Ch3. Asymptotic Notation

SNU
IDB Lab.

Preview of Chapters

- Chapter 2
  - How to analyze the space and time complexities of program
- Chapter 3
  - Review asymptotic notations such as O, Ω, Θ, o for simplifying the analysis
- Chapter 4
  - Show how to measure the actual run time of a program by using a clocking method

Bird's eye view

- In this chapter
  - We review asymptotic notations: O, Ω, Θ, o
  - The notations are for making statement about program performance when the input data is large
  - Big-Oh "O" is the most popular asymptotic notation
  - Asymptotic notations will be introduced in both informal and rigorous manner

Table of Contents

- Introduction
- Asymptotic Notation & Mathematics
- Complexity Analysis Example
- Practical Complexities

Introduction (1/3)

- Reasons to determine operation count and step count
  - To predict the growth in run time
  - To compare the time complexities of two programs

- Facts of the previous two approaches
  - The operation count method ignores all others except key operations
  - The step count method overcome the above shortage, but the notion of step is inexact
    - x=y and x=y+z+(y/z) treated as a same step?
    - Two analysts may arrive at 4n + 3 and 900n + 4 for the same program

- Asymptotic analysis focuses on determining the biggest terms (but not their coefficient) in the complexity function.

Introduction (2/3)

- If the step count is $c_1 n^2 + c_2 n + c_3$, coefficients and n term cannot give any particular meanings when the instance size is large

$$c_1 n^2 + c_2 n + c_3$$

$$\lim_{n \to \infty} \left(\frac{c_2}{c_1 n} + \frac{c_3}{c_1 n^2} \right) = 0$$

c1 is dominant factor when n is large

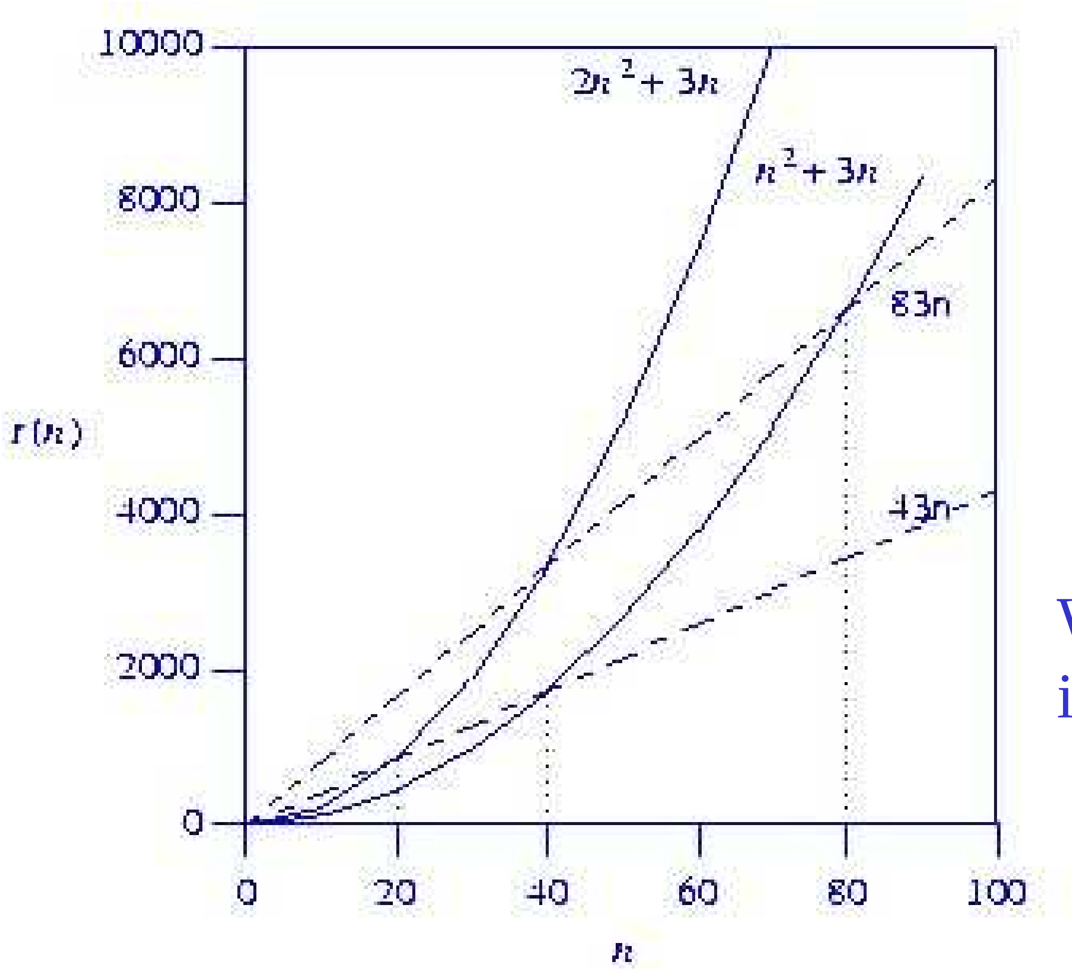
$$\therefore c_1 n^2 \text{ is important when n is very large!}$$

