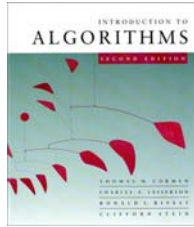




CS 5633 -- Spring 2010



Recurrences and Divide & Conquer

Carola Wenk

Slides courtesy of Charles Leiserson with small changes by Carola Wenk



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



Merging two sorted arrays

20 12

13 11

7 9

2 1



Merging two sorted arrays

20 12

13 11

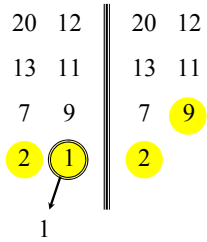
7 9

2 1

1



Merging two sorted arrays



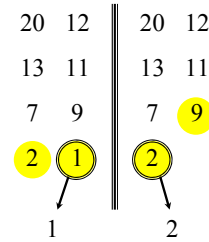
1/19/10

CS 5633 Analysis of Algorithms

5



Merging two sorted arrays



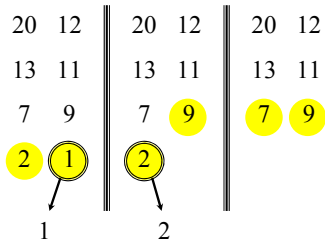
1/19/10

CS 5633 Analysis of Algorithms

6



Merging two sorted arrays



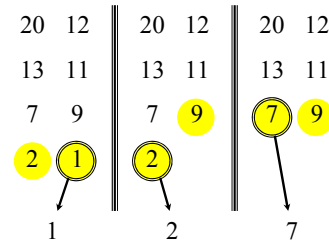
1/19/10

CS 5633 Analysis of Algorithms

7



Merging two sorted arrays



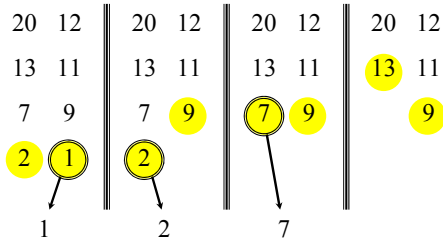
1/19/10

CS 5633 Analysis of Algorithms

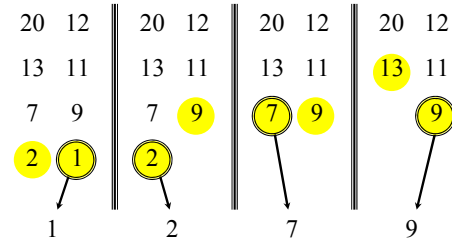
8



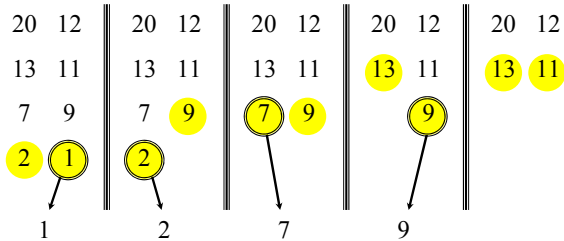
Merging two sorted arrays



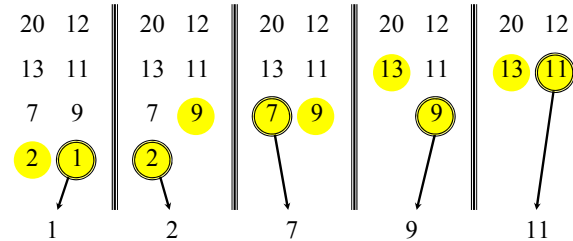
Merging two sorted arrays



Merging two sorted arrays

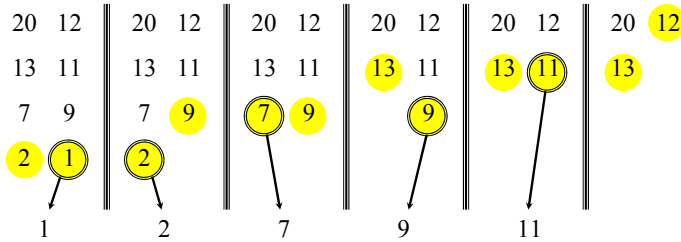


Merging two sorted arrays





Merging two sorted arrays



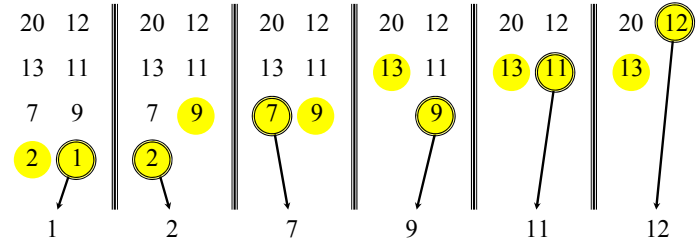
1/19/10

CS 5633 Analysis of Algorithms

13



Merging two sorted arrays



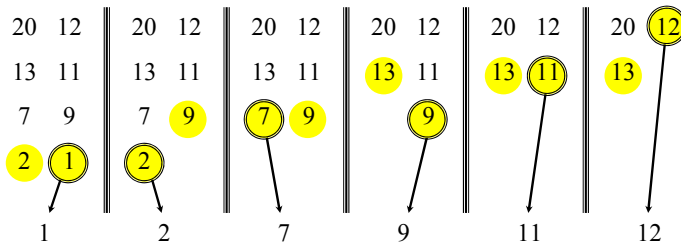
1/19/10

CS 5633 Analysis of Algorithms

14



Merging two sorted arrays



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

