CMP302: Algorithms

Lecture 02: Merge Sort and Asymptotic Analysis

Mohamed Alaa El-Dien Aly
Computer Engineering Department
Cairo University
Fall 2013

Agenda

- Merge Sort
- Recurrences
- Asymptotic Notation

Acknowledgment
A lot of slides adapted from the slides of Erik Demaine and Charles Leiserson

Merge Sort

- Divide-and-Conquer
  - *Divide* the problem into a number of sub-problems
  - *Conquer* the smaller problems
  - *Combine* the results of the sub-problems into a solution for the big problem

```
MERGE-SORT(A, p, r)
1  if p < r
2     then q ← ⌊(p + r)/2⌋
3          MERGE-SORT(A, p, q)
4          MERGE-SORT(A, q + 1, r)
5          MERGE(A, p, q, r)
```

Merge Sort

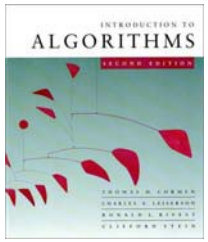
- Divide-and-Conquer
  - *Divide* the problem into a number of sub-problems
  - *Conquer* the smaller problems
  - *Combine* the results of the sub-problems into a solution for the big problem

```
void merge_sort(vector<int>& A, int p, int r) {
  if (p >= r) return;

  int q = (p + r) / 2;

  merge_sort(A, p, q);
  merge_sort(A, q+1, r);

  merge(A, p, q, r);
}
```

