Picture is from the cover of the textbook CLRS.
# Asymptotic Notations in a Nutshell

**Definition**

\[ \text{If } f(n) \in O(g(n)) \iff \exists M > 0, \exists n_0 > 0 \text{ s.t. } \forall n > n_0, f(n) \leq M \cdot g(n) \]

**Definition**

\[ \text{If } f(n) \in o(g(n)) \iff \forall M > 0, \exists n_0 > 0 \text{ s.t. } \forall n > n_0, f(n) < M \cdot g(n) \]

**Definition**

\[ \text{If } f(n) \in \Omega(g(n)) \iff \exists M > 0, \exists n_0 > 0 \text{ s.t. } \forall n > n_0, f(n) \geq M \cdot g(n) \]

**Definition**

\[ \text{If } f(n) \in \omega(g(n)) \iff \forall M > 0, \exists n_0 > 0 \text{ s.t. } \forall n > n_0, f(n) > M \cdot g(n) \]

**Definition**

\[ \text{If } f(n) \in \Theta(g(n)) \iff \exists M_1, M_2 > 0, \exists n_0 > 0 \text{ s.t.} \\
\forall n > n_0, M_1 \cdot g(n) \leq f(n) \leq M_2 \cdot g(n) \]
Common Growth Rates

- $\Theta(1) \rightarrow$ constant complexity
  - e.g., an algorithms that only samples a constant number of inputs
- $\Theta(\log n) \rightarrow$ logarithmic complexity
  - Binary search
- $\Theta(n) \rightarrow$ linear complexity
  - Most practical algorithms :)
- $\Theta(n \log n) \rightarrow$ pseudo-linear complexity
  - Optimal comparison based sorting algorithms, e.g., merge-sort
- $\Theta(n^2) \rightarrow$ Quadratic complexity
  - naive sorting algorithms (Bubble sort, insertion sort)
- $\Theta(n^3) \rightarrow$ Cubic Complexity
  - naive matrix multiplication
- $\Theta(2^n) \rightarrow$ Exponential Complexity
  - The ‘algorithm’ terminates but the universe is likely to end much earlier even if $n \approx 1000.$
Techniques for Comparing Growth Rates

Assume the running time of two algorithms are given by functions \( f(n) \) and \( g(n) \) and let

\[
L = \lim_{n \to \infty} \frac{f(n)}{g(n)}
\]

Then

\[
f(n) \in \begin{cases} 
  o(g(n)) & \text{if } L = 0 \\
  \Theta(g(n)) & \text{if } 0 < L < \infty \\
  \omega(g(n)) & \text{if } L = \infty 
\end{cases}
\]

If the limit is not defined, we need another method.

Note that we cannot compare two algorithms using big \( O \) and \( \Omega \) notations.

E.g., algorithm \( A \) can have complexity \( O(n^2) \) and algorithm \( B \) has complexity \( O(n^3) \). We cannot state that \( A \) is faster than \( B \) (why?)
Fun with Asymptotic Notations

- Compare the grow-rate of $\log n$ and $n^r$ where $r$ is a positive real number.
Fun with Asymptotic Notations

- Prove that $n(s\sin(n) + 2)$ is $\Theta(n)$.
- Use the definition since the limit does not exist.
  - Define $n_0, M_1, M_2$ so that $\forall n > n_0$ we have $M_1 n(s\sin(n) + 2) \leq n \leq qM_2 n(s\sin(n) + 2)$.
  - $M_1 = 1/3, M_2 = 1, n_0 = 1$ work!
Fun with Asymptotic Notations

- The same relationship that holds for relative values of numbers hold for asymptotic.
  - E.g., if \( f(n) \in O(g(n)) \) [\( f(n) \) is asymptotically smaller than or equal to \( g(n) \)], then we have \( g(n) \in \Omega(f(n)) \) [\( g(n) \) is asymptotically larger than or equal to \( f(n) \)].

- In order to prove \( f(n) \in \Theta(g(n)) \), we often show that \( f(n) \in O(n) \) and \( f(n) \in \Omega(g(n)) \).

- Similarly, we have transitivity in asymptotic notations: if \( f(n) \in O(g(n)) \) and \( g(n) \in O(h(n)) \), we have \( f(n) \in O(h(n)) \).

- Max rule: \( f(n) + g(n) \in \Theta(\max\{f(n), g(n)\}) \).
  - E.g., \( 2n^3 + 8n^2 + 16n \log n \in \Theta(\max\{2n^3, 8n^2, 16n \log n\}) = \Theta(n^3) \).
Fun with Asymptotic Notations

- **What is the time complexity of arithmetic sequences?**
  \[ \sum_{i=0}^{n-1} (a + di) = na + \frac{dn(n-1)}{2} \in \Theta(n^2) \]

- **What about geometric sequence?**
  \[ \sum_{i=0}^{n-1} ar^i = \begin{cases} 
  a \frac{1-r^n}{1-r} \in \Theta(1) & \text{if } 0 < r < 1 \\
  na \in \Theta(n) & \text{if } r = 1 \\
  a \frac{r^n-1}{r-1} \in \Theta(r^n) & \text{if } r > 1 
\end{cases} \]

