CP312
Algorithm Design and Analysis I

LECTURE 6: DIVIDE-AND-CONQUER

Divide-and-Conquer Algorithms

1. **Divide** the problem (instance) into subproblems
2. **Conquer** the subproblems by solving them recursively
3. **Combine** subproblem solutions

$$T(n) = aT\big(s(n)\big) + C(n) + D(n)$$

Running time on input size n

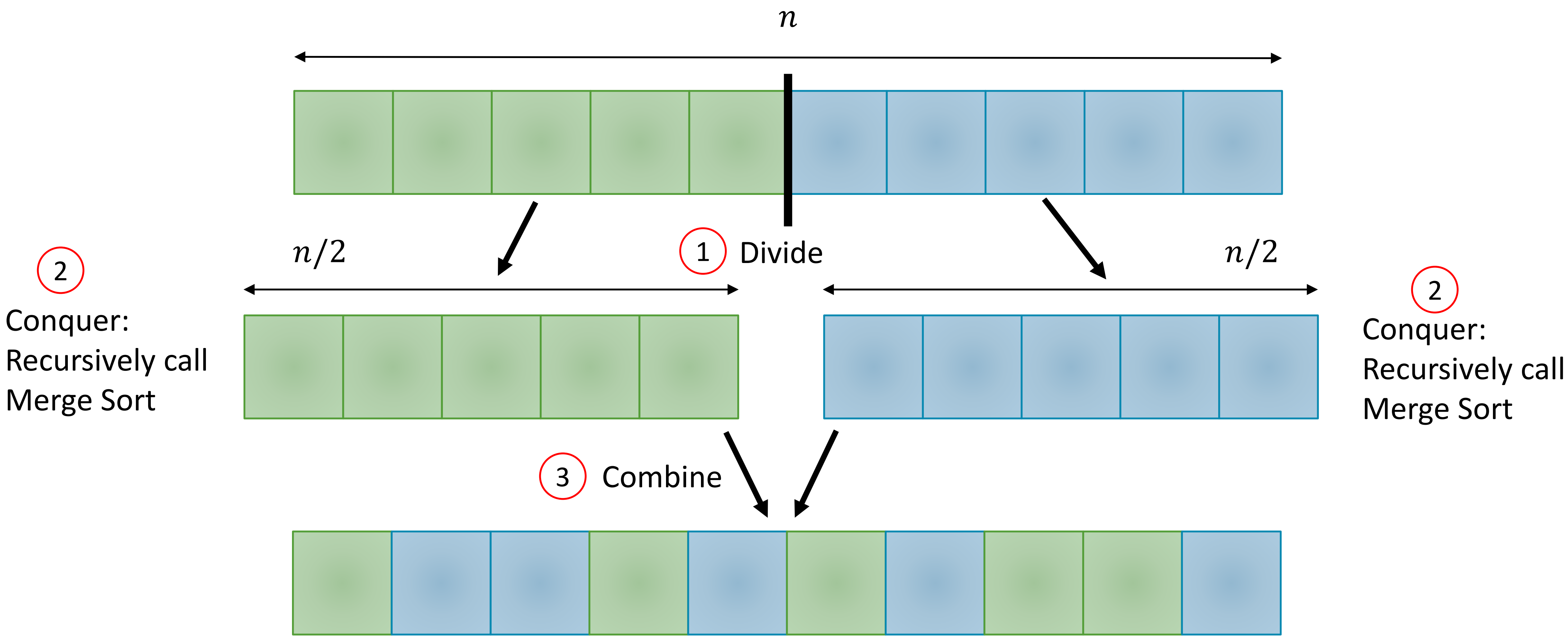
# of subproblems

Subproblem size

Work combining

Work dividing

Example: Merge Sort



Example: Merge Sort

- **Divide**: Split the array into 2 sub-arrays
- **Conquer:** Recursively sort the 2 sub-arrays
- **Combine**: Merge the two sorted sub-arrays

