The University of Auckland

Time Complexity of Algorithms

- If running time $T(n)$ is $O(f(n))$ then the function f measures time complexity
  - **Polynomial** algorithms: $T(n)$ is $O(n^k)$; $k = \text{const}$
  - **Exponential** algorithm: otherwise
- **Intractable problem**: if no polynomial algorithm is known for its solution

The University of Auckland

Answer these questions

Running time $T(n)$	Complexity $O(n)$
$n^2 + 100\,n + 1$	
$0.001n^3 + n^2 + 1$	
$23\,n$	
2^{3n}	
2^{3+n}	
$2 \cdot 3^n$	

The University of Auckland

Answer these questions

Running time $T(n)$	Complexity $O(n)$
$0.0001\,n + 10000$	
$100000\,n + 10000$	
$0.0001\,n^2 + 10000\,n$	
$100000\,n^2 + 10000\,n$	
$30 \log_{20}(23n)$ **actually NOT that hard…**	

The University of Auckland

Big-Omega

- The function $g(n)$ is $\Omega(f(n))$ iff there exist a real positive constant $c > 0$ and a positive integer n_0 such that $g(n) \geq cf(n)$ for all $n \geq n_0$
  - Big Omega is just opposite to Big Oh
  - It generalises the concept of "lower bound" (≥) in the same way as Big Oh generalises the concept of "upper bound" (≤)
  - If $f(n)$ is $O(g(n))$ then $g(n)$ is $\Omega(f(n))$

The University of Auckland

Big-Theta

- The function $g(n)$ is $\Theta(f(n))$ iff there exist two real positive constants $c_1 > 0$ and $c_2 > 0$ and a positive integer n_0 such that:

  $c_1 f(n) \geq g(n) \geq c_2 f(n)$ for all $n \geq n_0$
  - Whenever two functions, f and g, are of the same order, $g(n)$ is $\Theta(f(n))$, they are each Big-Oh of the other: $g(n)$ is $O(f(n))$ AND $f(n)$ is $O(g(n))$

The University of Auckland

Upper bounds of complexity

"Big-Oh" specifies an **upper bound of complexity** so that the following (and like) relationships hold:

$1 = O(\log n) = O(n) = O(n \log n) = \ldots$

$\log n = O(n) = O(n \log n) = O(n^\alpha);\ \alpha > 1 = \ldots$

$n = O(n \log n) = O(n^\alpha);\ \alpha > 1 = O(2^n) = \ldots$

$n \log n = O(n^\alpha);\ \alpha > 1 = O(n^k);\ k > \alpha = \ldots$

$n^k = O(n^\alpha);\ \alpha > k = O(2^n) = \ldots$

Time complexity growth

$f(n)$	Number of data items processed per:			
	1 minute	1 day	1 year	1 century
n	10	14,400	$5.26 \cdot 10^6$	$5.26 \cdot 10^8$
$n \log n$	10	3,997	883,895	$6.72 \cdot 10^7$
$n^{1.5}$	10	1,275	65,128	$1.40 \cdot 10^6$
n^2	10	379	7,252	72,522
n^3	10	112	807	3,746
2^n	10	20	29	35

Beware exponential complexity

☺ If a linear, $O(n)$, algorithm processes 10 items per minute, then it can process 14,400 items per day, 5,260,000 items per year, and 526,000,000 items per century.

☻ If an exponential, $O(2^n)$, algorithm processes 10 items per minute, then it can process only 20 items per day and 35 items per century...

Big-Oh vs. Actual Running Time

- Example 1: let algorithms **A** and **B** have running times $T_A(n) = 20n$ ms and $T_B(n) = 0.1n \log_2 n$ ms
- In the "Big-Oh"sense, **A** is better than **B**…
- But: on which data volume can **A** outperform **B**?

  $T_A(n) < T_B(n)$ if $20n < 0.1n \log_2 n$, or

  $\log_2 n > 200$, that is, when $n > 2^{200} \approx 10^{60}$!
- Thus, in all practical cases **B** is better than **A**…

Big-Oh vs. Actual Running Time

- Example 2: let algorithms **A** and **B** have running times $T_A(n) = 20n$ ms and $T_B(n) = 0.1n^2$ ms
- In the "Big-Oh"sense, **A** is better than **B**…
- But: on which data volumes **A** outperforms **B**?

  $T_A(n) < T_B(n)$ if $20n < 0.1n^2$, or $n > 200$
- Thus **A** is better than **B** in most practical cases except for $n < 200$ when **B** becomes faster…

"Big-Oh" Feature 1: Scaling

- Constant factors are ignored. Only the powers and functions of n should be exploited:

  for all $c > 0 \rightarrow c \cdot f = O(f)$ where $f \equiv f(n)$
- It is this ignoring of constant factors that motivates for such a notation!
- ***Examples***: $50n$, $50000000n$, and $0.0000005n$ are $O(n)$

"Big-Oh" Feature 2: Transitivity

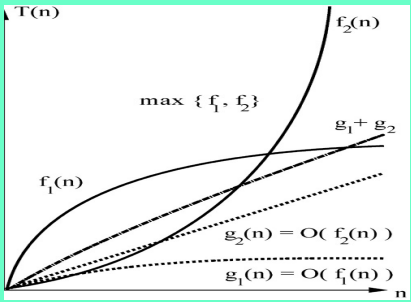
- If h does not grow faster than g and g does not grow faster than f, then h does not grow faster than f:

  $h = O(g)$ **AND** $g = O(f) \rightarrow h = O(f)$
- In other words, if f grows faster than g and g grows faster than h, then f grows faster than h
- ***Example***: $h = O(g)$; $g = O(n^2) \rightarrow h = O(n^2)$

Feature 3: The Rule of Sums

- The sum grows as its fastest term:
  $g_1 = O(f_1)$ AND $g_2 = O(f_2) \rightarrow g_1+g_2 = O(\max\{f_1,f_2\})$
  - if $g = O(f)$ and $h = O(f)$, then $g + h = O(f)$
  - if $g = O(f)$, then $g + f = O(f)$
- ***Examples***:
  - if $h = O(n)$ AND $g = O(n^2)$, then $g + h = O(n^2)$
  - if $h = O(n \log n)$ AND $g = O(n \log \log n)$, then $g + h = O(n \log n)$

The Rule of Sums



Feature 4: The Rule of Products

- The upper bound for the product of functions is given by the product of the upper bounds for the functions:
  $g_1 = O(f_1)$ AND $g_2 = O(f_2) \rightarrow g_1 \cdot g_2 = O(f_1 \cdot f_2)$
  - if $g = O(f)$ and $h = O(f)$, then $g \cdot h = O(f^2)$
  - if $g = O(f)$, then $g \cdot h = O(f \cdot h)$
- ***Example:***
  if $h = O(n)$ AND $g = O(n^2)$, then $g \cdot h = O(n^3)$

Ascending order of complexity

$1 \leftarrow \log \log n \leftarrow \log n \leftarrow n \leftarrow n \log n$
$\leftarrow n^{\alpha};\ 1<\alpha<2 \leftarrow n^2 \leftarrow n^3 \leftarrow n^m;\ m > 3 \leftarrow 2^n \ldots$

Questions:

- Where is the place of $n^2 \log n$?
- Where is the place of $n^{2.79}$?

- ***Answers:*** $\ldots \leftarrow n^2 \leftarrow n^2 \log n \leftarrow n^{2.79} \leftarrow n^3 \leftarrow \ldots$

Answers to the questions

Running time $T(n)$	Complexity $O(n)$
$n^2 + 100\,n + 1$	$O(n^2)$
$0.001n^3 + n^2 + 1$	$O(n^3)$
$23\,n$	$O(n)$
2^{3n}	$O(8^n)$ as $2^{3n} \equiv (2^3)^n$
2^{3+n}	$O(2^n)$ as $2^{3+n} \equiv 2^3 \cdot 2^n$
$2 \cdot 3^n$	$O(3^n)$

Answers to the questions

Running time $T(n)$	Complexity $O(n)$
$0.0001\,n + 10000$	$O(n)$
$100000\,n + 10000$	$O(n)$
$0.0001\,n^2 + 10000\,n$	$O(n^2)$
$100000\,n^2 + 10000\,n$	$O(n^2)$
$30 \log_{20}(23n)$ **actually NOT that hard…**	$O(\log n)$ as $\log_c(ab) = \log_c a + \log_c b$