Chapter 4. Divide-and-Conquer

Joon Soo Yoo

March 20, 2025

Assignment

- Read §4.4, §4.5
- Problems:
  - §4.4 - #1, 2
  - §4.5 - #1, 3

Chapter 4: Divide-and-Conquer

- Chapter 4.1: Multiplying square matrices
- Chapter 4.2: Strassen's algorithm for matrix multiplication
- Chapter 4.3: The substitution method for solving recurrences
- **Chapter 4.4: The recursion-tree method for solving recurrences**
- Chapter 4.5: The master method for solving recurrences

4.4 Using Recursion Trees to Solve Recurrences

Why use recursion trees?

- When it's hard to guess a solution to a recurrence, recursion trees help.
- Each node represents the cost of a single recursive call.
- Costs are summed level by level $\rightarrow$ total cost $=$ sum of per-level costs.

Two uses for recursion trees:

1. **Build intuition:** Use the tree to guess the form of the solution.
2. **Direct proof:** With careful calculation, a tree can justify the full bound.

Tips:

- It's okay to be a little sloppy when using the tree to guess.
- But if you want to prove the guess (e.g., via substitution), be precise!

When Is a Recursion Tree a Complete Proof?

Use Case	Is It a Proof?	Notes
Carefully calculated and summed	Yes	If you compute all levels precisely and sum correctly, the recursion tree provides a full asymptotic proof.
Used casually to guess complexity	Not yet	If you skip constants or ignore details, the tree is only a heuristic to help make a guess.
Used to guess, then verified with substitution	Yes	This is a common and safe strategy: use the tree to guide your guess, then prove it rigorously.

Takeaway: Recursion trees are powerful — but be clear whether you're using them for *intuition* or a *formal proof*.

An Illustrative Example: Guessing via Recursion Tree

Recurrence:

$$T(n) = 3T(n/4) + \Theta(n^2)$$

Goal: Use a recursion tree to find a good guess for the asymptotic bound.

Approach:

- Let the $\Theta(n^2)$ term be cn^2 for some constant $c > 0$.
- Build a recursion tree to track the cost per level.
- Add up all costs to estimate total work $T(n)$.

Let's assume:

- n is a power of 4 (simplifies the tree),
- base case occurs at $n = 1$ with cost $\Theta(1)$.

Recursion Tree Structure for $T(n) = 3T(n/4) + cn^2$

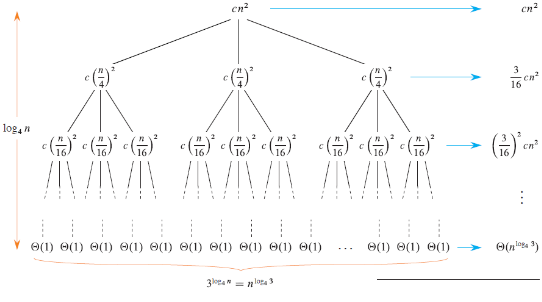
Each node's cost: $c \cdot \left(\frac{n}{4^i}\right)^2$ at level i

Number of nodes at level i**:** 3^i

Total cost at level i**:**

$$3^i \cdot c\left(\frac{n}{4^i}\right)^2 = \left(\frac{3}{16}\right)^i \cdot cn^2$$

Tree depth: $\log_4 n$



(d)

Total Cost via Geometric Series

Total cost over all levels (except leaves):

$$T(n) = \sum_{i=0}^{\log_4 n - 1} \left(\frac{3}{16}\right)^i \cdot cn^2 = cn^2 \cdot \sum_{i=0}^{\log_4 n - 1} \left(\frac{3}{16}\right)^i$$

This is a geometric series:

$$< cn^2 \cdot \sum_{i=0}^{\infty} \left(\frac{3}{16}\right)^i = cn^2 \cdot \frac{1}{1 - \frac{3}{16}} = \frac{16}{13} cn^2$$

Leaf cost:

Number of leaves $= 3^{\log_4 n} = n^{\log_4 3} \quad \Rightarrow \quad$ Total leaf cost $= \Theta(n^{\log_4 3})$

Conclusion:

$$T(n) = \Theta(n^2) \quad \text{(root dominates total cost)}$$

Substitution Method: Step 1 – The Setup

Recurrence:

$$T(n) = 3T(n/4) + cn^2$$

Assume $c > 0$. We want to prove:

$$T(n) = O(n^2)$$

Strategy: Prove that $T(n) \leq d\,n^2$ for some constant $d > 0$ using induction.

