Lecture 3 - Chpt 4: Divide-and-Conquer

Harley Caham Combest

Fa2025 CS4413 Lecture Notes – Mk1

4.1 — Multiplying Square Matrices

Historical Context. Matrix multiplication is a central computational primitive in algorithms. The straightforward $n \times n$ matrix multiplication runs in $\Theta(n^3)$ time. Divide-and-conquer strategies can restructure this computation and yield recurrences for analysis.

Motivation. By partitioning matrices into submatrices and applying recursive multiplication, we reveal opportunities for algorithmic improvement and asymptotic speedups.

Algorithm 1 MATRIX-MULTIPLY(A, B, C, n)

```
1: for i ← 1 to n do
2:     for j ← 1 to n do
3:         for k ← 1 to n do
4:             c_ij ← c_ij + a_ik · b_kj
5:         end for
6:     end for
7: end for
```

Analysis. Each of the three nested loops executes n iterations. Line 4 is constant time. Thus, the running time is $\Theta(n^3)$. Even accounting for the $\Theta(n^2)$ initialization cost of C, the asymptotics remain $\Theta(n^3)$.

A Divide-and-Conquer Approach. Partition each $n \times n$ matrix into four $(n/2) \times (n/2)$ submatrices:

$$A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix}, \quad B = \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix}, \quad C = \begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix}.$$

Then compute

$$C_{11} = A_{11}B_{11} + A_{12}B_{21},$$
$$C_{12} = A_{11}B_{12} + A_{12}B_{22},$$
$$C_{21} = A_{21}B_{11} + A_{22}B_{21},$$
$$C_{22} = A_{21}B_{12} + A_{22}B_{22}.$$

This requires 8 multiplications of $(n/2) \times (n/2)$ submatrices and 4 additions.

Algorithm 2 MATRIX-MULTIPLY-RECURSIVE(A, B, C, n)

```
1: if n = 1 then
2:     c_11 ← c_11 + a_11 · b_11                    ▷ Base case
3:     return
4: end if
5: Partition A, B, C into n/2 × n/2 submatrices
6: Recursively compute and add 8 products into C as per equations above
```

Recurrence. Let $T(n)$ be the running time. Then

$$T(n) = 8T(n/2) + \Theta(1).$$

Proposition 1. *The recurrence* $T(n) = 8T(n/2) + \Theta(1)$ *has solution* $T(n) = \Theta(n^3)$.

Concluding Remark. Divide-and-conquer reproduces the same $\Theta(n^3)$ bound as the straightforward method, but sets the stage for Strassen's breakthrough (Section 4.2), where the number of recursive multiplications is reduced.

4.2 — Strassen's Algorithm for Matrix Multiplication

Historical Context. In 1969, Strassen gave a divide-and-conquer algorithm that reduces the number of $(n/2) \times (n/2)$ submultiplications from 8 to 7, trading extra additions for one fewer multiplication. This yields an asymptotic speedup over the $\Theta(n^3)$ methods.

Motivation. Submatrix *multiplication* dominates cost; additions are cheaper. If we can reduce multiplications by a constant factor while adding only $\Theta(n^2)$ work, we can improve the overall exponent.

Setup. Partition as before:

$$A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix}, \quad B = \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix}, \quad C = \begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix}.$$

Form ten $(n/2) \times (n/2)$ sums/differences (all done by index arithmetic in $\Theta(n^2)$ total time):

$$\begin{aligned} &S_1 = B_{12} - B_{22},\ S_2 = A_{11} + A_{12},\ S_3 = A_{21} + A_{22},\ S_4 = B_{21} - B_{11}, \\ &S_5 = A_{11} + A_{22},\ S_6 = B_{11} + B_{22},\ S_7 = A_{12} - A_{22},\ S_8 = B_{21} + B_{22}, \\ &S_9 = A_{11} - A_{21},\ S_{10} = B_{11} + B_{12}. \end{aligned}$$

Compute seven products:

$$\begin{aligned} &P_1 = A_{11}S_1, \quad P_2 = S_2B_{22}, \quad P_3 = S_3B_{11}, \\ &P_4 = A_{22}S_4, \quad P_5 = S_5S_6, \quad P_6 = S_7S_8, \quad P_7 = S_9S_{10}. \end{aligned}$$

