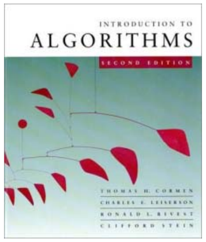
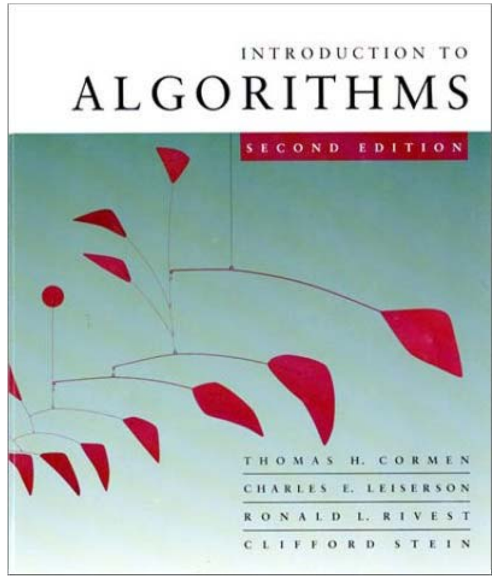


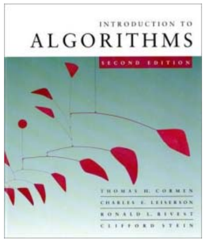
CS 3343 – Fall 2011



Merge Sort

Carola Wenk

Slides courtesy of Charles Leiserson with small changes by Carola Wenk

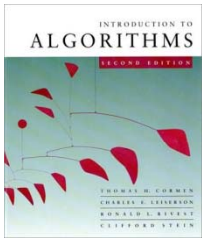


Merge sort

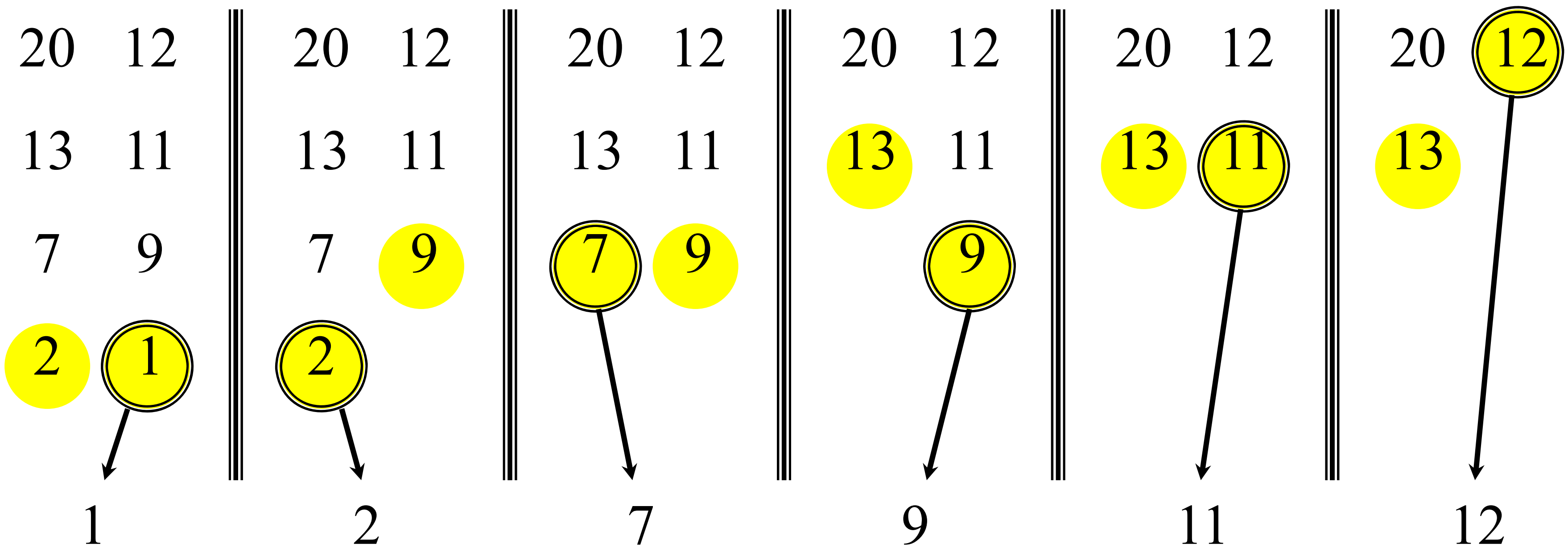
MERGE-SORT ($A[1\,..\,n]$)

