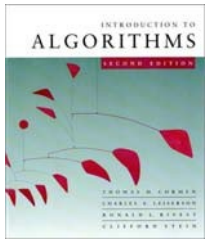
CS60020: Foundations of Algorithm Design and Machine Learning

Sourangshu Bhattacharya

DIVIDE AND CONQUER

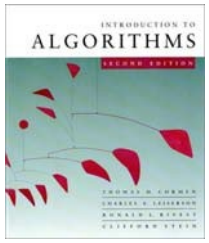


Merge sort

MERGE-SORT $A[1 \ldots n]$

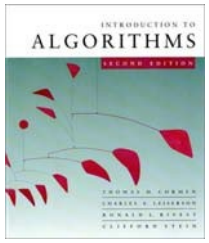
1. If $n = 1$, done.
2. Recursively sort $A[1 \ldots \lceil n/2 \rceil]$ and $A[\lceil n/2 \rceil + 1 \ldots n]$.
3. "***Merge***" the 2 sorted lists.

Key subroutine: **MERGE**

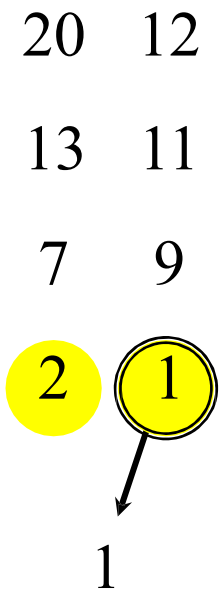


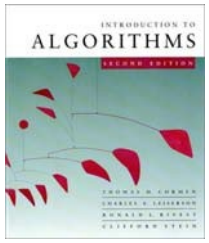
Merging two sorted arrays

20	12
13	11
7	9
2	1

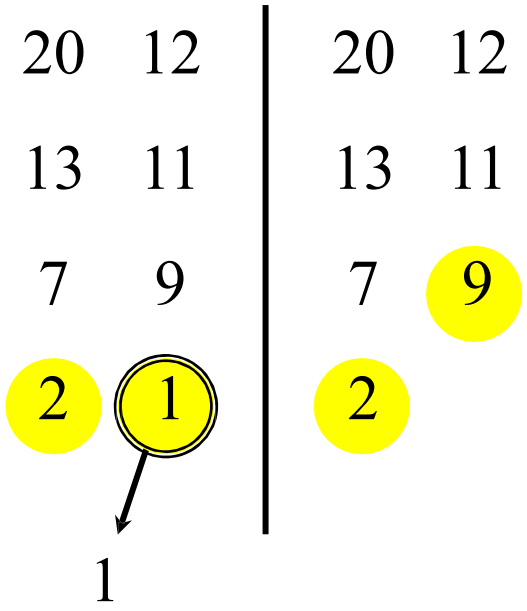


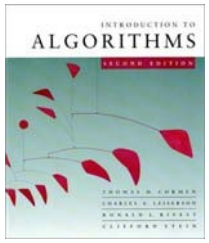
Merging two sorted arrays



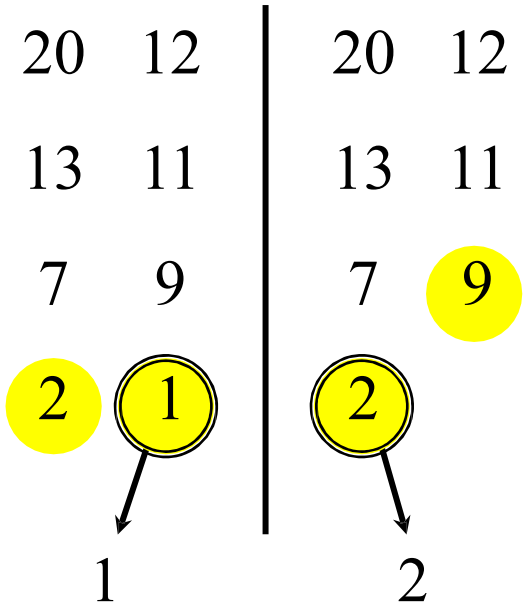


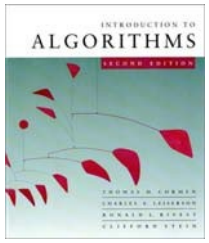
Merging two sorted arrays



