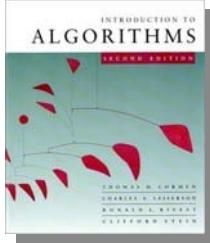


CS60020: Foundations of Algorithm Design and Machine Learning

Sourangshu Bhattacharya

DIVIDE AND CONQUER



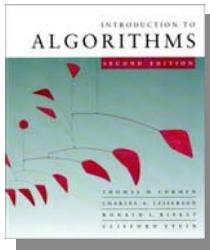
Merge sort

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

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

subproblems ↗
subproblem size ↗
work dividing
and combining

The equation $T(n) = 2 T(n/2) + \Theta(n)$ is displayed with three annotations. The term $2 T(n/2)$ is enclosed in a yellow oval, with an arrow pointing to it labeled "# subproblems". The term $\Theta(n)$ is also enclosed in a yellow oval, with an arrow pointing to it labeled "work dividing and combining". The label "subproblem size" is placed below the term $n/2$ in the original equation.



Master theorem

$$T(n) = a T(n/b) + f(n)$$

CASE 1: $f(n) = O(n^{\log_b a - \varepsilon})$, constant $\varepsilon > 0$

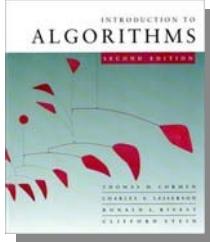
$$\Rightarrow T(n) = \Theta(n^{\log_b a}) .$$

CASE 2: $f(n) = \Theta(n^{\log_b a})$

$$\Rightarrow T(n) = \Theta(n^{\log_b a} \lg n) .$$

CASE 3: $f(n) = \Omega(n^{\log_b a + \varepsilon})$, constant $\varepsilon > 0$,
and regularity condition

$$\Rightarrow T(n) = \Theta(f(n)) .$$



Master theorem

$$T(n) = a T(n/b) + f(n)$$

CASE 1: $f(n) = O(n^{\log_b a - \varepsilon})$, constant $\varepsilon > 0$

$$\Rightarrow T(n) = \Theta(n^{\log_b a}) .$$

CASE 2: $f(n) = \Theta(n^{\log_b a})$

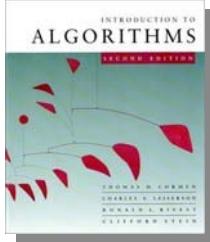
$$\Rightarrow T(n) = \Theta(n^{\log_b a} \lg n) .$$

CASE 3: $f(n) = \Omega(n^{\log_b a + \varepsilon})$, constant $\varepsilon > 0$,
and regularity condition

$$\Rightarrow T(n) = \Theta(f(n)) .$$

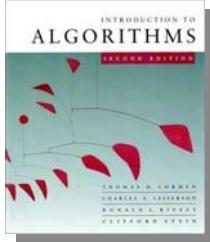
Merge sort: $a = 2$, $b = 2 \Rightarrow n^{\log_b a} = n^{\log_2 2} = n$

$$\Rightarrow \text{CASE 2} \qquad \Rightarrow T(n) = \Theta(n \lg n) .$$



Binary search

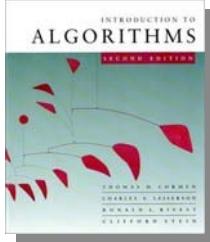
- Find an element in a sorted array:
 1. **Divide:** Check middle element.
 2. **Conquer:** Recursively search 1 subarray.
 3. **Combine:** Trivial.