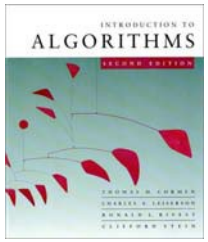


Merge sort

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

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

Key subroutine: MERGE



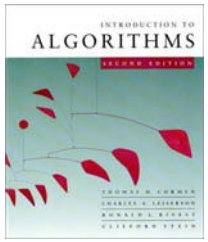
Merging two sorted arrays

20 12

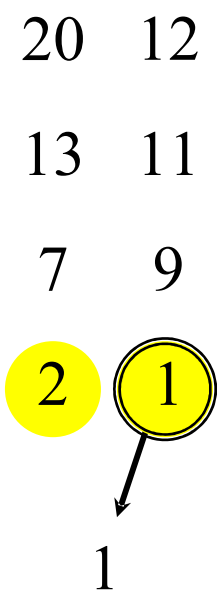
13 11

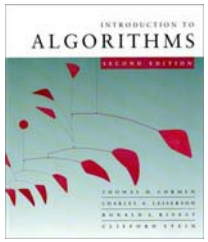
7 9

2 1

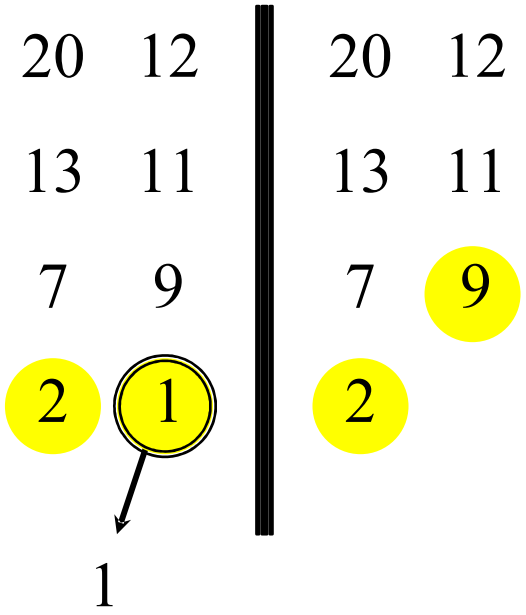


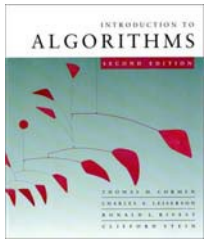
Merging two sorted arrays



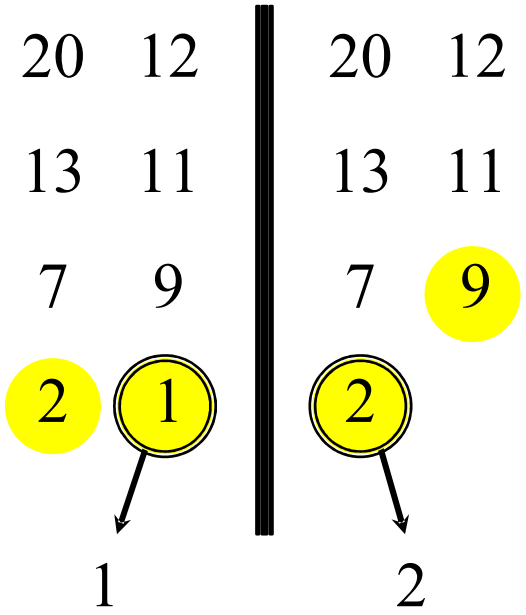


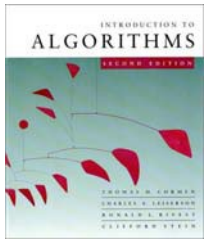
Merging two sorted arrays



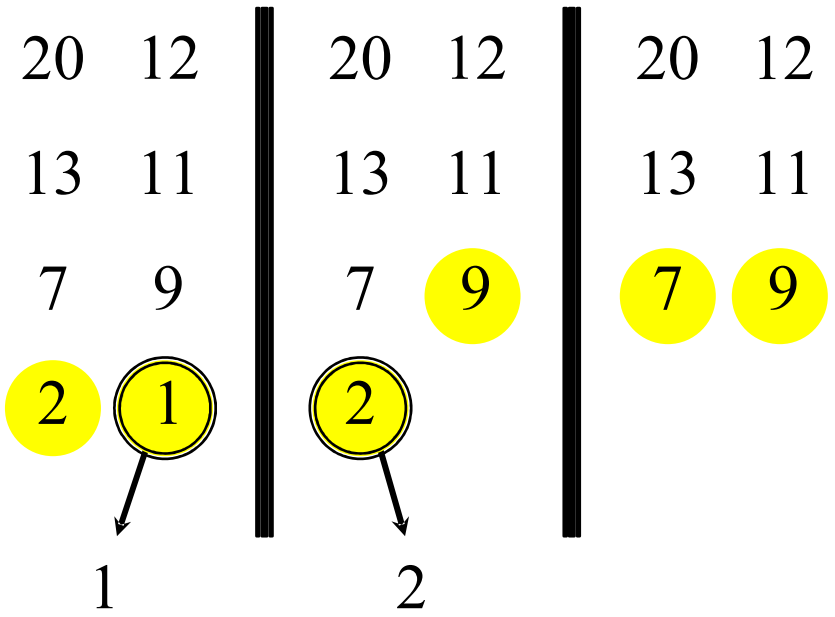


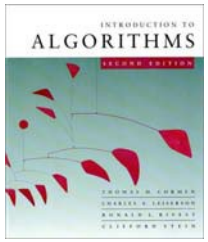
Merging two sorted arrays



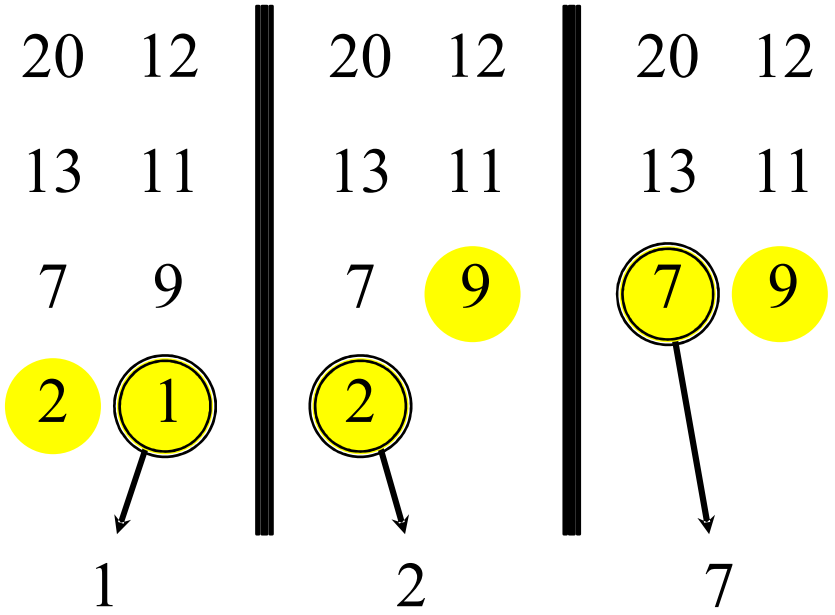


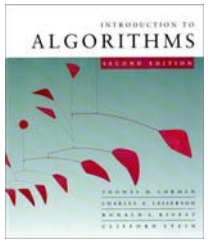
Merging two sorted arrays



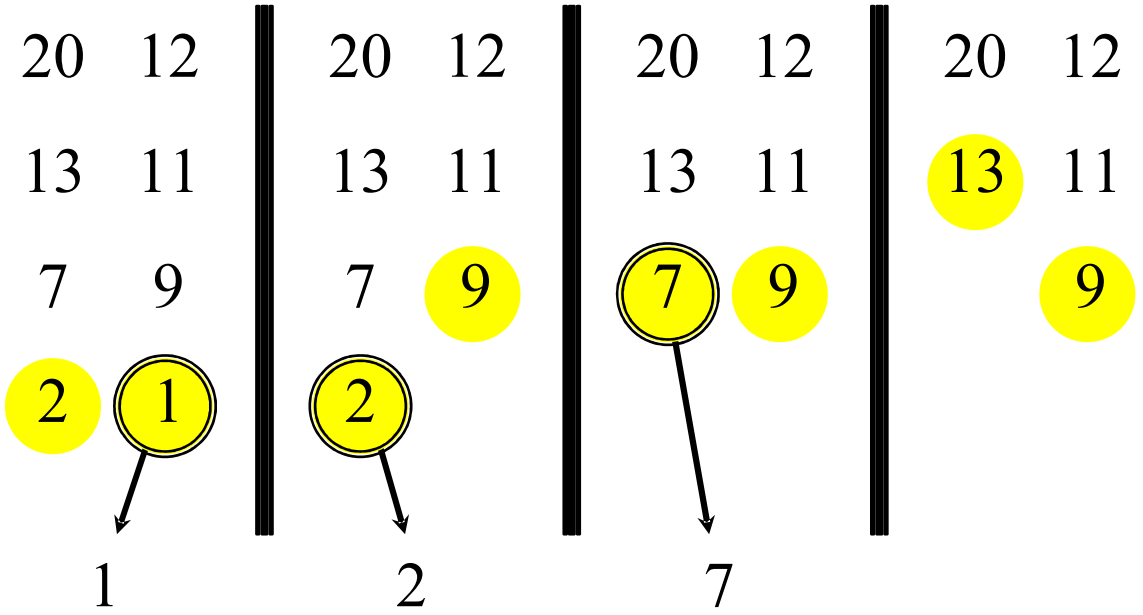


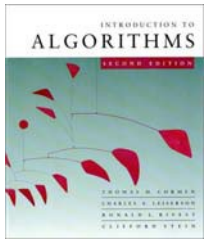
Merging two sorted arrays



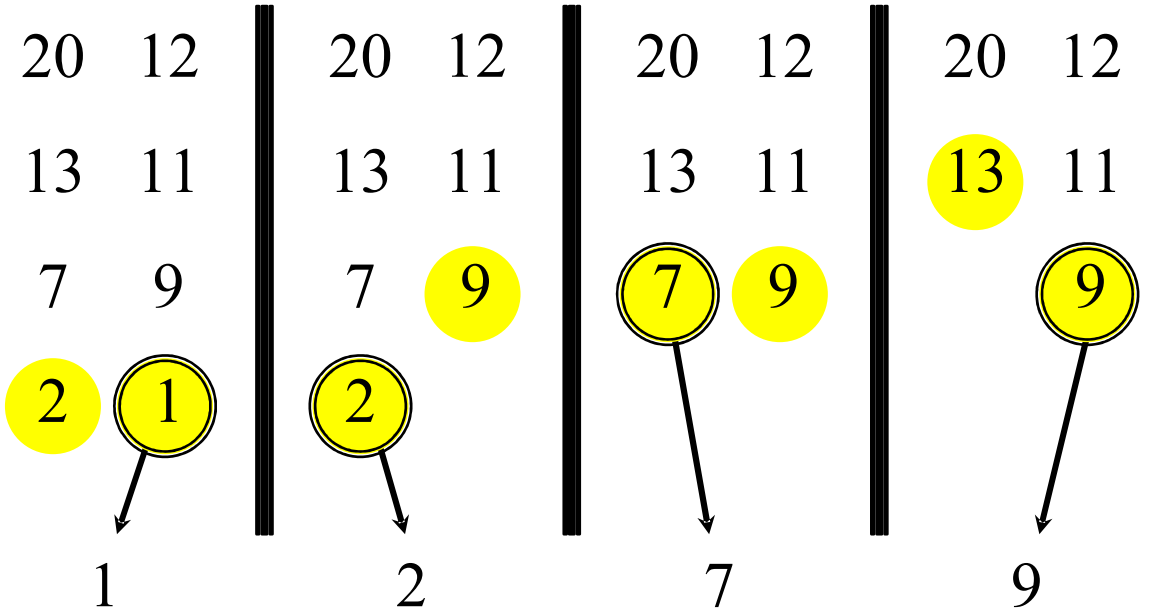


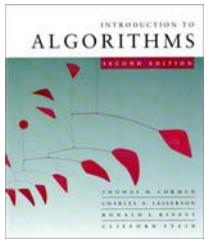
Merging two sorted arrays



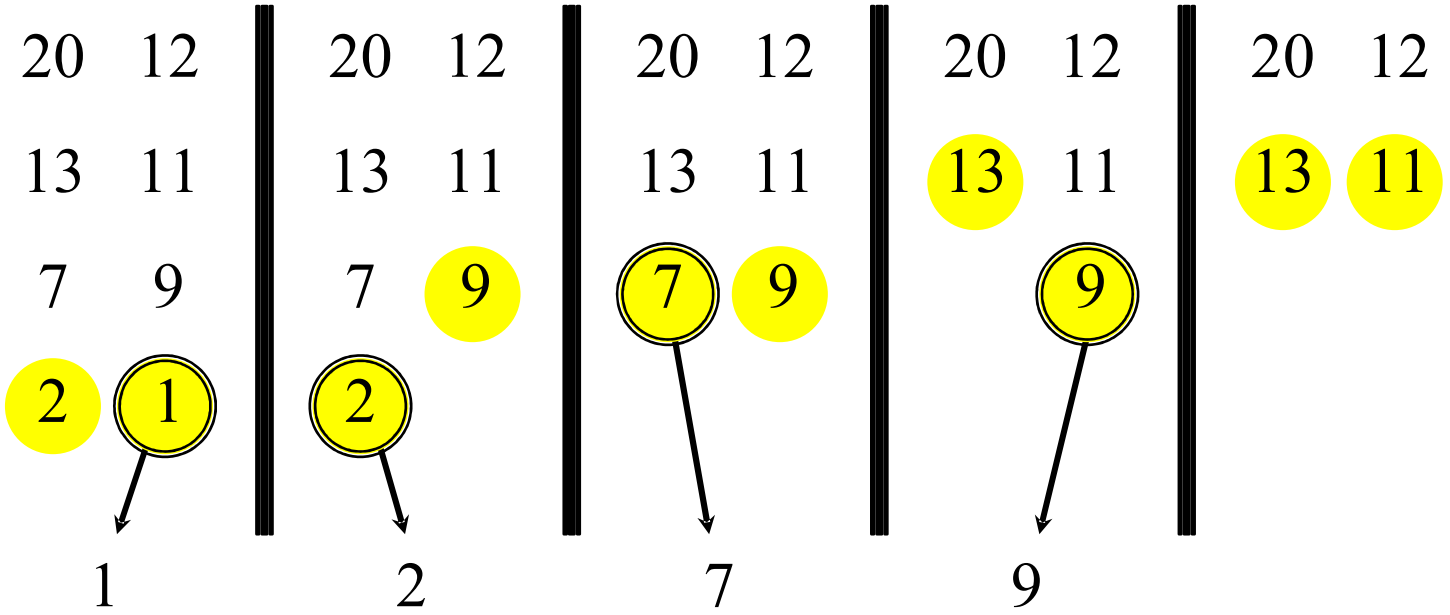


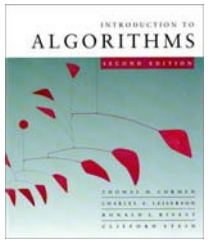
Merging two sorted arrays



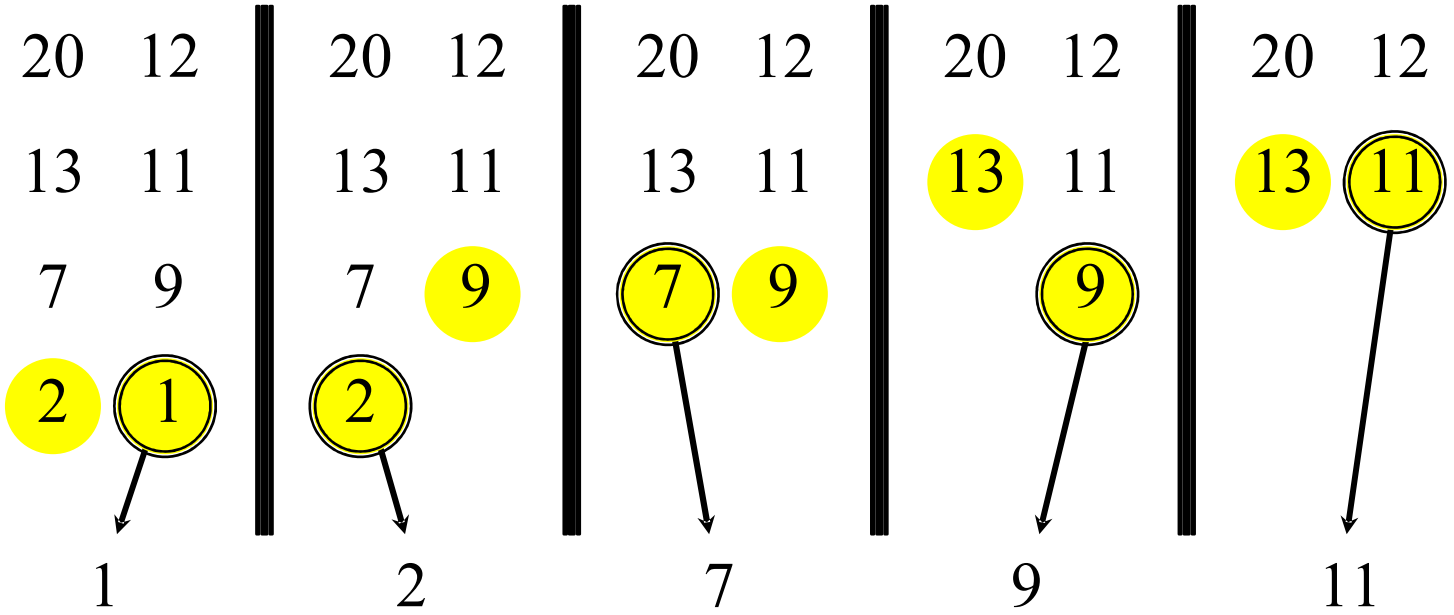


