CS 330 - Class 12, Tue Mar 2

Algorithms pt.3

Last time:

• Big-O notation and time complexity

• $f(x) \in O(g(x))$ means $g(x)$ is a rough upper bound for $f(x)$

• For some constant $c$, for almost all $x \in \mathbb{R}$, $|f(x)/g(x)| \leq c$

• For some $c$, $x_0 \in \mathbb{R}$, for all $x \in \mathbb{R}$, if $x > x_0$, then $|f(x)/g(x)| \leq c$

or, $|f(x)| \leq c \times |g(x)|$

• $(\log x)^1, (\log x)^2, (\log x)^3, \ldots$

• $x^1, x^2, x^3, \ldots$

• $2^x, 3^x, 4^x, \ldots$

• $x^b \in O(a^x)$ and $(\log_a x)^b \in O(x^c)$

---

Random Examples (T or F?)

• $17 \in O(1)$? $\text{True}$

• $3x + 27 \in O(x)$? $\text{True}$

• $3x + 27 \in O(27^{23})$? $\text{False}$

• $5x^2 + 75x + 12345 \in O(x^2)$? $\text{True}$

• $5x^2 + 75x + 12345 \in O(x)$? $\text{False}$

• $5x^2 + 75x + 12345 \in O(x^3)$? $\text{True}$

• $x^2 + 3^x \in O(x^3)$ $\text{False}$

• $x + x \log x \in O(x)$? $\text{False}$

• $\log x \in O(x^{1/2})$? $\text{True}$

• $x^2 + \log x \in O(\log(x)^3)$? $\text{False}$

$x + 6 \log x \in O(x)$

$\log x^5 = 5 \log x$
Big-O is an Upper Bound

- \( 1 \in O(x) \); \( 1, x \in O(x^2) \); \( 1, x, x^2 \in O(x^3) \), ...

- \( f(x) \) and \( g(x) \in O(h(x)) \) does not imply \( f(x) \) and \( g(x) \) are roughly equal.

Different Asymptotic Running Times Makes a Difference

- If two algorithms to solve the same problem have different \( O(\ldots) \) running times, the one with the lower \( O(\ldots) \) will always be faster than the other, for large enough \( n \).

---

Rank These Functions in Increasing Order of Growth (Slowest to Fastest)

1. \( f_1(x) = 17x^3 \)
2. \( f_2(x) = 2^x(x^2 - 1) \)  
3. \( f_3(x) = (\log x)^3 \)  
4. \( f_4(x) = 1,000,000 \)  
5. \( f_5(x) = x! \)  
6. \( f_6(x) = 3^x \)  
7. \( f_7(x) = 17x^3 - \log x \)  
8. \( f_8(x) = x(\log x)^2 \)  
9. \( f_9(x) = x^4/12345 \)  
10. \( f_{10}(x) = \log(\log x) \)
Asymptotic Lower Bound

- For rough lower bounds, we have $\Omega$.
  - $f(x) \in \Omega(g(x)) \iff$ for some constant $c$, $|f(x)/g(x)|$ is almost always $\geq c$
  - $(\exists c \in \mathbb{R}) \ (\exists x_0 \in \mathbb{R}) \ (\forall x \in \mathbb{R}) \ (|x| \geq x_0 \rightarrow |f(x)/g(x)| \geq c)$

  $|f(x)| \geq c \cdot g(x)$

- Since $|f(x)/g(x)| \geq c \Rightarrow |g(x)/f(x)| \leq 1/c$, so $g(x) \in O(f(x))$
  - $f(x) \in \Omega(g(x)) \iff f(x) \in O(g(x))$

- $\Omega(\ldots)$ isn't a strict lower bound (just as $O(\ldots)$ isn't a strict upper bound)

\[ T(n) \in \Omega(x) \]
\[ T(n) \in O(x^2) \]

Asymptotic Equality

- For rough equality, we have $\Theta$. We can combine $\Omega(\ldots)$ and $O(\ldots)$
  - $f(x) \in \Theta(g(x))$ iff $f(x) \in O(g(x))$ and $f(x) \in \Omega(g(x))$
  - $f(x) \in \Theta(g(x))$ iff $f(x)/g(x)$ is almost always between them:

  - $(\exists c_1, c_2 \in \mathbb{R}) \ (\exists x_0 \in \mathbb{R}) \ (\forall x \in \mathbb{R}) \ (|x| \geq x_0 \rightarrow c_1 \leq |f(x)/g(x)| \leq c_2)$
    - $c_1$ is the constant for $\Omega(\ldots)$, $c_2$ is the constant for $O(\ldots)$.