Substitution Method: Step 2 – Inductive Hypothesis

Inductive Hypothesis: Assume for all $n_0 \leq k < n$:

$$T(k) \leq d\,k^2$$

We aim to show this also holds for $T(n)$.

Substitution Method: Step 3 – Inductive Step

Plug into recurrence:

$$T(n) = 3T(n/4) + cn^2$$

Apply inductive hypothesis:

$$T(n) \leq 3 \cdot d(n/4)^2 + cn^2 = \frac{3}{16}dn^2 + cn^2$$

We want:

$$T(n) \leq dn^2$$

So it suffices that:

$$\frac{3}{16}d + c \leq d \quad \Rightarrow \quad d \geq \frac{16}{13}c$$

Substitution Method: Step 4 – Base Case

Assume $T(k) = a > 0$ for all $k < n_0$. We want:

$$T(k) = a \leq dk^2 \quad \text{for all } 1 \leq k < n_0$$

That is: $d \geq \frac{a}{k^2}$

$$\text{Worst case at } k = 1 \Rightarrow d \geq a$$

So: Pick $d \geq \max\left\{\frac{a}{k^2} \mid 1 \leq k < n_0\right\} = a$

Substitution Method: Final Conclusion

We showed:

- Inductive step holds for $n \geq n_0$ if $d \geq \frac{16}{13}c$
- Base case holds for $n < n_0$ if $d \geq a$

Thus:

$$T(n) \leq dn^2 \Rightarrow T(n) = O(n^2)$$

An Irregular Recurrence Example

Given recurrence:

$$T(n) = T(n/3) + T(2n/3) + \Theta(n)$$

This creates an **unbalanced recursion tree**:

- Left branch: subproblem of size $n/3$
- Right branch: subproblem of size $2n/3$

Goal: Find an upper bound on $T(n)$ using a recursion tree.

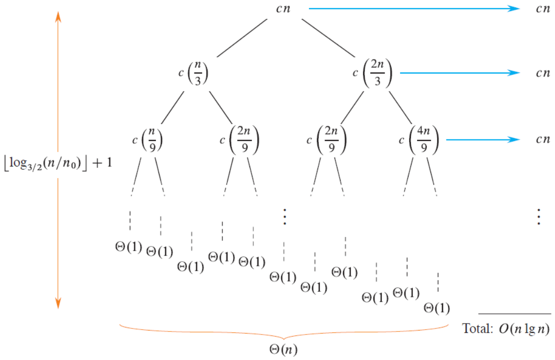
Recursion Tree Structure

Node Costs: $\Theta(n)$ at each internal node

Subproblem sizes:

- Left branch: $n/3$, $n/9$, $n/27$, ...
- Right branch: $2n/3$, $4n/9$, $8n/27$, ...

Rightmost path shrinks by $2/3$ each level



Tree Height: Following the Rightmost Path

Each level's right branch shrinks by a factor of 2/3:

$$\text{At depth } h : \left(\frac{2}{3}\right)^h \cdot n < n_0 \Rightarrow h = \left\lfloor \log_{3/2}(n/n_0) \right\rfloor + 1$$

Conclusion: Tree height is:

$$\boxed{\Theta(\log_{3/2} n) = \Theta(\log n)}$$

(since n_0 is constant, and log base changes are constant multiples)

Cost at Each Level

At each level:

- Work done across all nodes is $\leq cn$
- There are $\Theta(\log n)$ levels

Total cost from internal nodes:

$$\boxed{O(n \log n)}$$

What About the Leaves?