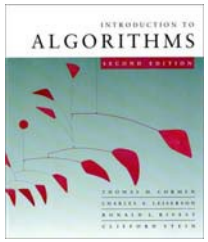
Merging two sorted arrays



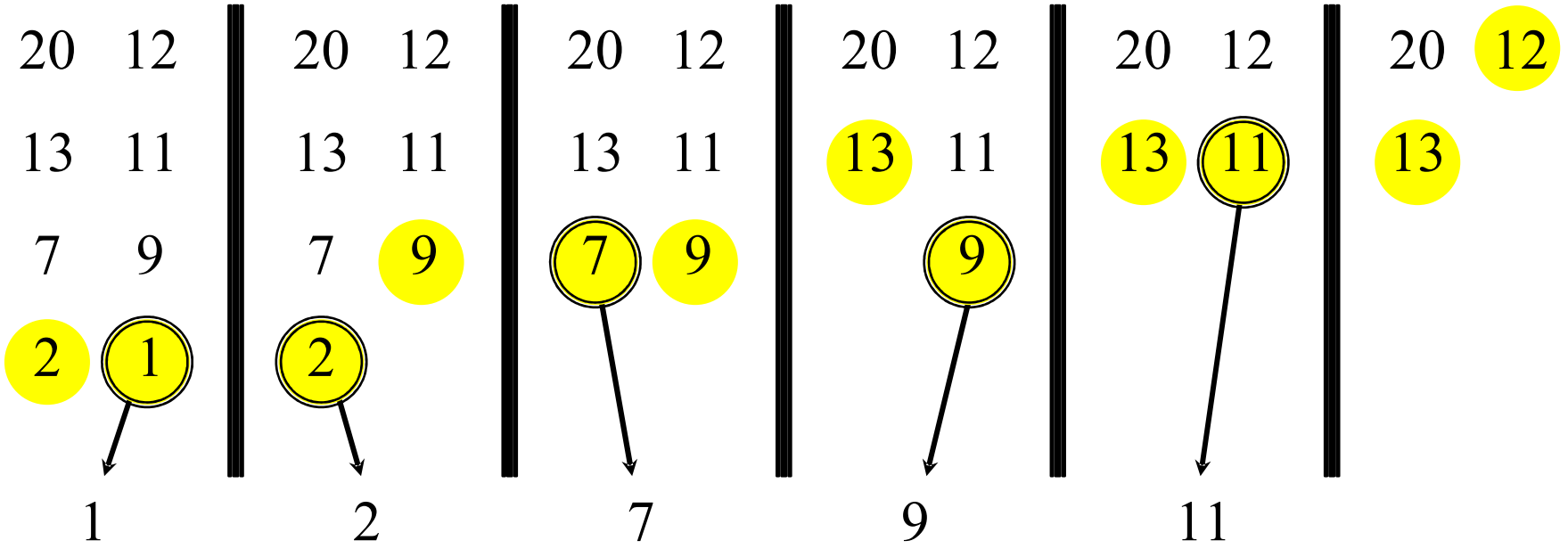


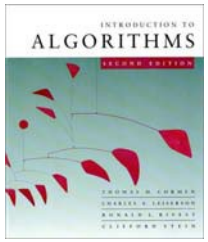
Merging two sorted arrays



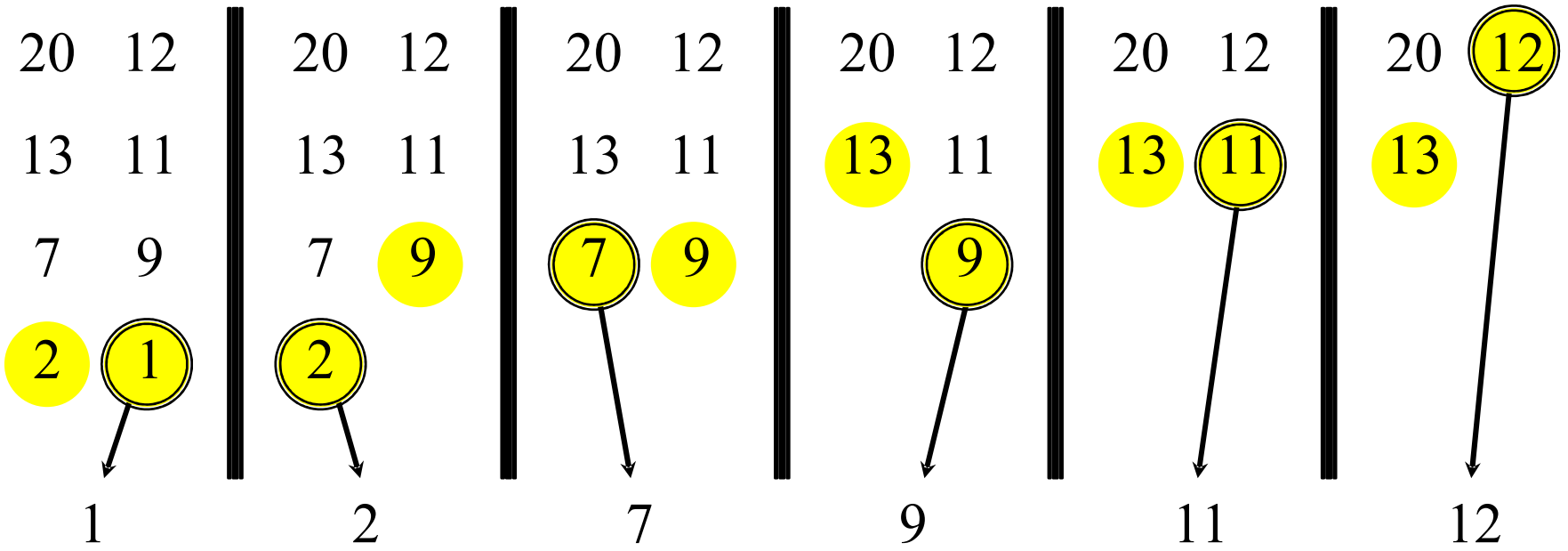


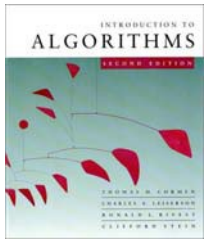
Merging two sorted arrays



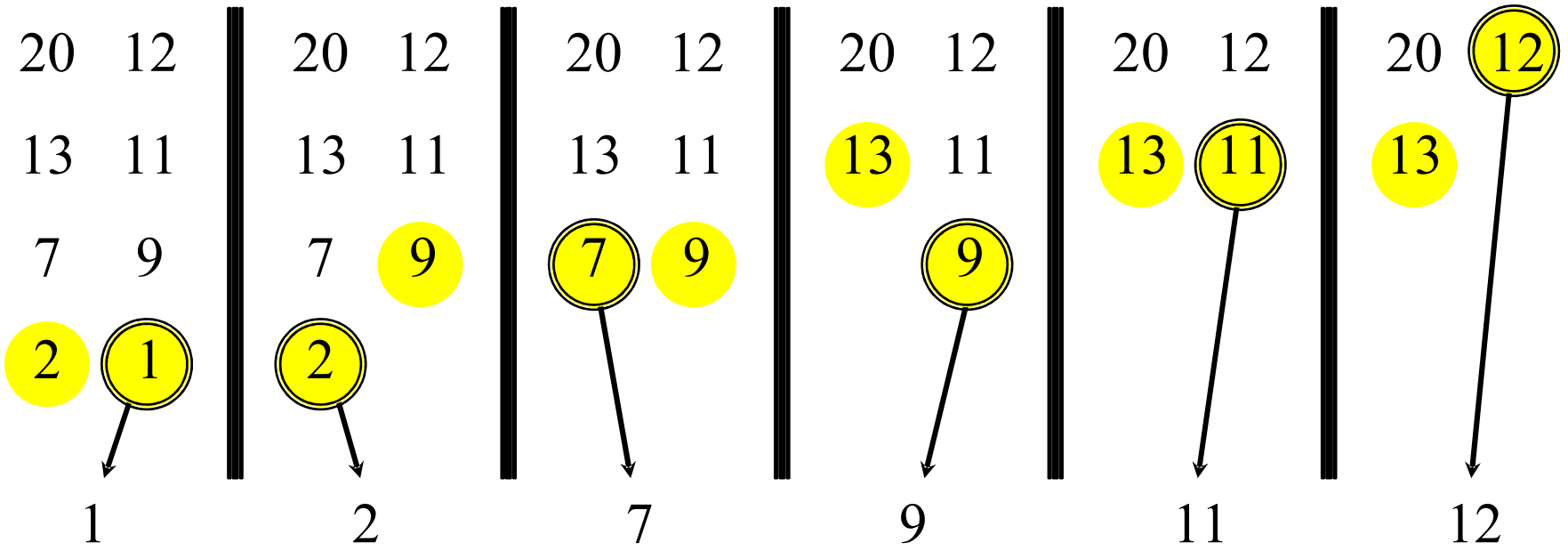


Merging two sorted arrays





Merging two sorted arrays



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

Merge

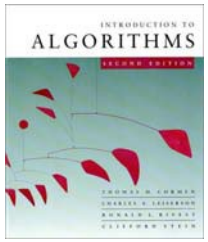
```
MERGE(A, p, q, r)
 1  n_1 = q − p + 1
 2  n_2 = r − q
 3  let L[1..n_1 + 1] and R[1..n_2 + 1] be new arrays
 4  for i = 1 to n_1
 5      L[i] = A[p + i − 1]
 6  for j = 1 to n_2
 7      R[j] = A[q + j]
 8  L[n_1 + 1] = ∞
 9  R[n_2 + 1] = ∞
10  i = 1
11  j = 1
12  for k = p to r
13      if L[i] ≤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

compute sizes

copy arrays to be merged

adds "sentinel"

merge



Analyzing merge sort

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

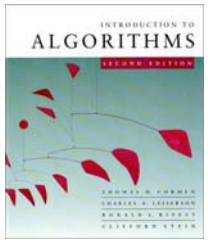
Abuse

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

Recurrence for Merge Sort

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

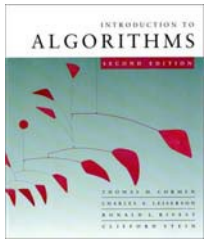
