

Running Time: Estimation Rules

- Running time is proportional to the most significant term in $T(n)$
- Once a problem size becomes large, the most significant term is that which has the largest power of n
- This term increases faster than other terms which reduce in significance

Lecture 3 COMPSCI.220.S1.T - 2004 1

Algorithm Versus Implementation

- Analysis of time complexity takes no account of the constant of proportionality c
- Analysis involves relative changes of running time

TOOLS for ANALYSING TIME COMPLEXITY

input data \rightarrow algorithm

Size $n \rightarrow$ Number $f(n)$ of computational steps

programming language, compiler, computer, ... constant c

Running time:
 $T(n) = c \cdot f(n)$

Relative running time change:
 $\frac{T(n_1)}{T(n_2)} = \frac{c \cdot f(n_1)}{c \cdot f(n_2)} = \frac{f(n_1)}{f(n_2)}$

Lecture 3 COMPSCI.220.S1.T - 2004 4

Running Time: Estimation Rules

- Running time $T(n) = 0.25n^2 + 0.5n \approx 0.25n^2$

n	$T(n)$	$0.25n^2$	$0.5n$	
			value	%
10	30	25	5	16.7
50	650	625	25	3.8
100	2,550	2,500	50	2.0
500	62,750	62,500	250	0.4

Lecture 3 COMPSCI.220.S1.T - 2004 2

Elementary Operations

- Basic arithmetic operations (+ ; - ; * ; / ; %)
 - Relational operators (==, !=, >, <, >=, <=)
- Boolean operations (AND, OR, NOT),
- Branch operations, ...

Input for problem domains (meaning of n):

Sorting: n items Graph / path: n vertices / edges
 Image processing: n pixels Text processing: string length

Lecture 3 COMPSCI.220.S1.T - 2004 5

Running Time: Estimation Rules

- Constants of proportionality depend on the compiler, language, computer, etc.
 - It is useful to ignore the constants when analysing algorithms.
- Constants of proportionality are reduced by using faster hardware or minimising time spent on the "inner loop"
 - But this would not effect behaviour of an algorithm for a large problem!

Lecture 3 COMPSCI.220.S1.T - 2004 3

"Big-Oh" Tool $O(\dots)$ for Analysing Algorithms

Typical curves of time complexity:

$T(n) \propto \log n$,
 $T(n) \propto n$
 $T(n) \propto n \log n$
 $T(n) \propto n^k$
 $T(n) \propto 2^n$

Lecture 3 COMPSCI.220.S1.T - 2004 6

Relative growth: $G(n) = f(n) / f(5)$

Complexity	Function $f(n)$	Input size n	5	25	125	625
Constant	1	1	1	1	1	1
Logarithm	$\log n$	1	2	3	4	
Linear	n	1	5	25	125	
" $n \log n$ "	$n \log n$	1	10	75	500	
Quadratic	n^2	1	25	625	15,625	
Cubic	n^3	1	125	15,625	5 ⁹	
Exponential	2^n	1	2 ²⁰	2 ¹²⁰	2 ⁶²⁰	

Lecture 3 COMPSCI.220.S1.T - 2004 7

$g(n)$ is $O(f(n))$, or $g(n) = O(f(n))$

Function $g(n)$ is "Big-Oh" of $f(n)$ if, starting from some $n > n_0$, there always exist a function $c \cdot f(n)$ that grows faster than the function $g(n)$

$n > (n_0 = 238): g(n) < (f(n) = n)$
 $n > (n_0 = 1000): g(n) < (f(n) = 0.3 \cdot n)$

Lecture 3 COMPSCI.220.S1.T - 2004 10

"Big-Oh" $O(\dots)$: Linear Complexity

Linear complexity \leftrightarrow
time $T(n) \propto n$
 $O(n) \leftrightarrow$ running time does not grow faster than a linear function of the problem size n

Lecture 3 COMPSCI.220.S1.T - 2004 8

"Big-Oh" $O(\dots)$: Formal Definition

for those who are not afraid of Maths:

- Let $f(n)$ and $g(n)$ be positive-valued functions defined on the positive integers n
- The function g is defined as $O(f)$ and is said to be of the order of $f(n)$ iff (read: if and only if) there are a real constant $c > 0$ and an integer n_0 such that $g(n) \leq c \cdot f(n)$ for all $n > n_0$