- Let n1 and n2 be two large values of the instance size

$$\frac{t(n_1)}{t(n_2)} = \frac{c_1 {n_1}^2}{c_1 {n_2}^2} = \left(\frac{n_1}{n_2}\right)^2$$

- We can conclude that if the instance size is doubled, the runtime increases by a factor of 4

Introduction (3/3)



- Program A

$2n^2 + 3n$ or $n^2 + 3n$

- Program B

$83n$ or $43n$

When n is large, program B is faster than program A

Table of Contents

- Introduction
- Asymptotic Notation & Mathematics
- Complexity Analysis Example
- Practical Complexities

Asymptotic Notation: concepts

- Definition

$$\lim_{n \to \infty} \frac{q(n)}{p(n)} = 0$$

  - p(n) is ***asymptotically bigger*** than q(n)
  - q(n) is ***asymptotically smaller*** than p(n)
  - p(n) and q(n) is ***asymptotically equal*** *iff* neither is *asymptotically bigger* than the other

Asymptotic Notation: terms

- Commonly occurring terms

Term	Name
1	constant
$\log n$	logarithmic
n	linear
$n \log n$	$n \log n$
n^2	quadratic
n^3	cubic
2^n	exponential
$n!$	factorial

$$1 < \log n < n < n \log n < n^2 < n^3 < 2^n < n!$$

Asymptotic Notation: Big Oh

$$f(n) = O(g(n)) \qquad \lim_{n \to \infty} \frac{f(n)}{g(n)} = 0 \text{ or Constant } C$$

- f(n) is big oh of g(n)
- The above notation means that f(n) is asymptotically smaller than or equal to g(n)
- g(n), multiplied by some constant C gives an asymptotic upper bound for f(n)

- The notation gives no clue to the value of this constant C, it only states that it exists

Big Oh arithmetic

- Definition

$f(n) = O(g(n))$ iff positive constants c and n_0

exist such that $f(n) \le cg(n)$ for all n, $n \ge n_0$

- Consider f(n) = 3n+2
  - When $c = 4, n_0 = 2$ then f(n) <= 4n
  - f(n) = O(n), therefore g(n) = n