- Next we need to solve this recurrence relation i.e. find *T(n)* as a function of *n* without *T(n/2)*



Recurrence for merge sort

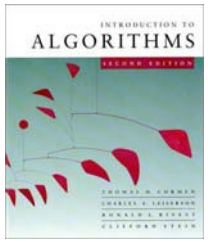
$$T(n) = \begin{cases} \Theta(1) \text{ if } n = 1; \\ 2T(n/2) + \Theta(n) \text{ if } n > 1. \end{cases}$$

- 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

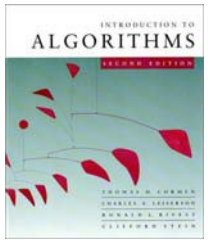
Solve $T(n) = 2T(n/2) + cn$, where $c > 0$ is constant.



Recursion tree

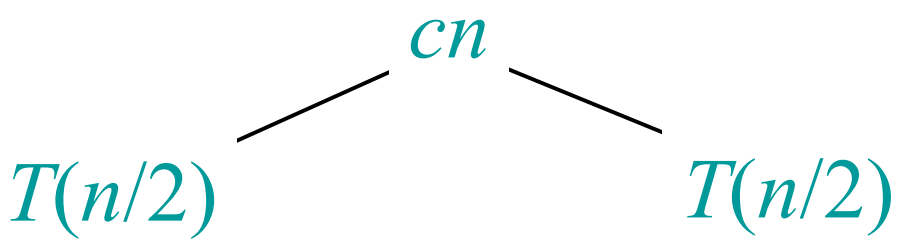
Solve $T(n) = 2T(n/2) + cn$, where $c > 0$ is constant.

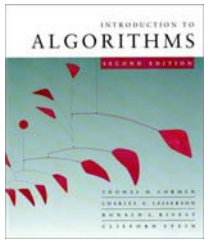
$T(n)$



Recursion tree

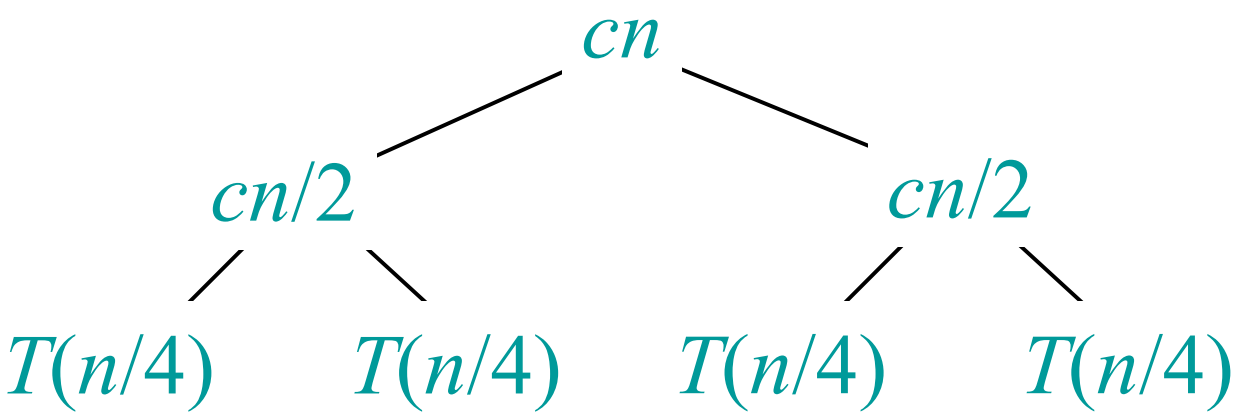
Solve $T(n) = 2T(n/2) + cn$, where $c > 0$ is constant.

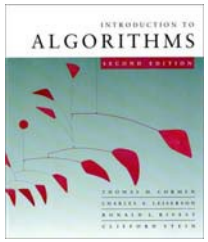




Recursion tree

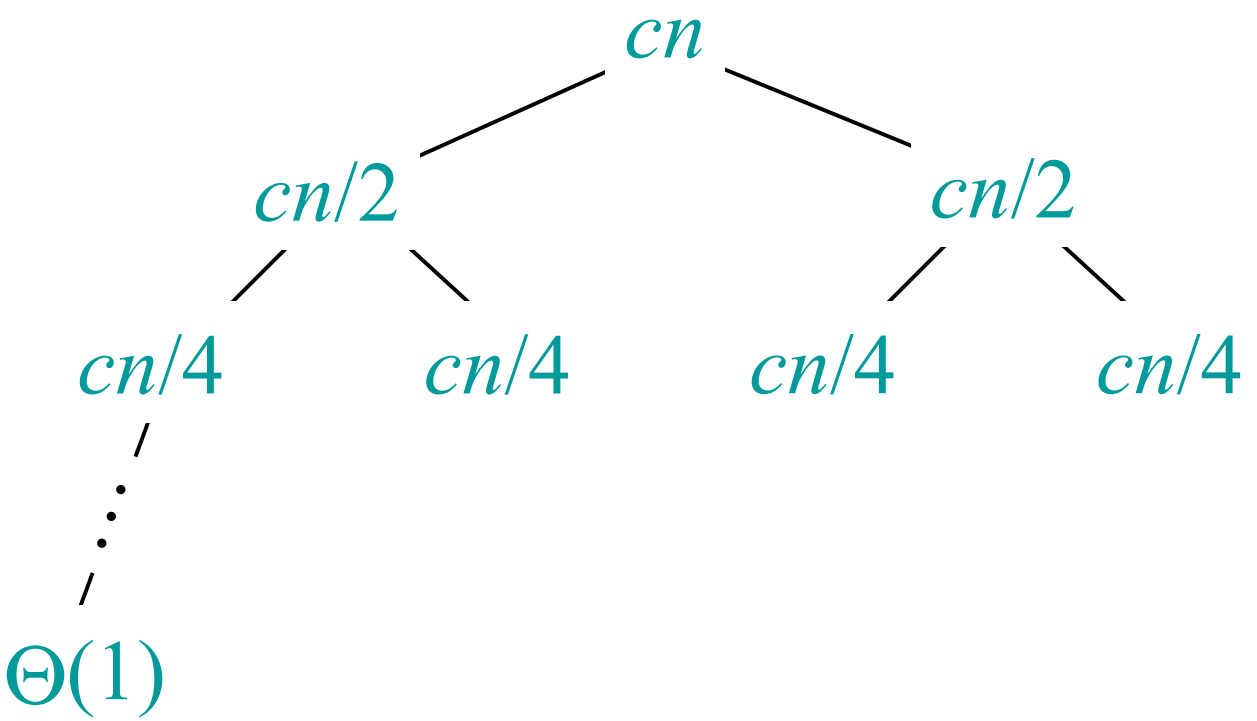
Solve $T(n) = 2T(n/2) + cn$, where $c > 0$ is constant.

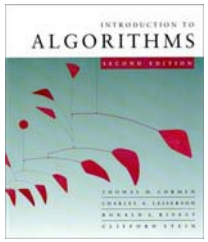




Recursion tree

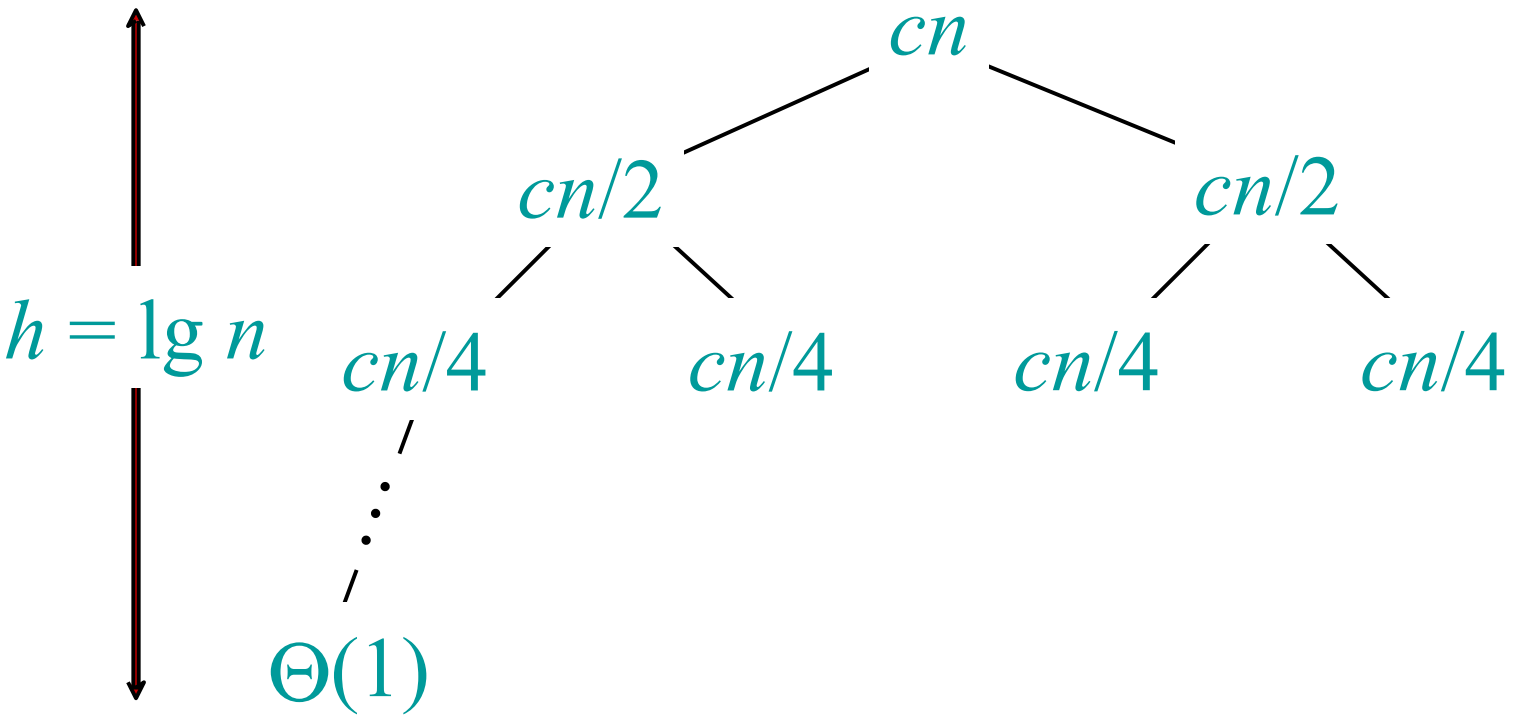
Solve $T(n) = 2T(n/2) + cn$, where $c > 0$ is constant.

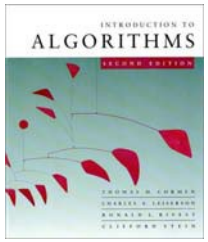




Recursion tree

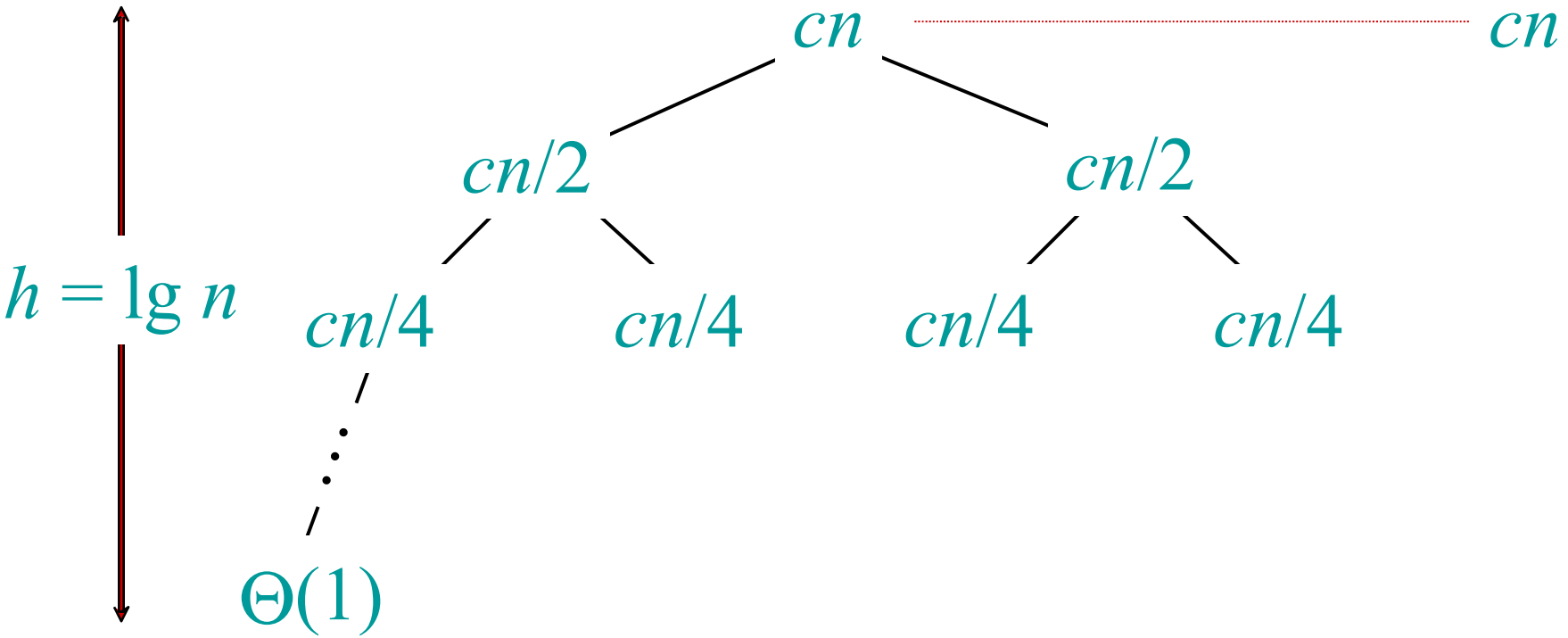
Solve $T(n) = 2T(n/2) + cn$, where $c > 0$ is constant.

