Requesting extensions

Running time of Algorithms:

- Implement the algorithm & check it against many inputs.

Drawbacks:

- Implement the alg.
- We cannot test against all inputs.
- If we want to compare two alg's then they have to be implemented using the same software & hardware environments.

Analytical analysis: take into account all inputs and allows us to compare
algorithms independent of the hardware and software environments.

RAM model of computation:

- all high-level instructions take constant time.
  - assignment, comparison, arithmetic operations,
  - indexing into an array, function call...

- all memory accesses take constant time.

Simplistic model.

Worst case.

Avg. case $\rightarrow$ prob. distr. of I/Os.
Asymptotic analysis of algorithms.

- rate of growth
- dropping lower order terms
- ignore the leading constants.

Big-O 'O' asymptotic upper bound.

\[ O(g(n)) = \{ f(n) \mid \exists \text{ positive constants } c \text{ and } n_0, \text{ such that } 0 \leq f(n) \leq cg(n), \text{ for } n \geq n_0 \} \]
\[ f(n) \in O(g(n)) \quad \equiv \quad f(n) = O(g(n)). \]

**Big-Omega (Ω)**

\[ \Omega(g(n)) = \left\{ f(n) \mid \exists \text{ positive const. } c \text{ and } n_0 \text{ s.t. } f(n) \geq cg(n), \forall n \geq n_0 \right\} \]

**Big-Theta (Θ)**

\[ \Theta(g(n)) = \left\{ f(n) \mid \exists \text{ positive consts., } c_1, c_2, n_0 \right\} \]
tight bound \[ c_1 g(n) \leq f(n) \leq c_2 g(n), \quad n \to \infty \]

Note that \( f(n) = \Theta(g(n)) \) if and only if
\[ f(n) = O(g(n)) \quad \text{and} \quad f(n) = \Omega(g(n)). \]

**Little-o, \( o \)**

\[ f(n) = o(g(n)) \quad \text{if} \quad f(n) = O(g(n)), \quad \text{but} \quad f(n) \neq \Theta(g(n)). \]

**Little omega, \( \omega \)**

\[ f(n) = \omega(g(n)) \quad \text{if} \quad f(n) = \Omega(g(n)), \quad \text{but} \quad f(n) \neq \Theta(g(n)). \]
Ex: Prove that \( T_{n-2} = \Theta(n) \).

Proof: We will start by showing that
\[ T_{n-2} = O(n) \]
\[ T_{n-2} \leq T_n, \quad \forall n \geq 1. \]
\[ c_2 = 7, \quad \mu_0 = 1 \]

What remains to show is that
\[ T_{n-2} = \Omega(n) \]
i.e., \( T_{n-2} \geq c_1 n, \quad \forall n \geq \mu_0' \).
i.e., \( (7 - c_4) n \geq 2, \quad \forall n \geq \mu_0'' \).

Let \( c_4 = 2, \quad \mu_0'' = 3 \)

\[ \therefore T_{n-2} = \Omega(n) \]
\[ c_1 = 1, \ c_2 = 2, \ \eta_0 = 3 \]

**Ex:** Prove that \( 100n^3 + 55n^2 \eta n + 25n + 18 = O(n^3) \)

**Soh:**

\[
100n^3 + 55n^2 \eta n + 25n + 18 \\
\leq 10000n^3 + 1000n^2 \eta + 100n^3 + 1000 \eta^2 n^2,
\]

\[
\forall \eta > 1.
\]

\[
= 4000n^3, \quad \forall n \geq 1.
\]

\[
\therefore c = 4000, \quad \eta_0 = 1.
\]

**Ex:** Prove that \( \underbrace{3^{100}}_{\text{100 times}} = O(1) \).

**Soh:** Note that

\[
3^{100} \leq \underbrace{3^{100}}_{\text{100 times}}, \quad \forall n \geq 1.
\]
\[ c = 3^{100}, \quad n_0 = 1. \Rightarrow \]

**Exp:** Prove that \( \log n = O(n) \).
\[
\left[ \log n = O(n) \right]
\]

**Proof:** We will prove that
\[
\log n \leq \frac{1}{3}n, \quad \forall n \geq 1
\]

**IH:** Assume that
\[
\log k \leq k, \quad \text{for some } n \leq k \geq 1.
\]

**BC:** \( \log 1 = 0 \leq 1 \)

**IS:** Want to prove that
\[
\log (k+1) \leq k+1
\]

**LHS
\[
= \log k + \log 1 \leq k + 1
\]

\( \times \) Bogus.

(By IH)
\[ 
\text{LHS} = \lg (k + 1) \\
\leq \lg (k + k) \\
= \lg (2k) \\
= \lg 2 + \lg k \\
\leq 1 + k \quad \text{(By IH)}. \checkmark \\
\]

\textbf{Ex: } Prove that \(3^{\frac{100}{n}} = O(2^n)\).

