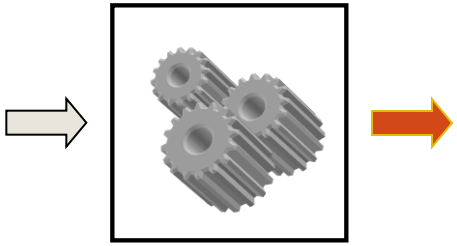
Asymptotic Notation, Review of Functions & Summations

CMS-A-CC-4-9-TH
Introduction to Algorithms & its Applications

Introduction

Data Structures: A systematic way of organizing and accessing data.
-- No single data structure works well for ALL purposes.



Input **Algorithm** **Output**

An **algorithm** is a step-by-step procedure for solving a problem in a finite amount of time.

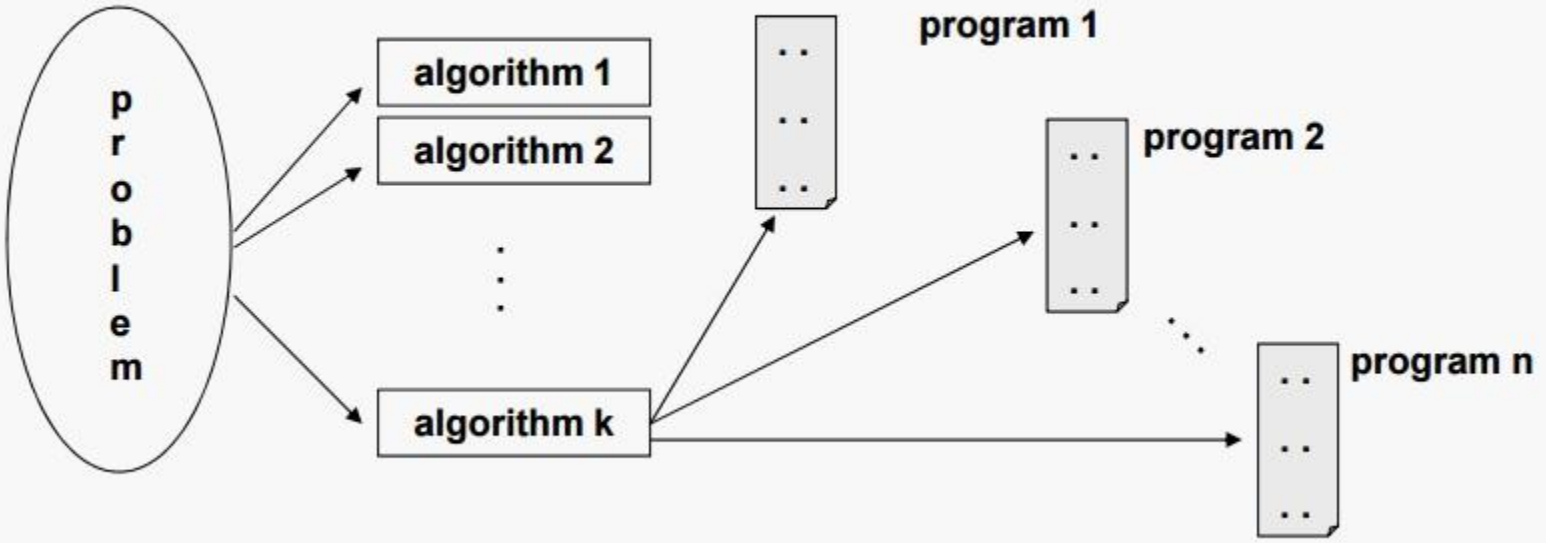
Program = algorithms + data structures

Properties of a good Algorithm

finiteness:	The algorithm must always terminate after a finite number of steps.
definiteness:	Each step must be precisely defined; the actions to be carried out must be rigorously and unambiguously specified for each case.
input:	An algorithm has zero or more inputs, taken from a specified set of objects.
output:	An algorithm has one or more outputs, which have a specified relation to the inputs.
effectiveness:	All operations to be performed must be sufficiently basic that they can be done exactly and in finite length.

Properties of a good Algorithm

- For each problem or class of problems, there may be many different algorithms.
- For each algorithm, there may be many different implementations (programs).



Analysis of Algorithms

Analysis invloves –

- Tracing the algorithm for logical correctness, implementing the algorithm and testing it on some data.
- Checking the algorithm for simplicity

However, the simplest way of solving a problem is sometimes not the best one, specially when, the simplest approach involves the use of too much computer time or space!!

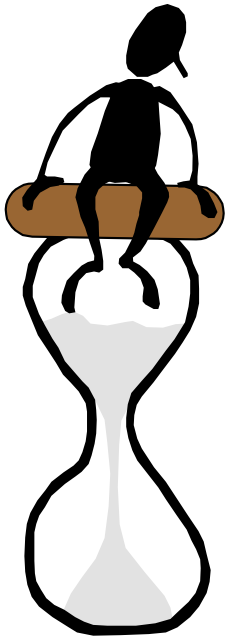
Analysis of Algorithms

- Estimate the running time – **Time Complexity T(n)**
- Estimate the memory space required – **Space Complexity S(n)**

Both depends on the input size n

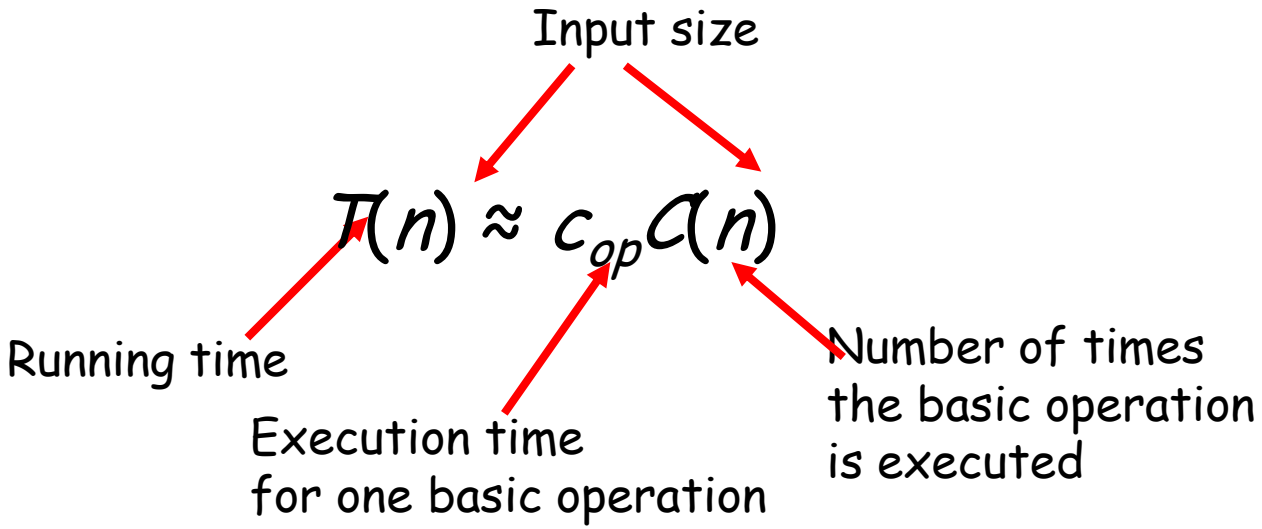
Definition of Time

- # of seconds (machine, implementation dependent).
- # lines of code executed.
- # of times a specific operation is performed (e.g., addition).



Theoretical analysis of time efficiency

- Time efficiency is analyzed by determining the number of repetitions of the *basic operation* as a function of *input size*
- *Basic operation:* the operation that contributes most towards the running time of the algorithm.



Best-case, Average-case, Worst-case

- **Worst case**: W(*n*) – "maximum" over inputs of size *n*
  - Provides an upper bound on running time
  - An absolute guarantee that the algorithm would not run longer, no matter what the inputs are

- **Best case**: B(*n*) – "minimum" over inputs of size *n*
  - Provides a lower bound on running time
  - Input is the one for which the algorithm runs the fastest

Lower bound ≤ Running time ≤ Upper bound

Best-case, Average-case, Worst-case (Contd..)

- **Average case**: $A(n)$ – "average" over inputs of size n
  - Provides a prediction about the running time
  - Assumes that the input is random
  - Under some assumption about the probability distribution of all possible inputs of size n, calculates the weighted sum of expected C(n) (numbers of basic operation repetitions) over all possible inputs of size n.

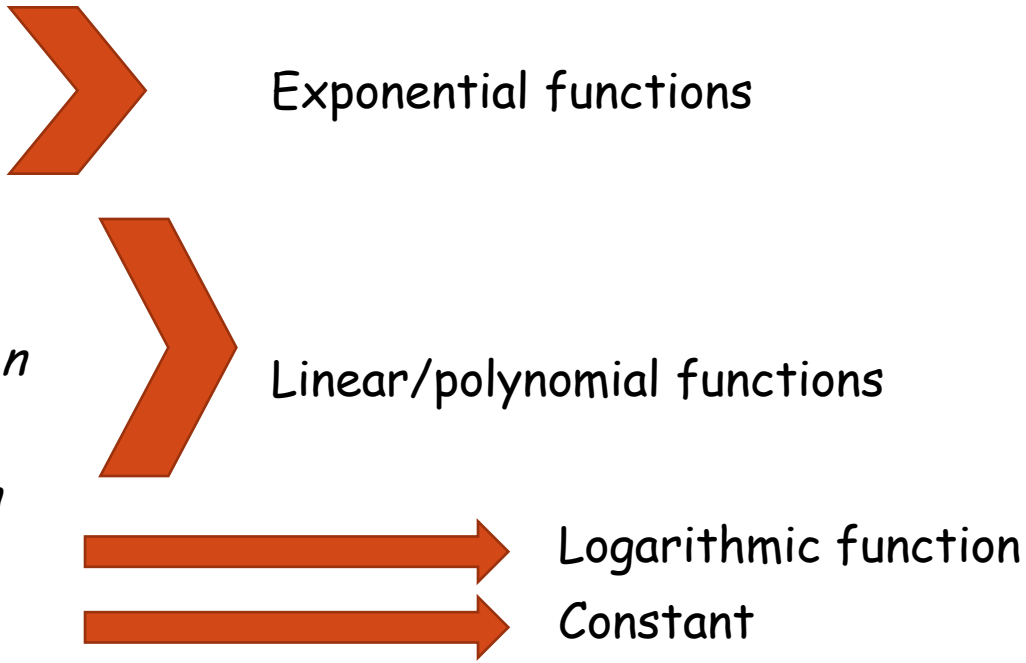
Asymptotic Complexity

- Running time of an algorithm as a function of input size n **for large n**.
- Expressed using only the **highest-order term** in the expression for the exact running time.
  - Instead of exact running time, say $\Theta(n^2)$.
- Describes behavior of function in the limit.
- Written using ***Asymptotic Notation.***

Growth of functions

- The rate of change of execution time **T(n)** with the change in the number of inputs is called **growth of function**.

- T(n) can be approximately equal to:
  - n^n
  - $n!$
  - 2^n
  - n^3
  - n^2
  - $n\log_2 n$
  - n
  - $\log_2 n$
  - 1



Comparison - for large n

$$\text{As } x \to \infty$$

Factorial	Exponential	Polynomial	Logarithmic
$x!$	e^x	$x\sqrt{1+x^2}$	$\ln(\ln x)$
	$\cosh x$	x^2+1	$(\ln x)^3$
	$3^{\sqrt{x}}$	$\sqrt{x}$	$\ln x$
	2^x		

Asymptotic Analysis

- To compare two algorithms with running times f(n) and g(n), we need a **rough measure** that characterizes **how fast each function grows.**

- *Hint:* use *rate of growth*

- Compare functions in the limit, that is, **asymptotically!**
  (i.e., for large values of *n*)

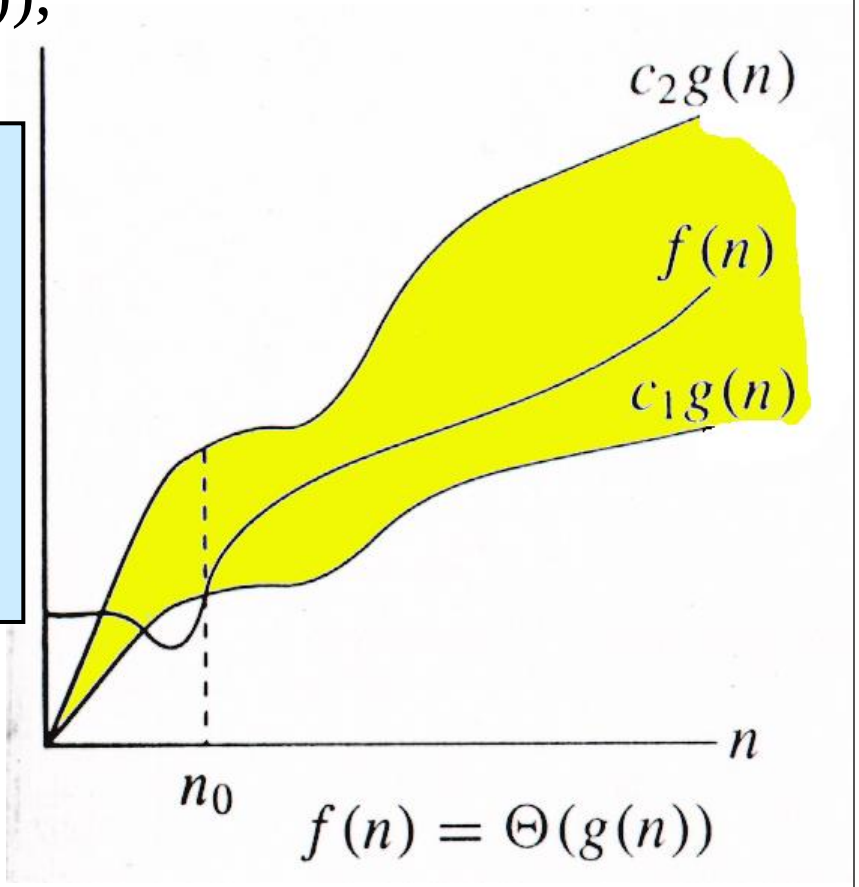
Asymptotic Notation

- **$\Theta, O, \Omega, o, \omega$**
- Defined for functions over the natural numbers.
  - **Ex:** $f(n) = \Theta(n^2)$.
  - Describes how $f(n)$ grows in comparison to n^2.
- Define a ***set*** of functions; in practice used to compare two function sizes.
- The notations describe different rate-of-growth relations between the defining function and the defined set of functions.

Θ-notation

For function $g(n)$, we define $\Theta(g(n))$, big-Theta of n, as the set:

$\Theta(g(n)) = \{f(n)$:
$\exists$ positive constants c_1, c_2, and n_0,
such that $\forall n \geq n_0$,
we have $0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n)$
$\}$



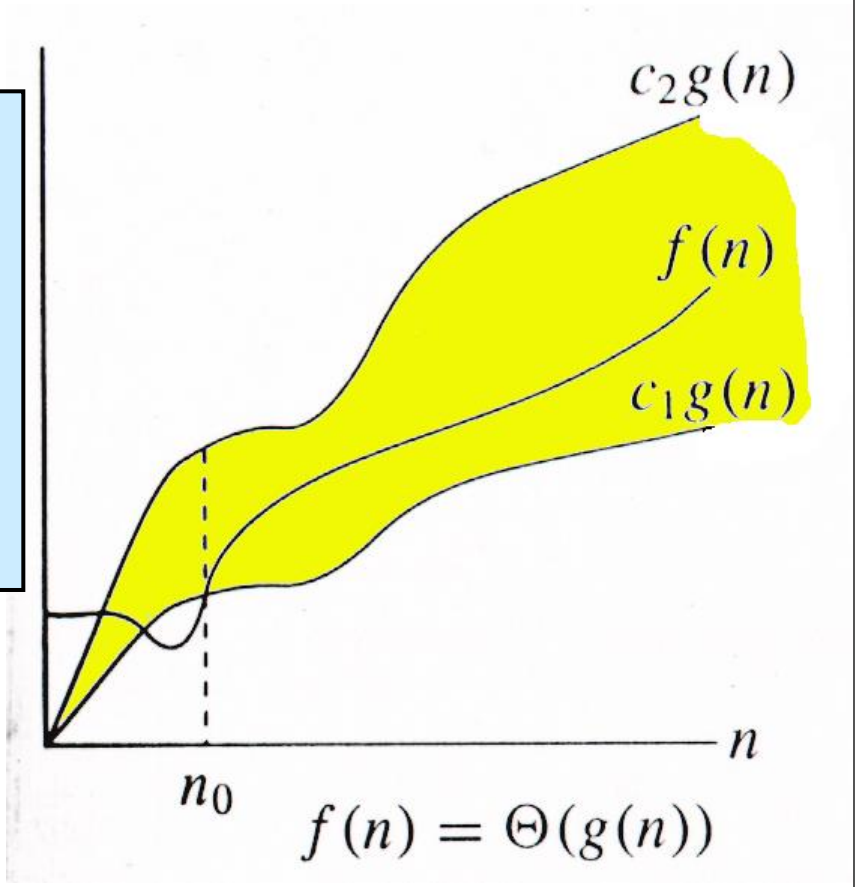
Intuitively: Set of all functions that have the same *rate of growth* as $g(n)$.

$g(n)$ is an *asymptotically tight bound* for $f(n)$.

Θ-notation

For function $g(n)$, we define $\Theta(g(n))$, big-Theta of n, as the set:

$\Theta(g(n)) = \{f(n)$:
$\exists$ positive constants c_1, c_2, and n_0,
such that $\forall n \geq n_0$,
we have $0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n)$
$\}$



Technically, $f(n) \in \Theta(g(n))$.
Older usage, $f(n) = \Theta(g(n))$.
I'll accept either…

***f*(*n*) and *g*(*n*) are nonnegative, for large *n*.**

Tight bounds

- $n(n-1)/2$ is $\Theta(n2)$

Upper bound

- $n(n-1)/2 = n2/2 - n/2 \leq n2/2$, for $n \geq 0$

Lower bound

- $n(n-1)/2 = n2/2 - n/2 \geq n2/2 - (n/2 \times n/2) \geq n2/4$,

for $n \geq 2$

- Choose $n0 = 2$, $c1 = 1/2$ and $c2 = 1/4$

Example

$\Theta(g(n)) = \{f(n) : \exists$ positive constants c_1, c_2, and n_0, such that $\forall n \geq n_0$, $0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n)\}$

- $10n^2 - 3n = \Theta(n^2)$
- What constants for n_0, c_1, and c_2 will work?
- Make c_1 a little smaller than the leading coefficient, and c_2 a little bigger.
- ***To compare orders of growth, look at the leading term.***
- Exercise: Prove that $n^2/2 - 3n = \Theta(n^2)$

Example

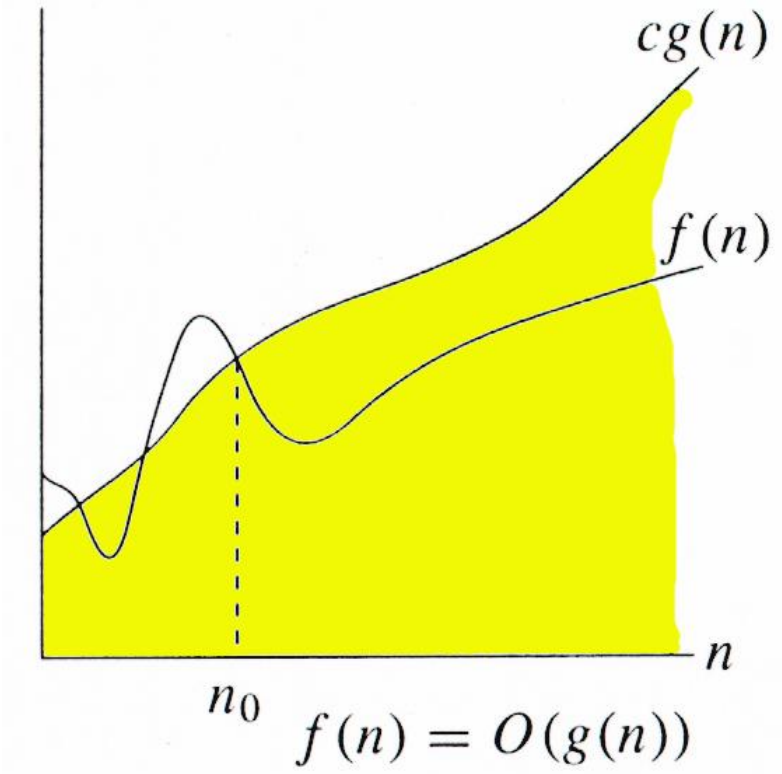
$\Theta(g(n)) = \{f(n) : \exists$ positive constants c_1, c_2, and n_0, such that $\forall n \geq n_0$, $0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n)\}$

- Is $3n^3 \in \Theta(n^4)$??
- How about $2^{2n} \in \Theta(2^n)$??

O-notation

For function $g(n)$, we define $O(g(n))$, big-O of n, as the set:

$O(g(n)) = \{f(n)$:
∃ positive constants c and n_0,
such that $\forall n \geq n_0$,
we have $0 \leq f(n) \leq cg(n)\}$



Intuitively: Set of all functions whose *rate of growth* is the same as or lower than that of $g(n)$.

$g(n)$ is an *asymptotic upper bound* for $f(n)$.

$f(n) = \Theta(g(n)) \Rightarrow f(n) = O(g(n))$.

$\Theta(g(n)) \subset O(g(n))$.

Examples: Big O

- 100n + 5 is O(n^2)
- 100n + 5
- ≤ 100n + n, for n ≥ 5
- = 101n ≤ 101n^2, so n0 = 5, c = 101
- Alternatively
- 100n + 5
- ≤ 100n + 5n, for n ≥1
- = 105n ≤ 105n^2, so n0 = 1, c = 105
- n0 and c are not unique!
- Of course, by the same argument, 100n+5 is also O(n)

Examples: Big O

- 100n^2 + 20n + 5 is O(n^2)
- 100n^2 + 20n + 5
- ≤ 100n^2 + 20n^2 + 5n^2, for n ≥ 1
- ≤ 125n^2
- n0 = 1, c = 125
- What matters is the highest term
- 20n + 5 dominated by 100n^2
- Is n^3 O(n^2)?
- No matter what c we choose, cn^2 will be dominated by n3 for n ≥ c

Examples

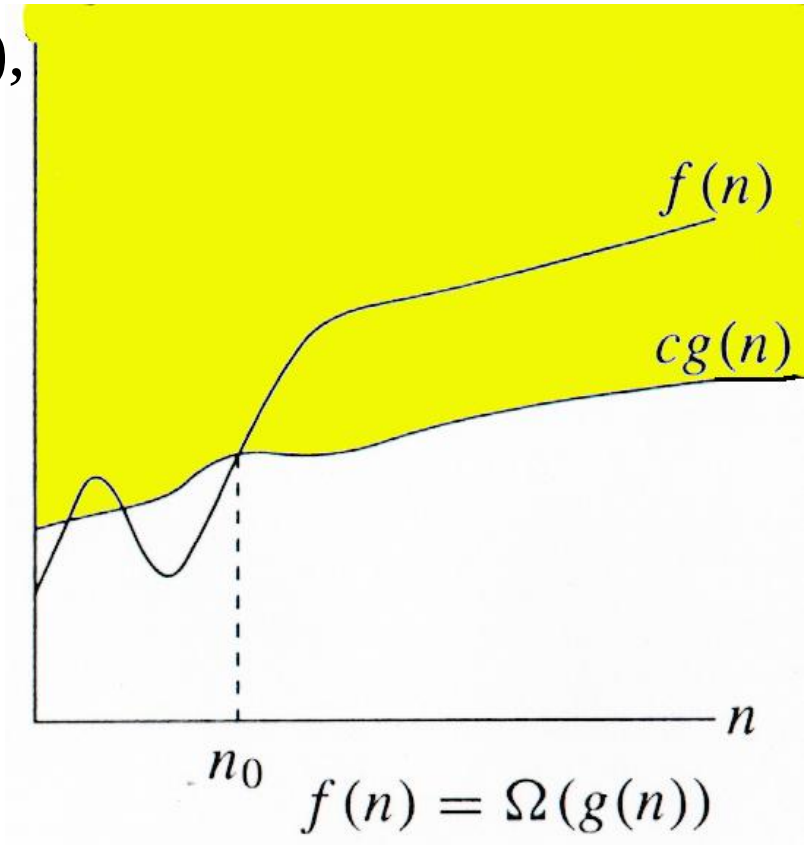
$O(g(n)) = \{f(n) : \exists$ positive constants c and n_0, such that $\forall n \geq n_0$, we have $0 \leq f(n) \leq cg(n)\}$

- Any linear *function* $an + b$ is in $O(n^2)$. **How?**
- Show that $3n^3 = O(n^4)$ for appropriate c and n_0.

Ω -notation

For function $g(n)$, we define $\Omega(g(n))$, big-Omega of n, as the set:

$\Omega(g(n)) = \{f(n)$:
$\exists$ positive constants c and n_0,
such that $\forall n \geq n_0$,
we have $0 \leq cg(n) \leq f(n)\}$



Intuitively: Set of all functions whose *rate of growth* is the same as or higher than that of $g(n)$.

$g(n)$ is an *asymptotic lower bound* for $f(n)$.

$f(n) = \Theta(g(n)) \Rightarrow f(n) = \Omega(g(n))$.
$\Theta(g(n)) \subset \Omega(g(n))$.

Example

$\Omega(g(n)) = \{f(n) : \exists$ positive constants c and n_0, such that $\forall n \geq n_0$, we have $0 \leq cg(n) \leq f(n)\}$

- $\sqrt{n} = \Omega(\lg n)$. Choose c and n_0.

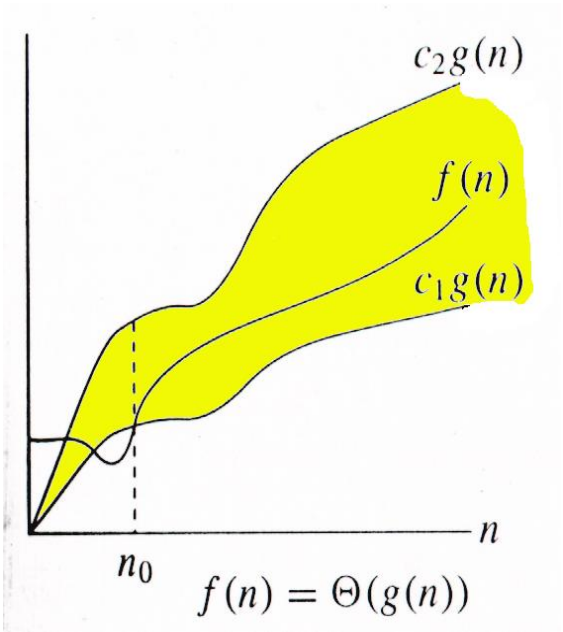
Example

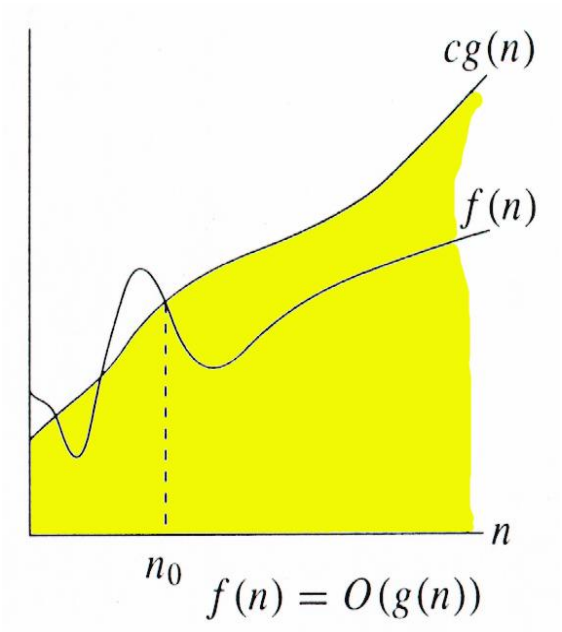
- n^3is Ω(n^2)
- n^3 ≥ n^2 for all n
- n0 = 0 and c = 1

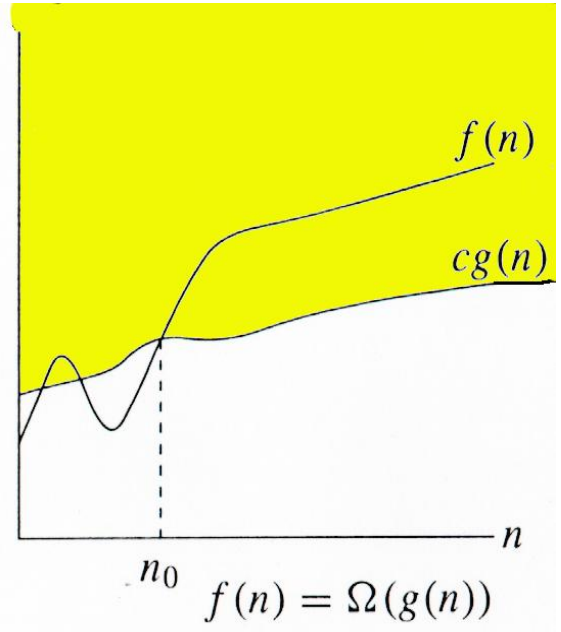
Relationship between Big O & Big Ω

$$f(n) = \Omega(g(n)) \text{ iff}$$
$$g(n) = O(f(n))$$

Relations Between Θ, O, Ω







Relations Between Θ, Ω, O

Theorem : For any two functions $g(n)$ and $f(n)$,

$f(n) = \Theta(g(n))$ iff

$f(n) = O(g(n))$ and $f(n) = \Omega(g(n))$.

- I.e., $\Theta(g(n)) = O(g(n)) \cap \Omega(g(n))$
- In practice, asymptotically tight bounds are obtained from asymptotic upper and lower bounds.

Running Times

- "Running time is $O(f(n))$" $\Rightarrow$ Worst case is $O(f(n))$
- $O(f(n))$ bound on the worst-case running time $\Rightarrow$ $O(f(n))$ bound on the running time of every input.
- $\Theta(f(n))$ bound on the worst-case running time $\Rightarrow$ $\Theta(f(n))$ bound on the running time of every input.
- "Running time is $\Omega(f(n))$" $\Rightarrow$ Best case is $\Omega(f(n))$
- