Analysis of Algorithms

http://www.geeksforgeeks.org/analysis-of-algorithms-set-3asymptotic-notations/

Book: http://ldc.usb.ve/~xiomara/ci2525/ALG_3rd.pdf

Background

Asymptotic Analysis

Worst, Average and Best Cases of Algorithms

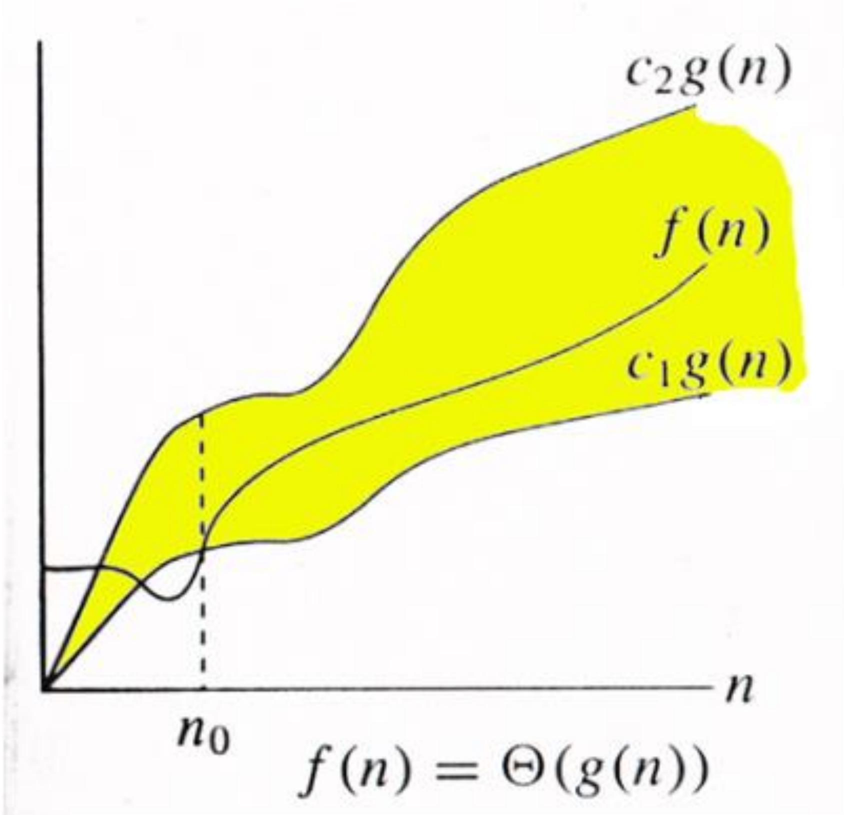
Asymptotic Analysis

The main idea is to have a measure of efficiency of algorithms that

- doesn’t depend on machine specific constants and
- doesn’t require algorithms to be implemented and time taken by programs to be compared
- Asymptotic notations are mathematical tools to represent time complexity of algorithms for asymptotic analysis
-

The following 3 asymptotic notations are mostly used to represent time complexity of algorithms.

1) Θ Notation



- The theta notation bounds a functions from above and below,
- so it defines exact asymptotic behavior
- A simple way to get Theta notation of an expression is to drop low order terms and ignore leading constants
- For example, consider the following expression

$$3n^3 + 6n^2 + 6000 = \Theta(n^3)$$

Dropping lower order terms is always fine because there will always be a n0 after which $\Theta(n^3)$ beats $\Theta(n^2)$ irrespective of the constants involved

For a given function $g(n)$, we denote $\Theta(g(n))$ is following set of functions

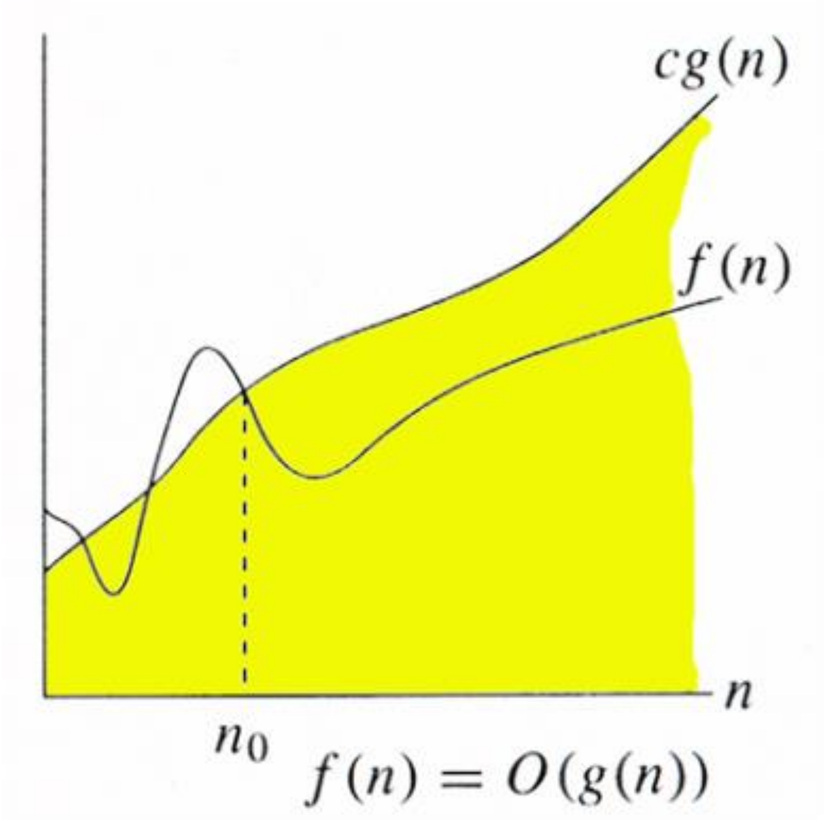
$\Theta((g(n)) = \{ f(n)$: there exist positive constants $c1$, $c2$ and $n0$ such that

$$0 <= c1*g(n) <= f(n) <= c2*g(n) \text{ for all } n >= n0 \}$$

The above definition means, if $f(n)$ is theta of $g(n)$, then the value $f(n)$ is always between $c1*g(n)$ and $c2*g(n)$ for large values of n ($n >= n0$)