Lecture 3 COMPSCI.220.S1.T - 2004 11

Logarithmic "Big-Oh" Complexity

Logarithmic complexity:
time $T(n) \propto \log n$
 $O(\log n) \leftrightarrow$ running time does not grow faster than a log function of the problem size n

Lecture 3 COMPSCI.220.S1.T - 2004 9


"Big-Oh" $O(\dots)$: Informal Meaning

for those who are afraid of Maths: g is $O(f)$ means that the algorithm with time complexity g runs (for large n) **at most** as fast, within a constant factor, as the algorithm with time complexity f

Note that the formal definition of "Big-Oh" differs slightly from the Stage I calculus:

$$\lim_{n \rightarrow \infty} \left\{ \frac{g(n)}{f(n)} \right\} = c$$


Lecture 3 COMPSCI.220.S1.T - 2004 12

 The University of Auckland

"Big-Oh" $O(\dots)$: Informal Meaning

- g is $O(f)$ means that the order of time complexity of the function g is **asymptotically less than or equal to** the order of time complexity of the function f
 - Asymptotical behaviour \leftrightarrow only for the large values of n
 - Two functions are of the same order when they each are "Big-Oh" of the other: $f = O(g)$ AND $g = O(f)$
 - This property is called "Big-Theta": $g = \Theta(f)$

Lecture 3 COMPSCL220.S1.T - 2004 13

 The University of Auckland

"Big-Oh" Examples - 2


Polynomial $P_k(n) = a_k n^k + a_{k-1} n^{k-1} + \dots + a_1 n + a_0$;
 $a_k > 0$, is $O(n^k)$

Do not write $O(P_k(n))$ as this means still $O(n^k)$!

$O(n^k)$ - running time:

- $T(n) = 3n^2 + 5n + 1$ is $O(n^2)$ Is it also $O(n^3)$?
- $T(n) = 10^{-8} n^3 + 10^8 n^2 + 30$ is $O(n^3)$
- $T(n) = 10^{-8} n^8 + 1000n + 1$ is $O(n^8)$


Lecture 3 COMPSCL220.S1.T - 2004 16

 The University of Auckland

$O(\dots)$ Comparisons: Two Crucial Ideas

- The exact running time function is not important, since it can be multiplied by any arbitrary positive constant.
- Two functions are compared only **asymptotically**, for large n , and not near the origin
 - If the constants involved are very large, then the asymptotical behaviour is of no practical interest!

Lecture 3 COMPSCL220.S1.T - 2004 14

 The University of Auckland

"Big-Oh" Examples - 3


Exponential $g(n) = 2^{n+k}$ is $O(2^n)$: $2^{n+k} = 2^k \cdot 2^n$ for all n

Exponential $g(n) = m^{n+k}$ is $O(l^n)$, $l \geq m > 1$:

$$m^{n+k} \leq l^{n+k} = l^k \cdot l^n \text{ for all } n, k$$

- Remember that a "brute-force" search for the best combination of n interdependent binary decisions by exhausting all the 2^n possible combinations has the exponential time complexity, and **try to find more efficient ways to solve your problem** if $n \geq 20 \dots 30$

Lecture 3 COMPSCL220.S1.T - 2004 17

 The University of Auckland

"Big-Oh" Examples - 1

Linear function $g(n) = an + b$; $a > 0$, is $O(n)$:

$$g(n) < (a + |b|) \cdot n \text{ for all } n \geq 1$$

Do not write $O(2n)$ or $O(an + b)$ as this means still $O(n)$!


$O(n)$ - running time:

$$T(n) = 3n + 1 \quad T(n) = 10^8 + n$$

$$T(n) = 50 + 10^{-8} \cdot n \quad T(n) = 10^6 \cdot n + 1$$

Remember that "Big-Oh" describes an "asymptotic behaviour" for large problem sizes

Lecture 3 COMPSCL220.S1.T - 2004 15

 The University of Auckland

"Big-Oh" Examples - 4

Logarithmic function

$$g(n) = \log_m n$$

is of order $\log_2 n$ because

$$\log_m n = \log_m 2 \cdot \log_2 n \text{ for all } n, m > 0$$

Do not write $O(\log_m n)$ as this means still $O(\log n)$!

You will find later that the most efficient search for data in an ordered array has the logarithmic complexity

Lecture 3 COMPSCL220.S1.T - 2004 18