

# 3. Recurrences and Complexity Analysis

# Goals

- Understand the relationship between recursive algorithms and recurrences.
- Learn asymptotic analysis of recurrences.

# Recurrence

- recurrence
  - Equation or inequality that describes a function in terms of its values on smaller inputs
- examples
  - $a_n = a_{n-1} + 2$
  - $f(n) = n f(n-1)$
  - $f(n) = f(n-1) + f(n-2)$
  - $f(n) = f(n/2) + n$

# Time Complexity of merge sort

```
mergeSort(A[ ], p, r)    ▷ sort A[p ... r]
{
    if (p < r) then {
        q ← ⌊(p + r)/2⌋; ----- ①    ▷ mid point
        mergeSort(A, p, q); ----- ②    ▷ sort left half
        mergeSort(A, q+1, r); ----- ③    ▷ sort right half
        merge(A, p, q, r); ----- ④    ▷ merge
    }
}

merge(A[ ], p, q, r)
{
    Combine two sorted arrays A[p ... q] and A[q+1 ... r]
    into one sorted array A[p ... r].
}
```

Recurrence for time:  $T(n) = 2T(n/2) + \text{overhead (time for ① and ④: } cn)$

# Asymptotic Analysis of Recurrences

- Repeated replacement
  - Repeatedly replace by functions on smaller inputs
- Guess and prove
  - Guess an answer and prove that it is correct by mathematical induction
- Master theorem
  - Theorem for general classes of recurrences

# Repeated Replacement

$$T(n) = T(n-1) + c$$

$$T(1) \leq c$$

$$\begin{aligned} T(n) &= T(n-1) + c \\ &= (T(n-2) + c) + c = T(n-2) + 2c \\ &= (T(n-3) + c) + 2c = T(n-3) + 3c \\ &\dots \\ &= T(1) + (n-1)c \\ &\leq c + (n-1)c \\ &= cn \end{aligned}$$

# Repeated Replacement

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

$$T(1) = 1$$

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

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

$$= 2^2(2T(n/2^3) + n/2^2) + 2n = 2^3T(n/2^3) + 3n$$

...

$$= 2^kT(n/2^k) + kn$$

$$= n + n \log n$$

$$= \Theta(n \log n)$$

# Guess and Prove

$$T(n) = 2T(n/2) + n, \quad T(1) = 1.$$

Guess:  $T(n) = O(n \log n)$ , that is,  $T(n) \leq cn \log n$

<Proof> (mathematical induction) Assume it holds for  $n' < n$ .

$$\begin{aligned} T(n) &= 2T(n/2) + n \\ &\leq 2c(n/2)\log(n/2) + n \\ &= cn \log n - cn \log 2 + n \\ &= cn \log n + (-c \log 2 + 1)n \\ &\leq cn \log n \quad \longleftarrow \text{There exists constant } c \geq 1/\log 2 \end{aligned}$$

Induction basis?

# Guess and Prove

$$T(n) = 2T(n/2) + 1, \quad T(1) = 1$$

Guess:  $T(n) = O(n)$ , that is,  $T(n) \leq cn$

<Proof> Assume it holds for  $n' < n$ .

$$\begin{aligned} T(n) &= 2T(n/2) + 1 \\ &\leq 2c(n/2) + 1 && \longleftarrow \text{induction hypothesis} \\ &= cn + 1 \end{aligned}$$

unsuccessful!

# Guess and Prove

$$T(n) = 2T(n/2) + 1, \quad T(1) = 1$$

Guess:  $T(n) \leq cn - 1$

<Proof> Assume it holds for  $n' < n$ .

$$\begin{aligned} T(n) &= 2T(n/2) + 1 \\ &\leq 2(c(n/2) - 1) + 1 && \longleftarrow \text{induction hypothesis} \\ &= cn - 1 \end{aligned}$$

# Master Theorem

- Theorem for recurrences of the form  $T(n) = aT(n/b) + f(n)$
- Let  $h(n) = n^{\log_b a}$

- ① If  $f(n) = O(h(n)/n^\epsilon)$  for some positive constant  $\epsilon$ ,  
 $T(n) = \Theta(h(n))$ .
- ② If  $f(n) = \Theta(h(n))$ ,  $T(n) = \Theta(h(n) \log n)$ .
- ③ If  $f(n) = \Omega(h(n)n^\epsilon)$  for some positive constant  $\epsilon$  and  
 $af(n/b) \leq cf(n)$  for some constant  $c (< 1)$  and all sufficient  
large  $n$ ,  $T(n) = \Theta(f(n))$ .

# Understanding Master Theorem

- ① If  $f(n)$  is polynomially smaller than  $h(n)$ ,  $h(n)$  determines the time.
- ② If  $f(n)$  is asymptotically equal to  $h(n)$ ,  $h(n) * \log n$  is the time.
- ③ If  $f(n)$  is polynomially larger than  $h(n)$ ,  $f(n)$  determines the time.

# Master Theorem

- $T(n) = 2T(n/3) + c$ 
  - $a=2, b=3, h(n) = n^{\log_3 2}, f(n) = c$
  - $T(n) = \Theta(n^{\log_3 2})$
- $T(n) = 2T(n/2) + n$ 
  - $a=b=2, h(n) = n^{\log_2 2} = n, f(n) = n$
  - $T(n) = \Theta(n \log n)$
- $T(n) = 2T(n/4) + n$ 
  - $a=2, b=4, h(n) = n^{\log_4 2}, f(n) = n$
  - $af(n/b) = n/2 \leq cf(n)$  for  $c=1/2$
  - $T(n) = \Theta(n)$



**Thank you**

---