Master Theorem

ku**upstream**: Divide and Conquer
topic:
type: #notes
2025-02-02

Master Theorem for Divide and Conquer Algorithms

Context: Why Study the Master Theorem?

When working with **divide and conquer** algorithms, it's crucial to understand their runtime complexity. Many such algorithms have a recurrence relation describing their runtime. The **Master Theorem** is a powerful tool that provides an efficient way to analyze these relations without manually expanding the recursion.

What Does the Master Theorem Solve?

The Master Theorem applies to recurrence relations of the form:

$$T(n) = aT\left(\frac{n}{b}\right) + O(n^d)$$

Where:

- $T(n)$: The runtime of the algorithm for input size n.
- a: The number of subproblems the algorithm splits into.
- b: The factor by which the problem size is reduced.
- $O(n^d)$: The cost of dividing the problem and combining the results.

The Master Theorem Statement

To analyze $T(n)$, compare n^d (the cost of the non-recursive work) with $n^{\log_b a}$ (the growth rate of the subproblem work). There are three cases:

1. **Case 1: Subproblem dominates ($n^{\log_b a} > n^d$)**
   - If $\log_b a > d$, then:

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

2. **Case 2: Balance between subproblem and combination ($n^{\log_b a} = n^d$)**

- If $\log_b a = d$, then:

$$T(n) = \Theta\left(n^d \log n\right)$$

3. **Case 3: Combination dominates ($n^{\log_b a} < n^d$)**
    - If $\log_b a < d$, then:

$$T(n) = \Theta\left(n^d\right)$$

Breaking Down the Symbols

- $T(n)$: Represents the runtime of the algorithm for input size n.
- $\Theta(f(n))$: Big-Theta notation, meaning the runtime is bounded both above and below by $f(n)$ for large n. In other words, $T(n)$ grows asymptotically at the same rate as $f(n)$.
- a: Number of subproblems.
- b: The factor by which the input size is reduced in each recursive step.
- d: Exponent of the non-recursive work, $O(n^d)$.
- $\log_b a$: Represents the growth rate of the recursive work.

How to Remember the Master Theorem

To remember the three cases of the Master Theorem, think of the acronym **SBC**:

- **S**: **Subproblem Dominates** ($\log_b a > d$): Recursive work grows faster, so $T(n) = \Theta(n^{\log_b a})$.
- **B**: **Balance** ($\log_b a = d$): Both recursive and non-recursive work grow at the same rate, so $T(n) = \Theta(n^d \log n)$.
- **C**: **Combination Dominates** ($\log_b a < d$): Non-recursive work grows faster, so $T(n) = \Theta(n^d)$.

Mnemonic: **"Solve By Comparing"** (Subproblem vs Balance vs Combination).

Practical Examples

1. **Merge Sort**
    - Recurrence: $T(n) = 2T\left(\frac{n}{2}\right) + O(n)$
    - Here:
        - $a = 2$, $b = 2$, $d = 1$
        - $\log_b a = \log_2 2 = 1$
        - Since $\log_b a = d$, this is **Case 2 (Balance)**:

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

2. **Binary Search**
   - Recurrence: $T(n) = T\left(\frac{n}{2}\right) + O(1)$
   - Here:
     - $a = 1$, $b = 2$, $d = 0$
     - $\log_b a = \log_2 1 = 0$
     - Since $\log_b a = d$, this is **Case 2 (Balance)**:

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

3. **Strassen's Matrix Multiplication**
   - Recurrence: $T(n) = 7T\left(\frac{n}{2}\right) + O(n^2)$
   - Here:
     - $a = 7$, $b = 2$, $d = 2$
     - $\log_b a = \log_2 7 \approx 2.81$
     - Since $\log_b a > d$, this is **Case 1 (Subproblem Dominates)**:

$$T(n) = \Theta(n^{\log_2 7})$$

Key Takeaways

1. **Compare recursive and non-recursive growth rates:** The Master Theorem simplifies the process of analyzing divide and conquer recurrences.
2. **Understand the three cases:** Determine which component of the algorithm dominates the runtime.
3. **Practical tool for efficiency analysis:** Use it to quickly estimate the runtime of algorithms like Merge Sort, Binary Search, and Strassen's Matrix Multiplication.

Practical Mnemonic

Use **SBC (Solve By Comparing)** to decide between Subproblem Dominates, Balance, or Combination Dominates.