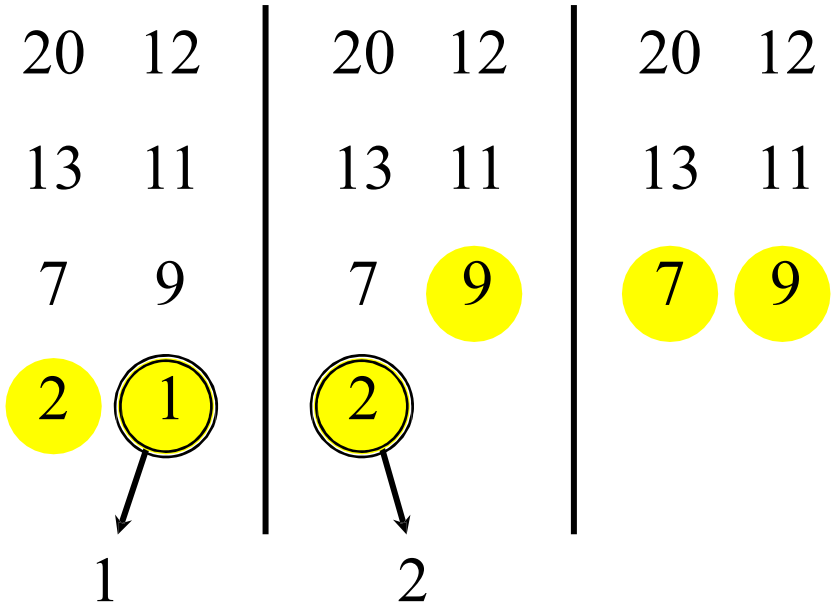


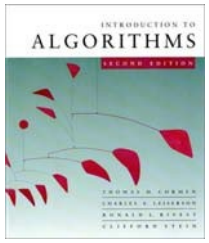
Merging two sorted arrays



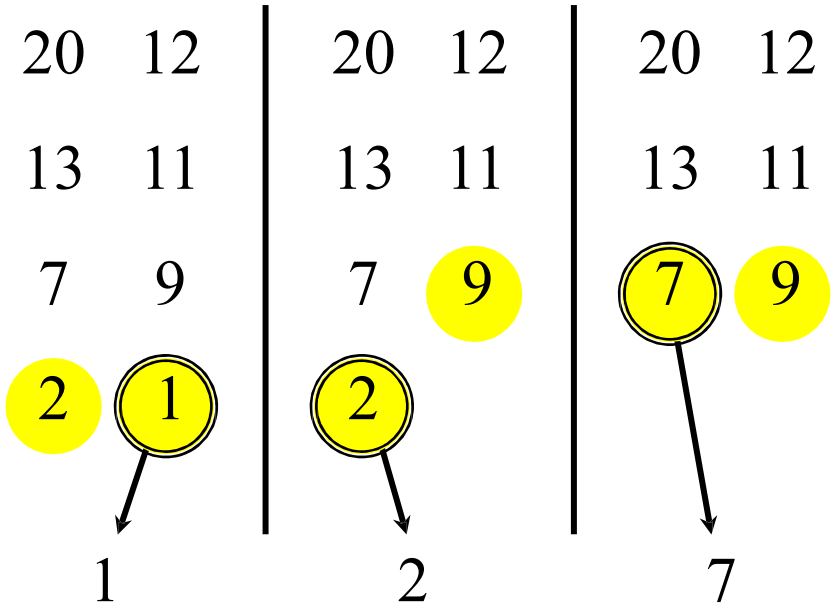


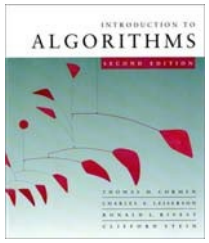
Merging two sorted arrays



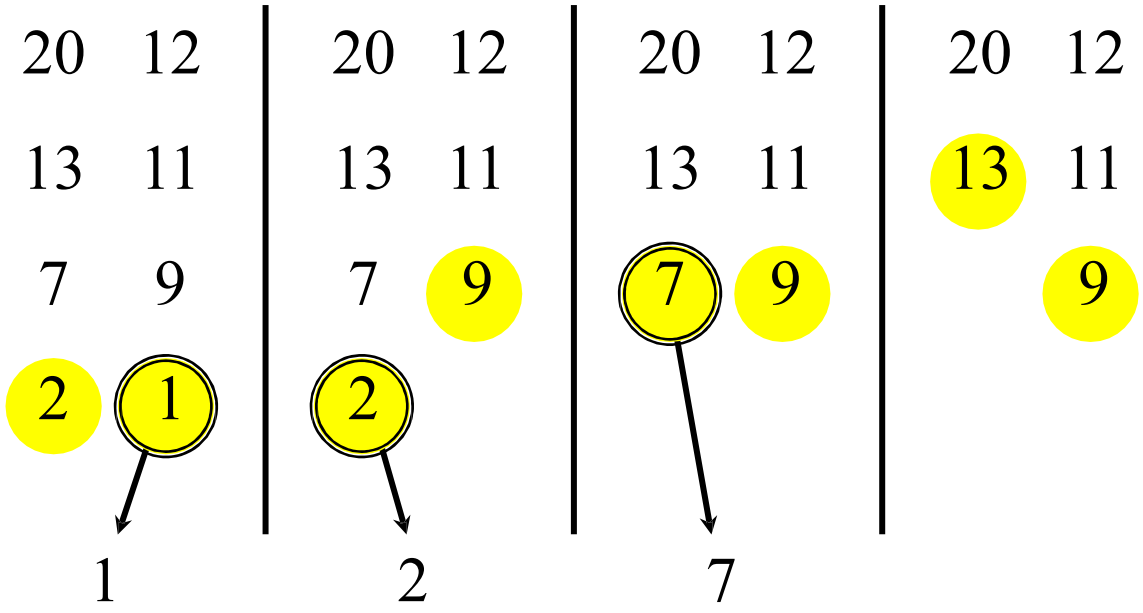


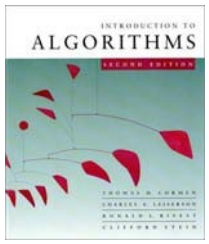
Merging two sorted arrays



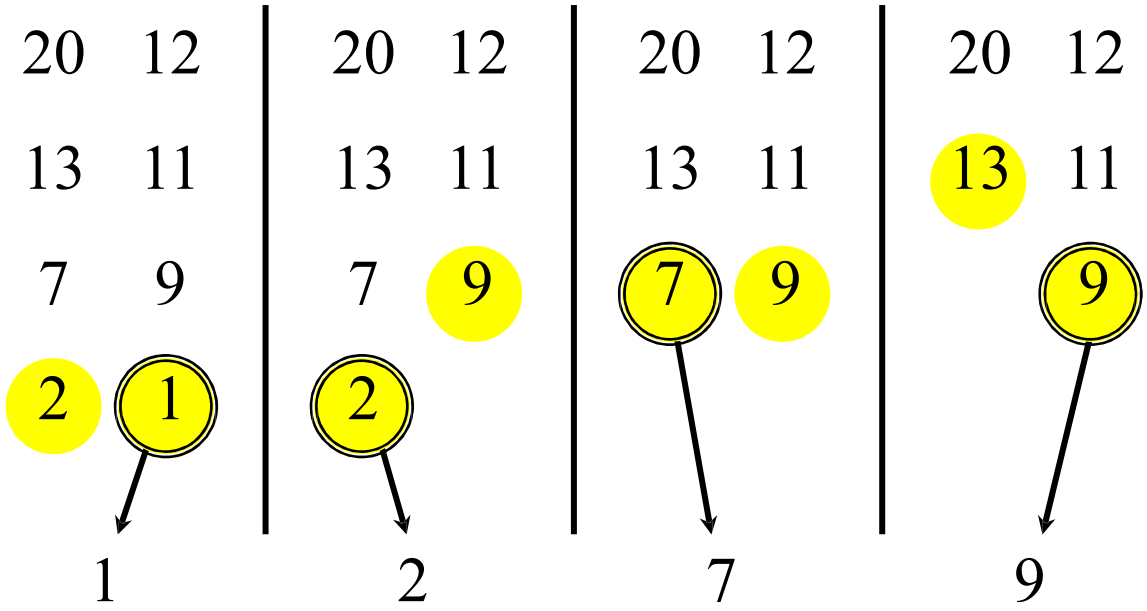


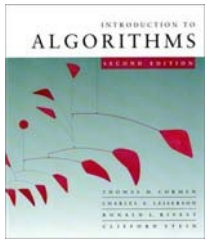
Merging two sorted arrays



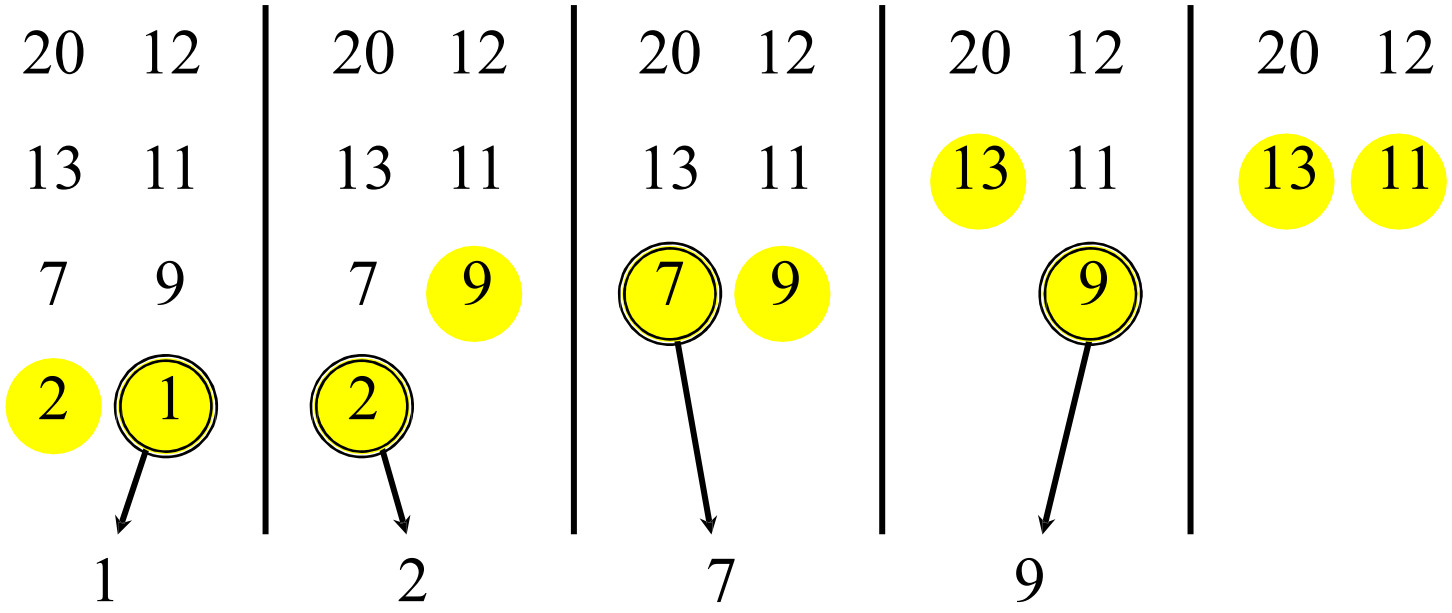


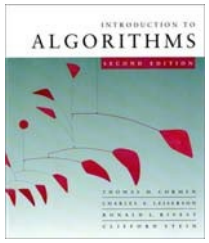
Merging two sorted arrays



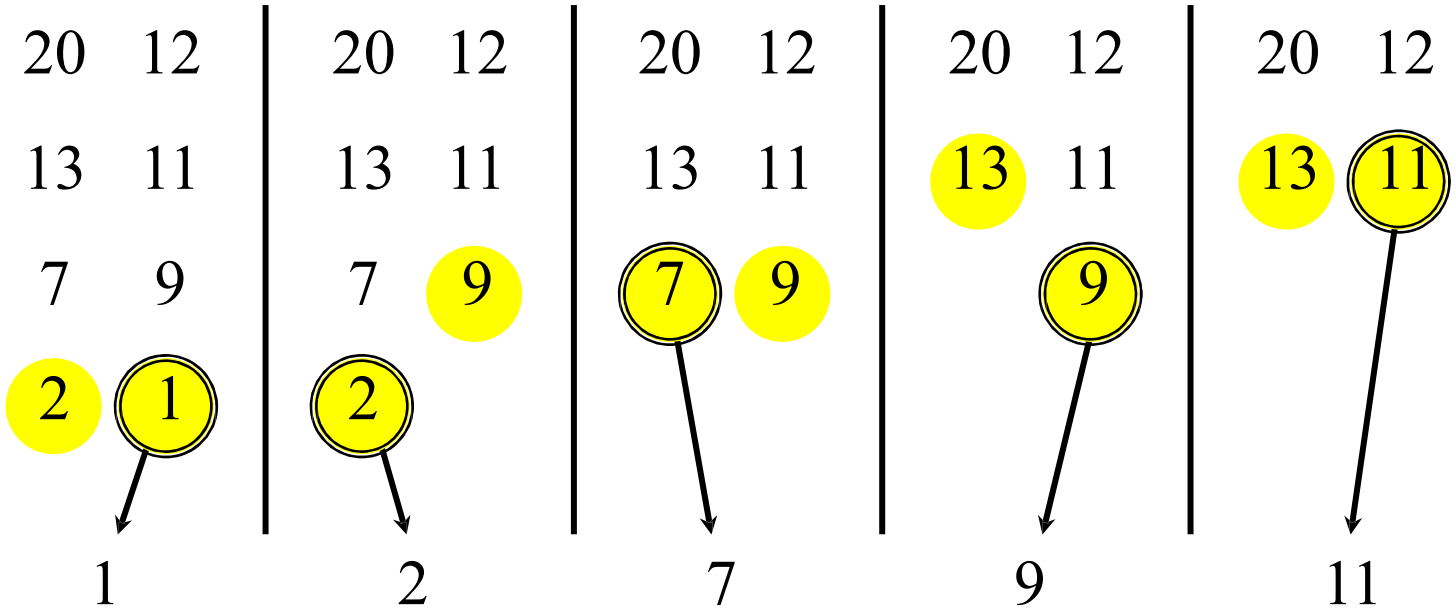


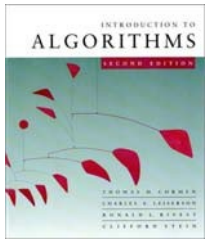
Merging two sorted arrays



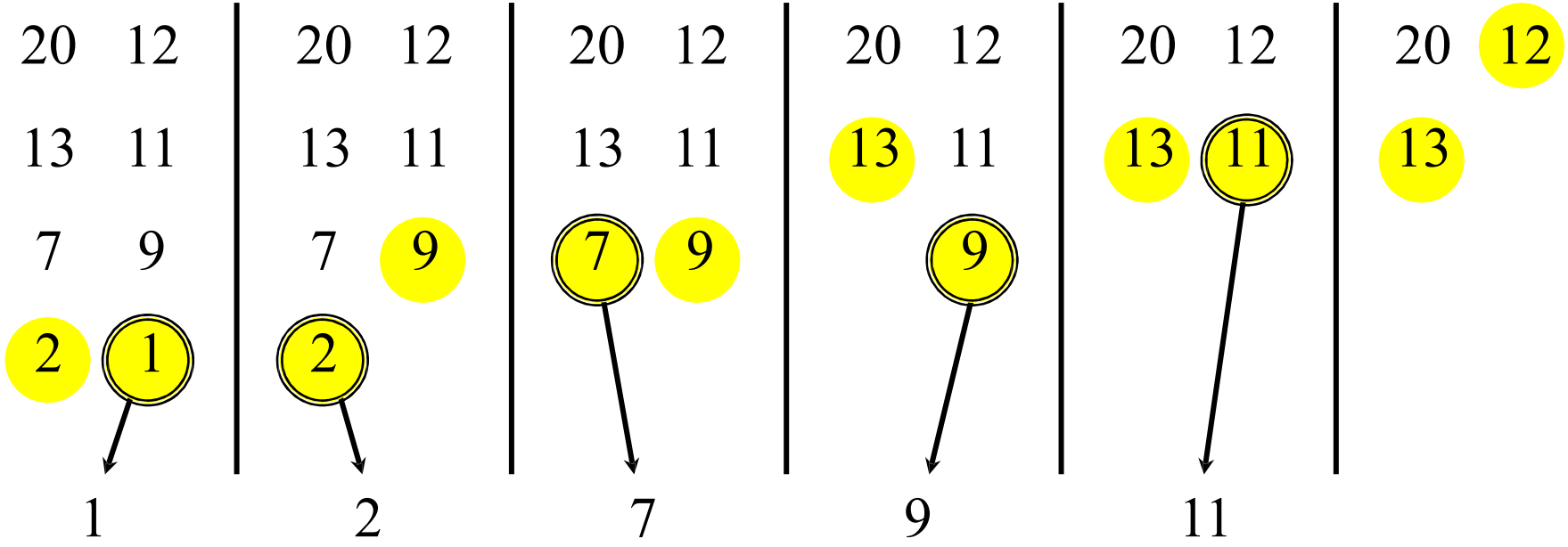


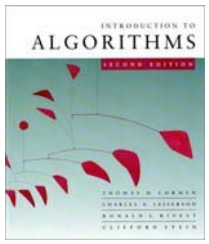
Merging two sorted arrays



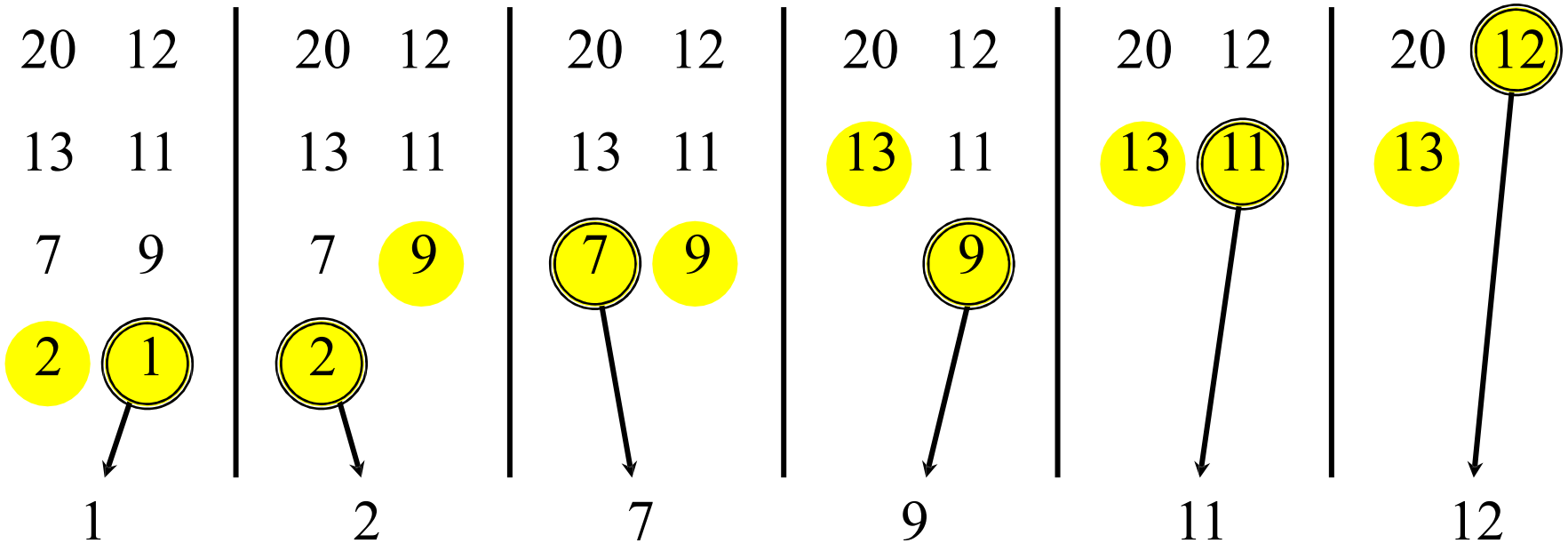