- f(n) is bounded above by some function g(n) at all points to the right of n_0

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

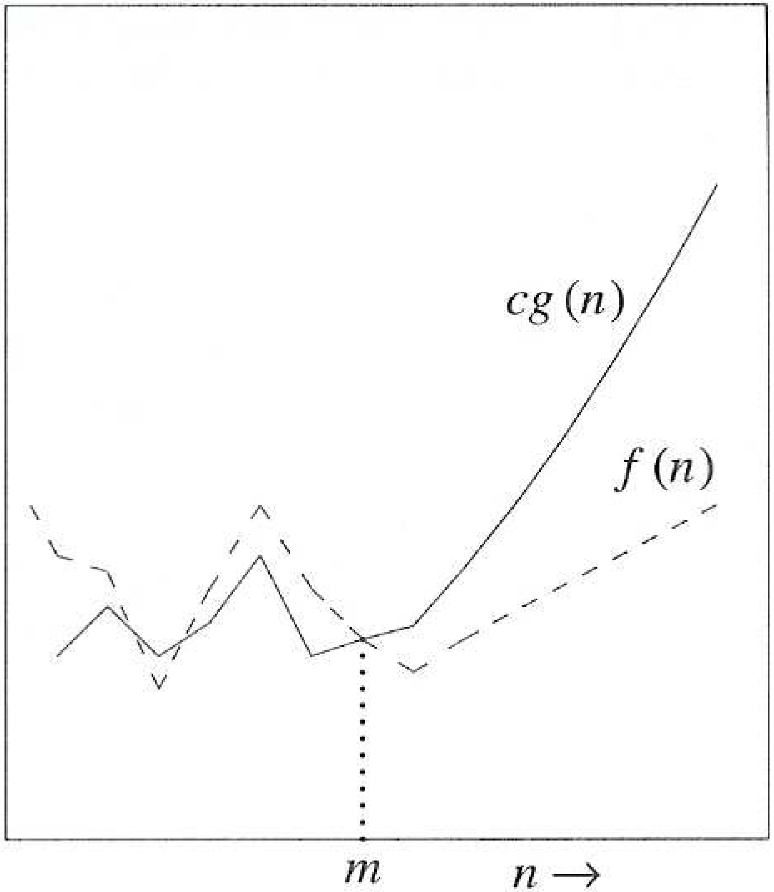


Figure 3.4 $g(n)$ is an upper bound (up to a constant factor c) on $f(n)$

n_0 is any integer greater than m

Big Oh example

$$f(n) = 3n^2 + 4n = O(n^3)$$

$$c = 7, n_0 = 0$$

$$3n^2 + 4n \leq 3n^3 + 4n^3 = 7n^3$$

Big O gives us an upper bound, but does not promise a careful (tight) upper bound!!!

$$f(n) = 3n^2 + 4n = O(n^2)$$

$$c = 4, n_0 = 4$$

$$3n^2 + 4n \leq 4n^2$$

$$\Theta\ n \geq 4 \rightarrow n^2 \geq 4n \rightarrow 4n^2 \geq 3n^2 + 4n$$

Asymptotic Notation: Big Theta

- Theta(Θ) Notations

$$f(n) = \Theta(g(n)) \qquad c_1 \le \lim_{n\to\infty} \frac{f(n)}{g(n)} \le c_2$$

  - f(n) is theta of g(n)
  - f(n) is asymptotically equal to g(n)
  - g(n) is an asymptotic tight bound for f(n)

Big Theta

- Definition $f(n) = \Theta(g(n))$ iff

  positive constants c_1 and c_2 and an n_0 exist

  such that $c_1 g(n) \le f(n) \le c_2 g(n)$ for all $n, n \ge n_0$

  $Example: 3n^2 + 4n = \Theta(n^2)$

  Proof : choose $c_1 = 3, c_2 = 7$ and $n_0 = 0$

  we have $3n^2 \le 3n^2 + 4n \le 7n^2$ for all $n, n \ge n_0$

- f(n) is bounded above and below by some function g(n) at all points to the right of n_0

