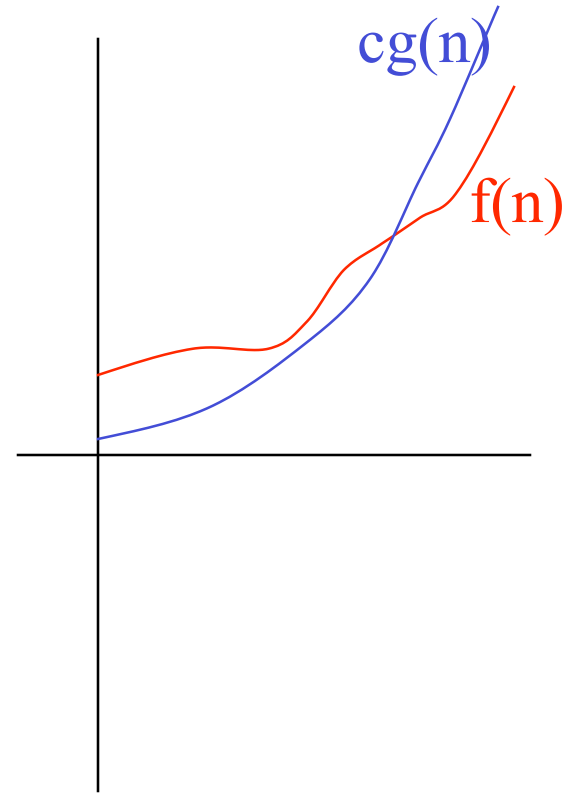
Complexity

- Complexity: study of how time&space to execute an algorithm vary with problem size
- Let n denote the problem size ("size" of data being manipulated)
- Let function t(n) denote the time to execute algorithm with size n
- The function t(n) depends on
  - implementation details,
  - language used,
  - compiler, etc.
- Solution:
  - Abstraction, *ie*, concentrate on general characteristics of t(n)
  - *eg*, growth of t(n) proportional to n.
    - n (linear),
    - n^2 (quadratic), or
    - log n (logarithmic) etc.

Big-O Notation

- Big-O notation is used to express an **upper bound** of a function. For example f(n) is O(g(n)) if f is bounded from above by a constant multiple of g(n) for sufficiently large values of n.



- More formally:
  there exist constants $c > 0$ and $m > 0$
  such that $f(n) \leq cg(n)$ whenever $n \geq m$

- Or using limit notation:

$$\lim_{n \to \infty} \frac{f(n)}{g(n)} = \text{ 0 or a finite positive constant}$$

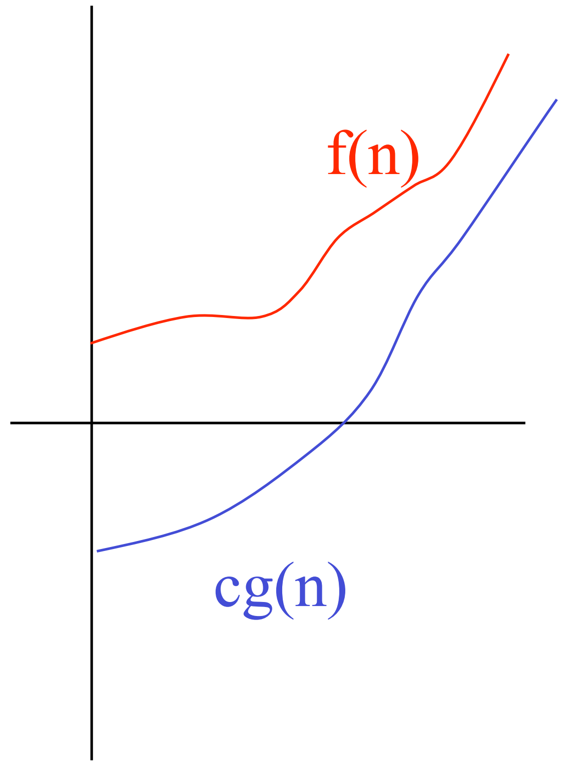
Relatives of Big-O Notation

- Big-Ω notation is used to express an **lower bound** of a function. For example f(n) is Ω(g(n)) if f is bounded from below by a constant multiple of g(n) for sufficiently large values of n.

- More formally:
  there exist constants $c > 0$ and $m > 0$
  such that $f(n) \geq cg(n)$ whenever $n \geq m$

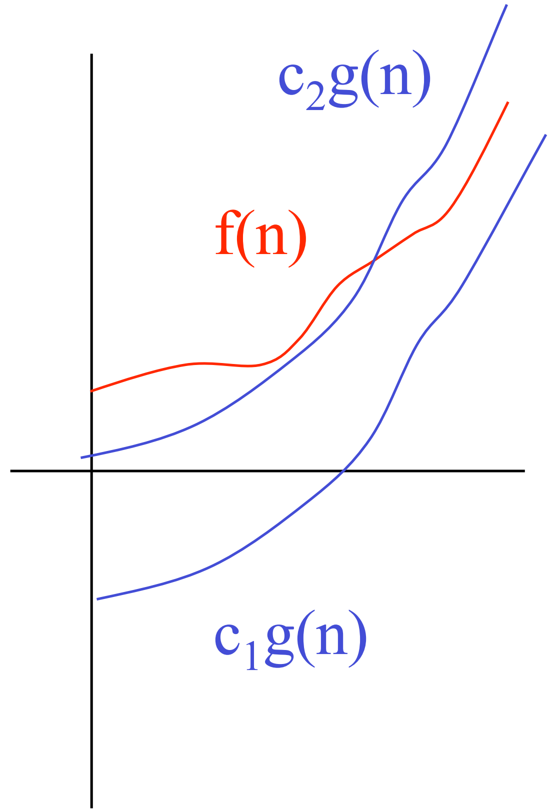
- Or using limit notation:

$$\lim_{n\to\infty} \frac{g(n)}{f(n)} = \text{ 0 or a finite positive constant}$$



Relatives of Big-O Notation

- Big-Θ notation is used to express a **tight bound** of a function.
- f(n) is Θ(g(n)) if f(n) is both O(g(n)) and Ω(g(n))
- Or using limit notation:



$$\lim_{n\to\infty}\frac{f(n)}{g(n)} = \lim_{n\to\infty}\frac{g(n)}{f(n)} = \text{a finite positive constant}$$

Examples of Big-O

- General rule: take the **dominant term** (increases the fastest) and discard the constant factor

- Let $f(n) = 3n^2 + 4n + 2$
  $f(n)$ is in $O(n^2)$

- Let $t(n) = 5\log_2 n$
  $t(n)$ is in $O(\log n)$

- Let $w(n) = 6n + 2\log_2 n$
  $w(n)$ is in $O(n)$