COMP3121 Lecture 7 Notes: Divide and Conquer

Module 2: Divide and Conquer

- **Instructor**: Paul Hunter (K17-217D)
- **Course Admins**: `cs3121@unsw.edu.au` (Song, Liam, James)
- **School of Computer Science and Engineering**, UNSW Sydney

- **Term 2, 2025**

Table of Contents

1. Introductory Examples

1.1 Coin Puzzle

- **Problem**: Find counterfeit (lighter) coin among 27 coins using only 3 weighings
- **Solution**:
    1. Divide into 3 groups of 9 (A, B, C)

2. Weigh A vs B
   - If unequal: lighter group contains counterfeit
   - If equal: counterfeit is in C
3. Repeat with groups of 3, then individual coins

- Classic divide-and-conquer example - reduces problem size by 1/3 each step

1.2 Binary Search

- **Steps**:
  - **Divide**: Test midpoint ($\Theta(1)$)
  - **Conquer**: Search one side recursively
  - **Combine**: Pass answer up ($\Theta(1)$)
- Time complexity: $\Theta(\log n)$ (recursion depth $\log_2 n$)
- **Extensions**:
  - Lower bound: smallest i where $A[i] \geq x$
  - Upper bound: largest i where $A[i] \leq x$
  - Equal range: find all indices with $A[\ell] = \cdots = A[r] = x$

1.3 Merge Sort

- **Steps**:
  - **Divide**: Split array equally ($\Theta(1)$)
  - **Conquer**: Sort halves recursively
  - **Combine**: Merge sorted subarrays ($\Theta(n)$)
- Time complexity: $\Theta(n \log n)$ ($\log n$ levels, $\Theta(n)$ work per level)
- **Counting Inversions**:
  - Measure of array "disorder" (pairs where $i < j$ but $A[i] > A[j]$)
  - Modified merge sort counts inversions in $\Theta(n \log n)$ time
  - Key insight: inversions = left inversions + right inversions + split inversions

1.4 Quick Sort

- **Steps**:
  - **Divide**: Choose pivot, partition ($\Theta(n)$)
  - **Conquer**: Sort partitions recursively
  - **Combine**: Pass answer up ($\Theta(1)$)
- Average case: $\Theta(n \log n)$
- Worst case: $\Theta(n^2)$ (bad pivot choices)

1.5 Divide and Conquer Paradigm

- **General Approach**:
  1. **Divide**: Split problem into smaller subproblems
  2. **Conquer**: Solve subproblems recursively
  3. **Combine**: Merge solutions to solve original problem
- **Correctness**: Proved by induction on problem size
  - Base case: trivial solution
  - Inductive step: assume works for smaller sizes, show combine step works

2. Recurrences

2.1 Framework

- General form: $T(n) = aT(n/b) + f(n)$
  - a: number of subproblems
  - b: size reduction factor
  - $f(n)$: divide/combine cost
- Examples:
  - Binary search: $T(n) = T(n/2) + \Theta(1)$
  - Merge sort: $T(n) = 2T(n/2) + \Theta(n)$
  - Karatsuba: $T(n) = 3T(n/2) + \Theta(n)$

2.2 Master Theorem

For recurrences of form $T(n) = aT(n/b) + f(n)$ with $a \geq 1$, $b > 1$:

1. If $f(n) = O(n^{c^*-\epsilon})$ for $\epsilon > 0$, then $T(n) = \Theta(n^{c^*})$
2. If $f(n) = \Theta(n^{c^*} \log^k n)$, then $T(n) = \Theta(n^{c^*} \log^{k+1} n)$
3. If $f(n) = \Omega(n^{c^*+\epsilon})$ and regularity condition holds, then $T(n) = \Theta(f(n))$

where $c^* = \log_b a$ is the critical exponent.

3. Integer Multiplication

3.1 Naive D&C Approach

- Split n-bit numbers A, B into halves:
$$A = A_1 2^{n/2} + A_0$$
$$B = B_1 2^{n/2} + B_0$$
- Compute recursively:
$$AB = A_1 B_1 2^n + (A_1 B_0 + A_0 B_1) 2^{n/2} + A_0 B_0$$
- Recurrence: $T(n) = 4T(n/2) + \Theta(n) \Rightarrow T(n) = \Theta(n^2)$ (no improvement)

3.2 Karatsuba Algorithm

- Key insight: Compute only three products:

$$\begin{aligned}
W &= A_1 B_1 \\
X &= A_0 B_0 \\
V &= (A_1 + A_0)(B_1 + B_0) \\
AB &= W 2^n + (V - W - X) 2^{n/2} + X
\end{aligned}$$

- Recurrence: $T(n) = 3T(n/2) + \Theta(n)$
- Solution: $T(n) = \Theta(n^{\log_2 3}) \approx \Theta(n^{1.585})$

4. Convolutions and FFT

4.1 Polynomials and Convolutions

- Polynomial multiplication equivalent to convolution:

$$C_t = \sum_{i+j=t} A_i B_j$$

- Naive approach: $\Theta(n^2)$ time
- **Value Representation**:
  - Represent degree-n polynomial by $n+1$ point-value pairs
  - Multiplication: pointwise multiply values ($\Theta(n)$)

4.2 Fast Fourier Transform

- **Strategy**:
  1. Evaluate polynomials at roots of unity (FFT)
  2. Pointwise multiply values
  3. Interpolate to get coefficients (inverse FFT)
- **Roots of Unity**:
  - $\omega_m = e^{2\pi i/m}$ (primitive m-th root)
  - Properties:
    * $\omega_m^m = 1$
    * $\omega_m^{m/2} = -1$
    * $\omega_m^{m+k} = \omega_m^k$
- **FFT Algorithm**:
  - Divide: Split into even/odd coefficients
  - Conquer: Compute FFT of halves
  - Combine: Use ω_m^k properties
  - Time: $T(n) = 2T(n/2) + \Theta(n) \Rightarrow \Theta(n \log n)$
- **Applications**:
  - Polynomial multiplication
  - integer multiplication (via polynomial evaluation)
  - Signal processing, image compression

5. Pirate Puzzle

- **Problem**: 5 pirates must divide 100 gold bars with voting rules
- **Priorities**:
  1. Survival
  2. Maximize gold
  3. See others walk plank
- **Solution Approach**: Backwards induction
  - Solve for 1 pirate, then 2, ..., up to 5
  - At each step, pirates anticipate next steps
- **Final Solution**: First pirate proposes (98, 0, 1, 0, 1)