$$f(n) = \Theta(g(n)) \quad \text{iff} \;\; c_1 g(n) \le f(n) \le c_2 g(n)$$

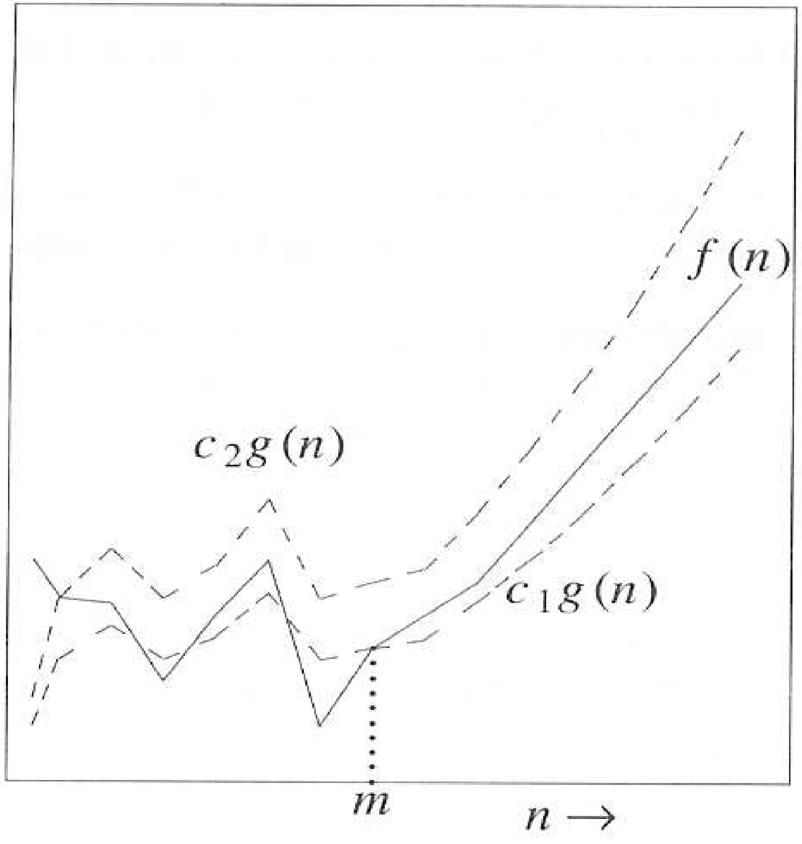


Figure 3.6 $g(n)$ is a lower and upper bound (up to a constant factor) on $f(n)$

Big Theta arithmetic

- Example

$$f(n) = 3n^2 + 2n + 1, \quad f(n) = \Theta(n^2)$$

$$when \quad c_1 = 3, c_2 = 4, n_0 = 1 + \sqrt{2}$$

$$3n^2 \le 3n^2 + 2n + 1 \le 4n^2 \; (n > 0)$$

$$\Rightarrow \; n \ge 1 + \sqrt{2} \text{ such that } n_0 = 1 + \sqrt{2}$$

Asymptotic Notation: Big Omega

- Omega(Ω) Notations

$$f(n) = \Omega(g(n)) \qquad \lim_{n\to\infty} \frac{f(n)}{g(n)} = \infty \text{ or Const.}$$

  - f(n) is omega of g(n)
  - f(n) is asymptotically bigger than or equal to g(n)
  - g(n) is an asymptotic lower bound for f(n)
  - It is the reverse of big-O notation

Big Omega arithmetic

- Definition

$$f(n) = \Omega(g(n)) \text{ iff } \quad \text{positive constants c and } \mathrm{n}_0$$

$$\text{exist such that } f(n) \geq cg(n) \text{ for all } n,\ n \geq n_0$$

- f(n)=3n+2 > 3n for all n, So $f(n) = \Omega(n)$

- f(n) is bounded below by a function g(n) at all points to the right of n_0

$$f(n) = \Omega(g(n)) \text{ iff } \quad f(n) \geq cg(n)$$

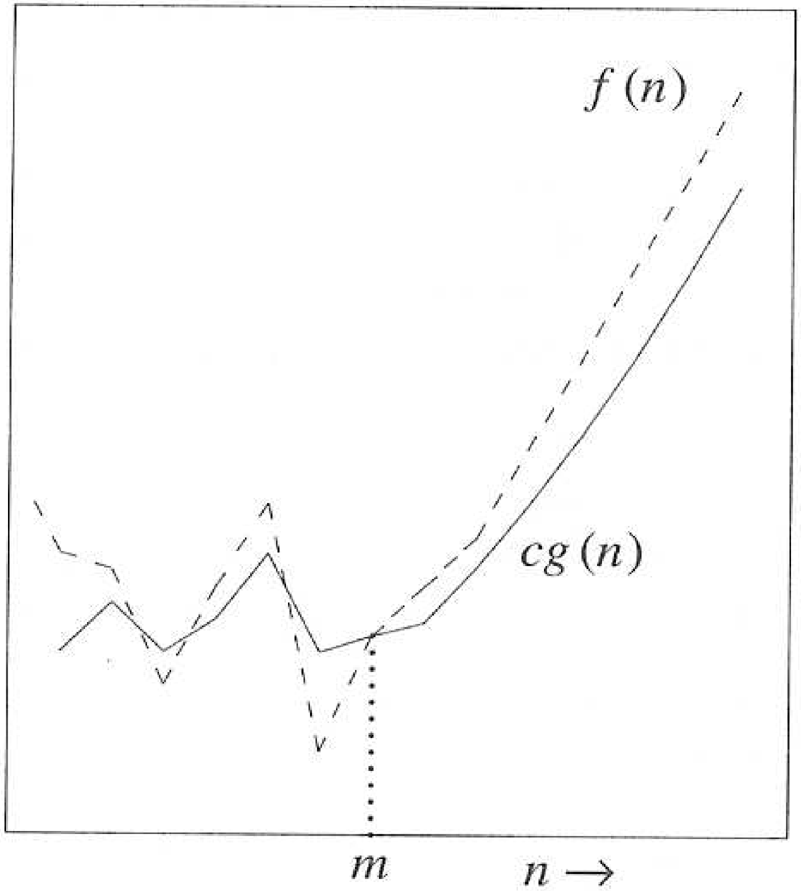


Figure 3.5 $g(n)$ is a lower bound (up to a constant factor c) on $f(n)$

Little Oh Notation (o)

- Definition

$$f(n) = o(g(n)) \quad \text{iff} \quad f(n) = O(g(n)) \;\; and \;\; f(n) \neq \Omega(g(n))$$

- Upper bound that is not asymptotically tight
- Example

$$3n + 2 = o(n^2) \quad \text{as} \;\; 3n + 2 = O(n^2) \quad and \;\; 3n + 2 \neq \Omega(n^2)$$

Big oh and Little oh

- Big O notation may or may not be asymptotically tight

$$2n^2 = O(n^2) : \text{tight} \quad \text{vs.} \quad 2n = O(n^2) : \text{not tight}$$

- We use little o notation to denote an upper bound that is not asymptotically tight

$$Ex: \quad 2n = o(n^2) \qquad 2n^2 \neq o(n^2)$$

Legend in Asymptotic Notation

- Roughly $f(n) = \Theta\ g(n)$ means $f(n) = g(n)$
- Roughly $f(n) = O\ g(n)$ means $f(n) <= g(n)$

- In general we use O even though we get Θ !
- In fact, O is a kind of Θ
- Another reason: In general, finding Θ is difficult!
- Our textbook will use O and Θ interchangeably!