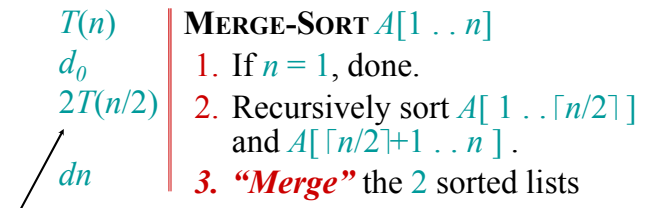
1/19/10

CS 5633 Analysis of Algorithms

15



Analyzing merge sort



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

1/19/10

CS 5633 Analysis of Algorithms

16



Recurrence for merge sort

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

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



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.



Example: merge sort

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

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

subproblems subproblem size work dividing and combining



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

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

subproblems subproblem size work dividing and combining



Recurrence for merge sort

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

- How do we solve $T(n)$? I.e., how do we find out if it is $O(n)$ or $O(n^2)$ or ...?



Recursion tree

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



Recursion tree

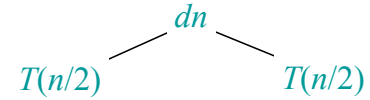
Solve $T(n) = 2T(n/2) + dn$, where $d > 0$ is constant.

$$T(n)$$



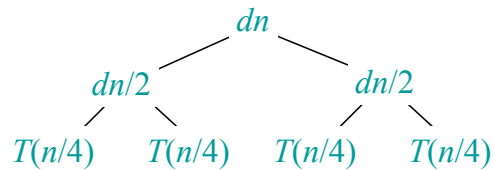
Recursion tree

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



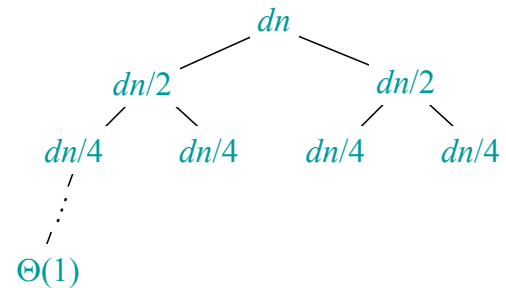
Recursion tree

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



Recursion tree

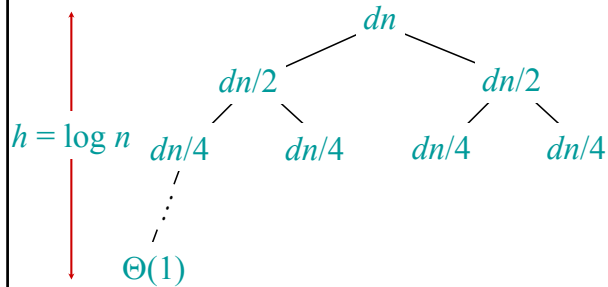
Solve $T(n) = 2T(n/2) + dn$, where $d > 0$ is constant.