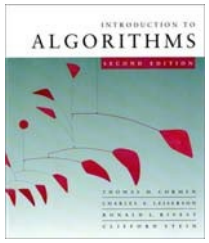
Merging two sorted arrays



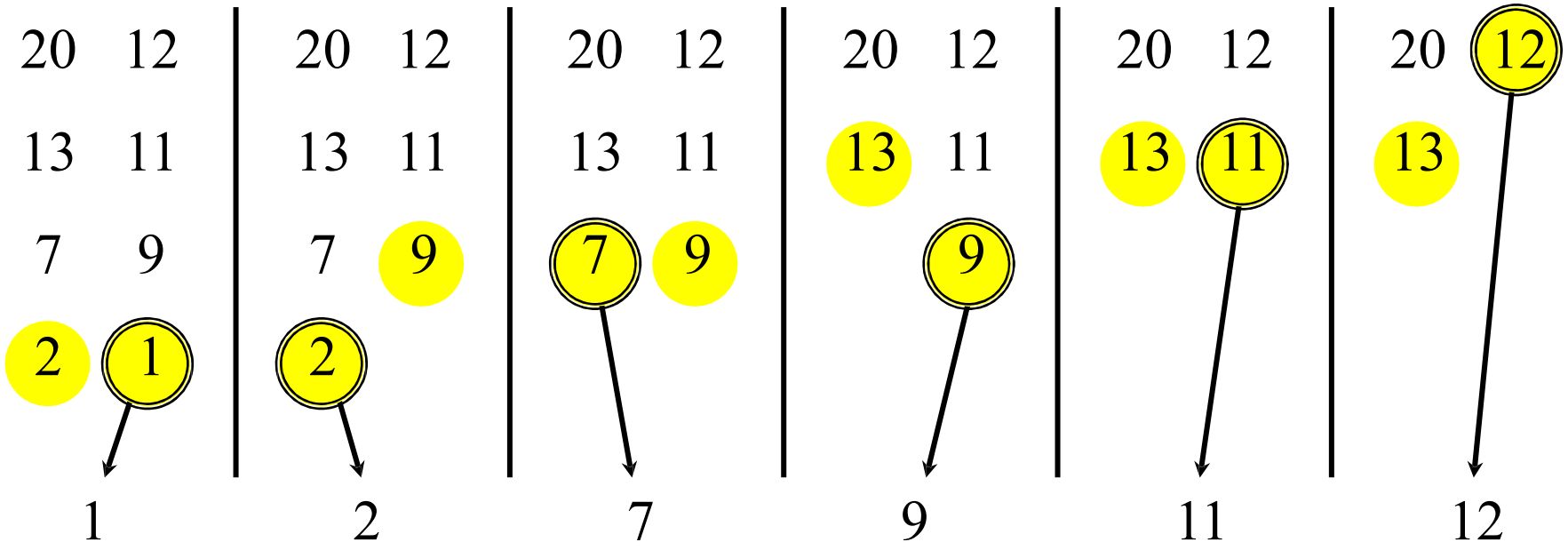


Merging two sorted arrays





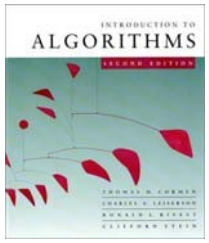
Merging two sorted arrays



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

Merge

```
MERGE(A, p, q, r)
 1  n_1 = q − p + 1
 2  n_2 = r − q
 3  let L[1 .. n_1 + 1] and R[1 .. n_2 + 1] be new arrays
 4  for i = 1 to n_1
 5      L[i] = A[p + i − 1]
 6  for j = 1 to n_2
 7      R[j] = A[q + j]
 8  L[n_1 + 1] = ∞
 9  R[n_2 + 1] = ∞
10  i = 1
11  j = 1
12  for k = p to r
13      if L[i] ≤ R[j]
14          A[k] = L[i]
15          i = i + 1
16      else A[k] = R[j]
17          j = j + 1
```



Analyzing merge sort

$T(n)$

$\Theta(1)$

$2T(n/2)$

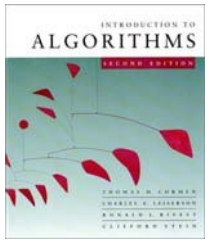
Abuse

$\Theta(n)$

MERGE-SORT $A[1 \ldots n]$