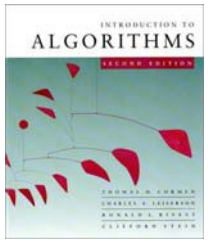




Recursion tree

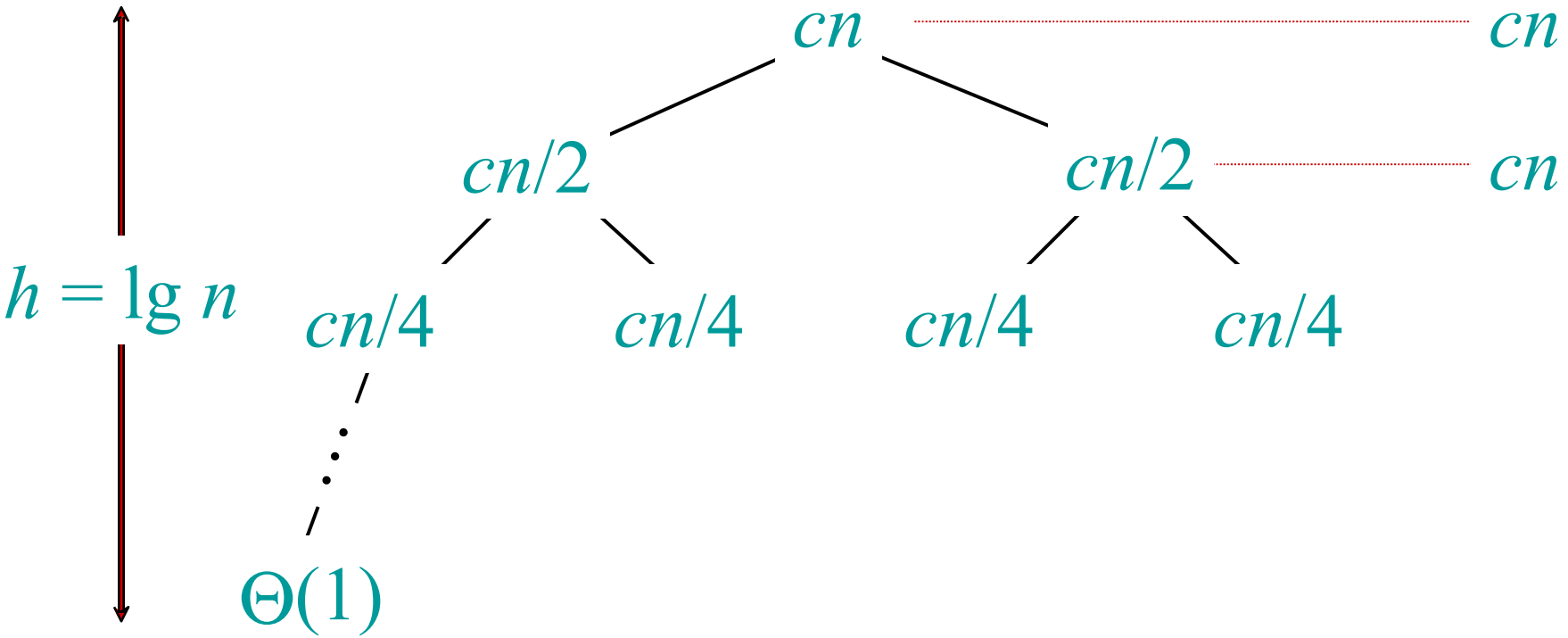
Solve $T(n) = 2T(n/2) + cn$, where $c > 0$ is constant.

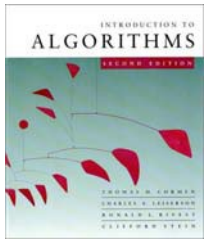




Recursion tree

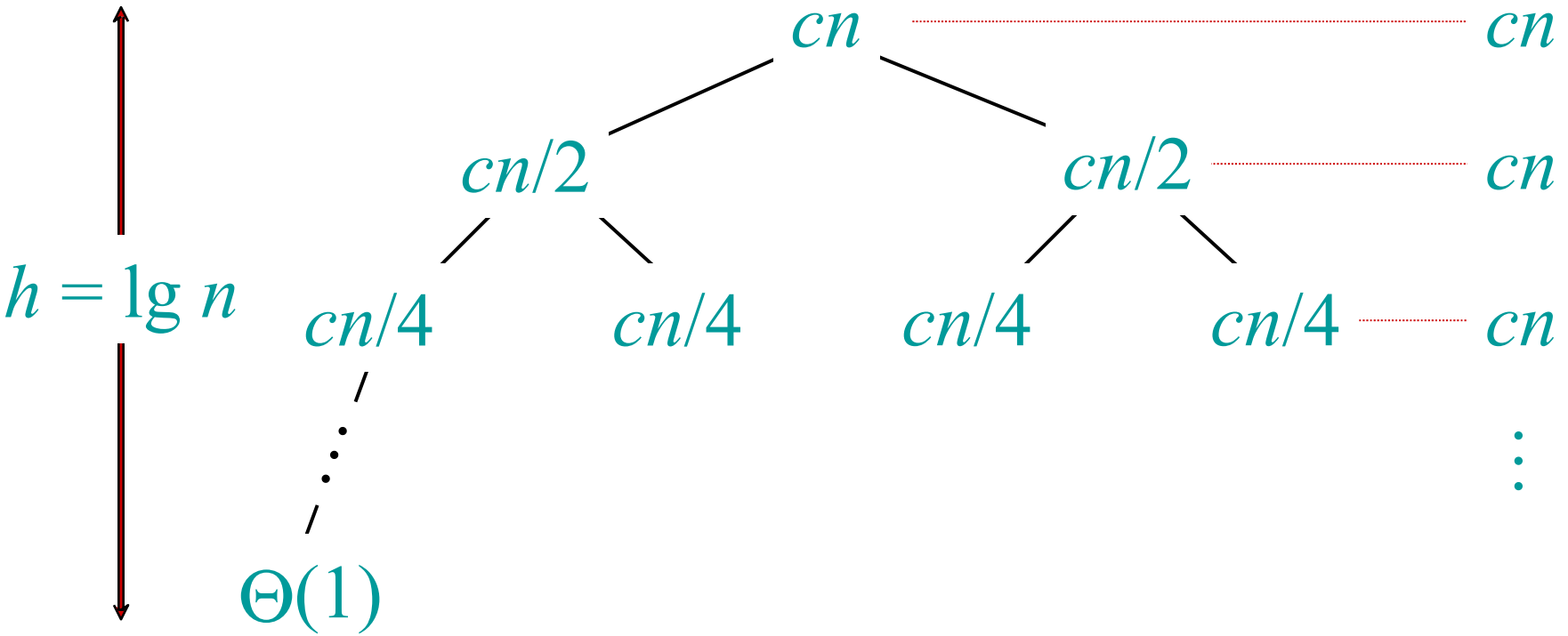
Solve $T(n) = 2T(n/2) + cn$, where $c > 0$ is constant.

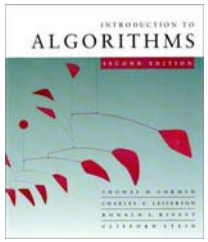




Recursion tree

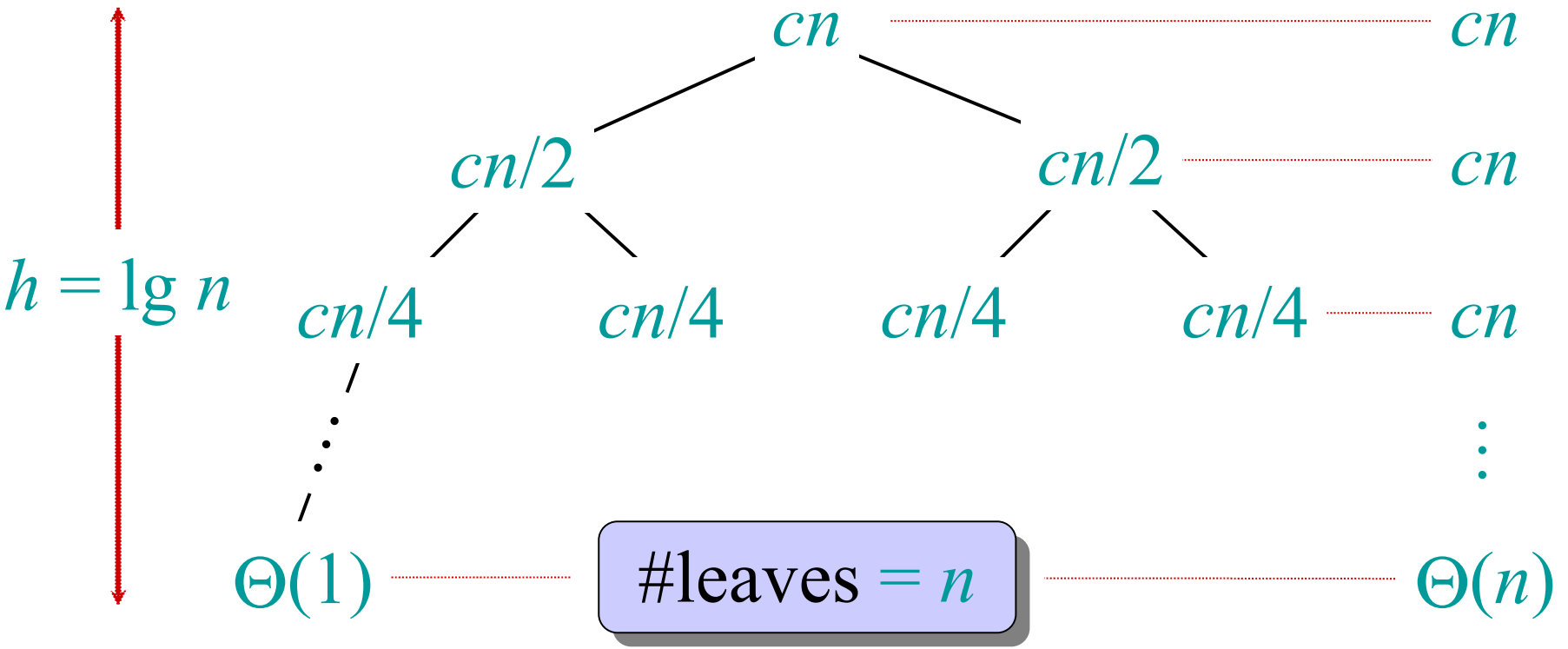
Solve $T(n) = 2T(n/2) + cn$, where $c > 0$ is constant.

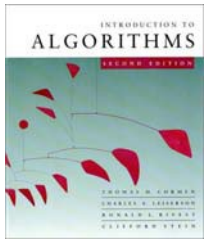




Recursion tree

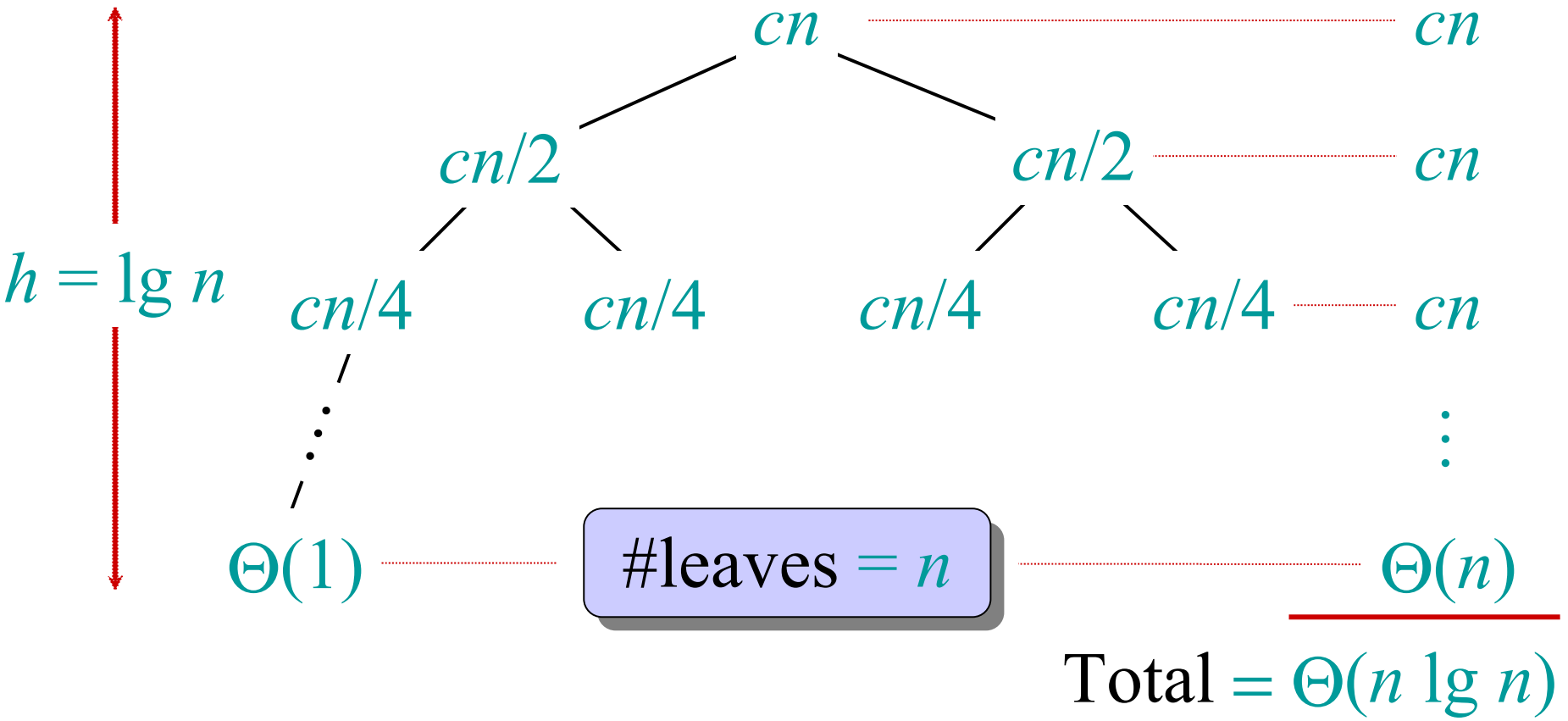
Solve $T(n) = 2T(n/2) + cn$, where $c > 0$ is constant.





Recursion tree

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

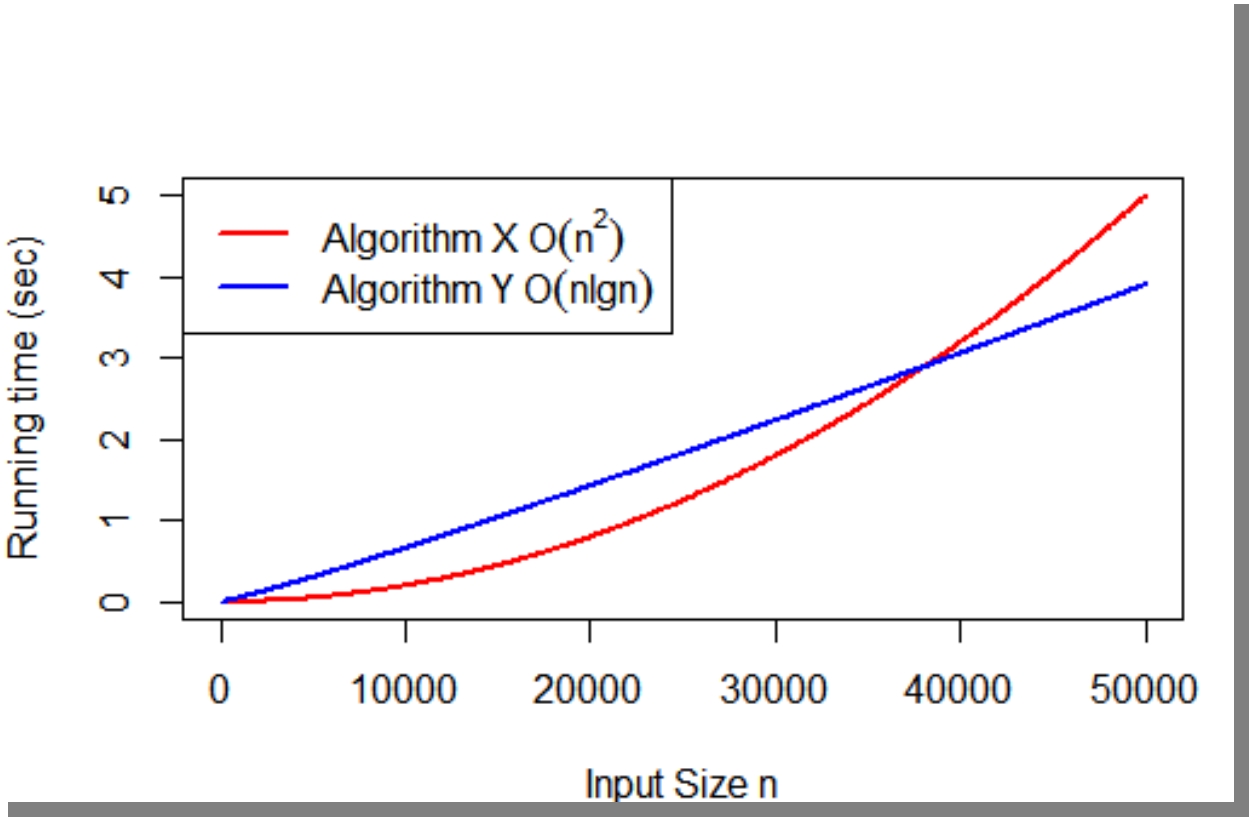


Merge Sort Analysis

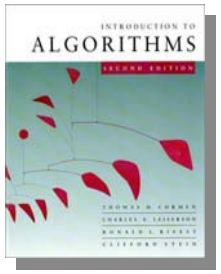
- What about space?
- Merge Sort does not sort "in place" as it needs temporary space for the "Merge" subroutine
- It needs space: $S(n) = \Theta(n)$

Conclusion

- Insertion Sort:
  - $T(n) = \Theta(n^2)$
  - $S(n) = \Theta(1)$
- Merge Sort:
  - $T(n) = \Theta(n \lg n)$
  - $S(n) = \Theta(n)$



- Insertion sort better for smaller n, merge sort for larger n

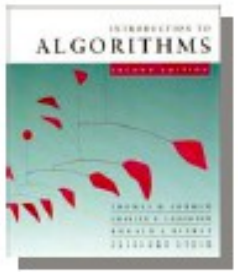