Each leaf contributes $\Theta(1)$ work. How many leaves are there?

- Upper bound: use a complete binary tree of height h
- Height $h = \log_{3/2}(n)$ implies at most:

$$2^{\log_{3/2} n} = n^{\log_{3/2} 2} \approx n^{1.71}$$

So:

$$\text{Leaf cost} = O(n^{1.71}) \quad \text{(too loose)}$$

Tighter Bound on Leaf Costs

- Many paths hit the base case sooner than max depth
- So not all leaves reach full height

Careful analysis shows:

$$\text{Total leaf cost} = \boxed{O(n)}$$

Final Conclusion

Internal nodes: $O(n \log n)$
Leaves: $O(n)$

Therefore:

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

This matches the recurrence behavior despite irregular subproblem sizes.

Leaf Count Recurrence $L(n)$

Goal: Bound the number of leaves in the recursion tree of $T(n)$.

Let $L(n)$ be the number of leaves (base-case calls) in the tree for $T(n)$.

Define:

$$L(n) = \begin{cases} 1 & \text{if } n < n_0 \\ L(n/3) + L(2n/3) & \text{if } n \geq n_0 \end{cases}$$

We want to show: $L(n) = O(n)$

Inductive Hypothesis

Assume for all $k < n$:

$$L(k) \leq d \cdot k \quad \text{for some constant } d > 0$$

Apply to recurrence:

$$L(n) = L(n/3) + L(2n/3) \leq d(n/3) + d(2n/3) = d \cdot n$$

So $L(n) \leq dn$ holds if the inductive step and base case work.

Base Case Verification

For all $k < n_0$, we assume:

$$L(k) = 1$$

We want to verify:

$$L(k) = 1 \leq dk \quad \Rightarrow \quad d \geq \frac{1}{k} \quad \text{for all } 1 \leq k < n_0$$

So: Choose:

$$\boxed{d \geq \max\left\{\frac{1}{k} \mid 1 \leq k < n_0\right\} = 1}$$

Conclusion: Leaf Count is Linear

We proved:

- Inductive step: $L(n) \leq dn$
- Base case: $L(k) = 1 \leq dk$ if $d \geq 1$

Therefore:

$$\boxed{L(n) = O(n)}$$

The number of leaves in the recursion tree is linear in n

Combining Internal and Leaf Costs

Recall:

- Internal nodes: total cost $= O(n \log n)$
- Leaves: $L(n) = O(n)$ leaves, each costing $\Theta(1)$

So:

$$\text{Total cost from leaves} = L(n) \cdot \Theta(1) = O(n)$$

Final Cost of $T(n)$

We add costs from both parts:

$$T(n) = O(n \log n) + O(n) = \boxed{O(n \log n)}$$

This completes the upper bound for:

$$T(n) = T(n/3) + T(2n/3) + \Theta(n)$$

Important: Recursion trees give intuition, but you should verify with:

- Substitution method (especially if you made approximations)

Chapter 4: Divide-and-Conquer

- Chapter 4.1: Multiplying square matrices
- Chapter 4.2: Strassen's algorithm for matrix multiplication
- Chapter 4.3: The substitution method for solving recurrences
- Chapter 4.4: The recursion-tree method for solving recurrences
- **Chapter 4.5: The master method for solving recurrences**

The Master Method: Overview

Purpose: Quickly solve divide-and-conquer recurrences of the form:

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

Where:

- $a > 0$, $b > 1$ are constants
- $f(n)$ is the "driving function"

Name: This is called a *master recurrence*

What Does the Master Recurrence Represent?

