

# CPSC 510: Algorithms

## Unit 2: Divide and Conquer Algorithms

Reading: Ch. 5

# Outline

- **Divide and Conquer Algorithms**
- Analyzing Divide and Conquer Algorithms
- Examples

# Divide and Conquer

- **Divide and conquer** is a common algorithm design technique.
- It has the following three steps:
  - 1. Divide** the problem into a number of subproblems that are smaller instances of the same problem.
  - 2. Conquer** the subproblems by solving them recursively.
    - For small subproblems sizes, simply solve the subproblems directly.
  - 3. Combine** the solutions to the subproblems into the solution for the original problem.

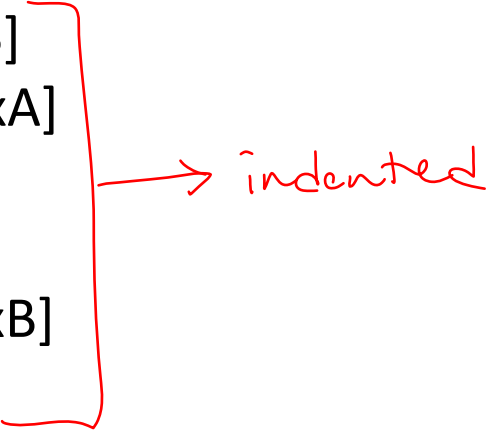
# Example: Merge Sort

- Divide: Divide the  $n$  element array into two subarrays of  $n/2$  elements.
- Conquer: Sort the two subarrays recursively using merge sort.
  - For subproblems of size one or less, the subarray is trivially sorted.
- Combine: Merge the two sorted subarrays to produce a single sorted array.

# Example: Merge Algorithm of Merge Sort

MERGE (A, B)

```
1  append  $\infty$  to end of A and B // A[A.length+1] = B[B.length+1] =  $\infty$ 
2  let C be new array of length A.length + B.length
3  let indexA = 1, indexB = 1
4  foreach indexC from 1 to C.length
5  if A[indexA] < B[indexB]
6      C[indexC] = A[indexA]
7      indexA++
8  else
9      C[indexC] = B[indexB]
10     indexB++
11 return C
```



# Outline

- Divide and Conquer Algorithms
- **Analyzing Divide and Conquer Algorithms**
- Examples

# Divide and Conquer Recurrence

- To analyze the running time of a divide and conquer algorithm, you can use the following recurrence:

$$T(n) = \begin{cases} \Theta(1) & \text{if } n \leq n' \\ aT(n/b) + D(n) + C(n) & \text{otherwise} \end{cases}$$

where...

- $a$  is the number of subproblems
- $n / b$  is the size of each subproblem
- $n'$  is the maximum size of a subproblem where a straightforward constant-time solution (of time  $c$ ) is used instead of recursion
- $D(n)$  is the time it takes to divide the problem into subproblems
- $C(n)$  is the time it takes to combine the solutions into the final answer

# Example: Merge Sort

- What are the values of the constants for merge sort?

$$T(n) = \begin{cases} \Theta(1) & \text{if } n \leq n' \\ aT(n/b) + D(n) + C(n) & \text{otherwise} \end{cases}$$

- $a = 2$

- $b = 2$

- $n' = 1$

- $D(n) = \Theta(1)$

- $C(n) = \Theta(n)$

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

# Solving Recurrences

- **Iteration method:** Expand the terms of the recursion.
- **Substitution method:** Use proof by induction to verify a solution.
- **Master method:** Returns a big-O solution of divide and conquer recurrences.

# Iteration Method

- The **iteration method** expands the terms of a the recursion with the goal of expression in a summation that further can be simplified.
- To make the math easier, we are going to assume..
  - The array size is an even power of 2. In other words, there exists an integer  $k$  such that  $n = 2^k$ 
    - Otherwise, the actual merge sort recurrence would be:
$$T(n) = T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + \Theta(n)$$
  - The time to merge the elements is  $cn$  where  $c$  is a positive constant. This replaces  $\Theta(n)$  in the recurrence.
- Neither simplification will not affect the final answer since we are only looking for a rate-of-growth in  $\Theta$ -notation.