Assemble C using 12 additions/subtractions:

$$\begin{aligned} C_{11} &\leftarrow C_{11} + P_5 + P_4 - P_2 + P_6, \\ C_{12} &\leftarrow C_{12} + P_1 + P_2, \\ C_{21} &\leftarrow C_{21} + P_3 + P_4, \\ C_{22} &\leftarrow C_{22} + P_5 + P_1 - P_3 - P_7. \end{aligned}$$

Algorithm 3 STRASSEN-MULTIPLY(A, B, C, n)

if $n = 1$ **then**
    $c_{11} \leftarrow c_{11} + a_{11} \cdot b_{11}$ ▷ base case
    return
end if
Partition A, B, C into $(n/2) \times (n/2)$ blocks (by indices).
Form $S_1, \ldots, S_{10}$ as above. ▷ $\Theta(n^2)$ work
Recursively compute $P_1, \ldots, P_7$ on $(n/2) \times (n/2)$ blocks. ▷ $7T(n/2)$
Update $C_{11}, C_{12}, C_{21}, C_{22}$ using $P_1, \ldots, P_7$. ▷ $\Theta(n^2)$

Recurrence. With index-based partitioning, the running time satisfies

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

Proposition 2. *The recurrence* $T(n) = 7T(n/2)+\Theta(n^2)$ *solves to* $T(n) = \Theta(n^{\lg 7}) = O(n^{2.81})$.

Intuition. Strassen makes the recursion tree *less bushy*: 7 children instead of 8. The $\Theta(n^2)$ extra additions at each level are dominated by the reduction in the number of multiplications across levels, yielding the smaller exponent.

Concluding Remark. Strassen's method is asymptotically faster but introduces more additions and potential numerical-error concerns in floating point. In practice, hybrid implementations switch to classical multiplication at small n for constant-factor wins.

4.3 — The Substitution Method

Historical Context. Divide-and-conquer recurrences often take the form

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

and must be solved to determine algorithmic efficiency. Before systematic theorems, the primary tool was the substitution method: guessing a bound and proving it by induction.

Motivation. The substitution method is flexible: it works even when the master theorem does not apply. The process is constructive, reinforcing intuition about how terms grow and dominate.

Method. To show $T(n) = O(g(n))$:

1. **Guess** a bound $g(n)$ based on the recurrence.
2. **Prove** by induction that $T(n) \leq cg(n)$ for some constant $c > 0$ and sufficiently large n.

Analogously, to show $\Omega(g(n))$, one proves $T(n) \geq cg(n)$.

Example 1. Consider $T(n) = 2T(\lfloor n/2 \rfloor) + n$. Guess $T(n) = O(n \log n)$. *Inductive step:*

$$T(n) \leq 2c\left(\frac{n}{2}\log\frac{n}{2}\right) + n = cn\log n - cn\log 2 + n \leq cn\log n,$$

for $c \geq 1$. Thus $T(n) = O(n \log n)$.

Example 2. Consider $T(n) = T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + \Theta(n)$. Guess $T(n) = O(n \log n)$. The inductive step proceeds similarly, showing the total recurrence cost is bounded by $cn \log n$ for appropriate c.

Theorem 3 (Substitution Method)**.** *If a correct guess $g(n)$ is made, then induction suffices to prove $T(n) = O(g(n))$ or $T(n) = \Omega(g(n))$ for many divide-and-conquer recurrences.*

Limitations. The method requires a good guess, and induction often demands "fudging" constants (subtracting lower-order terms) to make inequalities hold. It may be cumbersome for irregular recurrences.

Concluding Remark. The substitution method is powerful, flexible, and foundational. Even though later techniques (recursion-tree, master theorem) automate many cases, substitution remains essential—especially for non-standard recurrences.

4.4 — The Recursion-Tree Method

Historical Context. The recursion-tree method expands a recurrence into a tree whose nodes represent subproblems. By summing costs across levels, one can visually and arithmetically approximate the total running time. This method provides intuition about how work distributes across recursive calls.

Motivation. Recursion trees illustrate both the number of subproblems generated and the non-recursive work per level. They reveal whether cost is dominated at the root, leaves, or somewhere in between, and thus guide the guess for substitution proofs.

Method. Given a recurrence of the form $T(n) = aT(n/b) + f(n)$:

1. Draw the root with cost $f(n)$.
2. Expand into a children, each of size n/b, with cost $f(n/b)$ each.
3. Continue expanding until reaching base cases of constant size.
4. Sum the costs across levels to estimate the total $T(n)$.