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

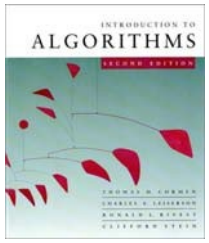
Sloppiness: Should be $T(\lceil n/2 \rceil) + T(\lfloor n/2 \rfloor)$, but it turns out not to matter asymptotically.



Recurrence for merge sort

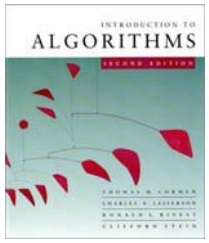
$$T(n) = \begin{cases} \Theta(1) \text{ if } n = 1; \\ 2T(n/2) + \Theta(n) \text{ if } n > 1. \end{cases}$$

- 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

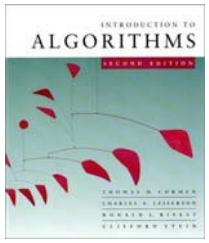
Solve $T(n) = 2T(n/2) + cn$, where $c > 0$ is constant.



Recursion tree

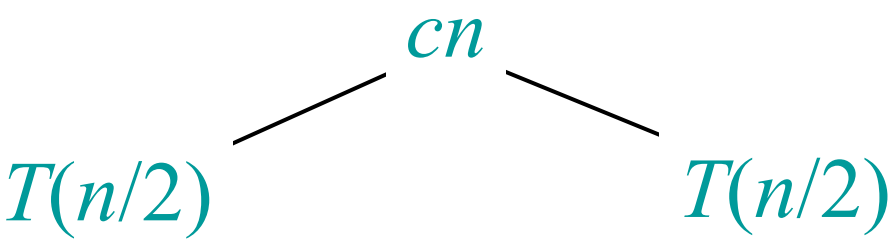
Solve $T(n) = 2T(n/2) + cn$, where $c > 0$ is constant.

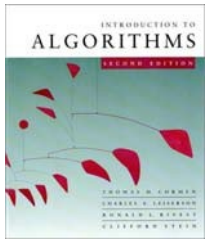
$T(n)$



Recursion tree

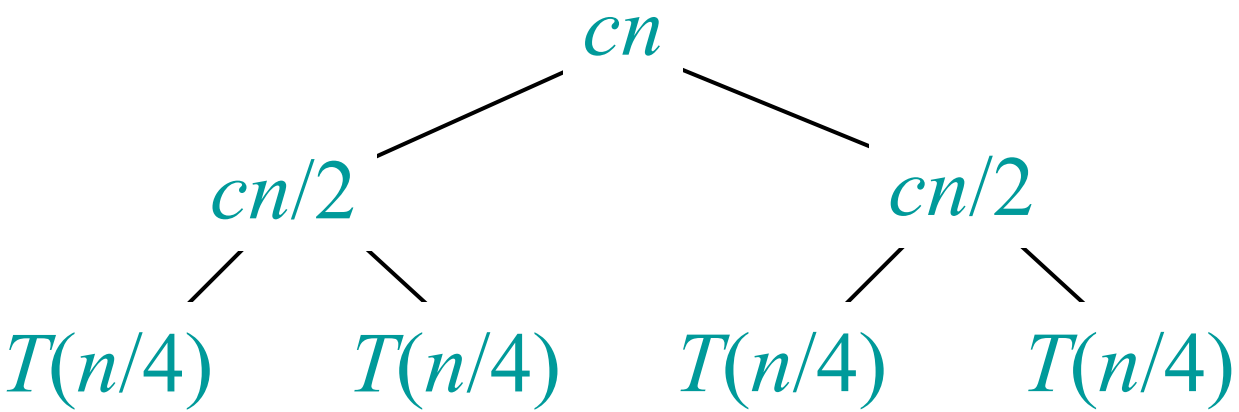
Solve $T(n) = 2T(n/2) + cn$, where $c > 0$ is constant.

