Divide and Conquer-2

Department of Computer Science, Tsinghua University

Review

- D&C is another powerful technique for algorithm design.
- In-class exercise (T/F):
  - D&C algorithms can only be implemented recursively.
  - The efficiency of D&C algorithms depends only on the number of subproblems and time for the divide and combine steps.

Solving Recurrences

- If your code has recursive calls, how to analyze the time complexity?
  - How to set up a recurrence? (from code to recurrence)

    Two elements of a recurrence:

    recursive relationship & exit condition
  - How to solve a recurrence?
- **We need to learn a few tricks!**
  - Substitution method
  - Recursion tree
  - Master Theorem

Method1: Substitution Method

The most general method:

1. Guess the form of the solution.

2. Verify by induction.

3. Solve for constants.

Example: $T(n) = 4T(n/2) + n$, $T(1) = d$

1. Guess**:** $O(n^3)$

2. Verify by induction: prove $T(n) \leq cn^3$ by induction

- $f(n) = O(g(n))$ iff $\exists\, c > 0, n_0 > 0$, when $n \geq n_0$, $0 \leq f(n) \leq cg(n)$

① **Base case:** $T(n) \leq cn^3$ for the base cases of the inductive proof.

② **Induction step:** assume that $T(k) \leq ck^3$ for $k < n$, prove $T(n) \leq cn^3$.

Example

3. Solve for constants.

Base case:

First, we handle the initial conditions, that is, ground the induction with base cases.

In this case:

For $n = 1$, we have $T(1) = d$. $T(1) = d \leq cn^3$ is true, if we pick c big enough, such that $c > d$.

3. *Solve* for constants.

Induction step:

$$
\begin{aligned}
T(n) &= 4T(n/2) + n \\
&\le 4c\left(\frac{n}{2}\right)^3 + n \\
&= (c/2)n^3 + n \\
&= cn^3 - \left(\left(\frac{c}{2}\right)n^3 - n\right)
\end{aligned}
$$

$\le cn^3$ holds, whenever $\left(\left(\frac{c}{2}\right)n^3 - n\right) \ge 0$.

For example, pick $c \ge 2$, for $n \ge 1, T(n) \le cn^3$ holds.

Example

$T(n) = O(n^3)$ ***This bound is not tight!***

==============================

- We shall prove that $T(n) = O(n^2)$

IDEA: Strengthen the inductive hypothesis.

- *Subtract* a low-order term.

Example: Select

- $T(n) = \begin{cases} T(n/5) + T(7n/10) + O(n) & n \geq 100 \\ O(1) & n < 100 \end{cases}$

- Guess: $T(n) = O(n)$

- Substitution Method (Mathematical Induction)
  - Prove $T(n) \leq cn$ for a constant c by induction
  - ***Base case:*** $T(n) \leq cn$ holds for some suitably large c and all $n < 100$
  - ***Induction step:*** Assume that $T(k) \leq ck$ for $k < n$, *prove* $T(n) \leq cn$

$$T(n) = T(n/5) + T(7n/10) + O(n) = O(n)$$

Prove by induction.

Assume that $T(k) \le ck$, for some suitable large c and $k < n$. Now we need to prove $T(n) \le cn$.

$$\begin{aligned} T(n) &\le \frac{1}{5}cn + \frac{7}{10}cn + an \\ &= \frac{9}{10}cn + an \\ &= cn + (-\frac{1}{10}cn + an) \\ &\le cn \quad (\text{when } -\frac{1}{10}cn + an \le 0, \text{ i.e., } c \ge 10a) \end{aligned}$$

