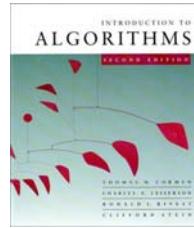




CS 3343 -- Spring 2009



Merge Sort

Carola Wenk

Slides courtesy of Charles Leiserson with small changes by Carola Wenk

1/27/09

CS 3343 Analysis of Algorithms

1



Merge sort

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

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

Key subroutine: MERGE

1/27/09

CS 3343 Analysis of Algorithms

2



Merging two sorted arrays

20 12

13 11

7 9

2 1

1/27/09

CS 3343 Analysis of Algorithms

3



Merging two sorted arrays

20 12

13 11

7 9

2 1

1

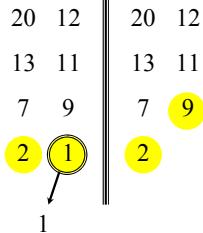
1/27/09

CS 3343 Analysis of Algorithms

4



Merging two sorted arrays



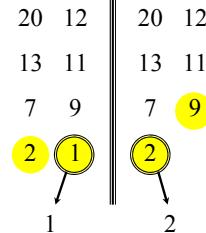
1/27/09

CS 3343 Analysis of Algorithms

5



Merging two sorted arrays



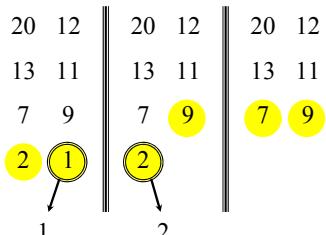
1/27/09

CS 3343 Analysis of Algorithms

6



Merging two sorted arrays



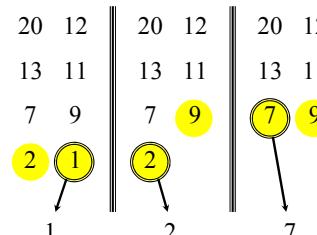
1/27/09

CS 3343 Analysis of Algorithms

7



Merging two sorted arrays



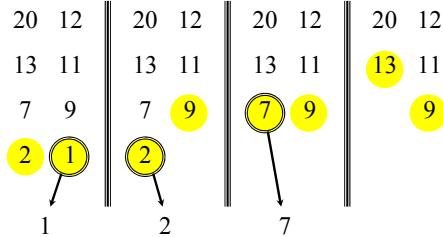
1/27/09

CS 3343 Analysis of Algorithms

8



Merging two sorted arrays



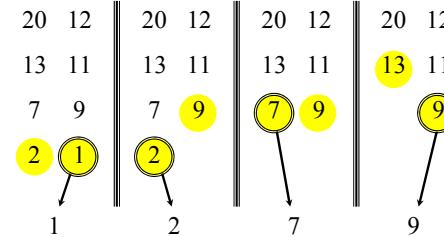
1/27/09

CS 3343 Analysis of Algorithms

9



Merging two sorted arrays



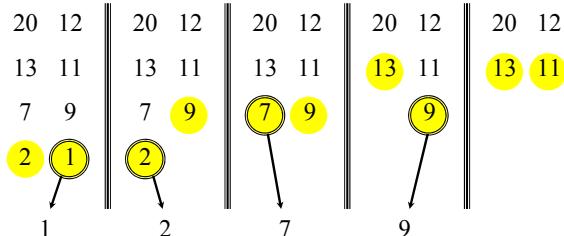
1/27/09

CS 3343 Analysis of Algorithms

10



Merging two sorted arrays



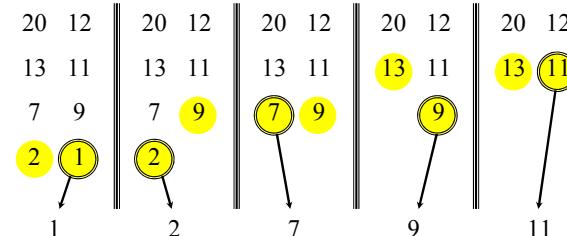
1/27/09

CS 3343 Analysis of Algorithms

11



Merging two sorted arrays



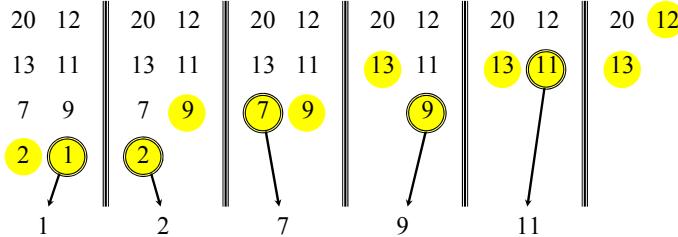
1/27/09

CS 3343 Analysis of Algorithms

12



Merging two sorted arrays



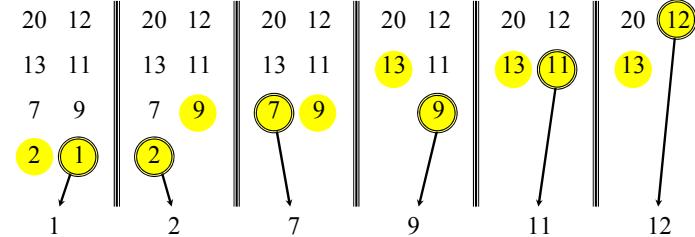
1/27/09

CS 3343 Analysis of Algorithms

13



Merging two sorted arrays



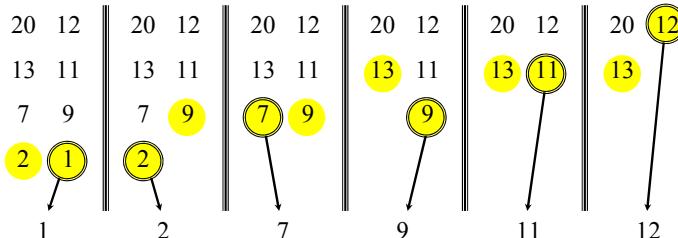
1/27/09

CS 3343 Analysis of Algorithms

14



Merging two sorted arrays



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

1/27/09

CS 3343 Analysis of Algorithms

15



Analyzing merge sort

$T(n)$ MERGE-SORT $A[1..n]$
 d_0 1. If $n = 1$, done.
 $2T(n/2)$ 2. Recursively sort $A[1..\lceil n/2 \rceil]$ and $A[\lceil n/2 \rceil + 1..n]$.
 dn 3. “Merge” the 2 sorted lists

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

1/27/09

CS 3343 Analysis of Algorithms

16



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 ...?