Asymptotic notation

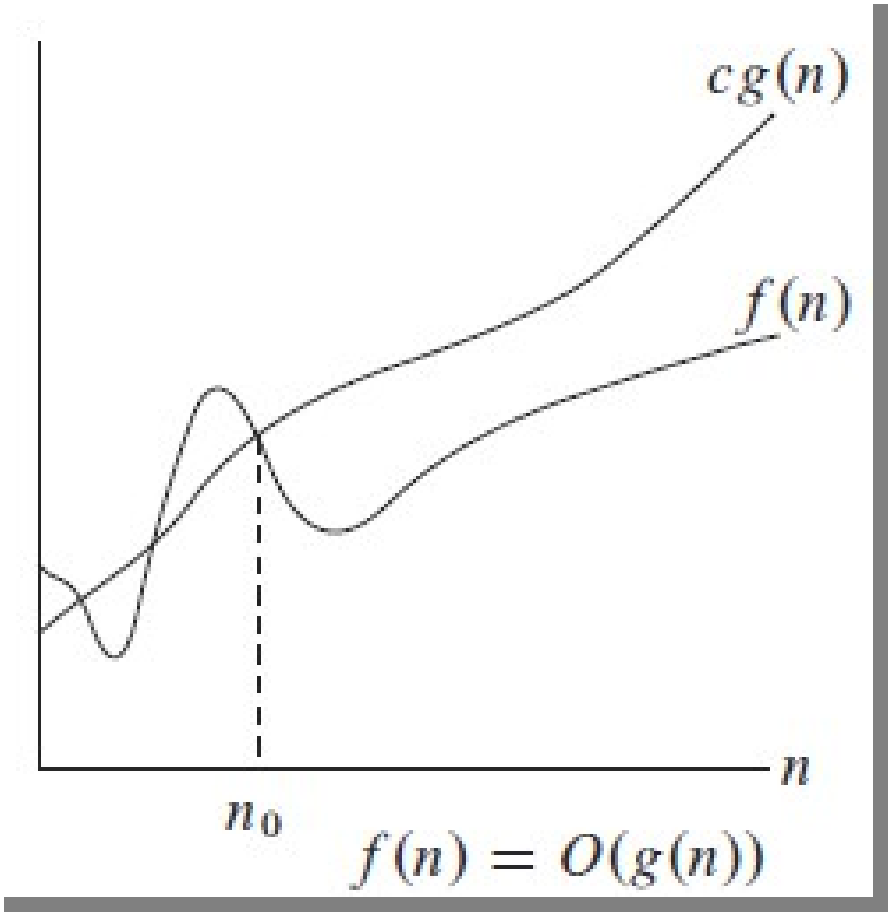
O-notation (upper bounds):

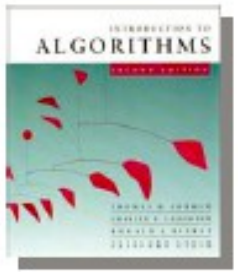
We write $f(n) = O(g(n))$ if there exist constants $c > 0$, $n_0 > 0$ such that $0 \le f(n) \le cg(n)$ for all $n \ge n_0$.

EXAMPLE: $2n^2 = O(n^3)$ $\quad (c = 1, n_0 = 2)$

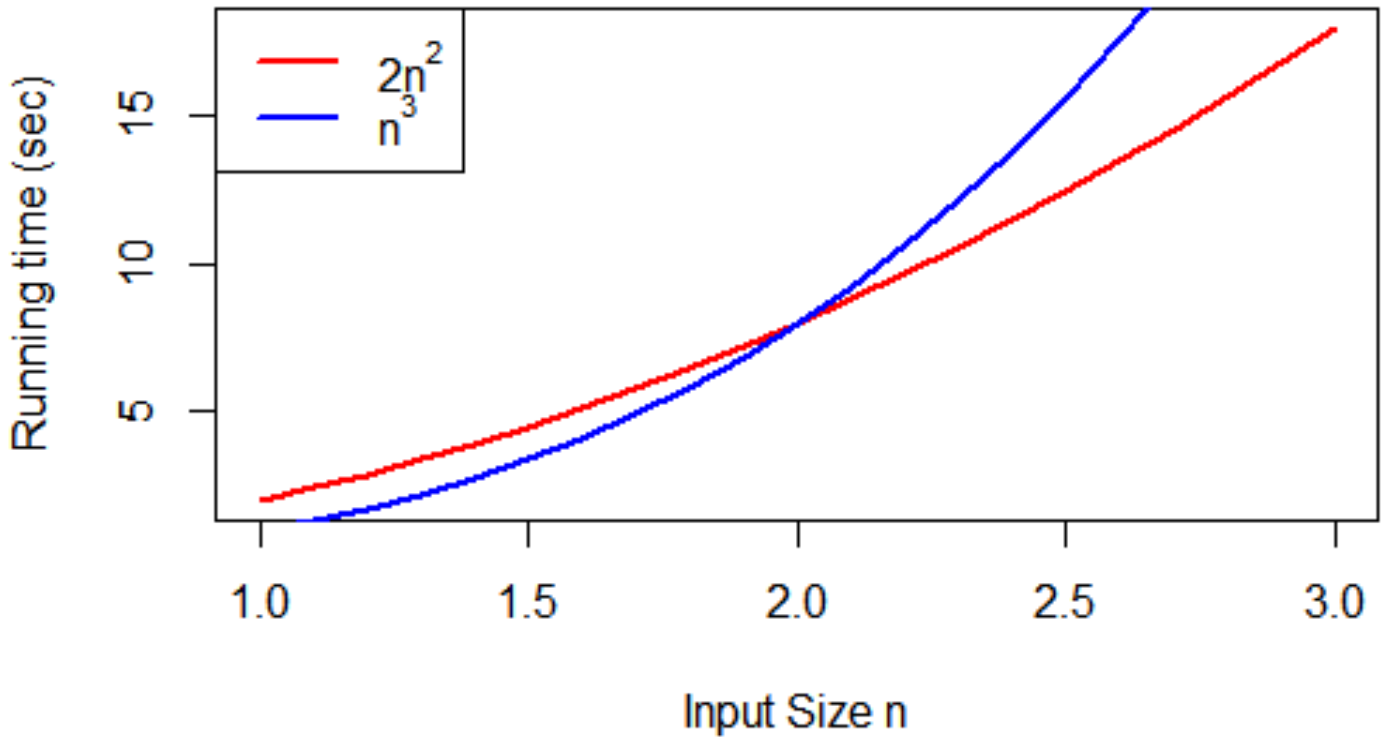


Asymptotic notation

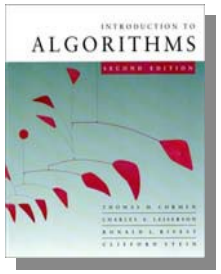




Asymptotic notation



EXAMPLE: $2n^2 = O(n^3)$ $(c = 1, n_0 = 2)$



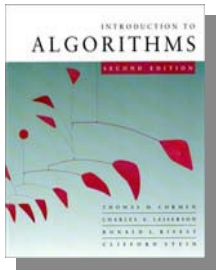
Asymptotic notation

O-notation (upper bounds):

We write $f(n) = O(g(n))$ if there exist constants $c > 0$, $n_0 > 0$ such that $0 \le f(n) \le cg(n)$ for all $n \ge n_0$.

EXAMPLE: $2n^2 = O(n^3)$ $\quad (c = 1, n_0 = 2)$

functions, not values



Asymptotic notation

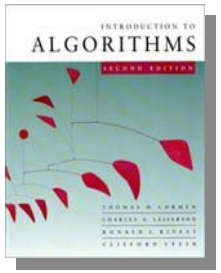
O-notation (upper bounds):

We write $f(n) = O(g(n))$ if there exist constants $c > 0$, $n_0 > 0$ such that $0 \le f(n) \le cg(n)$ for all $n \ge n_0$.

EXAMPLE: $2n^2 = O(n^3)$ $(c = 1, n_0 = 2)$

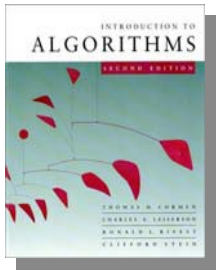
functions, not values

funny, "one-way" equality



Set definition of O-notation

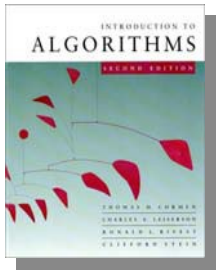