Preliminary Math

- Arithmetic Series

$$\sum_{k=1}^{n} k = \frac{1}{2} n(n+1)$$

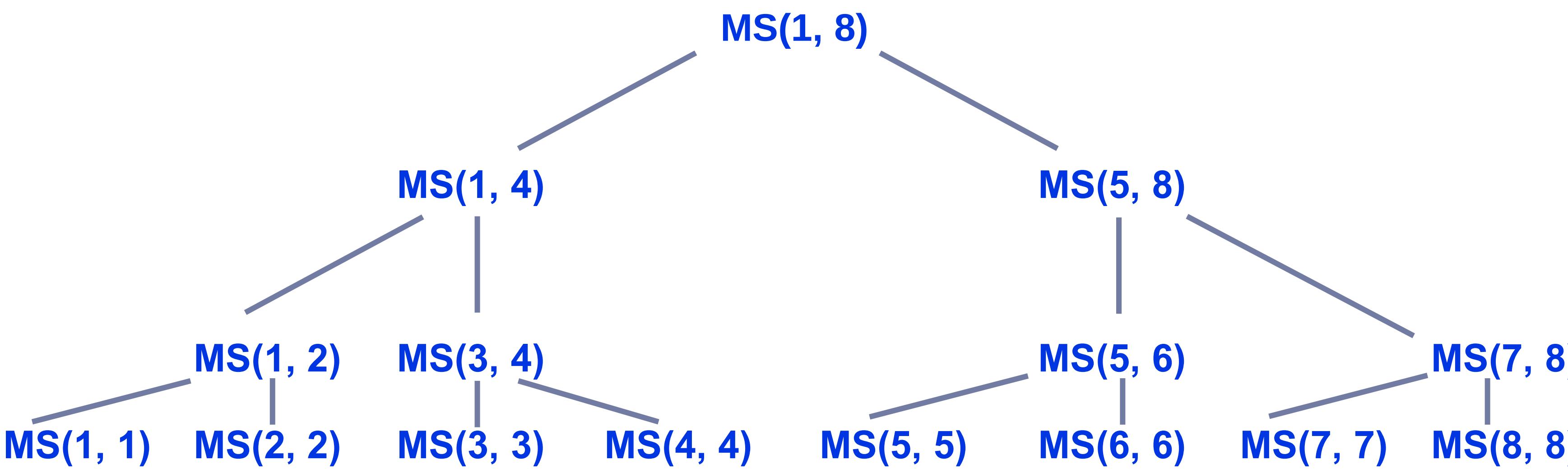
- Geometric Series

$$\sum_{k=0}^{n} x^k = \frac{1-x^{n+1}}{1-x}$$

$$\text{when } x < 1, \lim_{n \to \infty} \frac{1-x^{n+1}}{1-x} = \frac{1}{1-x} = \Theta(1)$$

Method 2: Recursion tree

Recursion tree vs. Call graph



Recursion Tree: each node represents the cost of $D(n) + C(n)$ of a single subproblem somewhere in the set of recursive function invocations. *The total running time equals the sum of all nodes.*

Example: Merge Sort

1. Divide: Trivial.

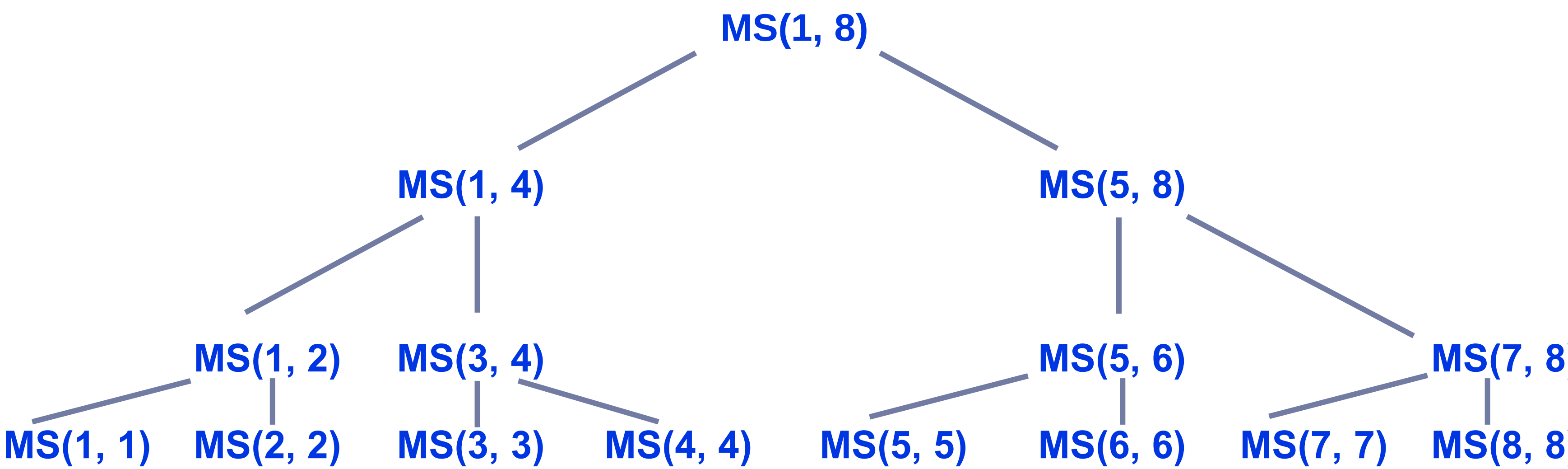
2. Conquer: Recursively sort 2 subarrays.

3. Combine: Linear-time merge.

```
MERGE-SORT(A, p, r)
    if p < r
        q = ⌊(p + r)/2⌋
        MERGE-SORT(A, p, q)
        MERGE-SORT(A, q + 1, r)
        MERGE(A, p, q, r)
```

Merge Sort

$$T(n) = 2T(n/2) + cn \qquad T(1) = d$$



Recursion Tree: each node represents the cost of $D(n) + C(n)$ of a single subproblem somewhere in the set of recursive function invocations. *The total running time equals the sum of all nodes.*

Merge Sort

Solve $T(n) = 2T(n/2) + cn \qquad T(1) = d$

Step1: Starts from the root: the cost of $D(n) + C(n)$ at the top level of the recursion, the number of subtrees equals the number of smaller recurrences.

Step2: The subtrees of the root: the cost of smaller recurrences.

Step3: Keep expanding on nodes.

Tree Structure: call graph

Calculation: check out *complete binary tree* from Appendix B.5 to see depth, height, # of nodes at each level etc.