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

Work dividing

Running time of merge sort on input size n

# of subproblems

Subproblem size

Work combining

Master Theorem (Review)

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

Case 1: $f(n) = O\left(n^{\log_b a-\epsilon}\right) \Rightarrow \boldsymbol{T(n) = \Theta\left(n^{\log_b a}\right)}$

Case 2: $f(n) = \Theta\left(n^{\log_b a} \lg^k n\right) \Rightarrow \boldsymbol{T(n) = \Theta\left(n^{log_b a} \lg^{k+1} n\right)}$

Case 3: $f(n) = \Omega\left(n^{\log_b a+\epsilon}\right)$ and $af(n/b) \leq cf(n) \Rightarrow \boldsymbol{T(n) = \Theta(f(n))}$

Where $\epsilon > 0, k \geq 0$

Case 2: $f(n) = \Theta\left(n^{\log_b a} \lg^k n\right) \Rightarrow \boldsymbol{T(n) = \Theta\left(n^{log_b a} \lg^{k+1} n\right)}$

Merge Sort

- Solving $T(n) = 2T(n/2) + \Theta(n)$ using Master Theorem:
- $a = 2, b = 2 \Rightarrow n^{\log_b a} = n$

- **Case 2** $(k = 0)$: $T(n) = \Theta(n \lg n)$

Binary Search

- **Problem**: Find an element in a sorted array.
- **Input:** Sorted array $A[1, \ldots, n]$ and target element x
- **Output:** Location of x if it exists in A or -1 otherwise.

3	5	7	8	12	14	15

Binary Search (Find $x = 7$)

- **Divide**: Check the middle element
- **Conquer:** Recursively search 1 sub-array
- **Combine**: Do nothing

3	5	7	8	12	14	15

Binary Search (Find $x = 7$)

- **Divide**: Check the middle element
- **Conquer:** Recursively search 1 sub-array
- **Combine**: Do nothing

3	5	7	8	12	14	15

Binary Search (Find $x = 7$)

- **Divide**: Check the middle element
- **Conquer:** Recursively search 1 sub-array
- **Combine**: Do nothing

3	5	7	8	12	14	15

Recurrence for Binary Search

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

Work dividing

# of subproblems

Subproblem size

Work combining

- $a = 1, b = 2$
- $n^{\log_b a} = n^{\log_2 1} = n^0 = 1$ vs. $f(n) = \Theta(1)$
- **Case 2** ($\boldsymbol{k = 0}$): $f(n) = \Theta\left(n^{\log_b a} \lg^k n\right) = \Theta\left(1. \lg^0 n\right) = \Theta(1)$
- Thus, $T(n) = \Theta(\lg n)$

Powering a Number

- **Problem**: Compute a^n where $n \in \mathbb{N}$
- **Input:** A real constant-sized number $a \in R$ and exponent n
- **Output:** a^n

- Naïve Algorithm: $a \times a \times \cdots \times a$ Time $= \Theta(n)$
- Can we do better?

Powering a Number: Divide & Conquer

- Using Divide & Conquer, we can first write a^n as follows:

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

- **Divide**: Divide n by 2
- **Conquer:** Recursively exponentiate the smaller terms
- **Combine**: Square the result of the sub-problem (and multiply by a only if n is odd)