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

- Level 0: cost n.
- Level 1: two subproblems of size $n/2$, each cost $n/2$, total n.
- Level 2: four subproblems of size $n/4$, each cost $n/4$, total n.

Each level contributes n, and there are $\log n$ levels until subproblems of size 1. Total cost $\approx n \log n$.

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

- Level i: 3^i nodes, each with cost $\frac{n}{2^i}$, so total $\left(\frac{3}{2}\right)^i n$.

- Height $\log_2 n$.

Thus the largest contribution is from the leaves, yielding $T(n) = \Theta(n^{\log_2 3}) \approx \Theta(n^{1.585})$.

Theorem 4 (Recursion-Tree Intuition)**.** *For divide-and-conquer recurrences, the total cost is the sum over levels of the recursion tree. The dominant term arises either at the root, the leaves, or across all levels equally.*

Limitations. Recursion trees provide approximation and intuition but may require careful bounding of geometric series. For precise bounds, one often verifies with the substitution method.

Concluding Remark. The recursion-tree method complements substitution by making costs visible. It gives intuition for where asymptotic growth originates and helps shape accurate guesses for rigorous proofs.

4.5 — The Master Method

Historical Context. The master method is a general tool to solve divide-and-conquer recurrences of the form

$$T(n) = aT\left(\frac{n}{b}\right) + f(n),$$

where $a \geq 1$ and $b > 1$. It unifies results from substitution and recursion-tree analysis into three broad cases, giving asymptotic bounds without guesswork.

Motivation. Many algorithms (merge sort, binary search trees, Strassen's multiplication) yield recurrences in this form. The master method provides an efficient way to classify and solve them, replacing ad hoc proofs with a standard theorem.

Theorem (Master Theorem). Let $a \geq 1$, $b > 1$, and $f(n)$ be asymptotically positive. Define $\alpha = \log_b a$. Then:

1. If $f(n) = O(n^{\alpha-\epsilon})$ for some $\epsilon > 0$, then $T(n) = \Theta(n^\alpha)$.
2. If $f(n) = \Theta(n^\alpha \log^k n)$ for some $k \geq 0$, then $T(n) = \Theta(n^\alpha \log^{k+1} n)$.
3. If $f(n) = \Omega(n^{\alpha+\epsilon})$ for some $\epsilon > 0$, and if $af(n/b) \leq cf(n)$ for some $c < 1$ and sufficiently large n (the regularity condition), then $T(n) = \Theta(f(n))$.

Example 1. Merge sort satisfies $T(n) = 2T(n/2) + \Theta(n)$. Here $a = 2$, $b = 2$, so $\alpha = \log_2 2 = 1$. Since $f(n) = \Theta(n)$, Case 2 applies with $k = 0$. Thus $T(n) = \Theta(n \log n)$.

Example 2. Binary search satisfies $T(n) = T(n/2) + \Theta(1)$. Here $a = 1$, $b = 2$, so $\alpha = \log_2 1 = 0$. Since $f(n) = \Theta(1) = \Theta(n^\alpha)$, Case 2 applies with $k = 0$. Thus $T(n) = \Theta(\log n)$.

Example 3. Strassen's algorithm satisfies $T(n) = 7T(n/2) + \Theta(n^2)$. Here $a = 7$, $b = 2$, so $\alpha = \log_2 7 \approx 2.81$. Since $f(n) = \Theta(n^2) = O(n^{\alpha-\epsilon})$ with $\epsilon \approx 0.81$, Case 1 applies. Thus $T(n) = \Theta(n^{\log_2 7})$.

Limitations. The master method does not apply if $f(n)$ does not fit any of the three cases (e.g., oscillating functions or functions between polynomial orders). In such cases, other tools (substitution or Akra–Bazzi theorem) are needed.

Concluding Remark. The master theorem is a powerful classification tool. It captures a wide swath of divide-and-conquer recurrences and provides immediate asymptotic bounds, becoming a cornerstone of algorithm analysis.

4.6 — Proof of the Master Theorem

Historical Context. The master theorem's power lies in its simplicity, but its proof rests on the recursion-tree method. By expanding the recurrence into a tree and bounding costs at each level, one establishes the three distinct asymptotic cases.

Motivation. A proof by recursion tree reveals why the three cases arise:

- Case 1: Work is dominated at the leaves.
- Case 2: Work is spread evenly across levels.
- Case 3: Work is dominated at the root.

This intuition also clarifies why the "regularity condition" is necessary in Case 3.

