Data Structures and Algorithms

Prof. Ganesh Ramakrishnan,
Prof. Ajit Diwan,
Prof. D.B. Phatak

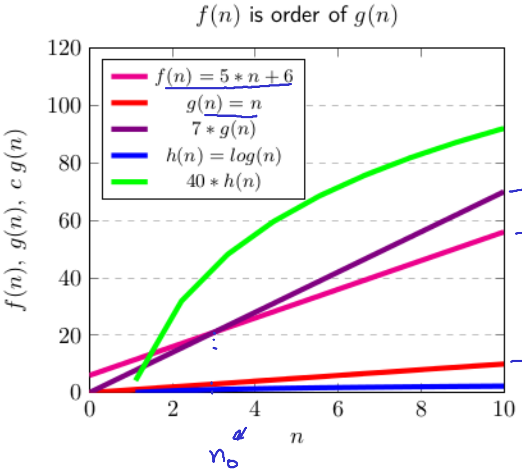
Department of Computer Science and Engineering
IIT Bombay

Session: Order of Running Time of an Algorithm
Big-oh($\mathcal{O}$), Small-oh(o), Omega(Ω), Theta(Θ)

Formalizing Definitions

- $T(N) = \mathcal{O}(f(N))$ if there are positive constants c and n_0 such that $T(N) \leq cf(N)$ when $N \geq n_0$
- This is also pronounced as $T(N)$ is Big-Oh $f(N)$
- This means $T(N)$ is of the order of $f(N)$ if you can find a point n_0 after which $T(N)$ is (asymptotically) smaller than a linearly scaled version of $f(N)$.
- Rougly speaking
  - The point n_0 helps ignore the additive constants
  - The factor c helps ignore the multiplicative constants
  - Focus is only on the dominating N term

Understanding Big-Oh



$f(n) = O(g(n))\ \forall\ n \geq n_0 \quad f(n) \leq c\,g(n)$

$5n+6 = O(n)$

$g(n) = \Omega(f(n))$ with $c' = \frac{1}{7}$

$f(n) = \Theta(g(n))$ if

$f(n) = O(g(n))$ &

$f(n) = \Omega(g(n))$

→ $c = 7 \; (7g(n))$

→ $f(n)$

$f(n) = \Omega(g(n))\ \forall\ n \geq n_0$

$f(n) \geq c'g(n)$

→ $c' = 1 \; (1g(n))$ $g(n) = O(f(n))$

with $C = 1$

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

Other Definitions

- $T(N) = \Omega(f(N))$ if there are positive constants c and n_0 such that $T(N) \geq cf(N)$ whenever $N \geq n_0$
  - Growth rate of $T(N)$ is asymptotically more than of $g(N)$
- $T(N) = \Theta(f(N))$ if $T(N) = O(f(N))$ and $T(N) = \Omega(f(N))$
  - Growth rate of $T(N)$ and $f(N)$ are the same
- $T(N) = o(f(N))$ if for all positive constants c there is an n_0 such that $T(N) < c\, f(N)$ when $N > n$
  - Growth rate of $T(N)$ is strictly less than of $f(N)$

if $T(N) = o(f(N)) \Rightarrow T(N) = O(f(N))$

More Definitions

- $T(N) = \Omega(f(N))$
  - $f(N)$ is $\mathcal{O}(T(N))$
- $T(N) = \Theta(f(N))$
  - Tighter (slightly advanced) analysis
- $T(N) = o(f(N))$: What is the real difference from big-Oh?
  - If $T(N)$ is $\mathcal{O}(f(N))$, it may still be $\Theta(f(N))$ (if $T(N) = \Omega(f(N))$)
  - But, if $T(N)$ is $o(f(N))$, it will **Not** be $\Theta(f(N))$

Examples (of functions)

- $2N + 3$ is $\mathcal{O}(N)$
  - $T(N) = 2N + 3$, $f(N) = N$
  - For $c = 6$, $n_0 = 1$, $T(N) < c\, f(N)$ for $n \geq n_0$
- Note that $2N + 3$ is also $\mathcal{O}(N^2)$, $\mathcal{O}(N^3)$ etc., but by convention we always state the (tightest) lowest order
- $f(N)$ is also $\mathcal{O}(T(N))$ $(c = 1, n_0 = 1)$
- So, $T(N)$ is $\Theta(f(N))$

Θ = upper bound

Generally by $\Theta(f(N)) = T(N)$ we mean "tightest upper bound"

If $T(N) = \Theta(f(N))$ & $f(N) = \Theta(g(N))$
$\therefore$ $T(N) = \Theta(g(N))$

Examples

$N^2 = O(N^2 + N)$

- $4N^2 + N + 5$ is conventionally described as $\mathcal{O}(N^2)$, although it is also $\mathcal{O}(N^2 + \cancel{N})$
  - Lower order terms usually not mentioned
- $5N + 3\log N \sim \mathcal{O}(N)$
- We don't formally prove finding of c and n_0, - just write the order intuitively, based on dominating term

Exercise: Analyse Interpolation Search

Remind you of Linear scan

- Interpolate call for 'mid' element with call for 'next' element in binary search
- For an interpolation search to be practical, two assumptions must be satisfied:
  - Each access **must be very expensive** compared to a typical instruction
    - E.g. The array might be on a disk instead of in memory, and each comparison requires a disk access.
  - The data must not only be sorted, it must also be **fairly uniformly distributed**.
    - E.g. A phone book is fairly uniformly distributed. If the input items are {1, 2, 4, 8, 16, }, the distribution is not uniform