\textbf{Soh: } We will prove that

\[3^{\frac{100}{n}} \leq 3^{\frac{100}{100}} 2^{\frac{200}{n}}, \quad \forall n \geq 100\]

use induction on \(n\).
IH: Assume that for some \( n \geq 100 \),
\[ 3 \cdot 100 \leq 3 \cdot 100 \cdot 2^{\frac{n}{200}}. \]

BC: \( n = 100 \)
\[ \text{LHS} = 3 \cdot 100 \]
\[ \text{RHS} = 3 \cdot 100 \cdot 2^{\frac{100}{200}} \]

\[ \text{LHS} \leq \text{RHS} \]

IS: Want to prove that
\[ 3 \cdot (k+1)^{100} \leq 3 \cdot 100 \cdot 2^{\frac{k+1}{200}}. \]

\[ \text{LHS} = 3 \cdot (k+1)^{100} \]
\[ = 3 \cdot \left[ (\binom{100}{0} \cdot k^{100}) + (\binom{100}{1} \cdot k^{99}) + (\binom{100}{2} \cdot k^{98}) + \cdots + (\binom{100}{100} \cdot k^{0}) \right] \]
\[ \leq 3 \cdot \left[ k^{100} + 100 \cdot k + 100 \cdot k + \cdots + 100 \cdot k \right] \]
\[
(\binom{n}{k}) \leq n^k
\]

\[
= 3^k \left[ 1 + \left( \frac{100}{k} \right) + \left( \frac{100}{k} \right)^2 + \cdots + \left( \frac{100}{k} \right)^{100} \right]
\]

\[
\leq 3 \cdot 100 \cdot 2 \left[ \sum_{i=0}^{100} \left( \frac{100}{k} \right)^i \right]
\]

\[
\leq 2, \quad \text{when} \quad k > 200.
\]

\[
\leq 3 \cdot 100 \cdot 2 \cdot k + 1
\]

\[
\lim_{n \to \infty} \frac{f(n)}{g(n)} \begin{cases} 0, & f(n) = o(g(n)) \\ \infty, & f(n) = \omega(g(n)) \\ \Theta, & f(n) = \Theta(g(n)) \end{cases}
\]

\[
100n^3 = O(n^3)
\]

\[
\neq o(n^3).
\]

\[
= \Theta(n^3).
\]
\[ 100 \, n^3 \leq 100 \, n^3, \quad \forall \, n \geq 1. \]
\[ \therefore \, \Theta(n^3) \]
\[ 100 \, n^3 \geq n^3, \quad \forall \, n \geq 1. \]
\[ \therefore \, \Omega(n^3). \]

The running time of IS (Courage) is \( \Theta(n^3) \).

Vishwesh comes & says, I have a better analysis & can show that
\[ T(n) = O(n \, \text{yn}). \]
\[ \therefore \text{become} \]
\[ T(n) = \Omega(n^3). \]

Vishwesh says I have a analysis that shows \( T(n) = \Omega(n \, \text{yn}) \).
\[ \eta^3 - 1 \neq \Theta(n^3). \]

\[ \eta^3 - 1 = \Theta(n^3). \]

\[ \eta^3 - 1 \leq n^3, \quad \forall n \geq 1. \]

\[ O(n^3). \]

\[ \eta^3 - 1 > c n^3, \quad \forall n \geq n_0. \]

\[ \eta^3 \left(1 - \frac{1}{2}\right) \geq 1. \]

\[ c = \frac{1}{2} \]

\[ \eta^3 \geq 1. \]
\[
\sqrt{2} \quad n_0 = 2
\]

5. \( n + 18 \cdot n^2 + 99 \cdot n^3 = \Theta(n^3) \)

\[\geq n^3, \quad \forall n \geq 1.\]

Ex: Prove that \( T[n-2] \leq 2 \cdot (n^{10}) \).

Sthn: \( T[n-2] \geq C \cdot n^{10}, \quad \forall n \geq n_0. \)

i.e., \( T[n] \geq C \cdot n^{10}, \quad \forall n \geq n_0. \)

\[
\therefore \quad T \geq C n^{10}, \quad \forall n \geq n_0.
\]

Suppose \( n = 2n \geq 9 \)
\[
T \geq C \cdot \left( 3 \cdot \left( \frac{1}{C} \right)^{\frac{9}{2}} \right)^9
\]

\[
= C \cdot \left( 3^9 \cdot \frac{1}{C} \right)
\]

\[
= \sqrt[9]{3^9}
\]

---

**Ex.**

1. \( f(n) = O(g(n)) \) and \( g(n) = O(h(n)) \) \( \Rightarrow \) \( f(n) = O(h(n)) \)

2. 

3. 

4. \( f(n) = O(h(n)) \) and \( g(n) = O(h(n)) \) then

\( f(n) + g(n) = O(h(n)) \).
\[ f: a_0 n + a_1 n + \ldots + a_d n^d \quad \text{(polynomial of degree)} \quad a_d \neq 0. \]

\[ f = O(n^d). \]

\[ \log n \quad \text{is upperbounded by any polylog.} \]

- Similarly, any poly in \( n \) is upperbounded by an \( \exp \) in \( n \).

\[ n^{1000} = O(2^n) \]

We have a program that performs a million high-level matrix multiplications.