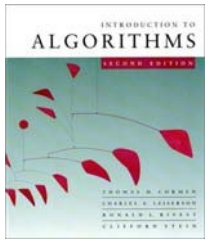




Recursion tree

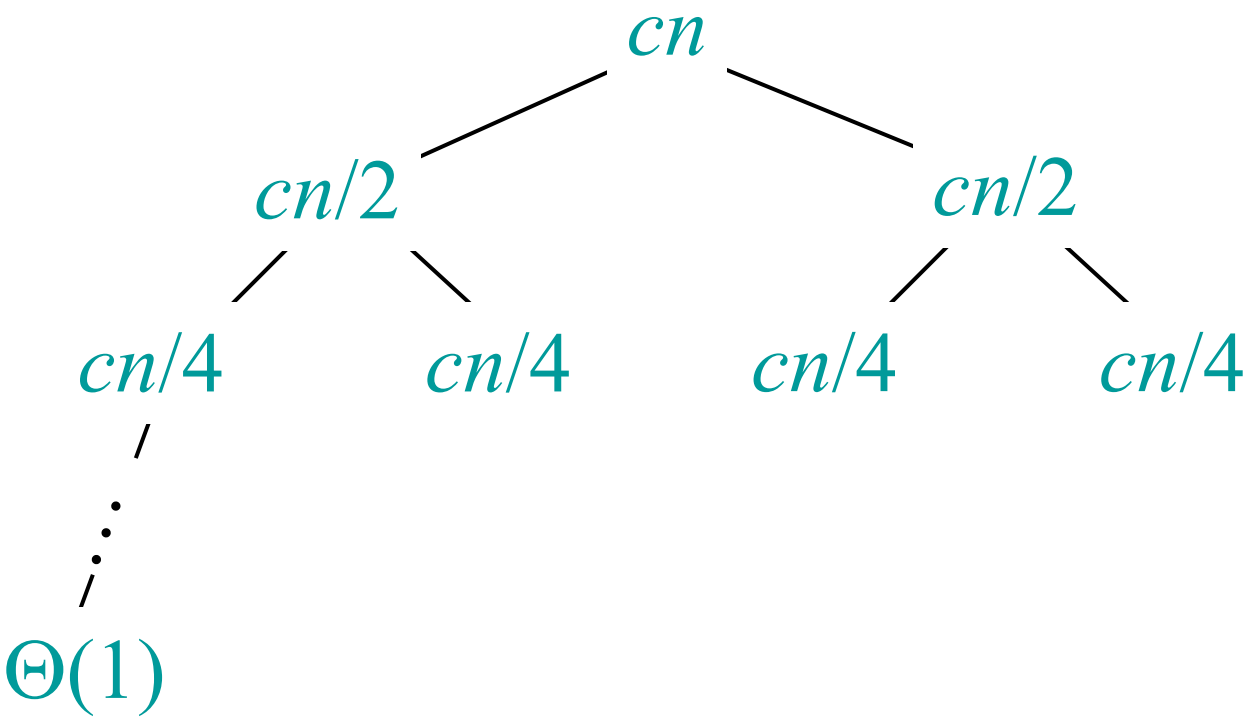
Solve $T(n) = 2T(n/2) + cn$, where $c > 0$ is constant.

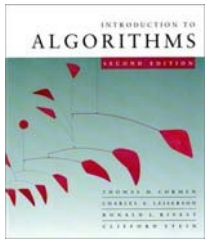




Recursion tree

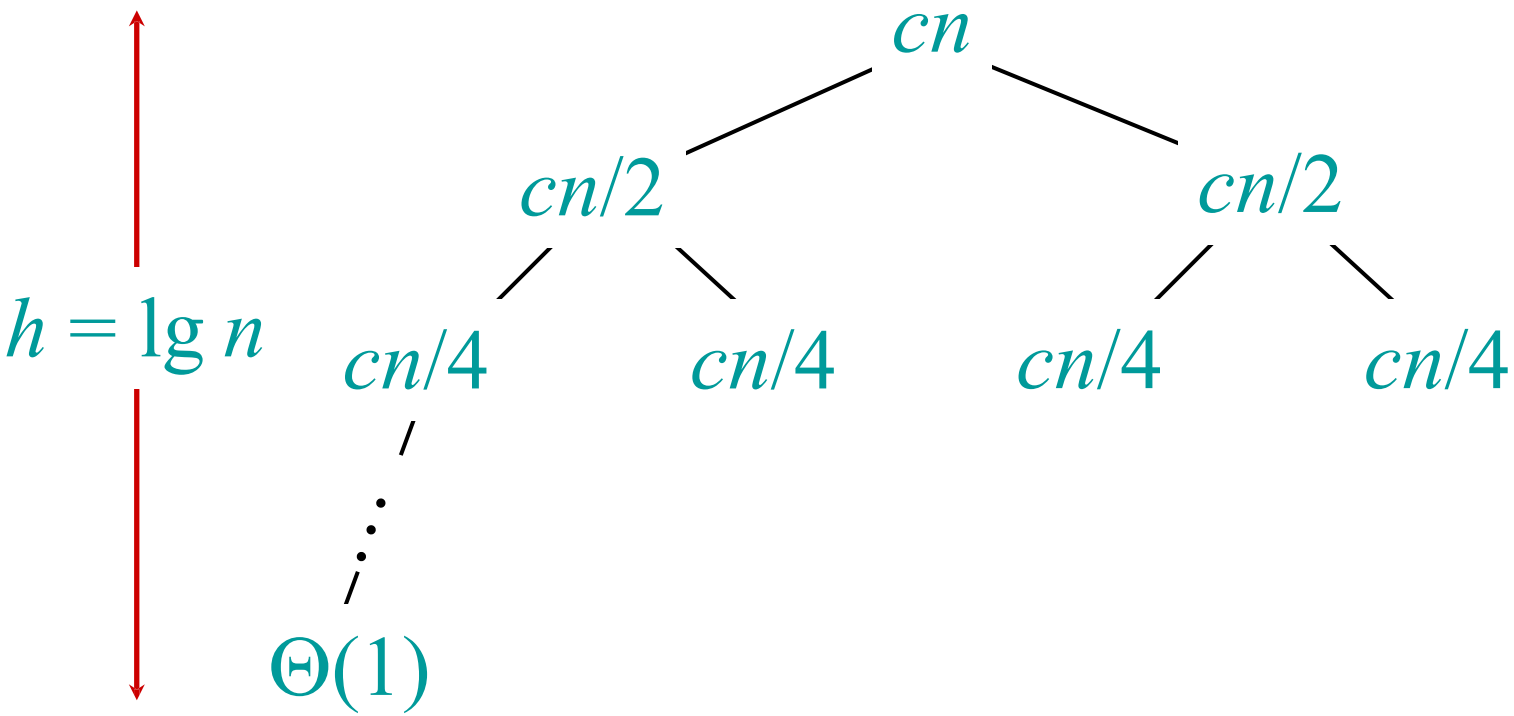
Solve $T(n) = 2T(n/2) + cn$, where $c > 0$ is constant.

