Recap

- **Recursion:** A problem-solving technique where a function calls itself to solve smaller instances of the same problem.
  - **Tree Method**
    - Visualizes the recursive calls as a tree.
    - Helps in making an **educated guess** for the time complexity by summing the work done at each level of the tree.
  - **Substitution Method**
    - Used to **verify** the correctness of a guessed solution.
    - Proving the guess using induction (base case and inductive step).
  - **Master Theorem**
    - Provides a direct way to solve recurrence relations.

Divide & Conquer Example: Median of Medians

Problem: Find the median of an unsorted array efficiently.

- **Divide:**
  - Split the array into groups of 5 elements
  - Find the median of each group
  - Recurse on the list of medians to find the "median of medians"
- **Conquer:**
  - Use the pivot to partition the array into:
    - Elements less than the pivot
    - Elements greater than the pivot
  - Recurse on the appropriate subarray to find the median
- **Combine:**
  - The pivot itself is the median or is used in further recursion

Time Complexity Analysis

Time Complexity:

- **Divide:** $T(n/5)$ — recurse on the list of medians.
- **Conquer:** $T(7n/10)$ — recurse on the larger subarray.
- **Combine:** $O(n)$ — partition the array around the pivot.
- **Overall:** $T(n) = T(n/5) + T(7n/10) + O(n) \Rightarrow O(n)$.

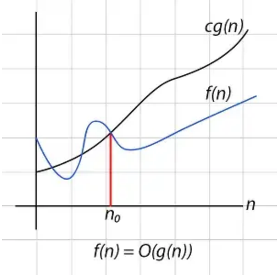
Recap of Asymptotic Notation: Big O (Upper Bound)

Definition:

$f(n) = O(g(n))$ if $\exists$ constants $c > 0$ and $n_0 \geq 1$ such that:

$$f(n) = O(g(n)) \iff f(n) \leq c \cdot g(n) \quad \forall n \geq n_0, \quad \exists c > 0,\ n_0 \geq 1$$

- **Intuition:**
  - $f(n)$ grows **no faster** than $g(n)$.
  - Describes the **upper bound** on growth.
- **Example:**
  - If $f(n) = 3n^2 + 2n + 1$, then $f(n) = O(n^2)$.
  - Here, $c = 4$ and $n_0 = 1$ satisfy the definition.



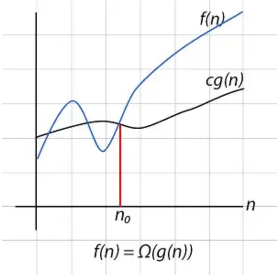
Recap of Asymptotic Notation: Big Omega (Lower Bound)

Definition:

$f(n) = \Omega(g(n))$ if $\exists$ constants $c > 0$ and $n_0 \geq 1$ such that:

$$f(n) = \Omega(g(n)) \iff f(n) \geq c \cdot g(n) \quad \forall n \geq n_0, \quad \exists n_0 \geq 1, \quad 0 < c \leq 1$$

- **Intuition:**
  - $f(n)$ grows **no slower than** $g(n)$.
  - Describes the **lower bound** on growth.
- **Example:**
  - If $f(n) = 3n^2 + 2n + 1$, then $f(n) = \Omega(n^2)$.
  - Here, $c = 3$ and $n_0 = 1$ satisfy the definition.



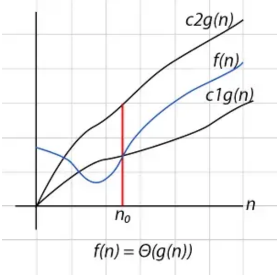
Recap of Asymptotic Notation: Big Theta (Tight Bound)

Definition:

$f(n) = \Theta(g(n))$ if $\exists$ constants $c_1, c_2 > 0 \wedge n_0 \geq 1$ such that:

$$f(n) = \Theta(g(n)) \iff f(n) = O(g(n)) \text{ and } f(n) = \Omega(g(n))$$

- **Intuition:**
  - $f(n)$ is **asymptotically equal** to $g(n)$.
  - $f(n) = O(g(n))$ and $f(n) = \Omega(g(n))$ simultaneously.
- **Example:**
  - If $f(n) = 3n^2 + 2n + 1$, then $f(n) = \Theta(n^2)$.
  - Here, $c_1 = 3$, $c_2 = 4$, and $n_0 = 1$ satisfy the definition.



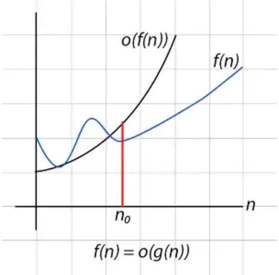
Little o

Definition:

$f(n) = o(g(n))$ if for **every** constant $c > 0$, there exists a constant $n_0 \geq 1$ such that:

$$0 \leq f(n) < c \cdot g(n) \quad \forall n \geq n_0$$

- **Intuition**: - $f(n)$ grows **strictly slower** than $g(n)$. - Unlike Big O, this is a **non-tight** upper bound.



