Asymptotic Analysis

Analysis of Algorithms

- An ***algorithm*** is a finite set of precise instructions for performing a computation or for solving a problem.
- What is the goal of analysis of algorithms?
  - To compare algorithms mainly in terms of running time but also in terms of other factors (e.g., memory requirements, programmer's effort, etc.)
- What do we mean by running time analysis?
  - **Determine how running time increases as the size of the problem increases**.

2

Input Size

Input size (number of elements in the input)

- size of an array
- polynomial degree
- # of elements in a matrix
- # of bits in the binary representation of the input
- vertices and edges in a graph

3

Types of Analysis

- **Worst case**
  - Provides an upper bound on running time
  - An absolute guarantee that the algorithm would not run longer, no matter what the inputs are
- **Best case**
  - Provides a lower bound on running time
  - Input is the one for which the algorithm runs the fastest
- **Average case**
  - Provides a prediction about the running time
  - Assumes that the input is random

$$\boxed{Lower\ Bound \le Running\ Time \le Upper\ Bound}$$

4

How do we compare algorithms?

Need to define a number of **objective measures.**

(1) Compare execution times?

Not good: times are specific to a particular computer.

(2) Count the number of statements executed?

Not good: number of statements vary with the programming language as well as the style of the individual programmer.

5

Solution

- Express running time as a function of the input size n (i.e., $f(n)$).
- Compare different functions corresponding to running times.
- Such analysis is independent of machine time, programming style, etc.

6

Example

- Associate a "cost" with each statement.
- Find the “total cost” by finding the total number of times each statement is executed.

Algorithm 1

	Cost
arr[0] = 0;	c_1
arr[1] = 0;	c_1
arr[2] = 0;	c_1
...	...
arr[N-1] = 0;	c_1

$c_1+c_1+\ldots+c_1 = c_1 \times N$

Algorithm 2

	Cost
for(i=0; i<N; i++)	c_2
arr[i] = 0;	c_1

$(N+1) \times c_2 + N \times c_1 = (c_2 + c_1) \times N + c_2$

7

Another Example

Algorithm 3	*Cost*
sum = 0;	c_1
for(i=0; i<N; i++)	c_2
for(j=0; j<N; j++)	c_2
sum += arr[i][j];	c_3

$c_1 + c_2 \times (N+1) + c_2 \times N \times (N+1) + c_3 \times N^2$

8

Asymptotic Analysis

- To compare two algorithms with running times *f(n)* and *g(n),* we need a **rough measure** that characterizes **how fast each function grows.**
- **Use rate of growth**
- **Asymptotic**:(of a function) **approaching a given value** as an expression containing **a variable tends to infinity.**
- Compare functions in the limit, that is, **asymptotically** (i.e., for large values of *n*)

9

Rate of Growth

- Consider the example of buying *elephants* and *goldfish:*

  Cost: cost_of_elephants + cost_of_goldfish

  Cost ~ cost_of_elephants (approximation)
- The low order terms in a function are relatively **insignificant** for **large** *n*

$$n^4 + 100n^2 + 10n + 50$$
$$\sim \quad n^4$$

i.e., We say that $n^4 + 100n^2 + 10n + 50$ and n^4 have the **same rate of growth**

10

Asymptotic Notation

- O notation (Big O): asymptotic "**less than**":
  - f(n)=O(g(n))
  - ➔ f(n) ≤ g(n)
- Ω notation (Big OMEGA): asymptotic "**greater than**":
  - f(n)= Ω (g(n))
  - ➔ f(n) ≥ g(n)
- Θ notation (Big THETA): asymptotic "**equality**":
  - f(n)= Θ (g(n))
  - ➔ f(n) = g(n)

11

Big-O Notation

- $f_A(n)=30n + 8$ is order n, or **O(n)**

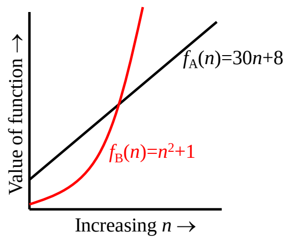
  It is, at most, roughly proportional to n.
- $f_B(n)=n^2+1$ is order n^2, or **O(n^2)**.

  It is, at most, roughly proportional to n^2.
- In general, any O(n^2) function is faster-growing than any O(n) function.