$O(g(n)) = \{ f(n)$: there exist constants $c > 0$, $n_0 > 0$ such that $0 \le f(n) \le cg(n)$ for all $n \ge n_0 \}$



Set definition of O-notation

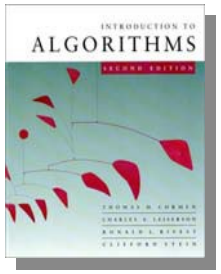
$O(g(n)) = \{ f(n)$: there exist constants $c > 0$, $n_0 > 0$ such that $0 \le f(n) \le cg(n)$ for all $n \ge n_0 \}$

EXAMPLE: $2n^2 \in O(n^3)$



Macro substitution

Convention: A set in a formula represents an anonymous function in the set.



Macro substitution

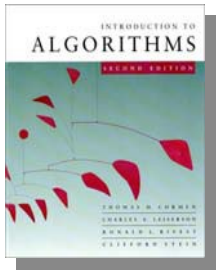
Convention: A set in a formula represents an anonymous function in the set.

EXAMPLE: $f(n) = n^3 + O(n^2)$

means

$f(n) = n^3 + h(n)$

for some $h(n) \in O(n^2)$.



Macro substitution

Convention: A set in a formula represents an anonymous function in the set.

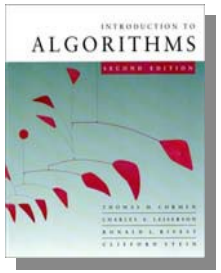
EXAMPLE: $n^2 + O(n) = O(n^2)$

means

for any $f(n) \in O(n)$:

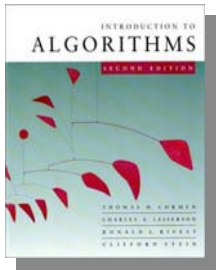
$$n^2 + f(n) = h(n)$$

for some $h(n) \in O(n^2)$.



Ω-notation (lower bounds)

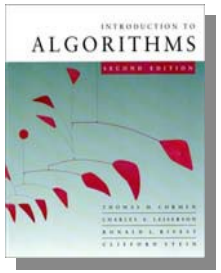
O-notation is an *upper-bound* notation. It makes no sense to say $f(n)$ is at least $O(n^2)$.



Ω-notation (lower bounds)