1/27/09

CS 3343 Analysis of Algorithms

17



Recursion tree

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

1/27/09

CS 3343 Analysis of Algorithms

18



Recursion tree

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

$T(n)$

1/27/09

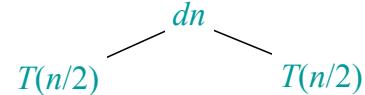
CS 3343 Analysis of Algorithms

19



Recursion tree

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



1/27/09

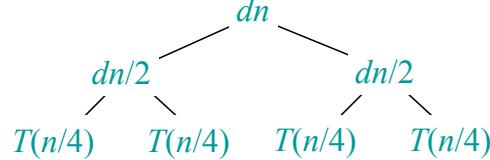
CS 3343 Analysis of Algorithms

20



Recursion tree

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



1/27/09

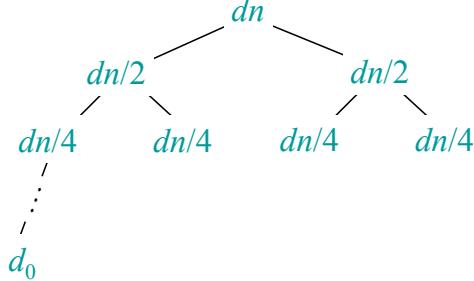
CS 3343 Analysis of Algorithms

21



Recursion tree

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



1/27/09

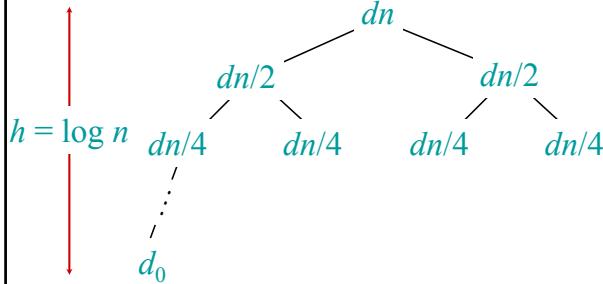
CS 3343 Analysis of Algorithms

22



Recursion tree

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



1/27/09

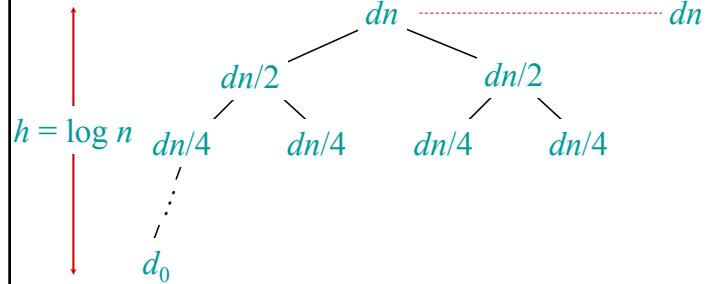
CS 3343 Analysis of Algorithms

23



Recursion tree

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



1/27/09

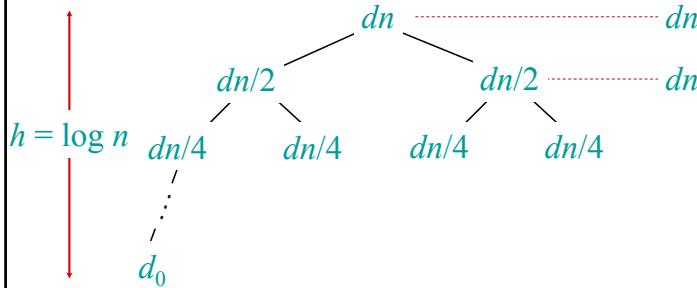
