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

Time complexity growth

$f(n)$	Number of data items processed per:			
	1 minute	1 day	1 year	1 century
n	**10**	**14,400**	**$5.26 \cdot 10^6$**	**$5.26 \cdot 10^8$**
$n \log_{10} n$	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: Scaling

For all $c > 0 \rightarrow cf$ is $O(f)$ where $f \equiv f(n)$

Proof: $cf(n) < (c+\varepsilon)f(n)$ holds for all $n > 0$ and $\varepsilon > 0$

- Constant factors are ignored. Only the powers and functions of n should be exploited
- It is this ignoring of constant factors that motivates for such a notation! In particular, f is $O(f)$
- ***Examples:*** $50n \in O(n)$ $\quad 0.05n \in O(n)$

  $50000000n \in O(n)$ $\quad 0.0000005n \in O(n)$

Big-Oh: Transitivity

If h is $O(g)$ and g is $O(f)$, then h is $O(f)$

Informally: if h grows at most as fast as g, which grows at most as fast as f, then h grows at most as fast as f

Examples: $h \in O(g)$; $g \in O(n^2) \rightarrow h \in O(n^2)$

$\log_{10} n \in O(n^{0.01})$; $n^{0.01} \in O(n) \rightarrow \log_{10} n \in O(n)$

$2^n \in O(3^n)$; $n^{50} \in O(2^n) \rightarrow n^{50} \in O(3^n)$

Big-Oh: Rule of Sums

If $g_1 \in O(f_1)$ and $g_2 \in O(f_2)$, then $g_1+g_2 \in O(\max\{f_1, f_2\})$

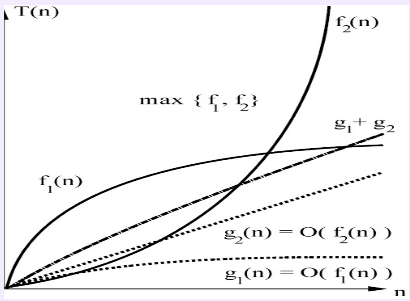
The sum grows as its fastest-growing term:

- if $g \in O(f)$ and $h \in O(f)$, then $g + h \in O(f)$
- if $g \in O(f)$, then $g + f \in O(f)$

Examples:

- if $h \in O(n)$ and $g \in O(n^2)$, then $g + h \in O(n^2)$
- if $h \in O(n \log n)$ and $g \in O(n)$, then $g + h \in O(n \log n)$

Rule of Sums



Big-Oh: Rule of Products

If $g_1 \in O(f_1)$ and $g_2 \in O(f_2)$, then $g_1 g_2 \in O(f_1 f_2)$

The product of upper bounds of functions gives an upper bound for the product of the functions:

- if $g \in O(f)$ and $h \in O(f)$, then $gh \in O(f^2)$
- if $g \in O(f)$, then $gh \in O(f h)$

Examples:

if $h \in O(n)$ and $g \in O(n^2)$, then $gh \in O(n^3)$

if $h \in O(\log n)$ and $g \in O(n)$, then $gh \in O(n \log n)$

Big-Oh: Limit Rule

Suppose $L \leftarrow \lim_{n\to\infty} f(n)/g(n)$ exists (may be ∞)

Then if $L = 0$, then f is $O(g)$

if $0 < L < \infty$, then f is $\Theta(g)$

if $L = \infty$, then f is $\Omega(g)$

To compute the limit, the standard **L'Hopital rule** of calculus is useful: if $\lim_{x\to\infty} f(x) = \infty = \lim_{x\to\infty} g(x)$ and f, g are positive differentiable functions for $x > 0$, then $\lim_{x\to\infty} f(x)/g(x) = \lim_{x\to\infty} f'(x)/g'(x)$ where $f'(x)$ is the derivative

Examples 1.23, 1.24, p.19

- **Ex.1.23**: Exponential functions grow faster than powers: n^k is $O(b^n)$ for all $b>1$, $n>1$, and $k\geq 0$

  Proof: by induction or by the limit L'Hopital approach
- **Ex. 1.24**: Logarithmic functions grow slower than powers: $\log_b n$ is $O(n^k)$ for all $b>1$, $k>0$
  - $\log_b n$ is $O(\log n)$ for all $b>1$: $\log_b n = \log_b a \log_a n$
  - $\log n$ is $O(n)$
  - $n \log n$ is $O(n^2)$