- Divide a problem of size n into a subproblems
- Each subproblem is of size n/b
- Recursive cost: $aT(n/b)$
- Driving cost: $f(n)$ (splitting and combining)

Example: Strassen's matrix multiplication

$$T(n) = 7T(n/2) + \Theta(n^2)$$

Ignoring Floors and Ceilings

Real-World Issue: Problem sizes must be integers

- **True recurrence (Merge Sort):**

$$T(n) = T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + \Theta(n)$$

- **Simplified form (for analysis):**

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

- **Why simplify?**
  - Floors and ceilings complicate the recurrence.
  - Asymptotic solution remains the same: $\Theta(n \log n)$.

Conclusion: It is safe to ignore floors/ceilings when applying methods like substitution or the master method.

Master Method in Practice

Key idea: You just need to:

- Recognize the form: $T(n) = aT(n/b) + f(n)$
- Memorize 3 cases (coming next!)
- Apply the case that matches $f(n)$

Result: Get the asymptotic bound for $T(n)$ in seconds

Theorem 4.1: Master Theorem

Let $a > 0$ and $b > 1$ be constants, and let $f(n)$ be a nonnegative function defined on all sufficiently large reals. Define the recurrence:

$$T(n) = a \cdot T(n/b) + f(n)$$

where $aT(n/b)$ represents $a_0 T(\lfloor n/b \rfloor) + a_1 T(\lceil n/b \rceil)$ with constants $a_0, a_1 \geq 0$ and $a = a_0 + a_1$.

Then the asymptotic behavior of $T(n)$ falls into one of the following cases:

1. If there exists $\epsilon > 0$ such that $f(n) = O(n^{\log_b a - \epsilon})$, then:

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

2. If $f(n) = \Theta(n^{\log_b a} \log^k n)$ for some $k \geq 0$, then:

$$T(n) = \Theta(n^{\log_b a} \log^{k+1} n)$$

3. If there exists $\epsilon > 0$ such that $f(n) = \Omega(n^{\log_b a + \epsilon})$, and if there exists $c < 1$ such that:

$$af(n/b) \leq cf(n) \quad \text{(for large } n\text{)},$$

then:

$$T(n) = \Theta(f(n))$$

Understanding the Master Theorem

Compare $f(n)$ to the watershed function:

$$\text{Watershed: } n^{\log_b a}$$

- **Case 1:** $f(n)$ grows *slower* than $n^{\log_b a}$
- **Case 2:** $f(n)$ grows *at the same rate*
- **Case 3:** $f(n)$ grows *faster*, and a regularity condition holds

Important: The technical details of the bounds (e.g., ε, k, and the regularity condition) are essential to apply the theorem correctly.

Master Theorem: Case 1 – Driving Function is Smaller

Condition:

- $f(n) = O(n^{\log_b a - \varepsilon})$ for some $\varepsilon > 0$

Result: $T(n) = \Theta(n^{\log_b a})$

Tree View:

- Cost per level *increases* geometrically from root to leaves.
- **Leaves dominate.**

Example: $T(n) = 4T(n/2) + n^{1.99}$

- $a = 4,\ b = 2,\ \log_b a = 2,\ f(n) = n^{1.99}$
- $f(n)$ is smaller by factor $n^{0.01} \rightarrow$ Case 1 applies
- $T(n) = \Theta(n^2)$

Master Theorem: Case 2 – Driving Function Matches

Condition:

- $f(n) = \Theta(n^{\log_b a} \cdot \log^k n)$ for some $k \geq 0$

Result: $T(n) = \Theta(n^{\log_b a} \cdot \log^{k+1} n)$

Tree View:

- Cost per level is approximately the same.
- **All levels contribute equally.**

Common Case: $f(n) = \Theta(n^{\log_b a}) \Rightarrow T(n) = \Theta(n^{\log_b a} \log n)$

Master Theorem: Case 3 – Driving Function is Larger

Condition:

- $f(n) = \Omega(n^{\log_b a+\varepsilon})$ for some $\varepsilon > 0$
- **Regularity condition:** $a \cdot f(n/b) \leq c \cdot f(n)$ for some $c < 1$

Result: $T(n) = \Theta(f(n))$

Tree View:

- Cost per level *decreases* geometrically from root to leaves.
- **Root dominates.**

Note: Regularity condition ensures that $f(n)$ doesn't fluctuate wildly.