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

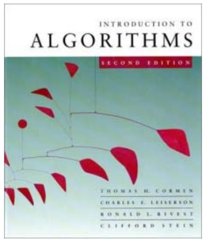
Key subroutine: **MERGE**



Merging two sorted arrays



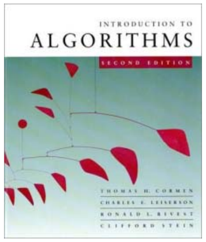
Time $dn \in \Theta(n)$ to merge a total of n elements (linear time).



Analyzing merge sort

$T(n)$	**MERGE-SORT** ($A[1 \,..\, n]$)
d_0	1. If $n = 1$, done.
$T(n/2)$	2. **MERGE-SORT** ($A[\,1 \,..\, \lceil n/2 \rceil\,]$)
$T(n/2)$	3. **MERGE-SORT** ($A[\,\lceil n/2 \rceil + 1 \,..\, n\,]$)
dn	4. "***Merge***" the 2 sorted lists.

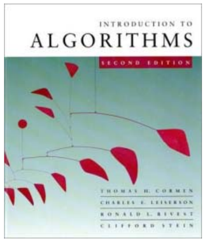
Sloppiness: Should be $T(\lceil n/2 \rceil) + T(\lfloor n/2 \rfloor)$, but it turns out not to matter asymptotically.



Recurrence for merge sort

$$T(n) = \begin{cases} d_0 \text{ if } n = 1; \\ 2T(n/2) + dn \text{ if } n > 1. \end{cases}$$

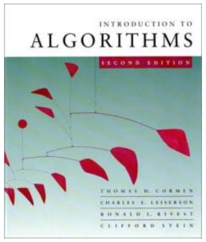
- But what does $T(n)$ solve to? I.e., is it O(n) or O(n^2) or O(n^3) or …?



Recursion tree

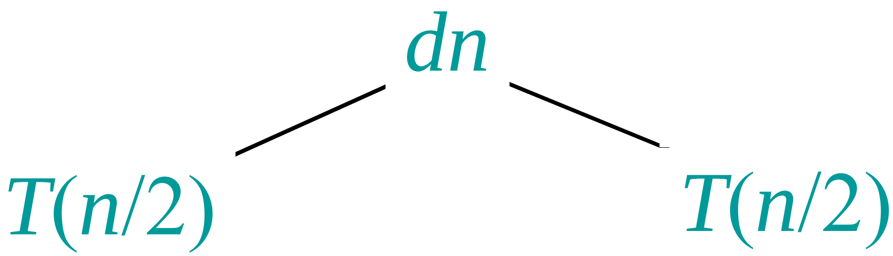
Solve $T(n) = 2T(n/2) + dn$, where $d > 0$ is constant.