12

Visualizing Orders of Growth

- On a graph, as you go to the right, a faster growing function eventually becomes larger.



More Examples

- $n^4 + 100n^2 + 10n + 50$ is $O(n^4)$
- $10n^3 + 2n^2$ is $O(n^3)$
- $n^3 - n^2$ is $O(n^3)$
- Constants
  - 10 is $O(1)$
  - 1273 is $O(1)$

Back to previous example

Algorithm 1

	Cost
arr[0] = 0;	c_1
arr[1] = 0;	c_1
arr[2] = 0;	c_1
...	
arr[N-1] = 0;	c_1

$c_1+c_1+...+c_1 = c_1 \times N$

Algorithm 2

	Cost
for(i=0; i<N; i++)	c_2
arr[i] = 0;	c_1

$(N+1) \times c_2 + N \times c_1 =$
$(c_2 + c_1) \times N + c_2$

Both algorithms are of the same order: O(N)

Example (cont’d)

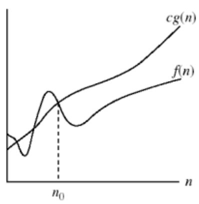
Algorithm 3	***Cost***
sum = 0;	c_1
for(i=0; i<N; i++)	c_2
for(j=0; j<N; j++)	c_2
sum += arr[i][j];	c_3

$c_1 + c_2 \times (N+1) + c_2 \times N \times (N+1) + c_3 \times N^2 = O(N^2)$

Asymptotic notations

- *O-notation*

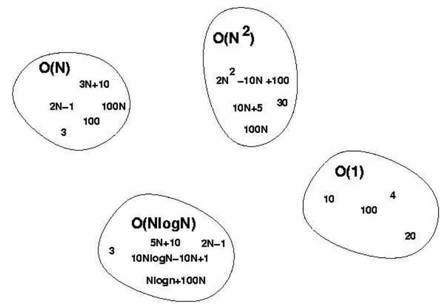
$O(g(n)) = \{f(n) :$ there exist positive constants c and n_0 such that $0 \le f(n) \le cg(n)$ for all $n \ge n_0\}$.



$g(n)$ is an ***asymptotic upper bound*** for $f(n)$.

Big-O Visualization

O(g(n)) is the set of functions with **smaller or same order of growth as g(n)**



Examples

- **$n \le cn^2$**

➔ $cn \ge 1$

➔ $c = 1$ and $n_0 = 1$

- **$2n^2 = O(n^3)$**

➔ $2n^2 \le cn^3$

➔ $2 \le cn$

➔ $c = 1$ and $n_0 = 2$

- **$n^2 = O(n^2)$**

➔ $n^2 \le cn^2$

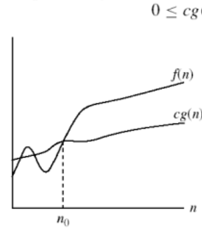
➔ $c \ge 1$

➔ $c = 1$ and $n_0 = 1$

Asymptotic notations (cont.)

- *Ω - notation*

$\Omega(g(n)) = \{f(n) :$ there exist positive constants c and n_0 such that $0 \le cg(n) \le f(n)$ for all $n \ge n_0\}$.



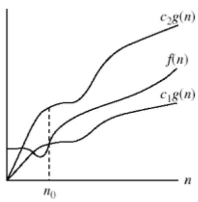
Ω(g(n)) is the set of functions with **larger or same order of growth as g(n)**

$g(n)$ is an ***asymptotic lower bound*** for $f(n)$.

Asymptotic notations (cont.)

- *Θ-notation*

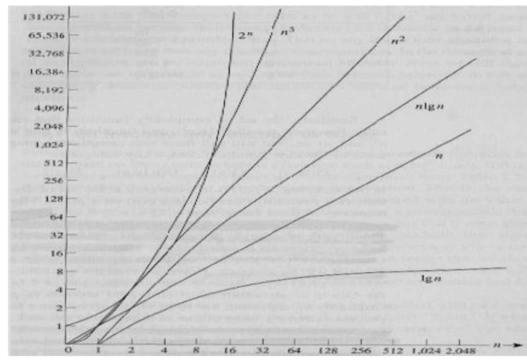
$\Theta(g(n)) = \{ f(n) :$ there exist positive constants c_1, c_2, and n_0 such that $0 \le c_1 g(n) \le f(n) \le c_2 g(n)$ for all $n \ge n_0 \}$.



