Algorithmic Complexity - II

Atul Gupta

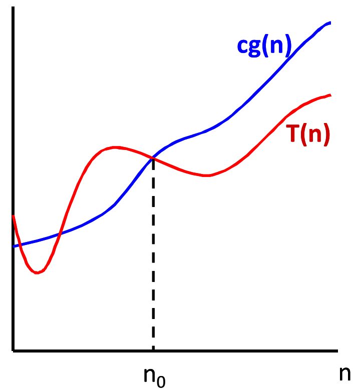
Algorithmic Complexity

- A measure of the performance of an algorithm with respect to input size
  - Space complexity
  - Time complexity (Mostly used)
- Expressed in terms of Asymptotic Notations
  - Bog-Oh (O), Big-Theta (Θ), Big-Omega (Ω)
- What to measure
  - Worst case performance
  - Average case
  - Best case

Big-Oh (O)

- An indicator of the worst case performance (upper bound)
- Defined as a function T(n) which is said to be of O(g(n)) if there exist positive constants c_0 and n_0 such that for all $n >= n_0$, we have

  $T(n) < = c_0\, g(n)$
- T(n) is asymptotically smaller than or equal to g(n)



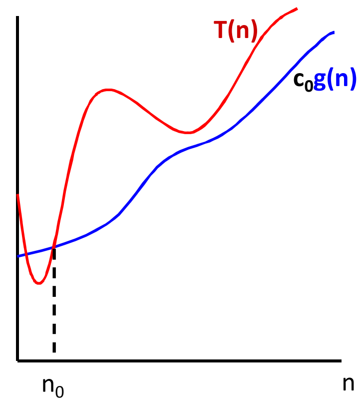
$T(n) <= c_0\, g(n)$

Big-Oh (O): Examples

- $T(n) = 3n^2+4n = O(n^2)$
- $T(n) = 3n^2+4n = O(n^3)$
- $T(n) = 3n^2+4n \neq O(n)$
- $T(n) = (n+1)^2 = O(n^2)$

Big-Omega (Ω)

- An indicator of the best case performance (lower bound)
- Defined as a function $T(n)$ which is said to be of $\Omega(g(n))$ if there exist positive constants c_0 and n_0 such that for all $n >= n_0$, we have
  $T(n) >= c_0\, g(n)$
- $T(n)$ is asymptotically greater than or equal to $g(n)$



$T(n) >= c_0\, g(n)$

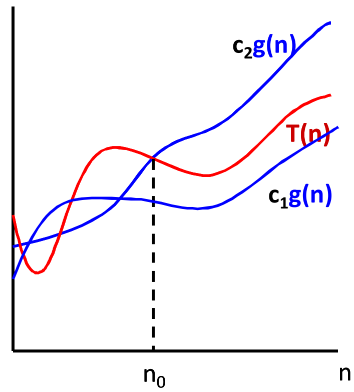
Big-Omega (Ω): Examples

- $T(n) = 3n^2+4n = \Omega(n^2)$
- $T(n) = 3n^2+4n = \Omega(n)$
- $Tn = 4n+2 = \Omega(n)$
- $Tn = 4n+2 = \Omega(1)$

Big-Theta (Θ)

- An indicator of the worse case and best case performance (asymptotically tight bound)
- Defined as a function T(n) which is said to be of Θ (g(n)) if there exist positive constants c_1, c_2 and n_0 such that for all n >= n_0, we have

  $c_1\ g(n) \le T(n) \le c_2\ g(n)$
- T(n) is asymptotically equal to g(n)



$T(n) \le c_0\ g(n)$

Big-Theta (Θ): Examples

- $T(n) = 3n^2+4n = \Theta(n^2)$
- $T(n) = 3n^2+4n \neq \Theta(n)$
- $Tn = 4n+2 = \Theta(n)$
- $Tn = 4n+2 \neq \Theta(1)$

"Divide and Conquer"

- Divide the problem into a number of subproblems.
- Conquer the subproblems by solving them recursively.
  - If the subproblem sizes are small enough, however, just solve the subproblems in a straightforward manner.
- Combine the solutions to the subproblems into the solution for the original problem.

Recursive Algorithms

- Recurrence (or Recurrence Equation)

$$T(n) = \begin{cases} \Theta(1) & \text{if } n <= c \\ aT(n/b) + D(n) + C(n) & \text{otherwise} \end{cases}$$

- Parts
  - Solution for smallest subdivision (Terminating condition)
  - Subdivision relation (aT(n/b))
  - Division efforts (D(n))
  - Combining effort (C(n))

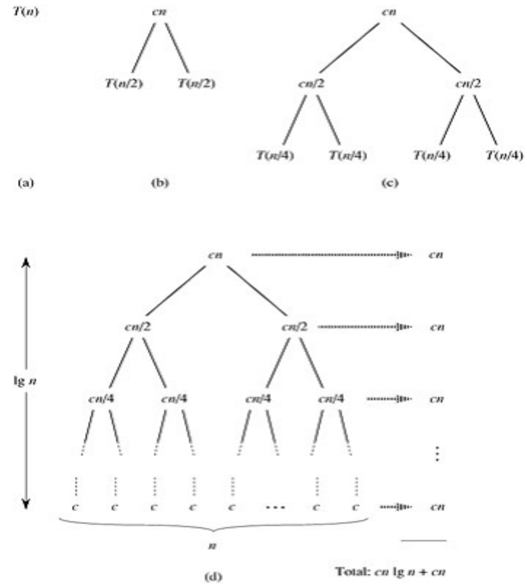
Merge Sort

- Recurrence Relation

Solving Recurrence

- Substitution Method
- Recursion-tree Method
- Master Method

Recursion Tree method



Master Theorem

T(n) = aT(n/b) + f(n)

1. If $f(n) = O(n^{\log_b a-\epsilon})$ for some constant $\epsilon > 0$, then $T(n) = \Theta(n^{\log_b a})$
2. If $f(n) = \Theta(n^{\log_b a})$, then $T(n) = \Theta(n^{\log_b a} \lg n)$.
3. If $f(n) = \Omega(n^{\log_b a+\epsilon})$ for some constant $\epsilon > 0$, and if $a f(n/b) \le cf(n)$ for some constant $c < 1$ and all sufficiently large n, then $T(n) = \Theta(f(n))$.

Master Theorem

Examples:

1. $T(n) = 9T(n/3) + n$
2. $T(n) = T(2n/3) + 1$
3. $T(n) = 3T(n/4) + n \log n$

Summary

- Recursion is a tool for solve problems using "Divide and Conquer" approach
- Time complexity of a recursive algorithm can be represented by a recurrence relation
- The recurrence relation can be solved by
  - Recursion Tree Method
  - Substitution
  - Master Theorem