Using the Master Method: Overview

Steps to apply:

1. Identify a, b, and $f(n)$ in $T(n) = aT(n/b) + f(n)$
2. Compute watershed: $n^{\log_b a}$
3. Compare $f(n)$ to $n^{\log_b a}$:
   - Case 1: $f(n) = O(n^{\log_b a - \varepsilon}) \Rightarrow T(n) = \Theta(n^{\log_b a})$
   - Case 2: $f(n) = \Theta(n^{\log_b a} \log^k n) \Rightarrow T(n) = \Theta(n^{\log_b a} \log^{k+1} n)$
   - Case 3: $f(n) = \Omega(n^{\log_b a + \varepsilon})$, with regularity condition $\Rightarrow T(n) = \Theta(f(n))$

Example 1: $T(n) = 9T(n/3) + n$

Parameters: $a = 9,\ b = 3,\ f(n) = n$
$\Rightarrow n^{\log_3 9} = n^2$
Compare: $f(n) = n = O(n^{2-\varepsilon})$ for $\varepsilon \leq 1$
Case 1 applies.
Solution: $T(n) = \Theta(n^2)$

Example 2: $T(n) = T(2n/3) + 1$

Master method parameters:

- $a = 1,\ b = 3/2$
- $f(n) = 1$
- Watershed function: $n^{\log_{3/2} 1} = n^0 = 1$

Compare growth rates:

- $f(n) = \Theta(1)$
- Same asymptotic growth as $n^{\log_b a}$ (since both are constant)

Case 2 applies:

- $f(n) = \Theta(n^{\log_b a} \log^k n)$ with $k = 0$
- $\Rightarrow T(n) = \Theta(n^{\log_b a} \log^{k+1} n) = \Theta(\log n)$

Example 3: $T(n) = 3T(n/4) + n\log n$

Master method parameters:

- $a = 3,\ b = 4$
- $f(n) = n\log n$
- Watershed: $n^{\log_4 3} \approx n^{0.793}$

Compare growth rates:

- $f(n) = \Omega(n^{0.793+\varepsilon})$ for $\varepsilon \approx 0.2$
- $f(n)$ grows polynomially faster than $n^{\log_b a}$

Check regularity condition:

$$af(n/b) = 3 \cdot \left(\frac{n}{4}\log\frac{n}{4}\right) \le \frac{3}{4}n\log n = cf(n),\ \text{for } c = \frac{3}{4} < 1$$

Case 3 applies: Driving function dominates and satisfies regularity.
Conclusion: $T(n) = \Theta(n\log n)$

Quick Check: Apply the Master Method!

Consider the recurrence:

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

Which case of the Master Theorem applies?

A. Case 1
B. Case 2
C. Case 3
D. None

What is the asymptotic solution for $T(n)$?

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

Parameters: $a = 4,\ b = 2,\ f(n) = n^2$

$$n^{\log_b a} = n^{\log_2 4} = n^2$$

Compare: $f(n) = \Theta(n^2) = \Theta(n^{\log_b a})$
Case 2 applies with $k = 0$

Final Answer: $T(n) = \Theta(n^2 \log n)$

Example 5: Classic Algorithms

Merge Sort: $T(n) = 2T(n/2) + \Theta(n)$
$\Rightarrow a = 2, b = 2, f(n) = n \Rightarrow \text{Case } 2 \Rightarrow \Theta(n \log n)$

Naive Matrix Mult.: $T(n) = 8T(n/2) + \Theta(1)$
$\Rightarrow a = 8, b = 2, f(n) = 1 \Rightarrow \text{Case } 1 \Rightarrow \Theta(n^3)$

Strassen's Alg.: $T(n) = 7T(n/2) + \Theta(n^2)$
$\Rightarrow a = 7, b = 2, f(n) = n^2 \Rightarrow \text{Case } 1 \Rightarrow \Theta(n^{\log_2 7})$

Question?