

- ***Example:*** Find 9



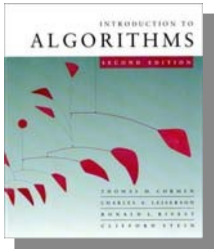
# Binary search

- Find an element in a sorted array:

1. ***Divide:*** Check middle element.
2. ***Conquer:*** Recursively search 1 subarray.
3. ***Combine:*** Trivial.

- ***Example:*** Find 9

3 5 7 8 9 12 15

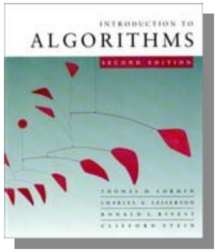


# Binary search

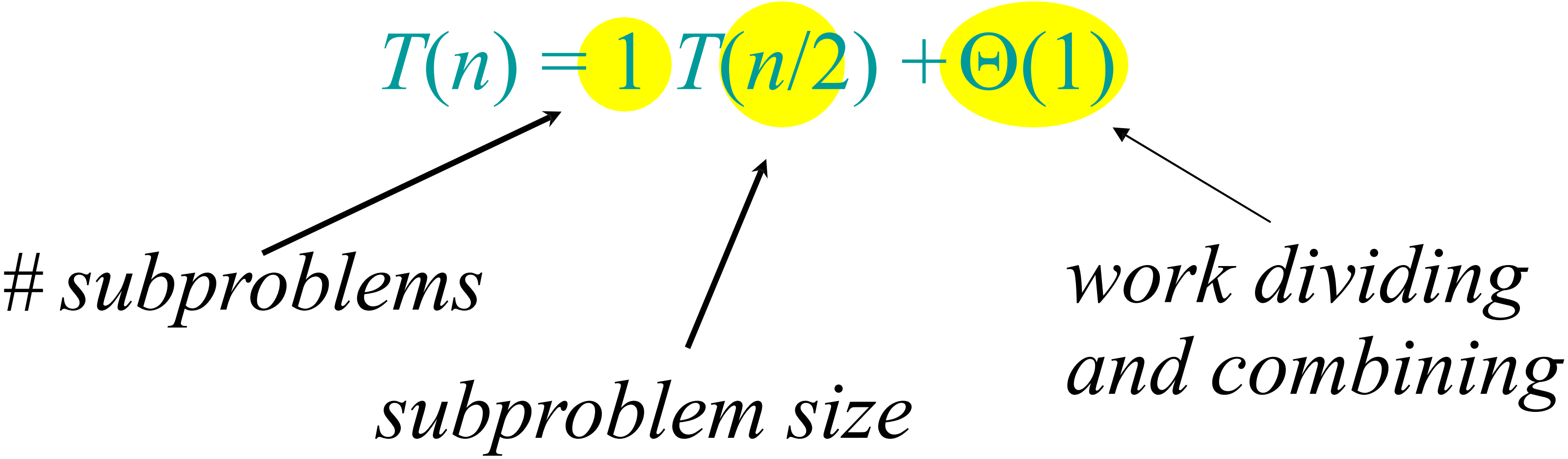
- Find an element in a sorted array:

1. ***Divide:*** Check middle element.
2. ***Conquer:*** Recursively search 1 subarray.
3. ***Combine:*** Trivial.

- ***Example:*** Find 9



# Recurrence for binary search

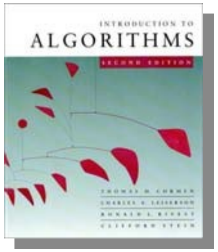


$$T(n) = 1\,T(n/2) + \Theta(1)$$

*# subproblems*

*subproblem size*

*work dividing*
*and combining*



# Recurrence for binary search

$$T(n) = 1\,T(n/2) + \Theta(1)$$

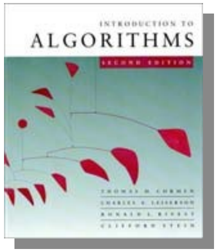
*# subproblems*

*subproblem size*

*work dividing and combining*

$$n^{\log_b a} = n^{\log_2 1} = n^0 = 1 \Rightarrow \quad \text{CASE } 2 \; (k = 0)$$

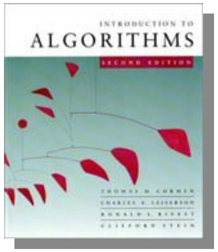
$$\Rightarrow \quad T(n) = \Theta(\lg n)\,.$$



# Powering a number

**Problem:** Compute $a^n$, where $n \in N$.

**Naive algorithm:** $\Theta(n)$.



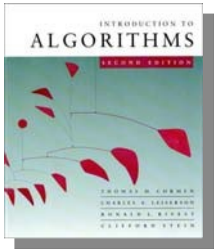
# Powering a number

**Problem:** Compute $a^n$, where $n \in N$.

**Naive algorithm:** $\Theta(n)$.

**Divide-and-conquer algorithm:**

$$a^n = \begin{cases} a^{n/2} \cdot a^{n/2} & \text{if } n \text{ is even;} \\ a^{(n-1)/2} \cdot a^{(n-1)/2} \cdot a & \text{if } n \text{ is odd.} \end{cases}$$



# Powering a number

**Problem:** Compute $a^n$, where $n \in N$.

**Naive algorithm:** $\Theta(n)$.

**Divide-and-conquer algorithm:**

$$a^n = \begin{cases} a^{n/2} \cdot a^{n/2} & \text{if } n \text{ is even;} \\ a^{(n-1)/2} \cdot a^{(n-1)/2} \cdot a & \text{if } n \text{ is odd.} \end{cases}$$

$$T(n) = T(n/2) + \Theta(1) \Rightarrow T(n) = \Theta(\lg n).$$