- $\Theta(\ldots)$ is symmetric in some sense

\[ f(x) \in \Theta(g(x)) \]
\[ g(x) \in \Theta(f(x)) \]
Constants Can Matter!

- Say algorithms 1 and 2 have runtimes \( T_1(n), T_2(n) \) both \( \in \Theta(g(n)) \)?

  Asymptotically equal

- Now, the constants are important:
  - \( c_1 \leq T_1(n) / g(n) \leq c_2 \) and \( d_1 \leq T_2(n) / g(n) \leq d_2 \)
  - How do \( c_1 \) and \( c_2 \) compare to \( d_1 \) and \( d_2 \)?

\[
\begin{array}{cccc}
2 & 3 & 4 & 6
\end{array}
\]

\( \neq O(\cdot) \) are importantly different

\[
O(n \log n) \text{ vs } O(n^2) \quad (10^3)^2 = 10^6
\]

\[
n = 10^3 \quad 3000 \text{ sec} \quad 50 \text{ min}
\]

\[
7.7 \times 10^6 \quad 1,000,000 = \text{less 2 yr}
\]

- Simple example:

  \[
  \text{Loop 1: } \text{for } x = 1 \text{ to } n \{ \text{for } y = 1 \text{ to } n \{ ... \} \}
  \]

  \[
  \text{Loop 2: } \text{for } x = 1 \text{ to } n \{ \text{for } y = 1 \text{ to } x \{ ... \} \}
  \]

  Both loops take \( \Theta(n^2) \) time, but loop 2 is roughly twice as fast as loop 1.

- How many times is the inner loop body run?
  - Loop 1: \( n \times n = n^2 \) times
  - Loop 2: \[
  \frac{1 + 2 + 3 + \ldots + n + n}{2} = \frac{n(n+1)}{2}
  \]
Knowing Your Data Can Make A Big Difference

- If we want to sort a list we know is 98% sorted, which is faster, insertion sort or selection sort?
- How many times does the inner loop of insertion sort runs?
- How many times does the inner loop of selection sort run?

True Example:

- Company looking up transaction code in a sorted table of codes.
- They do a simple linear search through a sorted list of length n.
- 90% of the time, the code you want is in the last few entries.
- They were certain they needed to buy much faster hardware.
- What's your recommendation?

\[ O(\log n) \text{ use binary search or reorder most visited values near top} \]

Tractable Problems

- Algorithms with run times \(\in O(n), O(n \log n), O(n^2)\) and (probably) \(O(n^3)\) are practical to run.
- Algorithms with run times \(\in O(2^n)\) are not practical
- Generalize to:
  - **Tractable problem**: Has a polynomial time algorithm
    - Runtime \(\in O(p(n))\) for some polynomial \(p(n)\).
  - **Intractable problem**: No polynomial time solution.
    - Known to be hard
    - Not known to be tractable
- Compare to **unsolvable problem**: No solution at all.
**Classes P and NP**

- **Class P** is the collection of problems that have polynomial time solutions. (Find an $x$ such that $P(x)$ is true) - runs in $O(\text{poly})$ time

- **Class NP** is the collection or problems where a solution can be checked for correctness in polynomial time. (Given $x$, is $P(x)$ true?)
  - General solution would be for all $x$, check $P(x)$.
  - **NP** short for **Non-deterministic Polynomial time**
  - If we could somehow guess the correct $x$ quickly, we could solve the problem in polynomial time by verifying $x$'s correctness.
  - The guess is done **non-deterministically** (we don't have a deterministic algorithm to generate it).

**Reducibility and NP Complete Problems**

- An **NP-Complete problem** is
  - **NP-hard** (has an NP solution)
  - Every other problem NP can be "reduced" to it.

- **Example: Boolean Decision Problem**
  - Can be solved in NP time: Guess a solution, verify it quickly.
  - Every problem in NP can be rephrased as a boolean decision problem.
  - $\text{Other\_Problem\_Soln}(x) = (\text{Boolean\_Decision\_Soln} \circ \text{Transform})(x)$
  - Best deterministic algorithm we know to solve B.D.P. is exponential.

solve any NP-complete problem tractably? solve all NP tractably
Is $P = NP$?

- We know $P \subseteq NP$; big question is if $P = NP$.
- We don't know, we think they probably aren't.
- Millennium Prize: $1,000,000

- If yes, then many problems currently considered intractable become tractable. \textit{yay}!
  - E.g., various decryption problems, un-hashing problems. \textit{uhoh}!
- If no, then we can give up looking for a polynomial-time solution for the NP-complete problems.
  - Focus on partial solutions / approximative algorithms.