The definition of theta also requires that $f(n)$ must be non-negative for values of n greater than $n0$

2) Big O Notation



- Big O notation defines an upper bound of an algorithm, it bounds a function only from above

- For example, in the case of Insertion Sort, it takes linear time in best case and quadratic time in worst case

- $\therefore$ time complexity of Insertion Sort is

  O(n^2)

- Note that O(n^2) also covers linear time

- If **Θ** notation is used to represent time complexity of Insertion Sort, two statements should be used for best and worst cases:

  1. The worst case time complexity of Insertion Sort is **Θ**(n^2)

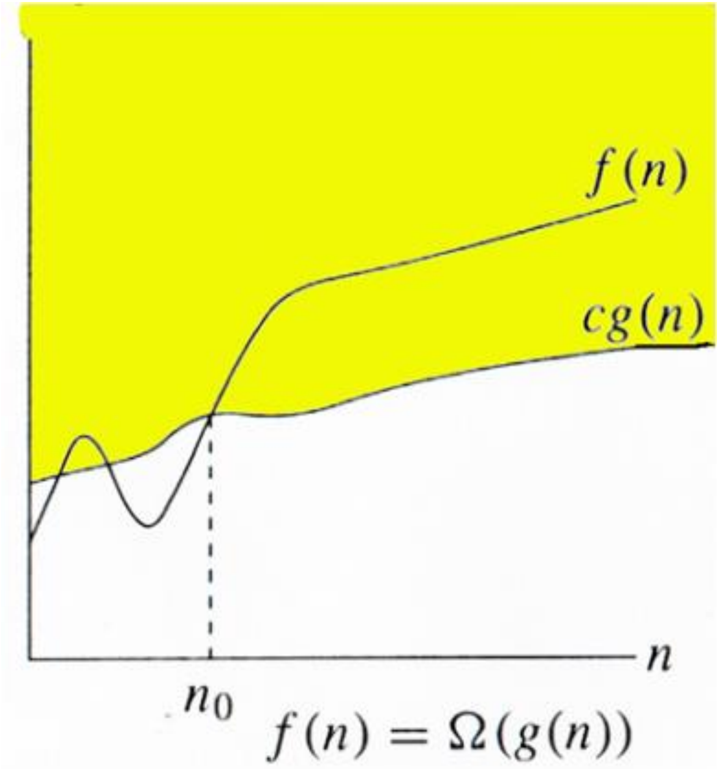
  2. The best case time complexity of Insertion Sort is **Θ**(n)

The Big O notation is useful when we only have upper bound on time complexity of an algorithm. Many times we easily find an upper bound by simply looking at the algorithm

$O(g(n))$ = { $f(n)$: there exist positive constants c and $n0$ such that

$$0 <= f(n) <= cg(n) \text{ for all } n >= n0\}$$

3) Ω Notation



- Just as Big O notation provides an asymptotic upper bound on a function, **Ω** notation provides an asymptotic lower bound

- Notation can be useful when we have lower bound on time complexity of an algorithm

For a given function g(*n*), we denote by **Ω**(g(*n*)) the set of functions

Ω(g(*n*)) = { f(*n*): there exist positive constants *c* and *n*0 such that

0 <= *c*g(*n*) <= f(*n*) for all *n* >= *n*0 }.

Let us consider the same Insertion Sort example here

The time complexity of Insertion Sort can be written as **Ω**(*n*), but it is not a very useful information about insertion sort, as we are generally interested in worst case and sometimes in average case

A good example on Alalysis of Buble Sort Algorithm

Exercise

Which of the following statements is/are valid?

1. Time Complexity of QuickSort is $\Theta(n\text{^}2)$

2. Time Complexity of QuickSort is $O(n\text{^}2)$

3. For any two functions $f(n)$ and $g(n)$, we have $f(n) = \Theta(g(n))$ if and only if $f(n) = O(g(n))$ and $f(n) = \Omega(g(n))$

4. Time complexity of all computer algorithms can be written as $\Omega(1)$

References

Lec 1 | MIT (Introduction to Algorithms)

Introduction to Algorithms 3rd Edition by Clifford Stein, Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest

This article is contributed by **Abhay Rathi**. Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.

Related Topics

- A Problem in Many Binary Search Implementations
- Analysis of Algorithms | Set 4 (Analysis of Loops)
- NP-Completeness | Set 1 (Introduction)
- Static and Dynamic Libraries | Set 1
- The Ubiquitous Binary Search | Set 1
- Reservoir Sampling
- Analysis of Algorithms | Set 2 (Worst, Average and Best Cases)
- Analysis of Algorithms | Set 1 (Asymptotic Analysis)

Writing code in comment? Please use **ideone.com** and share the link here.

- Advanced Data Structures
- Dynamic Programming
- Greedy Algorithms
- Backtracking
- Mathematical Algorithms
- Recursion
- Geometric Algorithms

Popular Posts

- All permutations of a given string
- Memory Layout of C Programs
- Understanding “extern” keyword in C
- Median of two sorted arrays
- Tree traversal without recursion and without stack!
- Structure Member Alignment, Padding and Data Packing
- Intersection point of two Linked Lists
- Lowest Common Ancestor in a BST.
- Check if a binary tree is BST or not
- Sorted Linked List to Balanced BST