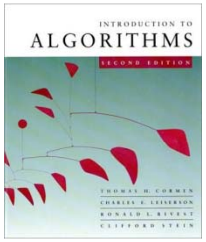
$$T(n)$$



Recursion tree

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





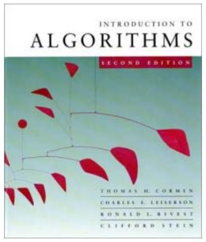
Recursion tree

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

dn

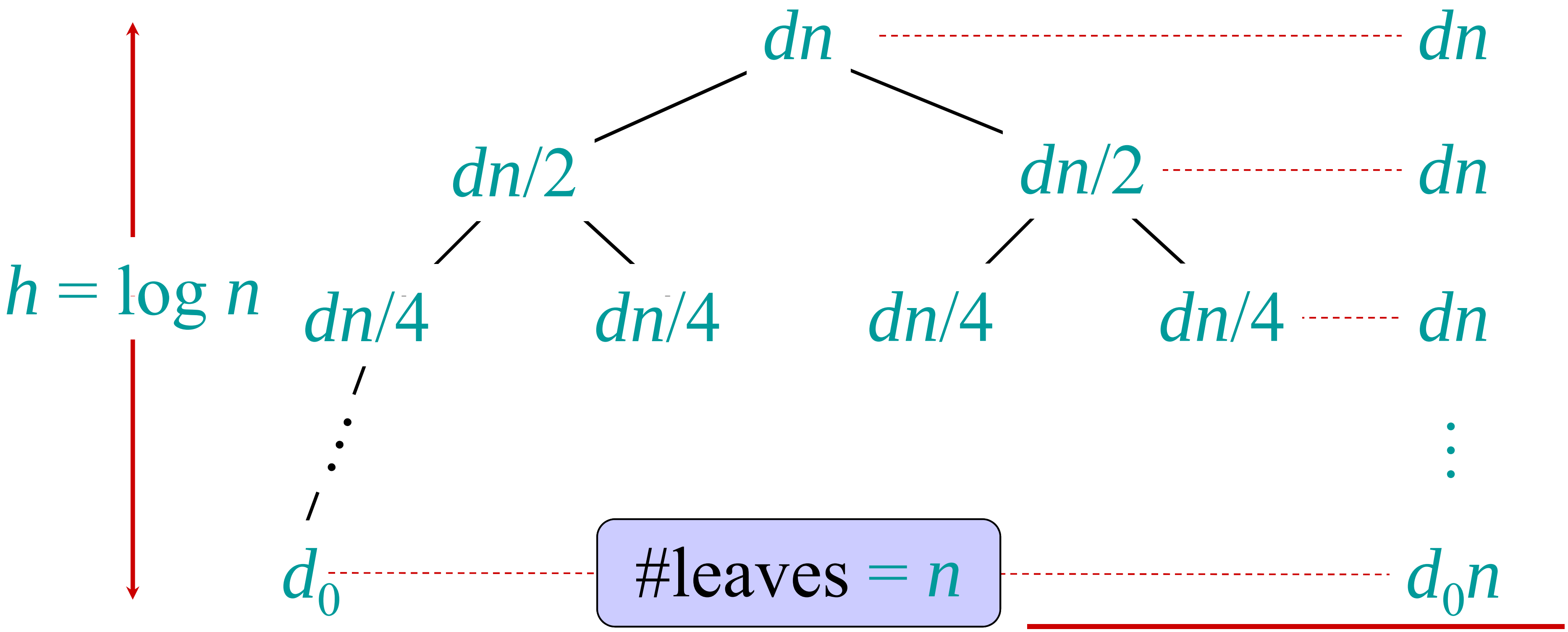
$dn/2$ $\qquad$ $dn/2$

$T(n/4)$ $\quad$ $T(n/4)$ $\quad$ $T(n/4)$ $\quad$ $T(n/4)$

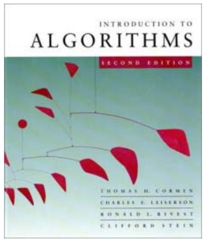


Recursion tree

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

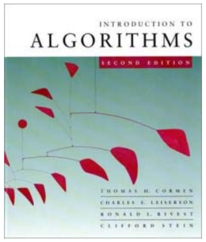


Total $dn \log n + d_0 n$



Conclusions

- Merge sort runs in $\Theta(n \log n)$ time.
- $\Theta(n \log n)$ grows more slowly than $\Theta(n^2)$.
- Therefore, merge sort asymptotically beats insertion sort in the worst case.
- In practice, merge sort beats insertion sort for $n > 30$ or so. (Why not earlier?)

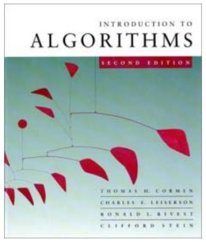


Recursion-tree method

- A recursion tree models the costs (time) of a recursive execution of an algorithm.
- The recursion-tree method can be unreliable, just like any method that uses ellipses (…).
- It is good for generating **guesses** of what the runtime could be.

But: Need to **verify** that the guess is right.

$\rightarrow$ Induction (substitution method)



Substitution method

The most general method to solve a recurrence (prove O and Ω separately):

1. ***Guess*** the form of the solution:
   (e.g. using recursion trees, or expansion)
2. ***Verify*** by induction (inductive step).
3. ***Solve*** for O-constants n_0 and c (base case of induction)