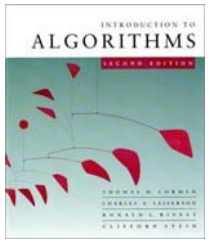




Recursion tree

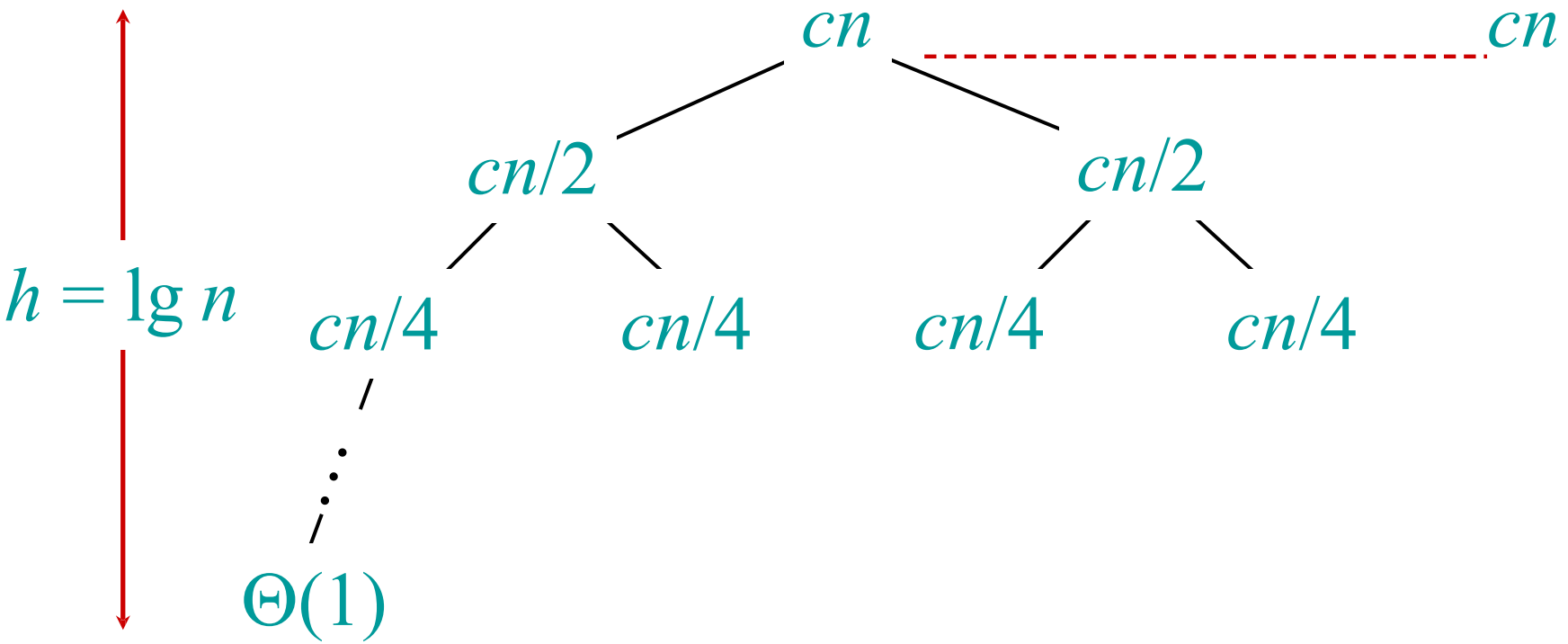
Solve $T(n) = 2T(n/2) + cn$, where $c > 0$ is constant.

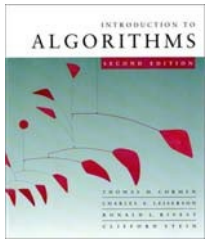




Recursion tree

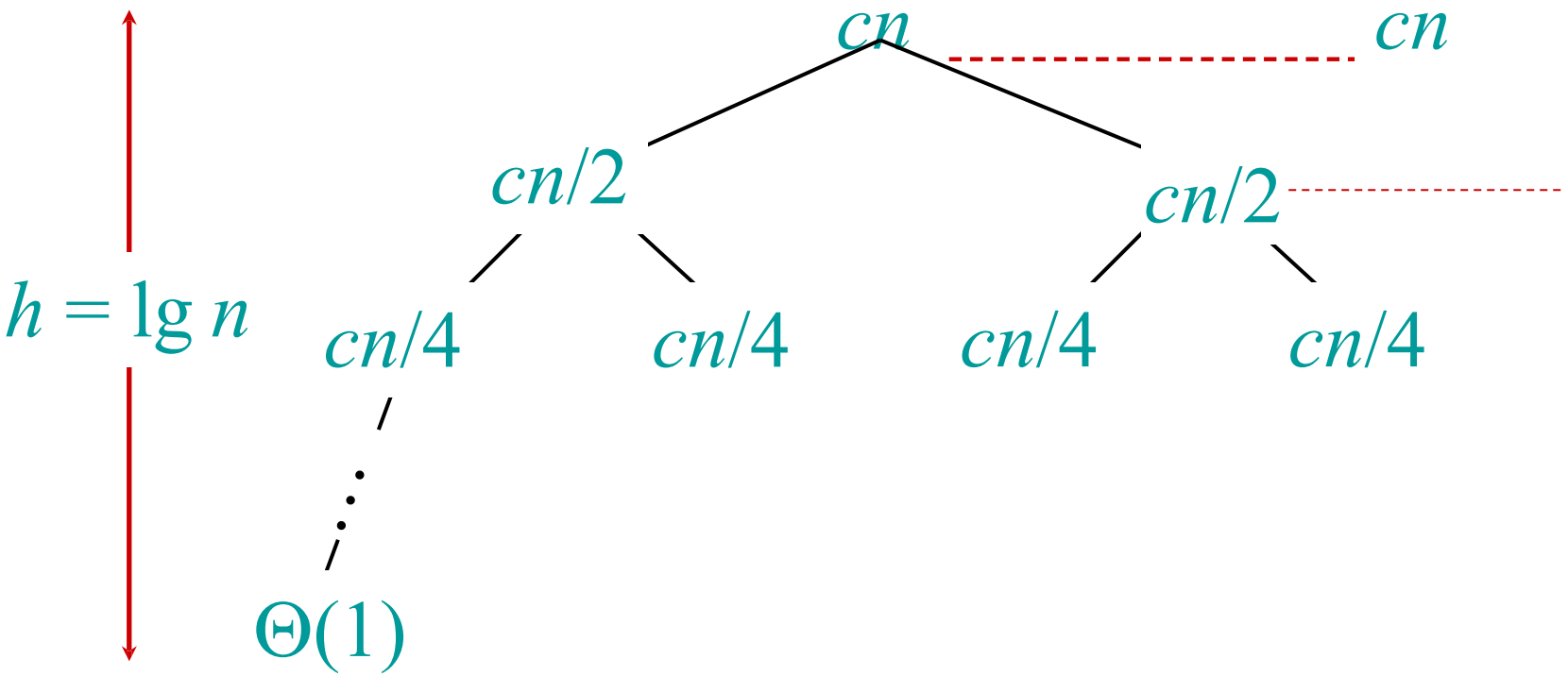
Solve $T(n) = 2T(n/2) + cn$, where $c > 0$ is constant.

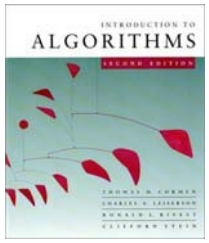




Recursion tree

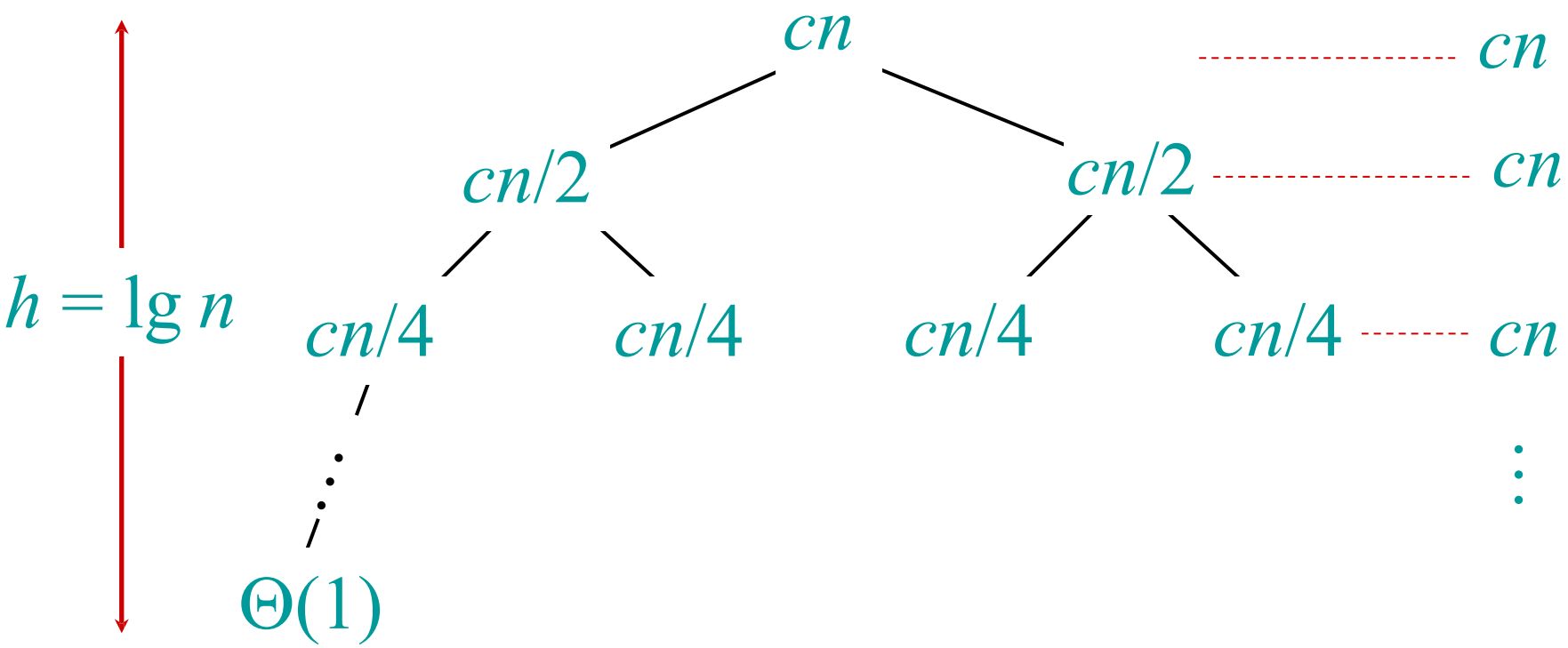
Solve $T(n) = 2T(n/2) + cn$, where $c > 0$ is constant.