Powering a Number: D&C Pseudocode

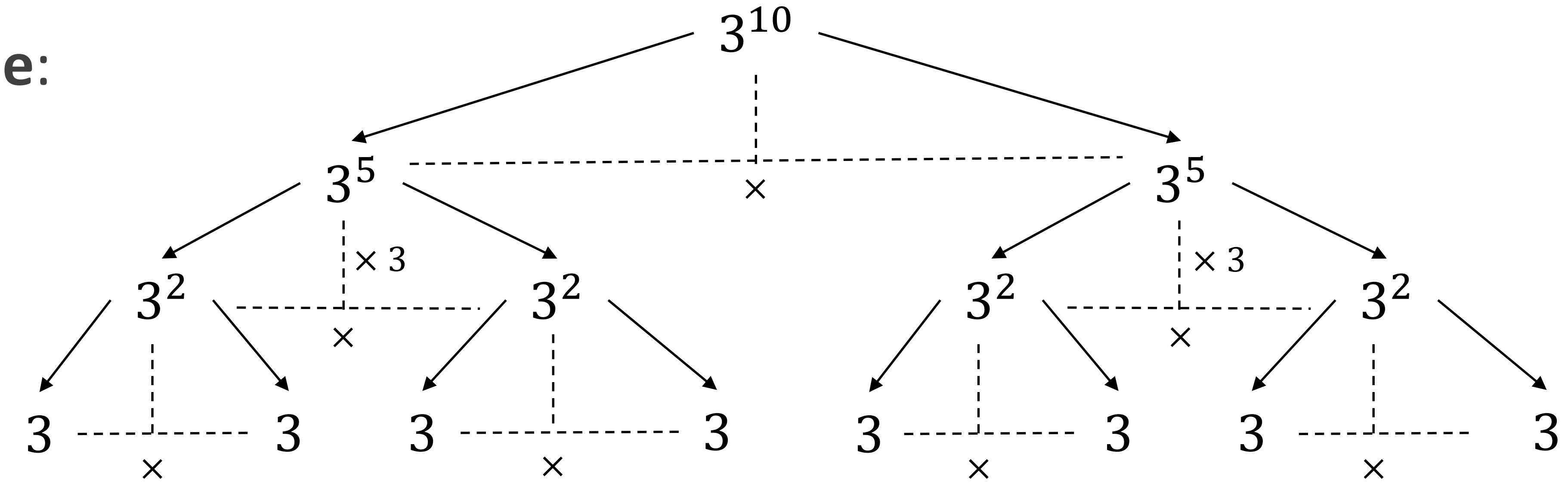
$$a^n = \begin{cases} a^{n/2} . a^{n/2} & \text{if } n \text{ is even} \\ a^{(n-1)/2} . a^{(n-1)/2} . a & \text{if } n \text{ is odd} \end{cases}$$

Pow(a, n):
    if n is even **then**
        $k = n/2$ // Divide
        $r = \text{Pow}(a, k)$ // Conquer
        return $r \times r$ // Combine
    else if n is odd **then**
        $k = \frac{n-1}{2}$ // Divide
        $r = \text{Pow}(a, k)$ // Conquer
        return $r \times r \times a$ // Combine

Powering a Number: Divide & Conquer

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

- **Example**:



Powering a Number: Running Time

- $T(n) = T(n/2) + \Theta(1) + \Theta(1)$
- Using Master Method
  - $f(n) = \Theta(1)$ $\qquad n^{\log_1 2} = 1$
  - Looks like Case 2 so let us try it first:
  - Is $f(n) = \Theta\left(n^{\log_1 2} \lg^k n\right)$?
    - Yes, for $k = 0$
  - So, it is **Case 2**
  - Therefore:

$$T(n) = \Theta(\lg n)$$

Pow(a, n):
    if n is even **then**
        $k = n/2$ // Divide $\Theta(1)$
        $r = \text{Pow}(a, k)$ // Conquer $T(n/2)$
        return $r \times r$ // Combine $\Theta(1)$
    else if n is odd **then** **// OR**
        $k = \frac{n-1}{2}$ // Divide $\Theta(1)$
        $r = \text{Pow}(a, k)$ // Conquer $T(n/2)$
        return $r \times r \times a$ // Combine $\Theta(1)$

Conclusion

- Divide-and-Conquer is just one of several powerful techniques for algorithm design.
- Divide-and-Conquer algorithms can be analyzed using recurrences and the master method (so practice this math).
- Can lead to more efficient algorithms.