Example of recursion tree

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

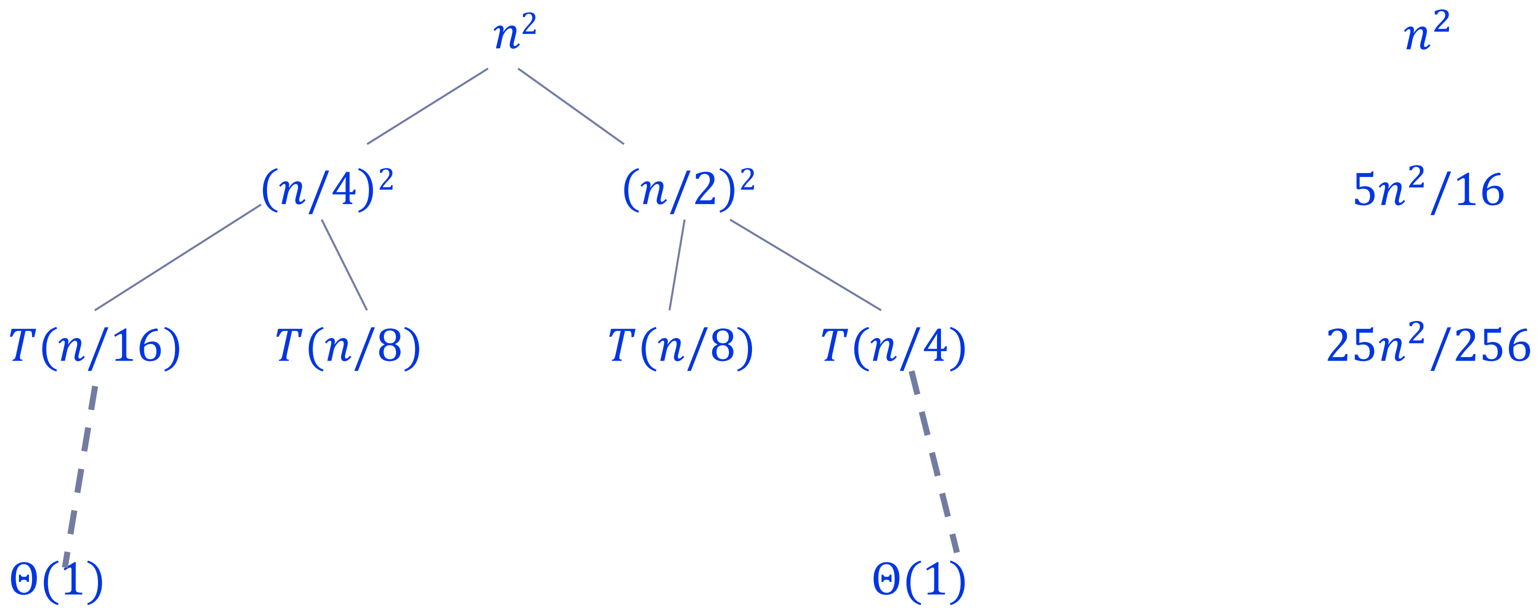
Step1: Starts from the root: the cost of $D(n) + C(n)$ at the top level of the recursion, the number of subtrees equals the number of smaller recurrences.

Step2: The subtrees of the root: the cost of smaller recurrences.

Step3: Keep expanding on nodes.

Example of recursion tree

Solve $T(n) = T(n/4) + T(n/2) + n^2$:



$$\text{Total} = n^2\left(1 + \frac{5}{16} + \left(\frac{5}{16}\right)^2 + \left(\frac{5}{16}\right)^3 + \cdots\right) = O(n^2)$$

A good guess, then verified by the substitution method

Group Discussion

- Draw the recursion tree for the following recurrences:
  - $T(n) = T(2n/3) + \Theta(1)$, $T(1) = \Theta(1)$
  - $T(n) = 3T(n/2) + cn$, $T(1) = d$

Group Discussion

- Let's solve $T(n) = 3T(n/2) + cn$, $T(1) = d$

Recursion tree	Level	Input Size	Recurrence equation	Time Cost
	0	n	$T(n) = 3T(n/2) + cn$	cn
	1	$n/2$	$T(n/2) = 3T(n/4) + cn/2$	$\frac{3}{2}cn$
	2	$n/4$	$T(n/4) = 3T(n/8) + cn/4$	$\left(\frac{3}{2}\right)^2 cn$
	…	…	…	…
	k	$\frac{n}{2^k}$	$T(n/2^k) = 3T(n/2^{k+1}) + cn/2^k$	$\left(\frac{3}{2}\right)^k cn$
	…	…	…	…
	lgn	1	d	$3^{lgn}d$

Group Discussion

- Let's solve $T(n) = 3T(n/2) + cn$, $T(1) = d$

$$T(n) = \sum_{k=0}^{\lg n - 1} \left(\frac{3}{2}\right)^k cn + 3^{\lg n} d$$

Summary

- Substitution method
  - Guess
  - Verify (by definition of θ, O, Ω etc.)
  - Solve for constants
- Recursion tree
  - Tree structure
  - Cost at each node
  - Cost of the entire tree
- Master Theorem