- **What about Harmonic sequence?**
  \[ H_n = \sum_{i=1}^{n} \frac{1}{i} \approx \ln(n) + \gamma \in \Theta(\log n) \] (\(\gamma\) is a constant \(\approx 0.577\))
Loop Analysis

- Identify **elementary operations** that require constant time.
- The complexity of a loop is expressed as the **sum** of the complexities of each iteration of the loop.
- Analyse independent loops separately, and then **add** the results (use “maximum rules” and simplify when possible).
- If loops are nested, start with the innermost loop and proceed outwards.
Example of Loop Analysis

**Algo1** \((n)\)
1. \(A \leftarrow 0\)
2. for \(i \leftarrow 1\) to \(n\) do
3. \hspace{1em} for \(j \leftarrow i\) to \(n\) do
4. \hspace{2em} \(A \leftarrow A / (i - j)^2\)
5. \hspace{1em} \(A \leftarrow A^{100}\)
6. return \(\text{sum}\)
Example of Loop Analysis

**Algo2** \((A, n)\)

1. \(max \leftarrow 0\)
2. \(\text{for } i \leftarrow 1 \text{ to } n \text{ do}\)
3. \(\text{for } j \leftarrow i \text{ to } n \text{ do}\)
4. \(X \leftarrow 0\)
5. \(\text{for } k \leftarrow i \text{ to } j \text{ do}\)
6. \(X \leftarrow A[k]\)
7. \(\text{if } X > max \text{ then}\)
8. \(max \leftarrow X\)
9. \(\text{return } max\)
Example of Loop Analysis

Algo3 \((n)\)

1. \(X \leftarrow 0\)
2. \(\text{for } i \leftarrow 1 \text{ to } n^2 \text{ do}\)
3. \(\quad j \leftarrow i\)
4. \(\quad \text{while } j \geq 1 \text{ do}\)
5. \(\quad \quad X \leftarrow X + i/j\)
6. \(\quad \quad j \leftarrow \lfloor j/2 \rfloor\)
7. \(\quad \text{return } X\)
MergeSort

Sorting an array $A$ of $n$ numbers

- **Step 1:** We split $A$ into two subarrays: $A_L$ consists of the first $\lceil \frac{n}{2} \rceil$ elements in $A$ and $A_R$ consists of the last $\lfloor \frac{n}{2} \rfloor$ elements in $A$.

- **Step 2:** Recursively run $MergeSort$ on $A_L$ and $A_R$.

- **Step 3:** After $A_L$ and $A_R$ have been sorted, use a function $Merge$ to merge them into a single sorted array. This can be done in time $\Theta(n)$. 


MergeSort

MergeSort($A, n$)
1. **if** $n = 1$ **then**
2. \hspace{1em} $S \leftarrow A$
3. **else**
4. \hspace{1em} $n_L \leftarrow \left\lceil \frac{n}{2} \right\rceil$
5. \hspace{1em} $n_R \leftarrow \left\lfloor \frac{n}{2} \right\rfloor$
6. \hspace{1em} $A_L \leftarrow [A[1], \ldots, A[n_L]]$
7. \hspace{1em} $A_R \leftarrow [A[n_L + 1], \ldots, A[n]]$
8. \hspace{1em} $S_L \leftarrow \text{MergeSort}(A_L, n_L)$
9. \hspace{1em} $S_R \leftarrow \text{MergeSort}(A_R, n_R)$
10. \hspace{1em} $S \leftarrow \text{Merge}(S_L, n_L, S_R, n_R)$
11. **return** $S$
The following is the corresponding sloppy recurrence (it has floors and ceilings removed):

\[
T(n) = \begin{cases} 
2T\left(\frac{n}{2}\right) + cn & \text{if } n > 1 \\
 d & \text{if } n = 1.
\end{cases}
\]

The exact and sloppy recurrences are identical when \( n \) is a power of 2.

The recurrence can easily be solved by various methods when \( n = 2^j \). The solution has growth rate \( T(n) \in \Theta(n \log n) \).

It is possible to show that \( T(n) \in \Theta(n \log n) \) for all \( n \) by analyzing the exact recurrence.
Analysis of Recursions

- **Substitution method**
  - **Guess** the growth function and prove it using induction.
    - For merge-sort, prove $T(n) < Mn \log n$.
    - This holds for $n = 2, n = 3$ (base of induction).
    - Fix a value of $n$ and assume the inequality holds for smaller values.
      we have $T(n) = 2T(n/2) + cn \leq 2M(n/2 \log n/2) + cn = Mn \log n - MN + cn \leq Mn \log n + cn$ (the inequality comes from induction hypothesis)

- **Limited Master theorem**

  $$T(n) = \begin{cases} a \cdot T \left( \frac{n}{b} \right) + n^c & \text{if } n > 1 \\ d & \text{if } n = 1. \end{cases}$$

  - if $\log_b a > c$, then $T(n) \in \Theta(n^{\log_b a})$
  - if $\log_b a = c$ then $T(n) \in \Theta(n^c \log n)$
  - if $\log_b a < c$ then $T(n) \in \Theta(n^c)$