Recursion tree

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



1/19/10

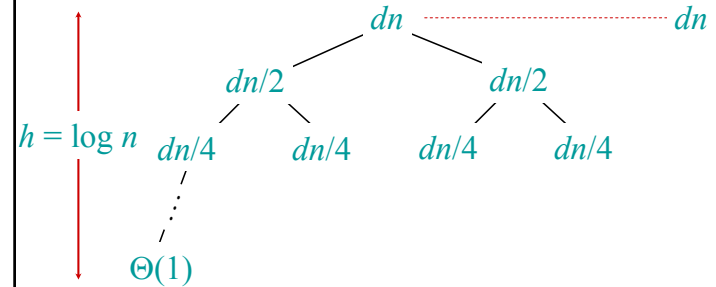
CS 5633 Analysis of Algorithms

33



Recursion tree

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



1/19/10

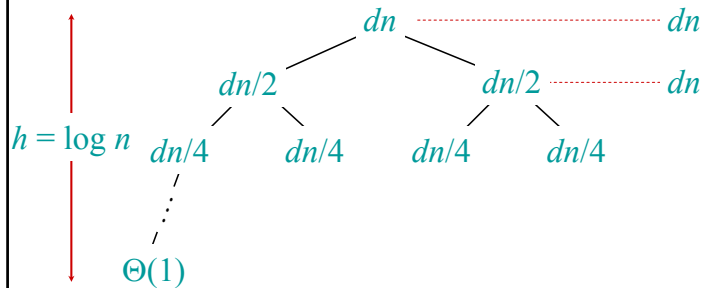
CS 5633 Analysis of Algorithms

34



Recursion tree

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



1/19/10

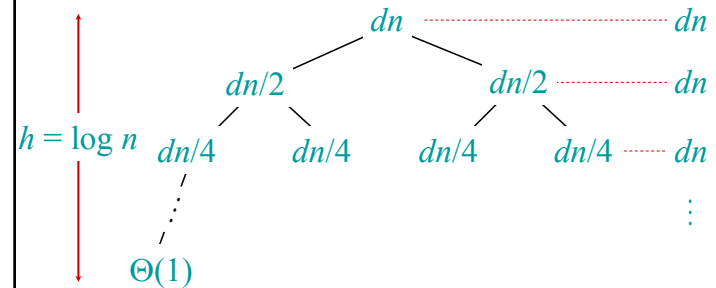
CS 5633 Analysis of Algorithms

35



Recursion tree

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



1/19/10

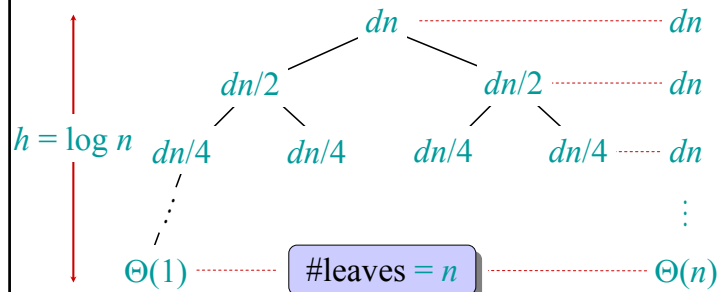
CS 5633 Analysis of Algorithms

36



Recursion tree

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



1/19/10

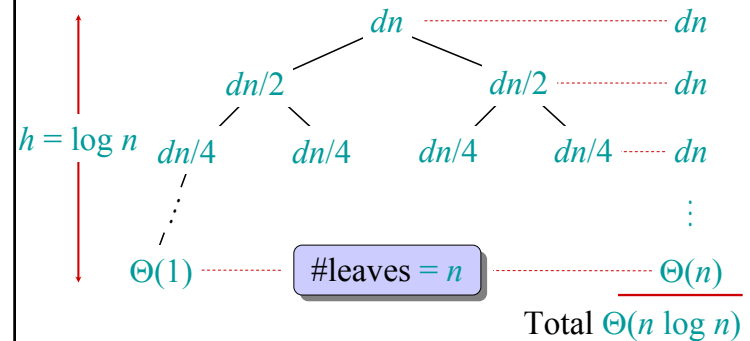
CS 5633 Analysis of Algorithms

37



Recursion tree

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



1/19/10

CS 5633 Analysis of Algorithms

38



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?)

1/19/10

CS 5633 Analysis of Algorithms

39



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)

1/19/10

CS 5633 Analysis of Algorithms

40



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)