Proof Sketch. Consider the recurrence

$$T(n) = aT\left(\frac{n}{b}\right) + f(n), \quad a \geq 1,\, b > 1.$$

Build the recursion tree:

- Level 0 (root): cost $f(n)$.
- Level 1: a nodes, each with cost $f(n/b)$, so total $af(n/b)$.
- Level 2: a^2 nodes, each with cost $f(n/b^2)$, so total $a^2 f(n/b^2)$.

Continue until level $\log_b n$ (base case size).

Let $\alpha = \log_b a$. Each level's contribution can be compared to n^α.

Case 1. If $f(n) = O(n^{\alpha-\epsilon})$, then $f(n/b^i) = O((n/b^i)^{\alpha-\epsilon})$. Multiplying by $a^i = (b^\alpha)^i$, the level cost is

$$a^i f(n/b^i) = O\left(n^{\alpha-\epsilon}\left(\frac{a}{b^{\alpha-\epsilon}}\right)^i\right).$$

Since $a/b^{\alpha-\epsilon} < 1$, costs decrease geometrically with i. Thus the root dominates and $T(n) = \Theta(n^\alpha)$.

Case 2. If $f(n) = \Theta(n^\alpha \log^k n)$, then each level contributes $\Theta(n^\alpha \log^k(n/b^i))$. There are $\Theta(\log n)$ levels, so summing yields $\Theta(n^\alpha \log^{k+1} n)$.

Case 3. If $f(n) = \Omega(n^{\alpha+\epsilon})$ and satisfies the regularity condition $af(n/b) \leq cf(n)$ for some $c < 1$, then costs decrease as we descend the tree. Thus the root dominates, and $T(n) = \Theta(f(n))$.

Conclusion of Proof. The three cases cover all polynomially comparable forms of $f(n)$, justifying the master theorem's classification.

Concluding Remark. The proof shows that the master theorem is not a "black box" but an immediate consequence of recursion-tree analysis. The regularity condition prevents pathological $f(n)$ that would otherwise cause lower levels to contribute comparably.

4.7 — The Akra–Bazzi Method

Historical Context. The Akra–Bazzi theorem (1990s) generalizes the master theorem. It handles recurrences where subproblem sizes are not equal divisions, or where additional polynomially bounded perturbations appear. This makes it a more powerful tool than the classical three-case master method.

Motivation. Many divide-and-conquer algorithms yield recurrences of the form

$$T(x) = \sum_{i=1}^{k} a_i T(b_i x + h_i(x)) + g(x),$$

where $0 < b_i < 1$, $a_i > 0$, and $h_i(x)$ are lower-order perturbations. The master theorem does not cover such irregular forms, but Akra–Bazzi does.

Theorem (Akra–Bazzi). Let $a_i > 0$, $0 < b_i < 1$, and suppose $g(x)$ is asymptotically positive and locally integrable. Define p as the unique real number satisfying

$$\sum_{i=1}^{k} a_i b_i^p = 1.$$

Then for sufficiently large x,

$$T(x) = \Theta\left(x^p \left(1 + \int_1^x \frac{g(u)}{u^{p+1}}\, du\right)\right).$$

Example 1. Consider $T(n) = T(n/2) + T(n/3) + n$. Here $a_1 = a_2 = 1$, $b_1 = 1/2$, $b_2 = 1/3$. Solve for p:

$$\left(\tfrac{1}{2}\right)^p + \left(\tfrac{1}{3}\right)^p = 1.$$

This yields $p \approx 0.79$. Since $g(n) = n$, the integral term contributes a factor $\Theta(n)$. Thus $T(n) = \Theta(n)$.

Example 2. For $T(n) = 2T(n/2 + \sqrt{n}) + n$, the perturbation $+\sqrt{n}$ prevents direct application of the master theorem, but Akra–Bazzi accommodates it. Solving $2(1/2)^p = 1$ gives $p = 1$. The integral term adds a $\log n$ factor, so $T(n) = \Theta(n \log n)$.

Intuition. The exponent p generalizes the $\log_b a$ from the master theorem: it balances the branching factor a_i with shrinkage factors b_i. The integral accounts for the cumulative effect of the additive function $g(x)$.

Concluding Remark. The Akra–Bazzi method completes the divide-and-conquer toolkit. Where substitution provides flexibility, recursion trees provide intuition, and the master theorem provides classification, Akra–Bazzi extends coverage to irregular recurrences beyond the master theorem's reach.