Important Announcements

The exams and the quizzes will be **on paper** from now on

Homework assignments should be **submitted online**

Please make sure your handwriting is legible for homework, quizzes, and exams

Announcements

Midterm Exam 1

- Thursday Feb. 10 in class
- Covered material: 1st class → the class on Tuesday Feb. 8th
- Do not forget to prepare your cheat sheet (a single-side letter-size paper)

Common Issues in Quiz 1

- **Simplest g(n)**
  - ignoring constant coefficients and terms with smaller order of growth, for example

  n, $logn$, $nlogn$, n^2, n^3, a^n, and **product** of them

- **Prove using definition or limit**

- **Simplify the ratio before applying L'Hôpital's Rule**

Analyze the Time Efficiency of An Algorithm

Nonrecursive Algorithm

- Matrix multiplication
- Selection sort
- etc

$$\begin{array}{l}
\textbf{ALGORITHM}\ \ Factorial(n) \\
f \leftarrow 1 \\
\textbf{for}\ i \leftarrow 1\ \textbf{to}\ n\ \textbf{do} \\
\quad f \leftarrow f * i \\
\textbf{return}\ f
\end{array}$$

Recursive Algorithm

- Fibonacci number
- Merge sort
- etc

$$\begin{array}{l}
\textbf{ALGORITHM}\ \ Factorial(n) \\
\textbf{if}\ n = 0 \\
\quad \textbf{return}\ 1 \\
\textbf{else} \\
\quad \textbf{return}\ Factorial(n-1) * n
\end{array}$$

Last Class: Time efficiency of Nonrecursive Algorithms

Steps in mathematical analysis of nonrecursive algorithms:

- Decide on parameter *n* indicating *input size*
- Identify algorithm's *basic operation*
- Determine *worst*, *average*, and *best* case for input of size *n*
- Set up summation for *t(n)* reflecting algorithm's loop structure
- Simplify summation using standard formulas (see Appendix A)

Last Class: Useful Formulas in Appendix A

$$\sum_{i=startInd}^{endInd} 1 = \underbrace{1+1+\cdots 1}_{\text{endInd-startInd+1 times}} = endInd - startInd + 1$$

$$\sum_{i=1}^{n} ca_i = c\sum_{i=1}^{n} a_i$$

$$\sum_{i=1}^{n} (a_i + b_i) = \sum_{i=1}^{n} a_i + \sum_{i=1}^{n} b_i$$

$$\sum_{i=1}^{n} i = 1 + 2 + \ldots + n = \frac{n(n+1)}{2} \in \Theta(n^2)$$

$$\sum_{i=1}^{n} i^k \in \Theta(n^{k+1}) \ldots$$

Analyze the Time Efficiency of A Recursive Algorithm

Recursive algorithm for $n!$

Input size: n

Basic operation: multiplication "*"

Let $C(n)$ be the number of multiplications needed to compute $n!$, then

ALGORITHM *Factorial*(n)

if $n = 0$

  return 1

else

  return $Factorial(n-1) * n$

$$C(0) = 0$$

$$C(n) = C(n-1) + 1 \text{ for } n > 0$$

To compute *Factorial*(n-1)

To multiply *Factorial*(n-1) by n

Solve the Recurrence

$C(0) = 0$

$C(n) = C(n-1) + 1 \quad \text{for } n > 0$

$$\Rightarrow C(n)$$

$$= \underbrace{\boxed{C(n-1)} + 1}$$

$$= \underbrace{\boxed{C(n-2)} + 1} + 1$$

$$= C(n-3) + 1 + 1 + 1$$

$$= \ldots$$

$$= C(n-n) + n$$

$$\boxed{= \mathrm{C}(0) + \mathrm{n}}$$

$$= \mathrm{n}$$

Therefore, the number of multiplications needed to compute *n*! in this algorithm is n.

→ The complexity of this algorithm is

$$\Theta(n)$$

Time Efficiency of Recursive Algorithms

Steps in mathematical analysis of recursive algorithms:

1. Decide on parameter n indicating ***input size***

2. Identify algorithm's ***basic operation***

3. Determine ***worst***, ***average***, and ***best*** case for input of size n

4. Set up a recurrence relation and initial condition(s) for $C(n)$- the number of times the basic operation will be executed for an input size n.