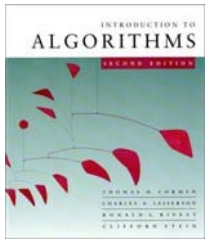




Recursion tree

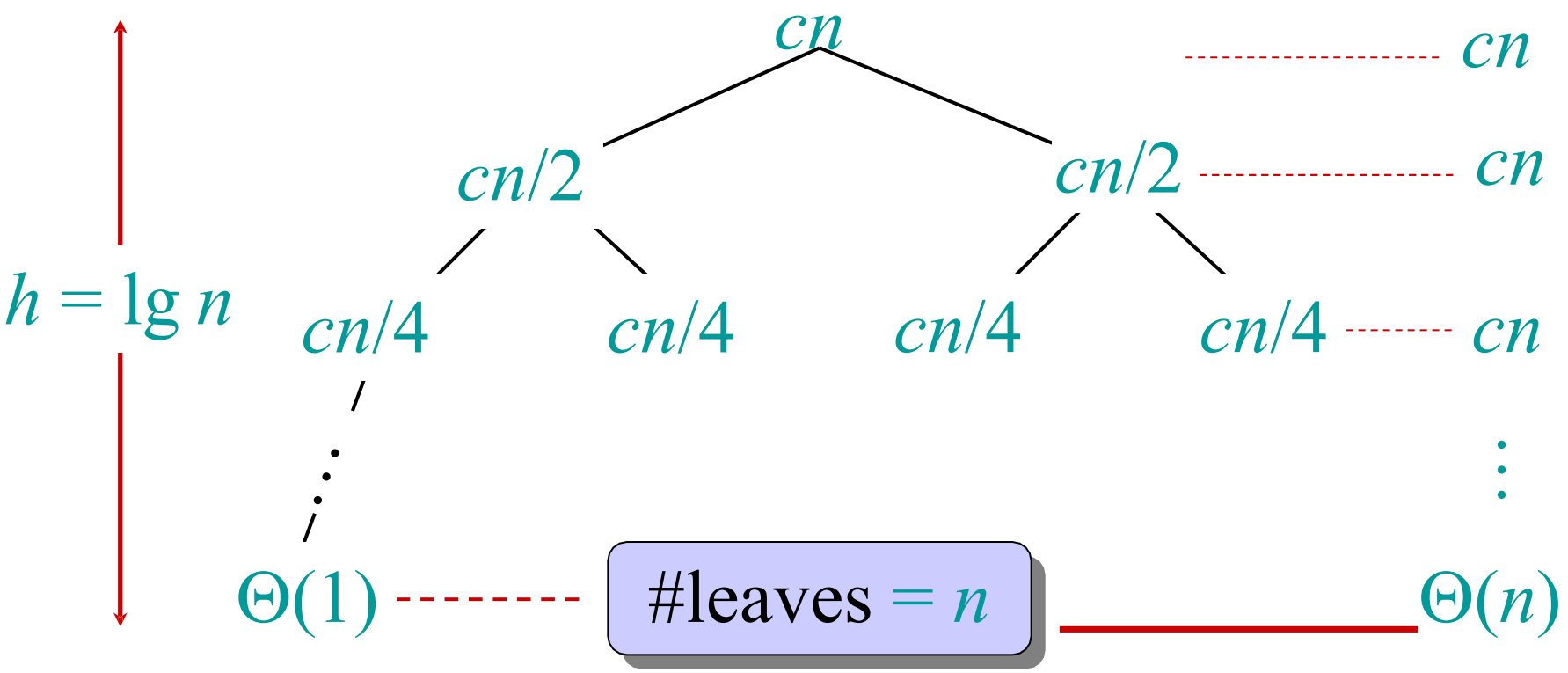
Solve $T(n) = 2T(n/2) + cn$, where $c > 0$ is constant.

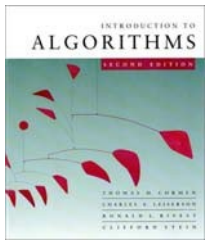




Recursion tree

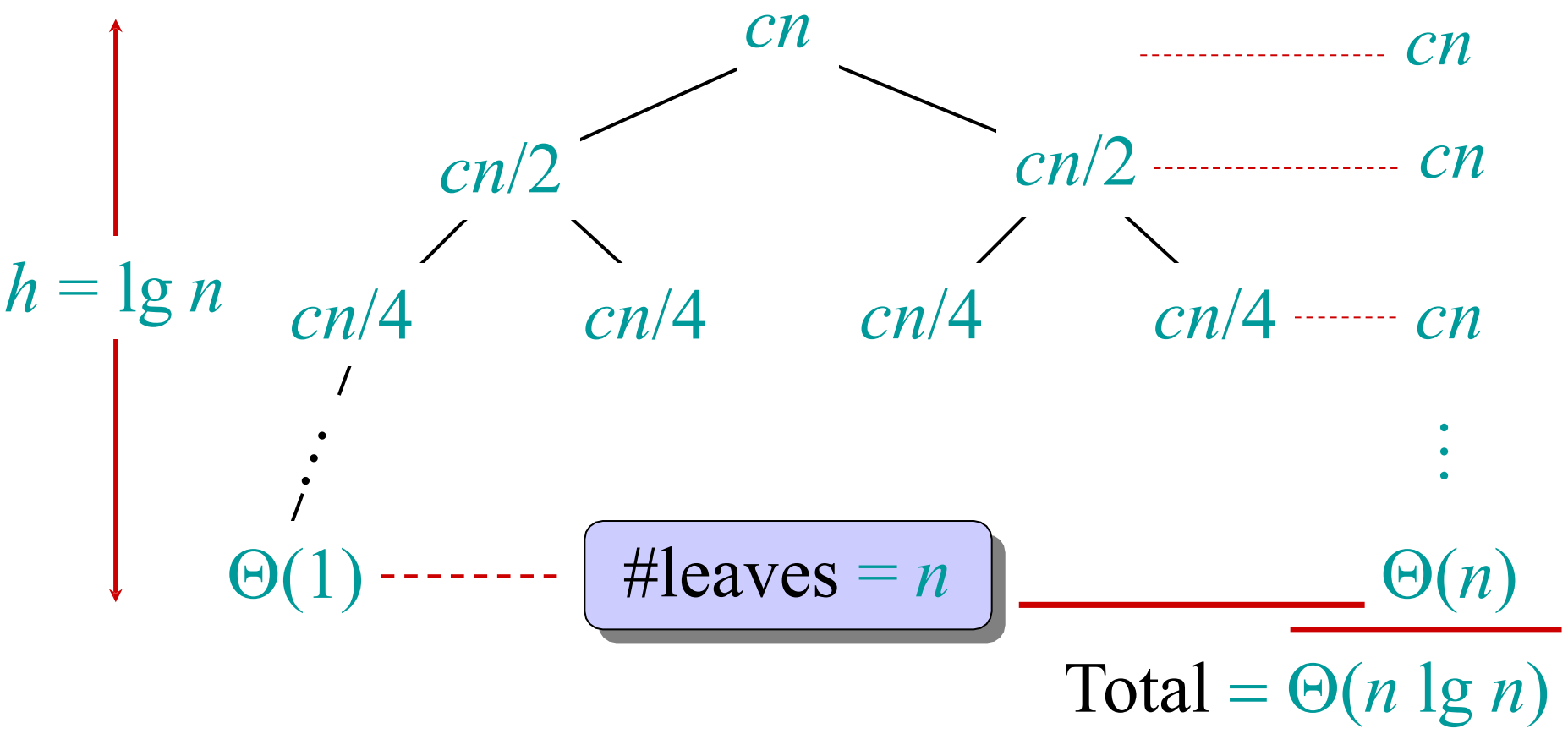
Solve $T(n) = 2T(n/2) + cn$, where $c > 0$ is constant.