# Iteration Method Example

- Expand the merge sort recurrence:

$$T(n) = 2T(n/2) + cn$$

$$= 2(2T(n/4) + c(n/2)) + cn$$

$$= 4T(n/4) + 2cn$$

$$= 4(2T(n/8) + c(n/4)) + 2cn$$

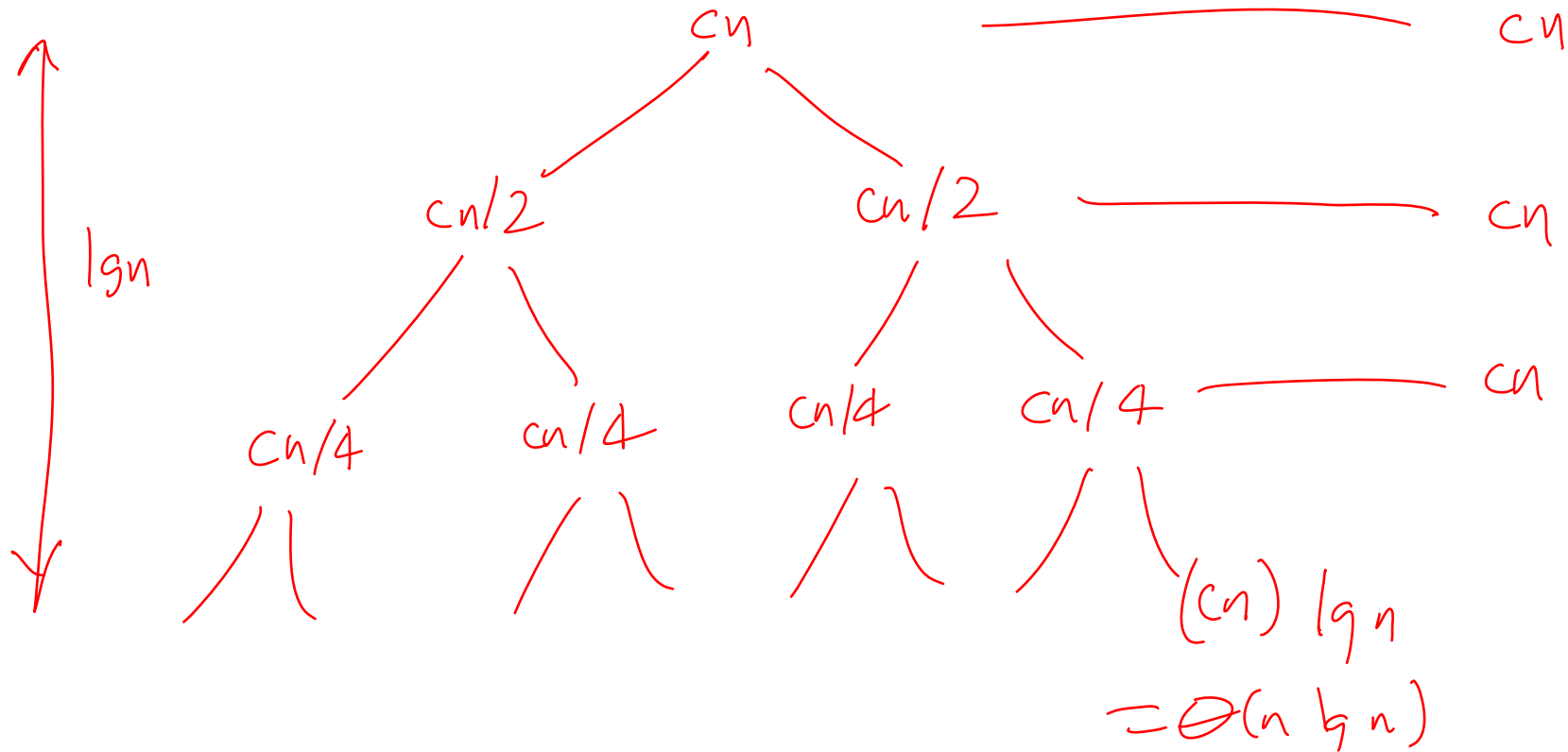
$$= 8T(n/8) + 3cn$$

$$\dots$$
$$= 16T(n/16) + 4cn$$

$$= \sum_{j=0}^{\lg n} cn = cn(\lg n) = \Theta(n \lg n)$$

# Recursion Tree

- A graphical way of expanding this sum is to use a **recursion tree**.



# Substitution Method

- The **substitution method** uses proof by induction to verify answers obtained using the iteration method or by simply guessing.
- Helpful in cases where you want to prove similar equations (such as the actual merge sort recurrence with floors and ceilings).
- To prove that  $T(n) = \Theta(f(n))$ , need to prove that  $T(n) = O(f(n))$  and  $T(n) = \Omega(f(n))$ .

# Substitution Method Example

- Prove that the merge sort recurrence is  $O(n \lg n)$  for  $n \geq 2$ .

# Master Method

- The **master method** can also be used to solve divide and conquer recurrences.
- Recurrence must be of the form  $T(n) = aT(n/b) + f(n)$ 
  - where  $a \geq 1$ ,  $b > 1$ , and  $f(n)$  is asymptotically positive.
  - $n/b$  can represent  $\lfloor n/b \rfloor$  and/or  $\lceil n/b \rceil$
- Only returns an answer in big-O notation, does not give a precise solution.
- For divide and conquer recurrences,  $f(n)$  is selected, usually using the most simple function, such that  $D(n) + C(n) = \Theta(f(n))$ .

# Master Method

The master method requires comparing  $f(n)$  to  $n^{\log_b a}$  to determine if it grows at a smaller, faster, or similar rate. This results in three cases:

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

↑                    ↑

1.  $f(n) = O(n^{\log_b a - \epsilon})$  for some constant  $\epsilon > 0$

- $f(n)$  grows polynomially slower than  $n^{\log_b a}$  (by an  $n^\epsilon$  factor).
- Solution:  $T(n) = \Theta(n^{\log_b a})$

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

- $f(n)$  and  $n^{\log_b a}$  grow at similar rates.
- Solution:  $T(n) = \Theta(n^{\log_b a} \lg n)$

3.  $f(n) = \Omega(n^{\log_b a + \epsilon})$  for some constant  $\epsilon > 0$  and that  $af(n/b) \leq cf(n)$  for some constant  $c < 1$  and all sufficiently large  $n$ .

- $f(n)$  grows polynomially faster than  $n^{\log_b a}$  (by an  $n^\epsilon$  factor)
- Solution:  $T(n) = \Theta(f(n))$

# Master Method Examples

- Examples: Use the master method to find the rate-of-growth using  $\Theta$ -notation for each of the following recurrences.

a.  $T(n) = 2T(n/2) + n$  (merge sort)

$$a = 2, \quad b = 2, \quad f(n) = n$$

$$n^{\log_b a} = n^{\log_2 2} = n^1 = n$$

$$\text{Case 2: } T(n) = \Theta(n^{\log_b a} \lg n) = \Theta(n \lg n)$$

# Master Method Examples

b.  $T(n) = 16T(n/4) + n$

$a=16, b=4, f(n)=n$

$n^{\log_b a} = n^{\log_4 16} = n^2$

Case 1:  $f(n) = O(n^{2-\epsilon})$  where  $\epsilon=1 \Rightarrow T(n) = \Theta(n^{\log_b a}) = \Theta(n^2)$

# Master Method Examples

$$c. T(n) = 3T(n/4) + n \lg n$$

$$a = 3, \quad b = 4, \quad f(n) = n \lg n$$

$$n^{\log_b a} = n^{\log_4 3} = n^{0.793} < n < n \lg n$$

Case 3:

Show regularity condition  $3(n/4) \lg(n/4) \leq c n \lg n$

Choose  $c = 3/4$

$$3/4 n \lg(n/4) \leq 3/4 n \lg n$$

$$T(n) = \theta(f(n)) = \theta(n \lg n)$$

# Master Method Examples

d.  $T(n) = 9T(n/3) + n^2 + 4$

$$a = 9, \quad b = 3, \quad f(n) = n^2 + 4$$

$$n^{\log_b a} = n^{\log_3 9} = n^2$$

$$T(n) = \Theta(n^{\log_b a} \lg n) = \Theta(n^2 \lg n)$$

Case 2

# Outline

- Divide and Conquer Algorithms
- Analyzing Divide and Conquer Algorithms
- **Examples**

# Example 1

A small investment company is doing a simulation in which they look at  $n$  consecutive days of a given stock in the past. Number the days  $i = 1, 2, \dots, n$  and for each day  $i$ , the price of the stock is represented using  $p(i)$  and is assumed to be fixed during the course of a single day. During the time period, they wanted to buy 1,000 shares on some day and sell all these shares on some later day. What is the most amount of money that could be made during these  $n$  days (noting it could be zero if the stock price never increases during the interval)?

# Example 1 Algorithm

FIND\_PROFIT(P)

- 1 if P.length == 1
- 2     return 0
- 3 if P.length == 2
- 4     if P[0] < P[1] return P[1] - P[0]
- 5     else return 0
- 6 divide P into two halves P<sub>1</sub> and P<sub>2</sub>
- 7 let profit<sub>1</sub> = FIND\_PROFIT(P<sub>1</sub>)
- 8 let profit<sub>2</sub> = FIND\_PROFIT(P<sub>2</sub>)
- 9 let min<sub>1</sub> = MIN(P<sub>1</sub>)
- 10 let max<sub>2</sub> = MAX(P<sub>2</sub>)
- 11 return MAX(profit<sub>1</sub>, profit<sub>2</sub>, max<sub>2</sub> - min<sub>1</sub>, 0)

$$T(n) = \begin{cases} \Theta(1) & \text{for } n \leq 2 \\ 2T(n/2) + \Theta(n) & \text{otherwise,} \end{cases}$$

↑

$$\Theta(n \lg n)$$

---

$$T(n) = \begin{cases} \Theta(1) & \text{for } n \leq 2 \\ 2T(n/2) + \Theta(1) & \text{otherwise} \end{cases}$$

$$\Theta(n)$$

## Example 2

Two separate databases contain  $n$  numerical values for a total of  $2n$  values. Assume that no two values are the same. The only way to access the database is through queries to the database. In a query, you can specify a value  $k$  to one of the two databases and the chosen database will return the  $k$ th smallest value that it contains. Develop an algorithm that finds the median of this set of  $2n$  values. Since the number of values is even, define the median to be the  $n$ th smallest value (the smaller of the two values in the middle).

# Example 2 Algorithm

Let  $A$  and  $B$  be arrays representing databases  $A$  and  $B$  respectively where the items are sorted in ascending order. A query is represented using the notation  $A(k)$  where  $A(1)$  has the smallest value.

$\text{MEDIAN}(n, a, b)$  takes integers  $n$ ,  $a$ , and  $b$  and finds the median of  $A[a+1..a+n]$  and  $B[b+1..b+n]$ .

$\text{MEDIAN}(n, a, b)$

```
1  if  $n = 1$ 
2    return  $\min(A(a+1), B(b+1))$ 
3  let  $k = \lfloor n/2 \rfloor$ 
4  if  $A(a+k) < B(b+k)$ 
5    then return  $\text{MEDIAN}(k, a + \lfloor n/2 \rfloor, b)$ 
6  else return  $\text{MEDIAN}(k, a, b + \lfloor n/2 \rfloor)$ 
```

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

$\Theta(\lg n)$