5. Solve the recurrence to obtain a closed form or estimate the order of growth of the solution

Example: Recursive evaluation of *n* !

Recursive algorithm for $n!$

Input size: n

Basic operation: multiplication "*"

Let $C(n)$ **be the number of multiplications needed to compute** $n!$**, then**

$$C(0) = 0$$

Initial condition

$$C(n) = C(n-1) + 1 \quad \text{for } n > 0$$

Recurrence relation

$$C(n) = n$$

Solve the recurrence

ALGORITHM $Factorial(n)$

if $n = 0$

return 1

else

return $Factorial(n-1) * n$

Sequences and Recurrence Relations

A sequence: an ordered list of numbers.

For example: 0, 2, 4, 6, … (even integers)

How to represent a sequence: x(n) -- General term of the sequence

The index in the sequence

- Explicit mathematic formula: e.g., $x(n) = n + 1$ for $n \geq 0$
- Recurrence relation:

  e.g., $x(n) = x(n-1) + 1$ for $n > 0$

  – Initial condition $x(0) = 1$

  – Initial condition defines the conditions that violate the recurrence relation with the valid input
- Solving the recurrence → finding the explicit formula

Solutions of Recurrence Relations

$$C(n) = C(n-1) + 1 \text{ for } n > 0$$

$$\boxed{C(n) = C(0) + n \text{ for } n > 0}$$

General solution

- A class of solutions **without specifying initial condition**
- Satisfying the recurrence relation with an arbitrary constant – any specified initial condition

Particular solution

- Satisfying the recurrence relation and the particular initial condition $\boxed{C(n) = n \text{ for } n > 0, C(0) = 0}$