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**

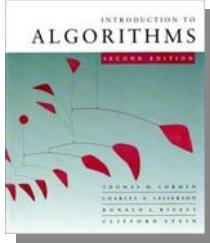
3 5 7 8 9 12 15



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

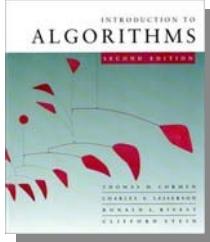


Binary search

- Find an element in a sorted array:

1. ***Divide:*** Check middle element.
 2. ***Conquer:*** Recursively search **1** subarray.
 3. ***Combine:*** Trivial.
- ***Example:*** Find **9**

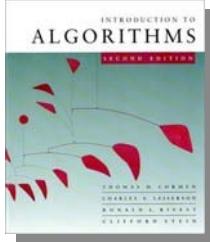
3 5 7 8 9 12 15



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**

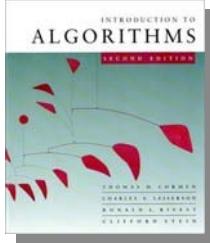




Binary search

- Find an element in a sorted array:
 1. ***Divide:*** Check middle element.
 2. ***Conquer:*** Recursively search **1** subarray.
 3. ***Combine:*** Trivial.
- ***Example:*** Find **9**

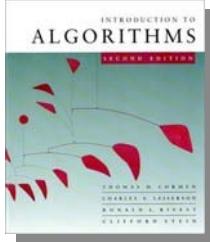




Binary search

- Find an element in a sorted array:
 1. *Divide*: Check middle element.
 2. *Conquer*: Recursively search 1 subarray.
 3. *Combine*: Trivial.
- *Example*: Find 9



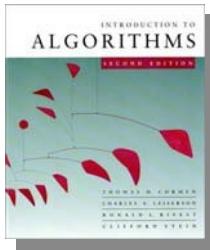


Recurrence for binary search

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

subproblems ↗
 ↓
 subproblem size
 ↗
 ↖
 work dividing
 and combining

The diagram illustrates the recurrence relation for binary search. The equation $T(n) = 1 T(n/2) + \Theta(1)$ is shown with three yellow circles highlighting the terms $T(n/2)$, $\Theta(1)$, and the constant 1. Arrows point from the text labels to these highlighted terms: one arrow from "# subproblems" points to the term 1, another from "subproblem size" points to the term $T(n/2)$, and a third from "work dividing and combining" points to the term $\Theta(1)$.



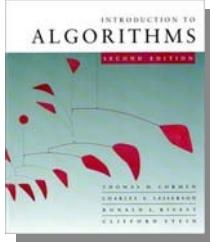
Recurrence for binary search

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

subproblems ↗
 ↓
 subproblem size

work dividing
and combining ↙

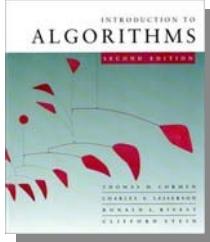
$$n^{\log_b a} = n^{\log_2 1} = n^0 = 1 \Rightarrow \text{CASE 2 } (k = 0)$$
$$\Rightarrow T(n) = \Theta(\lg n).$$



Powering a number

Problem: Compute a^n , where $n \in \mathbb{N}$.

Naive algorithm: $\Theta(n)$.



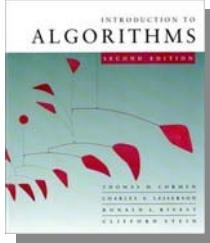
Powering a number

Problem: Compute a^n , where $n \in \mathbb{N}$.

Naive algorithm: $\Theta(n)$.

Divide-and-conquer algorithm:

$$a^n = \begin{cases} a^{n/2} \cdot a^{n/2} & \text{if } n \text{ is even;} \\ a^{(n-1)/2} \cdot a^{(n-1)/2} \cdot a & \text{if } n \text{ is odd.} \end{cases}$$



Powering a number

Problem: Compute a^n , where $n \in \mathbb{N}$.

Naive algorithm: $\Theta(n)$.

Divide-and-conquer algorithm:

$$a^n = \begin{cases} a^{n/2} \cdot a^{n/2} & \text{if } n \text{ is even;} \\ a^{(n-1)/2} \cdot a^{(n-1)/2} \cdot a & \text{if } n \text{ is odd.} \end{cases}$$

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