Recursion tree

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

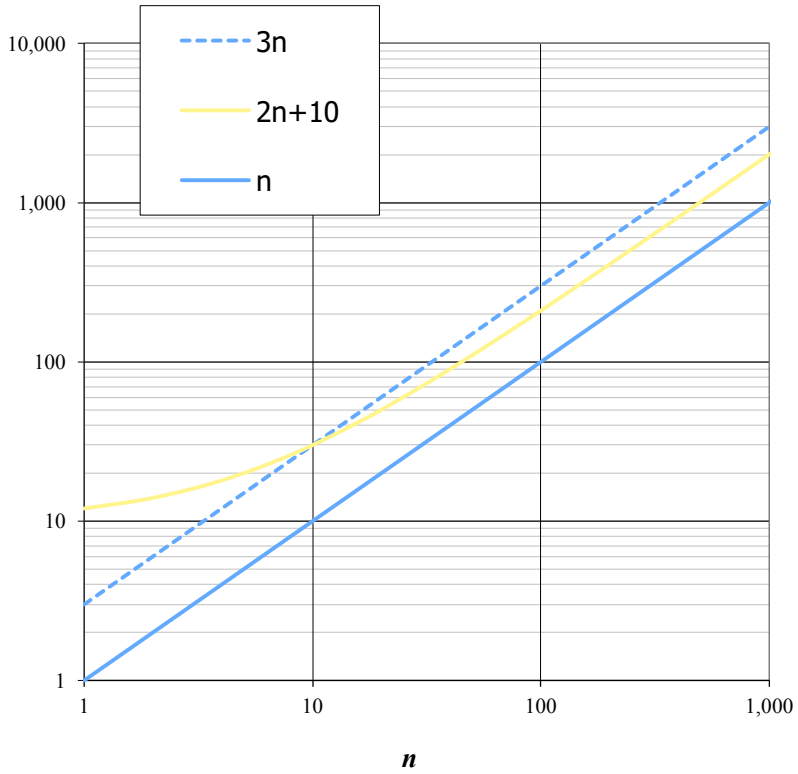


Big-Oh Notation

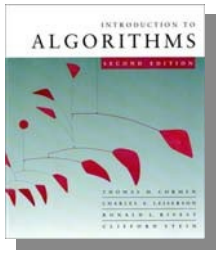
- Given functions $f(n)$ and $g(n)$, we say that $f(n)$ is $O(g(n))$ if there are positive constants c and n_0 such that

  $f(n) \leq cg(n)$ for $n \geq n_0$

- Example: $2n + 10$ is $O(n)$
  - $2n + 10 \leq cn$

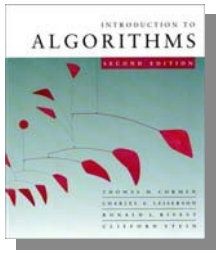


$$\Omega(g(n)) = \{f(n) : \text{there exist positive constants } c \text{ and } n_0 \text{ such that } 0 \leq cg(n) \leq f(n) \text{ for all } n \geq n_0\}.$$



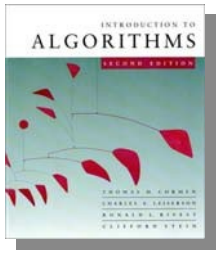
The divide-and-conquer design paradigm

1. ***Divide*** the problem (instance) into subproblems.
2. ***Conquer*** the subproblems by solving them recursively.
3. ***Combine*** subproblem solutions.



Merge sort

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



Merge sort

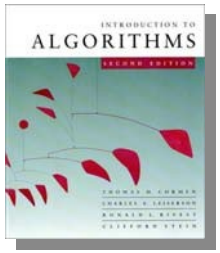
1. ***Divide:*** Trivial.
2. ***Conquer:*** Recursively sort 2 subarrays.
3. ***Combine:*** Linear-time merge.

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

 # subproblems

subproblem size

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 $\varepsilon > 0$

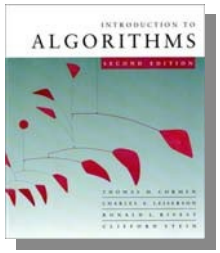
$\Rightarrow T(n) = \Theta(n^{\log_b a})$.

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

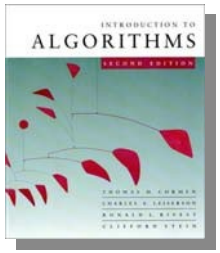
$\Rightarrow T(n) = \Theta(n^{\log_b a} \lg n)$.

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

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

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

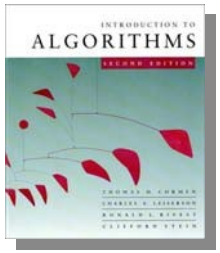
$\Rightarrow$ CASE 2 $\Rightarrow$ $T(n) = \Theta(n \lg n)$.



Binary search

- Find an element in a sorted array:

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



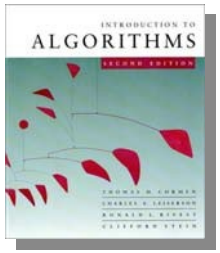
Binary search

- 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



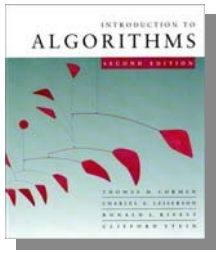
Binary search

- 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



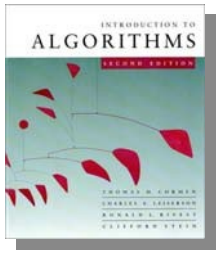
Binary search

- 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



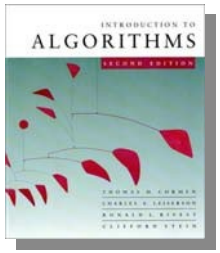
Binary search

- Find an element in a sorted array:

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

- ***Example:*** Find 9

3 5 7 8 9 12 15



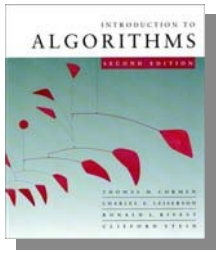
Binary search

- 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



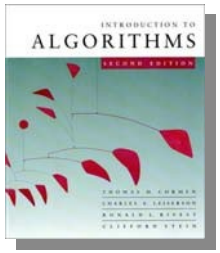
Binary search

- 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



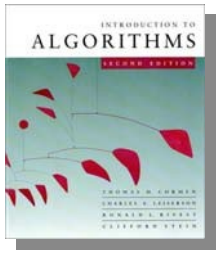
Recurrence for binary search

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

 # subproblems

subproblem size

work dividing and combining



Recurrence for binary search

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

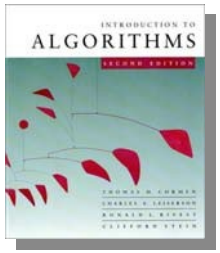
 # subproblems

subproblem size

work dividing and combining

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

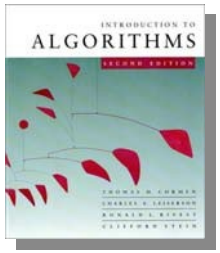
$$\Rightarrow \quad T(n) = \Theta(\lg n)\,.$$



Powering a number

Problem: Compute a^n, where $n \in \mathbb{N}$.

Naive algorithm: $\Theta(n)$.



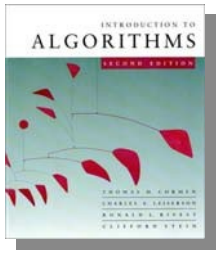
Powering a number

Problem: Compute a^n, where $n \in \mathbb{N}$.

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 $n \in \mathbb{N}$.

Naive algorithm: $\Theta(n)$.

Divide-and-conquer algorithm:

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

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