Solving Recurrence Relations: Forward Substitutions

Solving the recurrence by identifying the pattern of the sequence

$$x(n) = 2x(n-1) + 1 \text{ for } n > 1$$

$$x(1) = 1$$

$$x(1) = 1 \qquad x(2) = 2x(1) + 1 = 3$$

$$x(3) = 2x(2) + 1 = 7$$

$$x(4) = 2x(3) + 1 = 15$$

$$\Rightarrow \quad x(n) = 2^n - 1, \text{for n} \geq 1$$

Prove by induction or substitution

Forward substitution is difficult for complex patterns

Solving Recurrence Relations: Backward Substitutions

$$x(n) = x(n-1) + n \quad \text{for } n > 0, \qquad x(0) = 0$$

$$x(n) = [x(n-2) + n - 1] + n$$

$$x(n) = [x(n-3) + n - 2] + n - 1 + n$$

$$x(n) = x(n-i) + (n-i+1) + (n-i+2) + \cdots + n$$

$$x(n) = x(n-n) + (n-n+1) + (n-n+2) + \cdots + n$$

Solve $x(n)$ using the initial condition

$$x(n) = x(0) + (n-n+1) + (n-n+2) + \cdots + n = \frac{n(n+1)}{2}$$

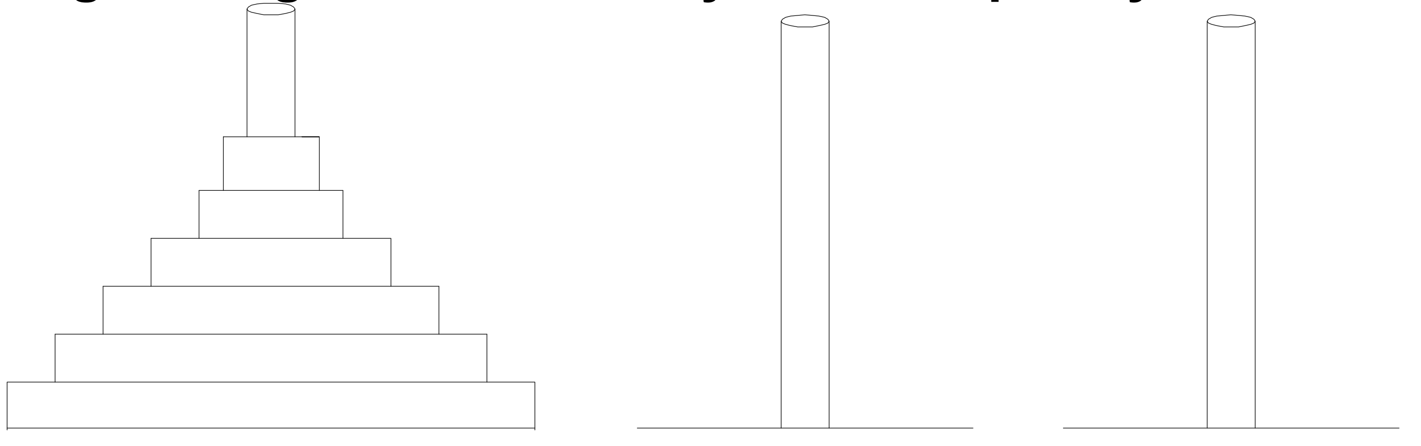
Example: Tower of Hanoi

***n* different-size disks, 3 pegs, move disks from the left peg to the right one using the middle one as an auxiliary**

Rules:

- move one disk each time
- cannot place a larger disk on top of a smaller one

Design an algorithm and analyze its complexity



Recursive Algorithm

Input size: *n* (disks)

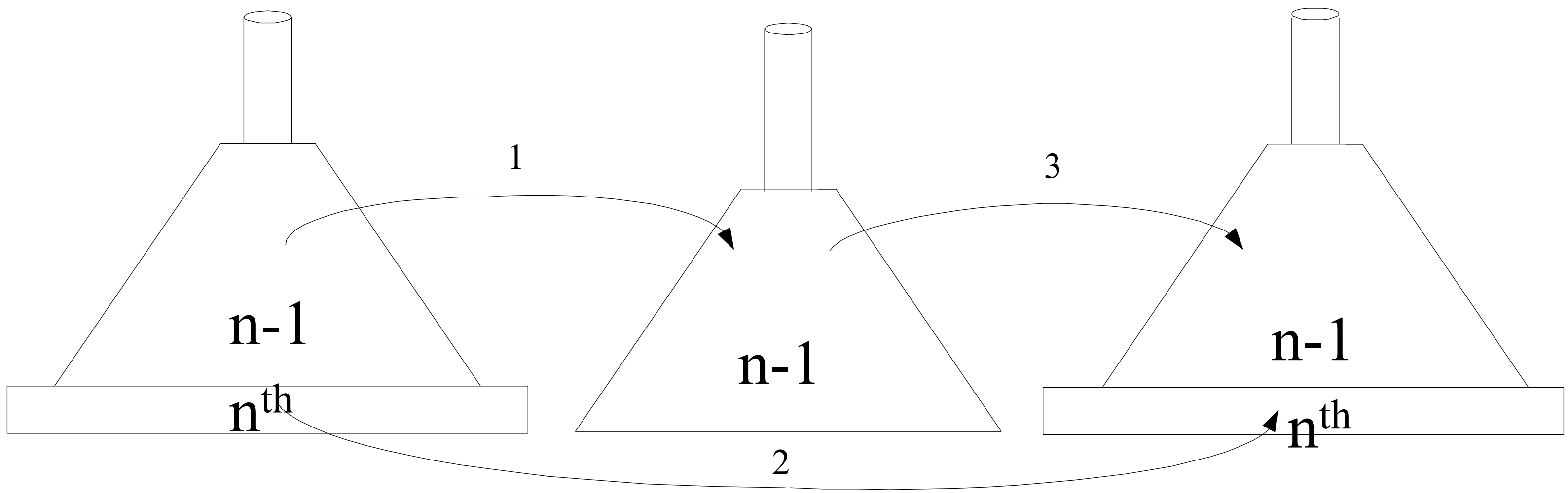
Basic operation: one move of a disk

Initial condition: *n*=1 → only one direct move

To build the recurrence: suppose you have a way to move *n*-1 disks.