CS 3343 Analysis of Algorithms

24



Recursion tree

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



1/27/09

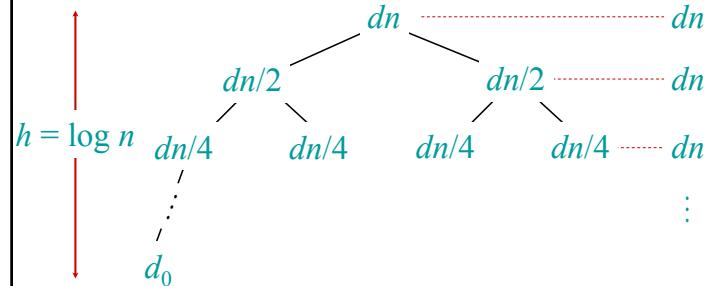
CS 3343 Analysis of Algorithms

25



Recursion tree

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



1/27/09

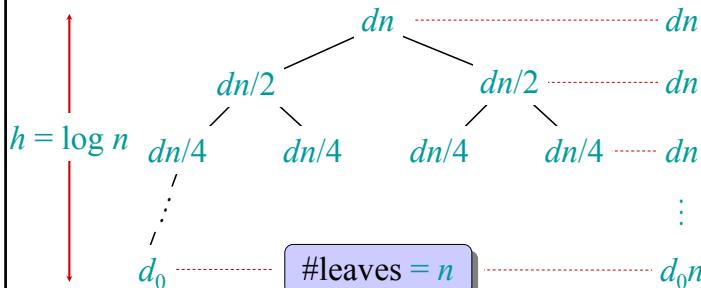
CS 3343 Analysis of Algorithms

26



Recursion tree

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



1/27/09

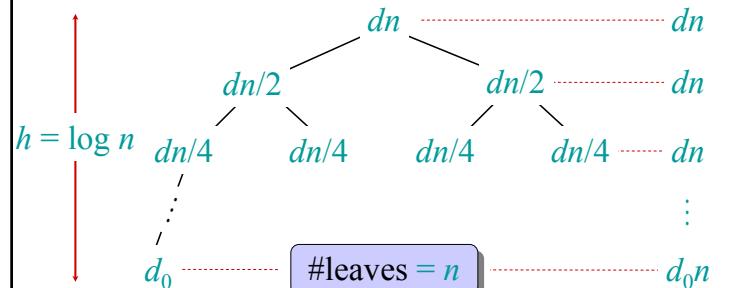
CS 3343 Analysis of Algorithms

27



Recursion tree

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



1/27/09

CS 3343 Analysis of Algorithms

28



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?)

1/27/09

CS 3343 Analysis of Algorithms

29



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.
→ Induction (substitution method)

1/27/09

CS 3343 Analysis of Algorithms

30



Substitution method

The most general method to solve a recurrence
(prove \mathcal{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 \mathcal{O} -constants n_0 and c (base case of induction)

1/27/09

CS 3343 Analysis of Algorithms

31