Introduction to Master Theorem

- Several algorithms (such as divide and conquer) are recursive in nature and can be solved using recurrence relations : $T(n) = T(n/a) + T((1-\frac{1}{a})n) + \ldots$
- It is enough to give asymptotic characterization for associating the cost of an algorithm $(\forall n \geq n_0)$
- Master Theorem is a tool for solving recurrence relations in the asymptotic case
- Provides a method for solving recurrences specifically of the form
  - $T(n) = aT(\frac{n}{b}) + f(n)$
  - where $a \geq 1, b > 1$ are constants and $f(n)$ is a function
- Ref: Section 4.5 of the Third Edition of CLRS

Master Theorem

- Applies to recurrence relations of the form $T(n) = aT(\frac{n}{b}) + f(n)$, with $\frac{n}{b}$ replaced by either $\lfloor \frac{n}{b} \rfloor$ or $\lceil \frac{n}{b} \rceil$
- The Master theorem defines the following asymptotic bounds for $T(n)$

Case 1: If $f(n) = \mathcal{O}(n^{\log_b a - \epsilon})$ for some constant $\epsilon > 0$, then $T(n) = \Theta(n^{\log_b a})$

Case 2: If $f(n) = \mathcal{O}(n^{\log_b a})$, then $T(n) = \Theta(n^{\log_b a} \log n)$

Case 3: If $f(n) = \Omega(n^{\log_b a + \epsilon})$ for some constant $\epsilon > 0$, and if $af(\frac{n}{b}) \le cf(n)$ for some constant, $c < 1$, and all sufficiently large n, then $T(n) = \Theta(f(n))$

Master Theorem

- Applies to recurrence relations of the form $T(n) = aT(\frac{n}{b}) \ + \ f(n)$, with $\frac{n}{b}$ replaced by either $\lfloor \frac{n}{b} \rfloor$ or $\lceil \frac{n}{b} \rceil$
- The Master theorem defines the following asymptotic bounds for $T(n)$

  Case 1: If $f(n) = \mathcal{O}(n^{\log_b a - \in})$ for some constant $\in > 0$, then $T(n) = \Theta(n^{\log_b a})$

  Case 2: If $f(n) = \mathcal{O}(n^{\log_b a})$, then $T(n) = \Theta(n^{\log_b a} \ \log \ n)$

  Case 3: If $f(n) = \Omega(n^{\log_b a + \in})$ for some constant $\in > 0$, and if $af(\frac{n}{b}) \leq cf(n)$ for some constant, $c < 1$, and all sufficiently large n, then $T(n) = \Theta(f(n))$
- Intuition: Compare function $f(n)$ with $n^{\log_b a}$. Larger of the two determines the solution

Master Theorem (Cases Elaborated)

Case 1: If the function $n^{\log_b a}$ is larger than $f(n)$, the solution is $T(n) = \Theta(n^{\log_b a})$

Case 2: If the functions $f(n)$ and $n^{\log_b a}$ are of the same size, we multiply by a logarithmic factor, and the solution is $T(n) = \Theta(n^{\log_b a} \log n)$

Case 3: If the function $f(n)$ is larger, then the solution is $T(n) = \Theta(f(n))$

Example 1

$T(n) = 9T(\frac{n}{3}) + n$

a b f(n)

$\underbrace{a = 9, b = 3}_{\log_b a = 2}, f(n) = n$

We have $n^{\log_b a} = n^{\log_3 9 - \epsilon} = \Theta(n^2)$

Since $f(n) = \mathcal{O}(n^{\log_3 9})$, where $\epsilon = 1$,

We can apply Case 1: $T(n) = \Theta(n^2)$

Example 2

$T(n) = T(\frac{5n}{3}) + 1$

$a = 1, b = \frac{5}{3}, f(n) = 1$

$\log_b a = \log_{5/3}(1)$

We have $n^{\log_b a} = n^{\log_{\frac{5}{3}} 1} = n^0 = 1$

Since $f(n) = \Theta(n^{\log_{\frac{5}{3}} 1}) = \Theta(1)$

We can apply Case 2: $T(n) = \Theta(\log n)$

Example 3

$T(n) = 3T(\frac{n}{4}) + n \ \log \ n$

$a = 3, b = 4, f(n) = n \ \log \ n$

We have $n^{\log_b a} = n^{\log_4 3} = \mathcal{O}(n^{0.793})$

Since $f(n) = \Omega(n^{\log_4 3 + \in})$, where $\in > 0$,

$af(\frac{n}{b}) = 3(\frac{n}{4}) \log(\frac{n}{4}) \leq (\frac{3}{4}) n \ \log \ n = cf(n)$ for $c = (\frac{3}{4})$

We can apply Case 3: $T(n) = \Theta(n \ \log \ n)$

Examples 4 (Cannot use Master Theorem)

- $T(n) = cos(n)$
  - $T(n)$ is not Monotone
- $T(n) = 3T(\frac{n}{3}) + 3^n$
  - $f(n)$ is not Polynomial
- $T(n) = \sqrt{n^2 + 3}$
  - b is not constant
- $T(n) = 2^n T(\frac{n}{2}) + n^n$
  - a is not constant
- $64T(\frac{n}{8}) - n^2 \log\ n$
  - $f(n)$ is not positive

$f(n) \longleftrightarrow n^{\log_b a}$

Thank you