- Then you can move the top *n*-1 disks from the left peg to the middle peg using the right peg as an auxiliary.
- Move the bottom disk from the left peg to the right peg.
- Move *n*-1 disks from the middle peg to the right peg using the left one as an auxiliary.

Illustration



Algorithm Complexity

Let *M*(*n*) be the number of needed moves

Initialization *M*(1)=1

Recurrence

$$\boxed{M(n) = \overbrace{M(n-1)}^{\text{Step 1}} + \underbrace{1}_{\text{Step 2}} + \overbrace{M(n-1)}^{\text{Step 3}} \quad \textbf{for } n > 1}$$

Algorithm Complexity

Let *M*(*n*) be the number of needed moves

Initialization *M*(1)=1

Recurrence $\boxed{M(n) = M(n-1) + 1 + M(n-1) \quad \textbf{for } n > 1}$

$$
\begin{aligned}
M(n) &= 2M(n-1) + 1 \quad \text{for } n > 1 \\
&= 2[2M(n-2) + 1] + 1 = 2^2 M(n-2) + 2 + 1 \\
&= 2^2[2M(n-3) + 1] + 2 + 1 = 2^3 M(n-3) + 2^2 + 2 + 1 \\
&= \ldots \\
&= 2^{n-2}\left[2M\big(n-(n-1)\big) + 1\right] + 2^{n-3} + \cdots + 2 + 1 \\
&= 2^{n-1}M(1) + \sum_{i=0}^{n-2} 2^i = 2^{n-1} + 2^{n-1} - 1 = 2^n - 1
\end{aligned}
$$

Example 2 – Solving Recurrence Relations Using Backward Substitutions

$$T(n) = T(n-1) + 2 \; for \; n > 1, \qquad T(1) = 2$$

$$
\begin{aligned}
T(n) &= T(n-1) + 2 \\
&= \overbrace{T(n-2) + 2} + 2 \\
&= T(n-3) + 2 + 2 + 2 \\
&= \quad \cdots \\
&= T[n-(n-1)] + \overbrace{2 + \cdots + 2}^{(n-1)*2} \\
&= T(1) + (n-1) * 2 \\
&= \quad 2n
\end{aligned}
$$

Example 3 – Solving Recurrence Relations Using Backward Substitutions

$$T(n) = T(n-1) + 2n \; for \; n > 0, \qquad T(0) = 2$$

$$T(n) = T(n-1) + 2n = \overbrace{T(n-2) + 2(n-1)}^{T(n-1)} + 2n$$
$$= T(n-3) + 2(n-2) + 2(n-1) + 2n = \cdots$$
$$= T[n-n] + 2[n-(n-1)] + \cdots + 2n$$
$$= T(0) + 2 * [1 + 2 + 3 + \cdots + n] = 2 + 2 * \sum_{i=1}^{n} i = 2 + 2 * \frac{n(n+1)}{2}$$
$$= n^2 + n + 2$$

Example 4 – Solving Recurrence Relations Using Backward Substitutions

$$T(n) = T(n/2) + 2n \; for \; n > 1, \qquad T(1) = 2$$

Let $n = 2^k$, k is an integer and k > 0

$$T(n) = T(n/2) + 2n \rightarrow T(2^k) = T(2^{k-1}) + 2 * 2^k$$

Example 4 – Solving Recurrence Relations Using Backward Substitutions

$$T(n) = T(n/2) + 2n \; for \; n > 1, \qquad T(1) = 2$$

$$T(2^k) = T(2^{k-1}) + 2 * 2^k = \overbrace{T(2^{k-2}) + 2 * 2^{k-1}}^{T(n/2) = T(2^{k-1})} + 2 * 2^k$$

$$= T(2^{k-3}) + 2 * 2^{k-2} + 2 * 2^{k-1} + 2 * 2^k$$

$$= T(2^{k-k}) + 2 * [2^{k-(k-1)} + \cdots + 2^{k-1} + 2^k]$$

$$= T(1) + 2 * \sum_{i=1}^{k} 2^i = T(1) + 2 * (2^{k+1} - 1 - 1)$$

$$n = 2^k$$

$$= 2 + 2 * (2n - 2) = 4n - 2$$