Θ(g(n)) is the set of functions with the **same order of growth as g(n)**

$g(n)$ is an ***asymptotically tight bound*** for $f(n)$.

Common orders of magnitude



Common orders of magnitude

Table 1.4 Execution times for algorithms with the given time complexities

n	$f(n) = \lg n$	$f(n) = n$	$f(n) = n \lg n$	$f(n) = n^2$	$f(n) = n^3$	$f(n) = 2^n$
10	0.003 µs*	0.01 µs	0.033 µs	0.1 µs	1 µs	1 µs
20	0.004 µs	0.02 µs	0.086 µs	0.4 µs	8 µs	1 ms†
30	0.005 µs	0.03 µs	0.147 µs	0.9 µs	27 µs	1 s
40	0.005 µs	0.04 µs	0.213 µs	1.6 µs	64 µs	18.3 min
50	0.006 µs	0.05 µs	0.282 µs	2.5 µs	125 µs	13 days
10^2	0.007 µs	0.10 µs	0.664 µs	10 µs	1 ms	4×10^{13} years
10^3	0.010 µs	1.00 µs	9.966 µs	1 ms	1 s	
10^4	0.013 µs	10 µs	130 µs	100 ms	16.7 min	
10^5	0.017 µs	0.10 ms	1.67 ms	10 s	11.6 days	
10^6	0.020 µs	1 ms	19.93 ms	16.7 min	31.7 years	
10^7	0.023 µs	0.01 s	0.23 s	1.16 days	31,709 years	
10^8	0.027 µs	0.10 s	2.66 s	115.7 days	3.17×10^7 years	
10^9	0.030 µs	1 s	29.90 s	31.7 years		

*1 µs = 10^{-6} second.
†1 ms = 10^{-3} second.

Logarithms and properties

- In algorithm analysis we often use the notation "**log n**" without specifying the base

Binary logarithm $\lg n = \log_2 n$

Natural logarithm $\ln n = \log_e n$

$\lg^k n = (\lg n)^k$

$\lg \lg n = \lg(\lg n)$

$\log x^y = y \log x$

$\log xy = \log x + \log y$

$\log \frac{x}{y} = \log x - \log y$

$a^{\log_b x} = x^{\log_b a}$

$\log_b x = \frac{\log_a x}{\log_a b}$

Common Summations

- Arithmetic series: $\sum_{k=1}^{n} k = 1+2+...+n = \frac{n(n+1)}{2}$
- Geometric series: $\sum_{k=0}^{n} x^k = 1+x+x^2+...+x^n = \frac{x^{n+1}-1}{x-1} \left(x \neq 1\right)$
  - Special case: $|x| < 1$: $\sum_{k=0}^{\infty} x^k = \frac{1}{1-x}$
- Harmonic series: $\sum_{k=1}^{n} \frac{1}{k} = 1 + \frac{1}{2} + ... + \frac{1}{n} \approx \ln n$
- Other important formulas: $\sum_{k=1}^{n} \lg k \approx n \lg n$

$$\sum_{k=1}^{n} k^p = 1^p + 2^p + ... + n^p \approx \frac{1}{p+1} n^{p+1}$$

27

Exercise 1

- For each of the following pairs of functions, either f(n) is O(g(n)), f(n) is Ω(g(n)), or f(n) = Θ(g(n)). Determine which relationship is correct.
  - f(n) = n; g(n) = log n^2 — f(n) = Ω(g(n))
  - f(n) = log log n; g(n) = log n — f(n) = O(g(n))
  - f(n) = n; g(n) = $\log^2 n$ — f(n) = Ω(g(n))
  - f(n) = n log n + n; g(n) = log n — f(n) = Ω(g(n))
  - f(n) = 10; g(n) = log 10 — f(n) = Ω(g(n))
  - f(n) = 2^n; g(n) = 3^n — f(n) = O(g(n))
  - f(n) = log n^2; g(n) = log n + 5 — f(n) = Θ (g(n))
  - f(n) = 2^n; g(n) = $10n^2$ — f(n) = Θ(g(n))

28