

CMPS 2200 – Fall 2012

Divide-and-Conquer

Carola Wenk

Slides courtesy of Charles Leiserson
with changes and additions by Carola Wenk

The divide-and-conquer design paradigm

- 1. *Divide*** the problem (instance) into subproblems of sizes that are fractions of the original problem size.
- 2. *Conquer*** the subproblems by solving them recursively.
- 3. *Combine*** subproblem solutions.

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

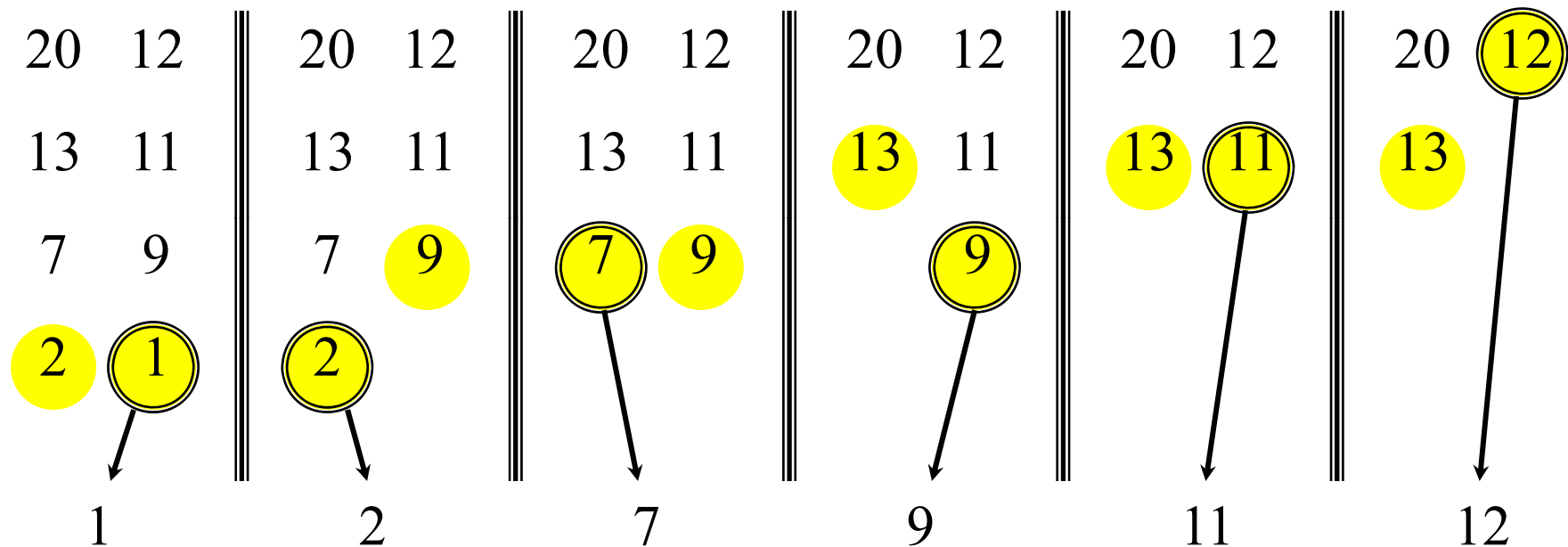
Merge sort

1. *Divide*: Trivial.
2. *Conquer*: Recursively sort 2 subarrays of size $n/2$
3. *Combine*: Linear-time key subroutine **MERGE**

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

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

Merging two sorted arrays



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

Analyzing merge sort

$T(n)$

d_0

$T(n/2)$

$T(n/2)$

dn

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

1. If $n = 1$, done.
2. **MERGE-SORT** ($A[1 \dots \lceil n/2 \rceil]$)
3. **MERGE-SORT** ($A[\lceil n/2 \rceil + 1 \dots n]$)
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 ...?