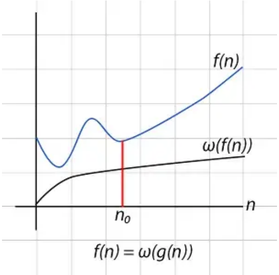
Little ω

Definition:

$f(n) = \omega(g(n))$ if for **every** constant $c > 0$, there exists a constant $n_0 \geq 1$ such that:

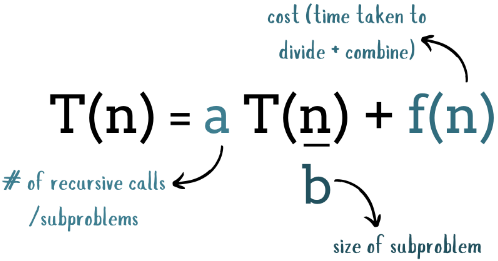
$$0 \leq c \cdot g(n) < f(n) \quad \forall n \geq n_0$$

- **Intuition**: - $f(n)$ grows **strictly faster** than $g(n)$. - Unlike Big Omega, this is a **non-tight** lower bound.



Master Theorem

The Master Theorem determines the asymptotic time complexity $T(n)$ by comparing $f(n)$ with $O(n^{\log_b a})$. The final complexity depends on which term dominates.



Case 1: Cost remains equal to each level

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

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

- **Intuition**:
  - The cost $f(n)$ is **equal** to $n^{\log_b a}$ at every level.

Case 2: Cost decreases at each level

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

- **Intuition**:
  - The cost $f(n)$ is **asymptotically smaller** than $n^{\log_b a}$.

Case 3: Cost increases at each level

$$f(n) = \omega(n^{\log_b a})$$

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

- **Intuition**:
  - The cost $f(n)$ is **asymptotically larger** than $n^{\log_b a}$.

Book vs Professor's Terminology: Asymptotic Equivalences

Case 2: Growth Rate Below Threshold

$$f(n) = o(n^{\log_b a}) \iff f(n) = O(n^{\log_b a - \epsilon}), \quad \epsilon > 0$$

Case 3: Growth Rate Above Threshold

$$f(n) = \omega(n^{\log_b a}) \iff f(n) = \Omega(n^{\log_b a + \epsilon}), \quad \epsilon > 0$$

Example 1

Recurrence

$$T(n) = 8T(n/2) + cn$$

Solution:

- Compare $f(n) = cn$ with $n^{\log_b a} = n^{\log_2 8} = n^3$.
- Since $f(n) = O(n^{\log_b a - \epsilon})$ for $\epsilon > 0$, **Case 2** applies.
- $T(n) = \Theta(n^3)$.

Example 2

Recurrence

$$T(n) = 8T(n/2) + c^2 n$$

Solution:

- Similar to Example 1, $f(n) = c^2 n$ is dominated by n^3.
- **Case 2** applies.
- $T(n) = \Theta(n^3)$.

Example 3

Recurrence

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

Solution:

- Compare $f(n) = cn^2$ with $n^{\log_b a} = n^{\log_2 4} = n^2$.
- Since $f(n) = \Theta(n^{\log_b a})$, **Case 1** applies.
- $T(n) = \Theta(n^2 \log n)$.

Example 4

Recurrence

$$T(n) = T(n/4) + T(5n/8) + cn$$

Solution:

- This recurrence does not fit the standard Master Theorem form.
- Use the **convexity property**: $T(n/4) + T(5n/8) \leq T(7n/8) + cn$.
- $T(n) = \Theta(n)$.

Example 5

Recurrence

$$T(n) = \sqrt{n}\,T(n/\sqrt{n}) + c\sqrt{n}$$

Solution:

- a and b are not constants here.
- The Master Theorem **cannot be applied**.
- Use other methods (e.g., substitution or recursion tree).

Example 6

Recurrence

$$T(n) = 2T(n/2) + \frac{n}{\log n}$$

Solution:

- Compare $f(n) = \frac{n}{\log n}$ with $n^{\log_b a} = n^{\log_2 2} = n$.
- $f(n)$ is not polynomially larger or smaller than n.
- The Master Theorem **does not apply directly**.
- Use the **extended Master Theorem** or other techniques.
- $T(n) = \Theta(n^{\log_b a} \log \log n)$.

Example 7

Recurrence

$$T(n) = 2T(n/2) - n$$

Solution:

- $f(n) = -n$ is **negative**.
- The Master Theorem **cannot be applied**.
- Use other methods (e.g., substitution or recursion tree).

Key Notes

- The Master Theorem requires a, b to be constants.
- If $f(n)$ is negative, the Master Theorem **cannot be applied**.
- For recurrences not fitting the standard form, use **recursion tree**, or **substitution methods**.

Why Do We Need Sorting?

- **Searching:** If we can search faster, every computing problem can be solved.
- Suppose there is a problem P, define its solution space (geometric space in higher dimensions).
- All solutions (both valid and invalid) are present in this space.
- The only problem left is **searching** for the correct solution.

Quick Sort: Randomized Select Variant

Algorithm:

1. If $|A| < k$ (small size), use **brute force**:
   - Return the i-th element directly.
2. Guess a median g.
3. Partition the array into:
   - L (elements $< g$),
   - R (elements $> g$).
4. Recursively call **QuickSort** on L and R.
5. Return the concatenated result: $\text{ceil}(L, g, R)$.

Best Case Analysis

- The guessed median g is the **actual median**.
- Recurrence relation:

$$T(n) = 2T\left(\frac{n}{2}\right) + cn$$

- Using the **Master Theorem**:

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

Worst Case Analysis

- The guessed median g is the **first or last element**.
- Recurrence relation:

$$T(n) = T(n-1) + cn$$

- Solving the recurrence:

$$T(n) = \Theta(n^2)$$