Lecture 4: Properties of and Rules for

Asymptotic Big-Oh, Big-Omega, and Big-Theta Notation

Georgy Gimel'farb

COMPSCI 220 Algorithms and Data Structures

1. Time complexity
2. Big-Oh rules
   - Scaling
   - Transitivity
   - Rule of sums
   - Rule of products
   - Limit rule
3. Examples

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

$f(n)$	Approximate number of data items processed per:			
	1 minute	1 day	1 year	1 century
n	10	$14,400$	5.3×10^6	5.3×10^8
$n \log_{10} n$	10	$4,000$	8.8×10^5	6.7×10^7
$n^{1.5}$	10	1.3×10^3	6.5×10^4	1.4×10^6
n^2	10	380	7.3×10^3	7.3×10^4
n^3	10	110	810	3.7×10^3
2^n	10	20	29	35

Beware Exponential Complexity!

- A linear, $O(n)$, algorithm processing **10** items per minute, can process $\mathbf{1.4 \times 10^4}$ items per day, $\mathbf{5.3 \times 10^6}$ items per year, and $\mathbf{5.3 \times 10^8}$ items per century.
- An exponential, $O(2^n)$, algorithm processing **10** items per minute, can process only **20** items per day and only **35** items per century...

Time Complexity Growth

$f(n)$	**Approximate number of data items processed per:**			
	1 minute	1 day	1 year	1 century
n	10	14, 400	5.3×10^6	5.3×10^8
$n \log_{10} n$	10	4, 000	8.8×10^5	6.7×10^7
$n^{1.5}$	10	1.3×10^3	6.5×10^4	1.4×10^6
n^2	10	380	7.3×10^3	7.3×10^4
n^3	10	110	810	3.7×10^3
2^n	10	20	29	35

Beware Exponential Complexity!

- A linear, $O(n)$, algorithm processing **10** items per minute, can process $\mathbf{1.4 \times 10^4}$ items per day, $\mathbf{5.3 \times 10^6}$ items per year, and $\mathbf{5.3 \times 10^8}$ items per century.
- An exponential, $O(2^n)$, algorithm processing **10** items per minute, can process only **20** items per day and only **35** items per century...

Big-Oh vs. Actual Running Time

Example 1:

Algorithms A and B with running times $T_A(n) = 20n$ time units and $T_B(n) = 0.1n \log_2 n$ time units, respectively.

- In the "Big-Oh" sense, the linear algorithm A is better than the linearithmic algorithm B...
- **But:** on which data volume can A outperform B, i.e. for which value n the running time for A is less than for B?

$$T_A(n) < T_B(n) \quad \text{if} \quad 20n < 0.1n \log_2 n,$$
$$\text{or} \quad \log_2 n > 200, \qquad \text{that is, when } \mathbf{n > 2^{200} \approx 10^{60}!}$$

Thus, in all practical cases the algorithm B is better than A...

Big-Oh vs. Actual Running Time

Example 1:

Algorithms A and B with running times $T_A(n) = 20n$ time units and $T_B(n) = 0.1n \log_2 n$ time units, respectively.

- In the "Big-Oh" sense, the linear algorithm A is better than the linearithmic algorithm B...
- **But:** on which data volume can A outperform B, i.e. for which value n the running time for A is less than for B?

$$T_A(n) < T_B(n) \quad \text{if} \quad 20n < 0.1n \log_2 n,$$
$$\text{or} \quad \log_2 n > 200, \quad \text{that is, when } \mathbf{n > 2^{200} \approx 10^{60}}!$$

Thus, in all practical cases the algorithm B is better than A...

Big-Oh vs. Actual Running Time

Example 1:

Algorithms A and B with running times $T_A(n) = 20n$ time units and $T_B(n) = 0.1n \log_2 n$ time units, respectively.

- In the "Big-Oh" sense, the linear algorithm A is better than the linearithmic algorithm B...
- **But:** on which data volume can A outperform B, i.e. for which value n the running time for A is less than for B?

$$T_A(n) < T_B(n) \quad \text{if} \quad 20n < 0.1n \log_2 n,$$

or $\log_2 n > 200$, that is, when $\mathbf{n > 2^{200} \approx 10^{60}}$!

Thus, in all practical cases the algorithm B is better than A...

Big-Oh vs. Actual Running Time

Example 1:

Algorithms A and B with running times $T_A(n) = 20n$ time units and $T_B(n) = 0.1n \log_2 n$ time units, respectively.

- In the "Big-Oh" sense, the linear algorithm A is better than the linearithmic algorithm B...
- **But:** on which data volume can A outperform B, i.e. for which value n the running time for A is less than for B?

$$T_A(n) < T_B(n) \quad \text{if} \quad 20n < 0.1n \log_2 n,$$
$$\text{or} \quad \log_2 n > 200, \quad \text{that is, when } \mathbf{n > 2^{200} \approx 10^{60}}!$$

Thus, in all practical cases the algorithm B is better than A...

Big-Oh vs. Actual Running Time

Example 1:

Algorithms A and B with running times $T_A(n) = 20n$ time units and $T_B(n) = 0.1n\log_2 n$ time units, respectively.

- In the "Big-Oh" sense, the linear algorithm A is better than the linearithmic algorithm B...
- **But:** on which data volume can A outperform B, i.e. for which value n the running time for A is less than for B?

$$T_A(n) < T_B(n) \quad \text{if} \quad 20n < 0.1n\log_2 n,$$

$$\text{or} \quad \log_2 n > 200, \quad \text{that is, when } \mathbf{n > 2^{200} \approx 10^{60}}!$$

Thus, in all practical cases the algorithm B is better than A...

Big-Oh vs. Actual Running Time

Example 2:

Algorithms A and B with running times $T_A(n) = 20n$ time units and $T_B(n) = 0.1n^2$ time units, respectively.

- In the "Big-Oh" sense, the linear algorithm A is better than the quadratic algorithm B...
- **But:** on which data volume can A outperform B, i.e. for which value n the running time for A is less than for B?

$$T_A(n) < T_B(n) \text{ if } 20n < 0.1n^2, \text{ or } n > 200$$

Thus the algorithm A is better than B in most of practical cases except for $n < 200$ when B becomes faster...

Big-Oh vs. Actual Running Time

Example 2:

Algorithms A and B with running times $T_A(n) = 20n$ time units and $T_B(n) = 0.1n^2$ time units, respectively.

- In the "Big-Oh" sense, the linear algorithm A is better than the quadratic algorithm B...
- **But:** on which data volume can A outperform B, i.e. for which value n the running time for A is less than for B?

$$T_A(n) < T_B(n) \text{ if } 20n < 0.1n^2, \text{ or } n > 200$$

Thus the algorithm A is better than B in most of practical cases except for $n < 200$ when B becomes faster...

Big-Oh vs. Actual Running Time

Example 2:

Algorithms A and B with running times $T_A(n) = 20n$ time units and $T_B(n) = 0.1n^2$ time units, respectively.

- In the "Big-Oh" sense, the linear algorithm A is better than the quadratic algorithm B...
- **But:** on which data volume can A outperform B, i.e. for which value n the running time for A is less than for B?

$$T_A(n) < T_B(n) \text{ if } 20n < 0.1n^2, \text{ or } n > 200$$

Thus the algorithm A is better than B in most of practical cases except for $n < 200$ when B becomes faster...

Big-Oh vs. Actual Running Time

Example 2:

Algorithms A and B with running times $T_A(n) = 20n$ time units and $T_B(n) = 0.1n^2$ time units, respectively.

- In the "Big-Oh" sense, the linear algorithm A is better than the quadratic algorithm B...
- **But:** on which data volume can A outperform B, i.e. for which value n the running time for A is less than for B?

$$T_A(n) < T_B(n) \text{ if } 20n < 0.1n^2, \text{ or } n > 200$$

Thus the algorithm A is better than B in most of practical cases except for $n < 200$ when B becomes faster...

Big-Oh vs. Actual Running Time

Example 2:

Algorithms A and B with running times $T_A(n) = 20n$ time units and $T_B(n) = 0.1n^2$ time units, respectively.

- In the "Big-Oh" sense, the linear algorithm A is better than the quadratic algorithm B...
- **But:** on which data volume can A outperform B, i.e. for which value n the running time for A is less than for B?

$$T_A(n) < T_B(n) \text{ if } 20n < 0.1n^2, \text{ or } n > 200$$

Thus the algorithm A is better than B in most of practical cases except for $n < 200$ when B becomes faster...

Big-Oh: Scaling

Scaling (Lemma 1.15; p.15)

For all constant factors $c > 0$, the function $cf(n)$ is $O(f(n))$, or in shorthand notation cf is $O(f)$.

The 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: $\begin{cases} 50n \in O(n) & 0.05n \in O(n) \\ 50,000,000n \in O(n) & 0.0000005n \in O(n) \end{cases}$

Big-Oh: Scaling

Scaling (Lemma 1.15; p.15)

For all constant factors $c > 0$, the function $cf(n)$ is $O(f(n))$, or in shorthand notation cf is $O(f)$.

The 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: $\begin{cases} 50n \in O(n) & 0.05n \in O(n) \\ 50,000,000n \in O(n) & 0.0000005n \in O(n) \end{cases}$

Big-Oh: Scaling

Scaling (Lemma 1.15; p.15)

For all constant factors $c > 0$, the function $cf(n)$ is $O(f(n))$, or in shorthand notation cf is $O(f)$.

The 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: $\begin{cases} 50n \in O(n) & 0.05n \in O(n) \\ 50,000,000n \in O(n) & 0.0000005n \in O(n) \end{cases}$

Big-Oh: Scaling

Scaling (Lemma 1.15; p.15)

For all constant factors $c > 0$, the function $cf(n)$ is $O(f(n))$, or in shorthand notation cf is $O(f)$.

The 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 $O(f)$.

$$\text{Examples: } \begin{cases} 50n \in O(n) & 0.05n \in O(n) \\ 50,000,000n \in O(n) & 0.0000005n \in O(n) \end{cases}$$

Big-Oh: Transitivity

Transitivity (Lemma 1.16; p.15)

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.

$$\text{Examples:} \begin{cases} h \in O(g); & g \in O(n^2) & \rightarrow & h \in O(n^2) \\ \log_{10} n \in O(n^{0.01}); & n^{0.01} \in O(n) & \rightarrow & \log_{10} n \in O(n) \\ 2^n \in O(3^n); & n^{50} \in O(2^n) & \rightarrow & n^{50} \in O(3^n) \end{cases}$$

The proof: If $h(n) \leq c_1 g(n)$ for $n > n_1$ and $g(n) \leq c_2 f(n)$ for $n > n_2$, then $h(n) \leq \underbrace{c_1 c_2}_{c} f(n)$ for $n > \underbrace{\max\{n_1, n_2\}}_{n_0}$.

Big-Oh: Transitivity

Transitivity (Lemma 1.16; p.15)

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.

$$\text{Examples:} \left\{ \begin{array}{llcl} h \in O(g); & g \in O(n^2) & \rightarrow & h \in O(n^2) \\ \log_{10} n \in O(n^{0.01}); & n^{0.01} \in O(n) & \rightarrow & \log_{10} n \in O(n) \\ 2^n \in O(3^n); & n^{50} \in O(2^n) & \rightarrow & n^{50} \in O(3^n) \end{array} \right.$$

The proof: If $h(n) \leq c_1 g(n)$ for $n > n_1$ and $g(n) \leq c_2 f(n)$ for $n > n_2$, then $h(n) \leq \underbrace{c_1 c_2}_{c} f(n)$ for $n > \underbrace{\max\{n_1, n_2\}}_{n_0}$.

Big-Oh: Rule of Sums

Rule-of-sums (Lemma 1.17; p.15)

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:

$$\begin{cases} \text{If } h \in O(n) \text{ and } g \in O(n^2), \text{ then } g + h \in O(n^2) \\ \text{If } h \in O(n \log n) \text{ and } g \in O(n), \text{ then } g + h \in O(n \log n) \end{cases}$$

The proof: If $g_1(n) \le c_1 f_1(n)$ for $n > n_1$ and $g_2(n) \le c_2 f_2(n)$ for $n > n_2$, then $g_1(n) + g_2(n) \le c_1 f_1(n) + c_2 f_2(n) \le \underbrace{\max\{c_1, c_2\}}_{c} (f_1(n) + f_2(n)) \le$

$\underbrace{2 \cdot \max\{c_1, c_2\}}_{c} \cdot \max\{f_1(n), f_2(n)\}$ for $n > \underbrace{\max\{n_1, n_2\}}_{n_0}$.

Big-Oh: Rule of Sums

Rule-of-sums (Lemma 1.17; p.15)

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 $g + f \in O(f)$.

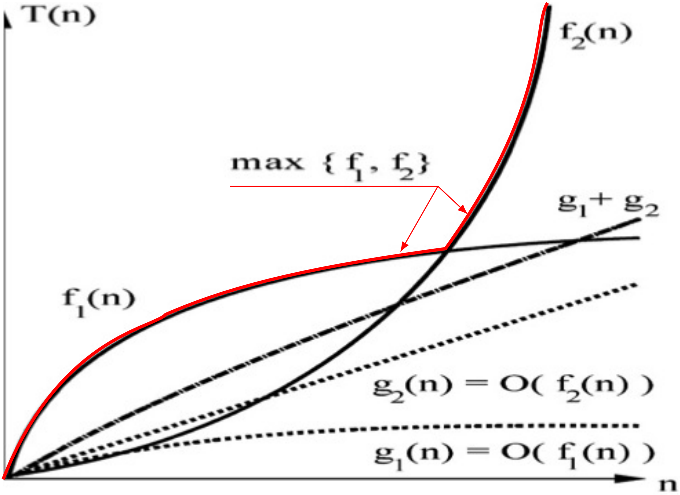
Examples:

$$\begin{cases} \text{If} \quad h \in O(n) & \text{and} \quad g \in O(n^2), \quad \text{then} \quad g + h \in O(n^2) \\ \text{If} \quad h \in O(n \log n) & \text{and} \quad g \in O(n), \quad \text{then} \quad g + h \in O(n \log n) \end{cases}$$

The proof: If $g_1(n) \leq c_1 f_1(n)$ for $n > n_1$ and $g_2(n) \leq c_2 f_2(n)$ for $n > n_2$, then $g_1(n) + g_2(n) \leq c_1 f_1(n) + c_2 f_2(n) \leq \underbrace{\max\{c_1, c_2\}}_{c} (f_1(n) + f_2(n)) \leq$

$\underbrace{2 \cdot \max\{c_1, c_2\}}_{c} \cdot \max\{f_1(n), f_2(n)\}$ for $n > \underbrace{\max\{n_1, n_2\}}_{n_0}$.

Big-Oh: Rule of Sums



Big-Oh: Rule of Products

Rule-of-products (Lemma 1.18; p.16)

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 $gh \in O(fh)$.

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 $gh \in O(n \log n)$.

The proof: If $g_1(n) \leq c_1 f_1(n)$ for $n > n_1$ and $g_2(n) \leq c_2 f_2(n)$ for $n > n_2$, then $g_1(n) g_2(n) \leq \underbrace{c_1 c_2}_{c} f_1(n) f_2(n)$ for $n > \underbrace{\max\{n_1, n_2\}}_{n_0}$.

Big-Oh: Rule of Products

Rule-of-products (Lemma 1.18; p.16)

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 $gh \in O(fh)$.

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 $gh \in O(n \log n)$.

The proof: If $g_1(n) \leq c_1 f_1(n)$ for $n > n_1$ and $g_2(n) \leq c_2 f_2(n)$ for $n > n_2$, then $g_1(n)g_2(n) \leq \underbrace{c_1 c_2}_{c} f_1(n) f_2(n)$ for $n > \underbrace{\max\{n_1, n_2\}}_{n_0}$.

Big-Oh: Rule of Products

Rule-of-products (Lemma 1.18; p.16)

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 $gh \in O(fh)$.

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 $gh \in O(n \log n)$.

The proof: If $g_1(n) \le c_1 f_1(n)$ for $n > n_1$ and $g_2(n) \le c_2 f_2(n)$ for $n > n_2$, then $g_1(n)g_2(n) \le \underbrace{c_1 c_2}_{c} f_1(n) f_2(n)$ for $n > \underbrace{\max\{n_1, n_2\}}_{n_0}$.

Big-Oh: Rule of Products

Rule-of-products (Lemma 1.18; p.16)

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 $gh \in O(fh)$.

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 $gh \in O(n \log n)$.

The proof: If $g_1(n) \leq c_1 f_1(n)$ for $n > n_1$ and $g_2(n) \leq c_2 f_2(n)$ for $n > n_2$, then $g_1(n) g_2(n) \leq \underbrace{c_1 c_2}_{c} f_1(n) f_2(n)$ for $n > \underbrace{\max\{n_1, n_2\}}_{n_0}$.

Big-Oh: Rule of Products

Rule-of-products (Lemma 1.18; p.16)

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 $gh \in O(fh)$.

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 $gh \in O(n \log n)$.

The proof: If $g_1(n) \leq c_1 f_1(n)$ for $n > n_1$ and $g_2(n) \leq c_2 f_2(n)$ for $n > n_2$, then $g_1(n)g_2(n) \leq \underbrace{c_1 c_2}_{c} f_1(n) f_2(n)$ for $n > \underbrace{\max\{n_1, n_2\}}_{n_0}$.

Big-Oh: The Limit Rule

Suppose the ratio's limit $\lim_{n\to\infty} \frac{f(n)}{g(n)} = L$ exists (may be infinite, ∞).

$$\text{Then} \begin{cases} \text{if} \quad L = 0 & \text{then} \quad f \in O(g) \\ \text{if} \quad 0 < L < \infty & \text{then} \quad f \in \Theta(g) \\ \text{if} \quad L = \infty & \text{then} \quad f \in \Omega(g) \end{cases}$$

When f and g are positive and differentiable functions for $x > 0$, but $\lim_{x\to\infty} f(x) = \lim_{x\to\infty} g(x) = \infty$ or $\lim_{x\to\infty} f(x) = \lim_{x\to\infty} g(x) = 0$, the limit L can be computed using the standard **L'Hopital** rule of calculus:

$$\lim_{x\to\infty} \frac{f(x)}{g(x)} = \lim_{x\to\infty} \frac{f'(x)}{g'(x)}$$

where $z'(x)$ denotes the first derivative of the function $z(x)$.

Examples 1.22 and 1.23 (Textbook, p.16)

Example 1.22: 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 rule (the L'Hopital approach):

Successive (k times) differentiation of n^k and b^n by n:

$$\lim_{n\to\infty} \frac{n^k}{b^n} = \lim_{n\to\infty} \frac{kn^{k-1}}{b^n \ln b} = \lim_{n\to\infty} \frac{k(k-1)n^{k-2}}{b^n (\ln b)^2} = \ldots = \lim_{n\to\infty} \frac{k!}{b^n (\ln b)^k} = 0.$$

Example 1.23: Logarithmic functions grow slower than powers: $\log_b n$ is $O(n^k)$ for all $b > 1$, $k > 0$.

Proof: This is the inverse of the preceding feature.

As a result, $\log n \in O(n)$ and $n \log n \in O(n^2)$.

$\log_b n$ is $O(\log n)$ for all $b > 1$ because $\log_b n = \log_b a \times \log_a n$

Examples 1.22 and 1.23 (Textbook, p.16)

Example 1.22: 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 rule (the L'Hopital approach):

Successive (k times) differentiation of n^k and b^n by n:

$$\lim_{n\to\infty} \frac{n^k}{b^n} = \lim_{n\to\infty} \frac{kn^{k-1}}{b^n \ln b} = \lim_{n\to\infty} \frac{k(k-1)n^{k-2}}{b^n (\ln b)^2} = \ldots = \lim_{n\to\infty} \frac{k!}{b^n (\ln b)^k} = 0.$$

Example 1.23: Logarithmic functions grow slower than powers: $\log_b n$ is $O(n^k)$ for all $b > 1$, $k > 0$.

Proof: This is the inverse of the preceding feature.

As a result, $\log n \in O(n)$ and $n \log n \in O(n^2)$.

$\log_b n$ is $O(\log n)$ for all $b > 1$ because $\log_b n = \log_b a \times \log_a n$