Lecture Title: Recursion

Dept. of Computer Science
Faculty of Science and Technology

Lecture No:	03	Week No:	03	Semester:	
Lecturer:					

Recurrences
&
Master Method

Lecture Outline

1. **Divide and Conquer**
2. **Recurrences in Divide and Conquer and methodologies for recurrence Solutions**
3. **Repeated Backward Substitution Method**
4. **Substitution Method**
5. Recursion Tree (Next Week)
6. Master Method (Next Week)

The divide- and- conquer design paradigm

- 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

The divide- and- conquer design paradigm

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

The divide- and- conquer design paradigm

Example: merge sort

1. Divide: Trivial (array is halved).

2. Conquer: Recursively sort 2 subarrays.

3. Combine: Linear- time merge.

Recurrence for merge sort

Let T(*n*) = Time required for size *n*

T(*n*) =

Sub-problem Size (n/2)

2 * T(n / 2) + O(n)

Number Of Subproblems * ***Time required for each Subproblem*** + ***Work Dividing and Combining***

The divide- and- conquer design paradigm

Binary search

Find an element in a sorted array:

1. Divide: Check middle element.

2. Conquer: Recursively search 1 subarray.

3. Combine: Trivial.

Recurrence for binary search

$T(n) =$

$$1 \; * \; T(n/2) \; + \; \Theta(1)$$

Number Of Subproblems	*	***Time required for each Subproblem***	+	***Work Dividing and Combining***

Recurrences

- Running times of algorithms with **recursive calls** can be described using recurrences.
- A **recurrence** is an equation or inequality that describes a function in terms of its value on smaller inputs.
- For **divide and conquer** algorithms:

$$T(n) = \begin{cases} \text{solving\_trivial\_problem} & \text{if } n = 1 \\ \text{num\_pieces } T(n/\text{subproblem\_size\_factor}) + \text{dividing} + \text{combining} & \text{if } n > 1 \end{cases}$$

- Example: Merge Sort

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

Solving Recurrences

- **Repeated (backward) substitution method**
  - Expanding the recurrence by substitution and noticing a pattern (this is not a strictly formal proof).
- **Substitution method**
  - guessing the solutions
  - verifying the solution by the mathematical induction
- **Recursion-trees**
- **Master method**
  - templates for different classes of recurrences

Repeated Substitution

- Let's find the running time of the merge sort

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

$T(n) \quad = 2T(n/2) + n$ **substitute**

$= 2(2T(n/4) + n/2) + n$ **expand**

$= 2^2T(n/4) + 2n$ **substitute**

$= 2^2(2T(n/8) + n/4) + 2n$ **expand**

$= 2^3T(n/8) + 3n$ **observe pattern**

$= 2^3T(n/2^3) + 3n$ **observe pattern**

$$let\ \frac{n}{2^i} = 1$$

$$2^i = \mathrm{n}$$

$$\log_2 2^i = \log_2 n$$

$$\log_2 n = \mathrm{i}$$

$$T(n) = 2^iT(n/2^i) + in$$
$$= n\,T(1) + n \lg n$$
$$= n + n \lg n$$

$$\boxed{T(n) = O(n \lg n)}$$

Repeated Substitution Method

- The procedure is straightforward:
  - **Substitute**, **Expand**, **Substitute**, **Expand**, …
  - Observe a ***pattern*** and determine the expression after the *i*-th substitution.
  - Find out what the highest value of *i* (number of iterations, e.g., `lg` *n*) should be to get to the base case of the recurrence (e.g., *T(1)*).
  - Insert the value of *T(1)* and the expression of *i* into your expression.

Another Example…

$$T(n) = \begin{cases} 2 & \text{if } n = 1 \\ 2T(n/2) + 2n + 3 & \text{if } n > 1 \end{cases}$$

$$let\ \frac{n}{2^i} = 1$$
$$2^i = n$$
$$\log_2 2^i = \log_2 n$$
$$\log_2 n = i$$

$T(n) = 2\boldsymbol{T(n/2)} + 2n + 3$ **substitute**

$= 2(\boldsymbol{2T(n/4) + n + 3}) + 2n + 3$ **expand**

$= 2^2\boldsymbol{T(n/4)} + 4n + 2x3 + 3$ **substitute**

$= 2^2(\boldsymbol{2T(n/8) + n/2 + 3}) + 4n + 2x3 + 3$ **expand**

$= 2^3T(n/2^3) + 2x3n + 3x(2^2 + 2^1 + 2^0)$ **observe pattern**

$T(n) = 2^iT(n/2^i) + 2ni + 3\sum_{j=0}^{i-1} 2^j$

$= 2^iT(1) + 2ni + 3(2^i - 1)$

$= \boldsymbol{n + 2n \lg n + 3(n - 1)}$

$= \boldsymbol{n + 2n \lg n + 3n - 3}$

$= \boldsymbol{4n + 2n \lg n - 3}$

$$\boxed{T(n) = O(n \lg n)}$$

Substitution Method

- The substitution method to solve recurrences entails two steps:
  - **Guess** the solution.
  - Use **induction** to prove the solution.
- Example:
  - $T(n) = 4T(n/2) + n$

…Substitution Method

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

1) Guess $T(n) = O(n^3)$, i.e., $T(n)$ is of the form cn^3

2) Prove $T(n) \leq cn^3$ by induction

$T(n)$	***$= 4T(n/2) + n$***	**recurrence**
	$\leq 4c(n/2)^3 + n$	**induction hypothesis**
	$= 0.5cn^3 + n$	**simplify**
	$= cn^3 - (0.5cn^3 - n)$	**rearrange**
	$\leq cn^3$ if $c>=2$ and $n>=1$	

Thus *$T(n) = O(n^3)$*

Exercise-1

$$T(1) = 1$$
$$T(n) = T\left(\frac{n}{2}\right) + \sqrt{n}$$

$T(n) = O(\sqrt{n})$

N=1, c=4

…Substitution Method

- Tighter bound for $T(n) = 4T(n/2) + n$:

  Try to show $T(n) = O(n^2)$

  Prove $T(n) \le cn^2$ by induction

  $$\begin{aligned} \mathbf{T(n)} &= \mathbf{4T(n/2) + n} \\ &\le \mathbf{4c(n/2)^2 + n} \\ &= \mathbf{cn^2 + n} \\ &\not\le \mathbf{cn^2} \end{aligned}$$

…Substitution Method

- What is the problem? Rewriting

  $T(n) = O(n^2) = cn^2 - (\text{something positive})$

- As $T(n) \leq cn^2$ does not work with the inductive proof.
- Solution: **Strengthen the hypothesis** for the inductive proof:
  - T(n) $\leq$ (**answer you want**) - (something > 0)

…Substitution Method

- Fixed proof: strengthen the inductive hypothesis by subtracting lower-order terms:

Prove $T(n) \le cn^2 - dn$ by induction

$$
\begin{aligned}
\mathbf{T(n)} &= \mathbf{4T(n/2) + n} \\
&\le \mathbf{4(c(n/2)^2 - d(n/2)) + n} \\
&= \mathbf{cn^2 - 2dn + n} \\
&= \mathbf{(cn^2 - dn) - (dn - n)} \\
&\le \mathbf{cn^2 - dn \text{ if } d \ge 1}
\end{aligned}
$$

- Don’t miss the next class !!!!!!!!!!!!!!!!!!