Table of Contents

- Introduction
- Asymptotic Notation & Mathematics
- Complexity Analysis Example
- Practical Complexities

Sequential Search

- Expression of step count as an asymptotic notation (program 2.1)

Statement	s/e	Frequency	Total Steps
`public static int`			
`sequentialSearch(···)`	0	0	$\Theta(0)$
`{`	0	0	$\Theta(0)$
`int i;`	1	1	$\Theta(1)$
`for (i = 0; i < a.length &&`			
`!x.equals(a[i]); i++);`	1	$\Omega(1)$, $O(n)$	$\Omega(1)$, $O(n)$
`if (i == a.length) return -1;`	1	1	$\Theta(1)$
`else return i;`	1	$\Omega(0)$, $O(1)$	$\Omega(0)$, $O(1)$
`}`	0	0	$\Theta(0)$

- Ignore terms without n!

Best case $t_{sequentialSearch}(n) = \Omega(1)$ ==> lower bound is 1

Worst case $t_{sequentialSearch}(n) = O(n)$ ==> upper bound is n

- In fact, the worst case is Big-Theta(n): Remember Big O is a kind of Big-Theta

Example 3.24

Binary Search

```
public static int binarySearch(Comparable [] a, Comparable x)
 {// Search a[0]  <=  a[1]  <=  ...  <=  a[a.length-1] for x.
   int left = 0;
   int right = a.length - 1;
   while (left <= right)
      { int middle = (left + right)/2;
        if (x.equals(a[middle])              return middle;
        if (x.compareTo(a[middle]) > 0)  left  = middle + 1;
        else                                 right = middle - 1;
    }
   return -1; // x not found
 }
```

- Each iteration of the while loop ➔ Decrease in search space by a factor about 2
- Best case complexity ➔ $\Omega(1)$
- Worst case complexity ➔ $\Theta(\log a.length)$ // because we have to go down to leaf nodes!

Table of Contents

- Introduction
- Asymptotic Notation & Mathematics
- Complexity Analysis Example
- Practical Complexities

Practical Complexities

- 1,000,000,00 instructions per second computer
- To execute a program of complexity f(n)

	$f(n)$						
n	n	$n \log_2 n$	n^2	n^3	n^4	n^{10}	2^n
10	.01μs	.03μs	.1μs	1μs	10μs	10s	1μs
20	.02μs	.09μs	.4μs	8μs	160μs	2.84h	1ms
30	.03μs	.15μs	.9μs	27μs	810μs	6.83d	1s
40	.04μs	.21μs	1.6μs	64μs	2.56ms	121d	18m
50	.05μs	.28μs	2.5μs	125μs	6.25ms	3.1y	13d
100	.10μs	.66μs	10μs	1ms	100ms	3171y	$4 * 10^{13}$y
10^3	1μs	9.96μs	1ms	1s	16.67m	$3.17 * 10^{13}$y	$32 * 10^{283}$y
10^4	10μs	130μs	100ms	16.67m	115.7d	$3.17 * 10^{23}$y	
10^5	100μs	1.66ms	10s	11.57d	3171y	$3.17 * 10^{33}$y	
10^6	1ms	19.92ms	16.67m	31.71y	$3.17 * 10^7$y	$3.17 * 10^{43}$y	

Summary

- Big-O ➔ upper bound
- Big-theta ➔ tight (upper & lower) bound
- Big-omega ➔ lower bound

$$n^2 \neq O(n) \quad n^2 = O(n^2) \quad n^2 = O(n^3)$$

$$n^2 \neq \Theta(n) \quad n^2 = \Theta(n^2) \quad n^2 \neq \Theta(n^3)$$

$$\mathrm{n}^2 = \Omega(\mathrm{n}) \quad n^2 = \Omega(n^2) \quad n^2 \neq \Omega(n^3)$$

Summary

- In this chapter
  - We reviewed asymptotic notation O, Ω, Θ, o
  - Asymptotic notation is for making statement about program performance when the input data is large
  - Big O notation is the most popular asymptotic notation
  - Asymptotic notations were introduced in both informal and rigorous manner