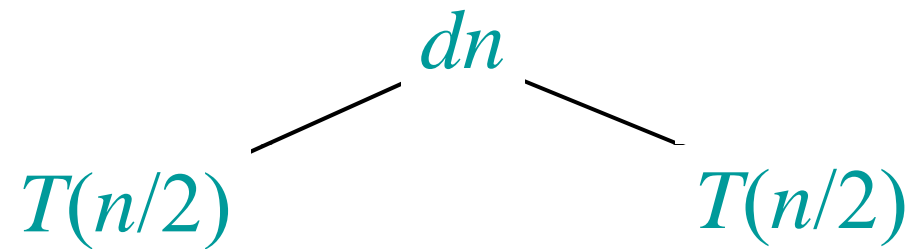
Recursion tree

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

$$T(n)$$

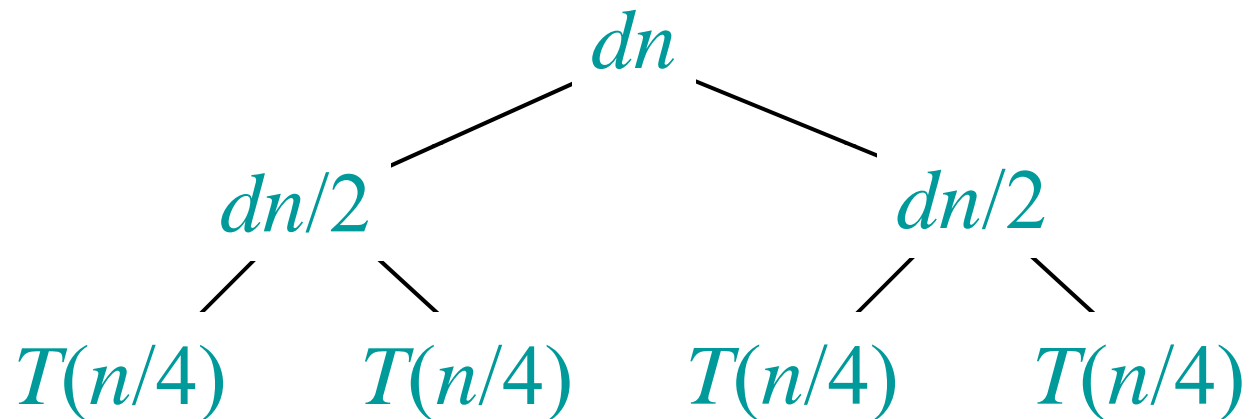
Recursion tree

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



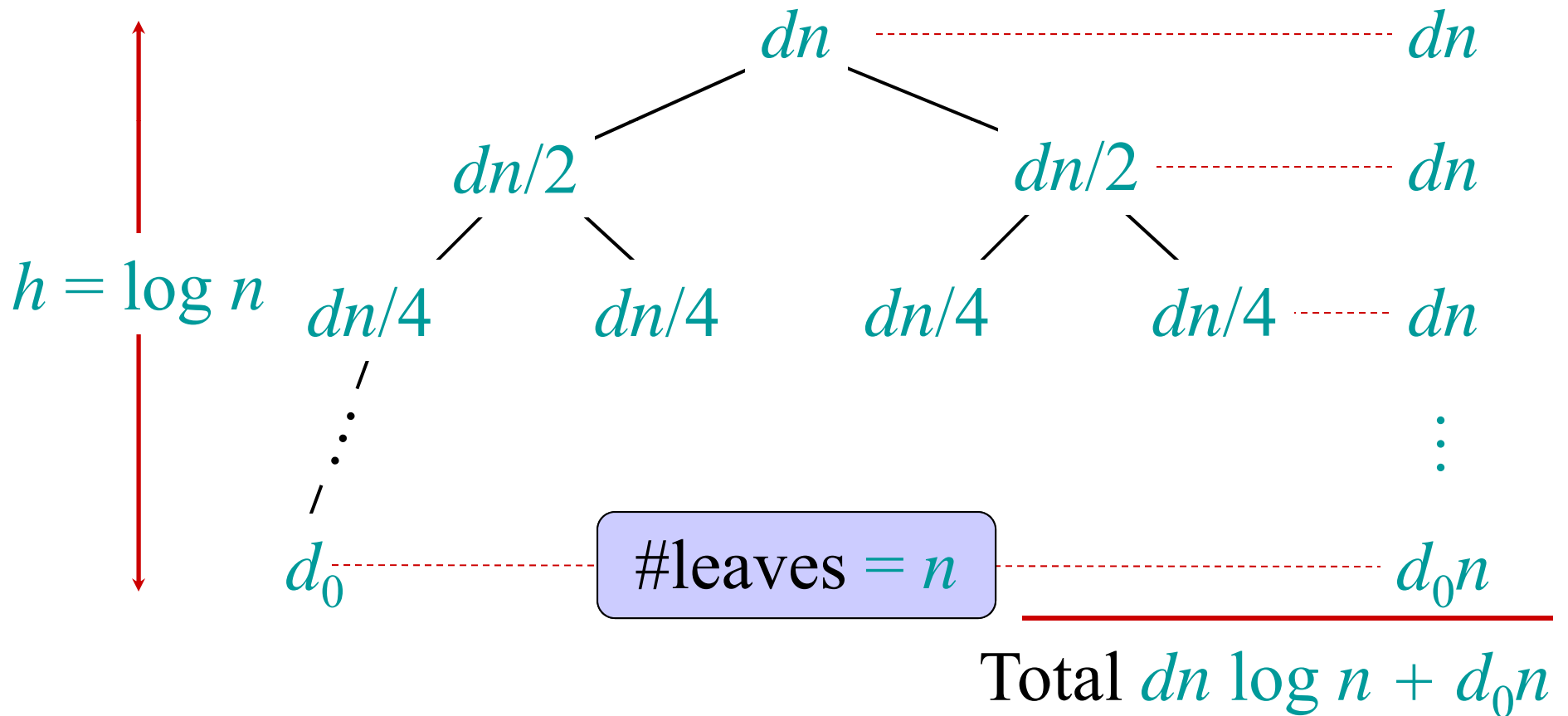
Recursion tree

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



Recursion tree

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



Mergesort 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 correct.
→ 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)