O-notation is an *upper-bound* notation. It makes no sense to say $f(n)$ is at least $O(n^2)$.

$\Omega(g(n)) = \{ f(n) :$ there exist constants $c > 0$, $n_0 > 0$ such that $0 \le cg(n) \le f(n)$ for all $n \ge n_0 \}$

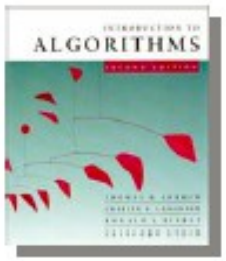


Ω-notation (lower bounds)

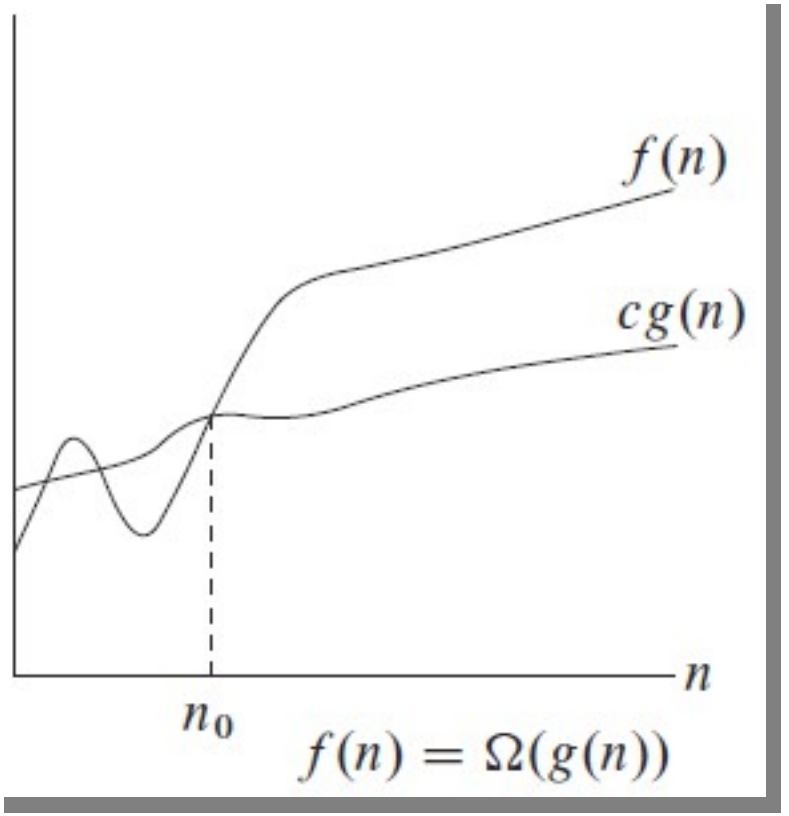
O-notation is an *upper-bound* notation. It makes no sense to say $f(n)$ is at least $O(n^2)$.

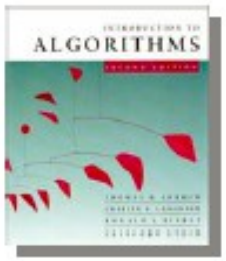
$\Omega(g(n)) = \{ f(n)$: there exist constants $c > 0, n_0 > 0$ such that $0 \le cg(n) \le f(n)$ for all $n \ge n_0 \}$

EXAMPLE: $\sqrt{n} = \Omega(\lg n)$ $(c = 1, n_0 = 16)$

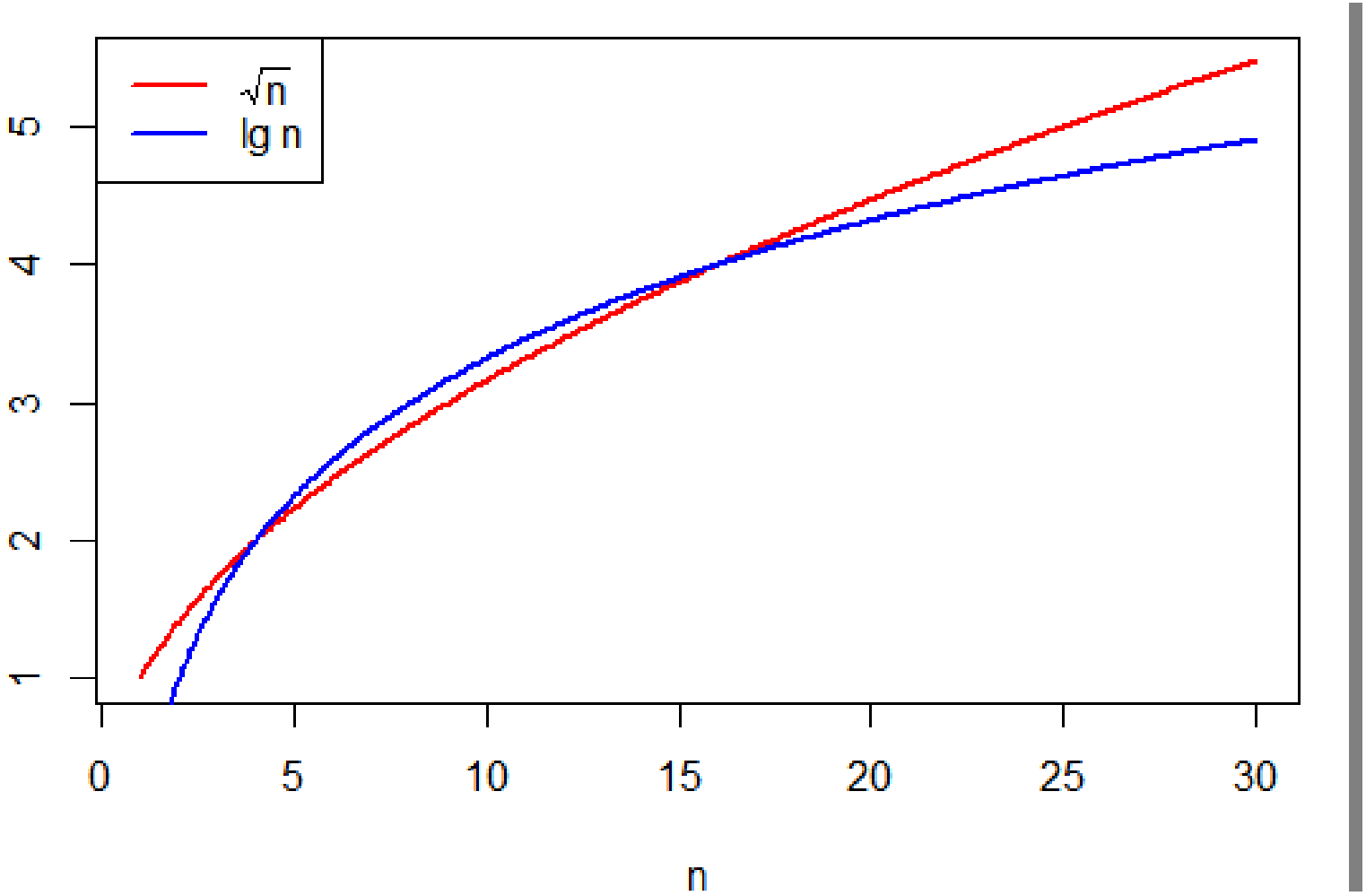


Ω-notation (lower bounds)

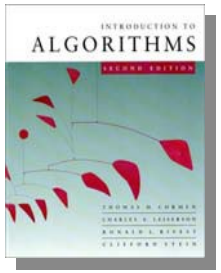




Ω-notation (lower bounds)

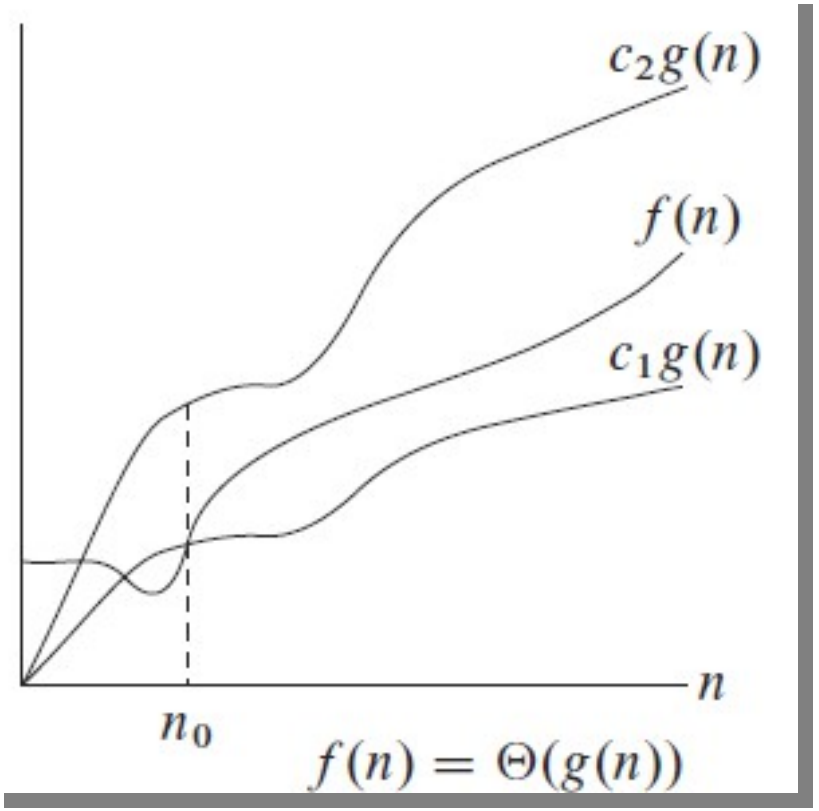


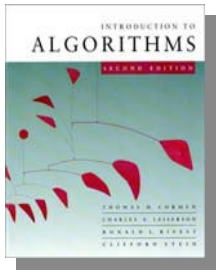
EXAMPLE: $\sqrt{n} = \Omega(\lg n) \quad (c = 1, n_0 = 16)$



Θ-notation (tight bounds)

$$\Theta(g(n)) = O(g(n)) \cap \Omega(g(n))$$





Θ-notation (tight bounds)

$$\Theta(g(n)) = O(g(n)) \cap \Omega(g(n))$$

EXAMPLE: $\frac{1}{2}n^2 - 2n = \Theta(n^2)$

Recap

- Merge Sort
- Recurrences
- Asymptotic Notation
- Next: More on recurrences and Divide and Conquer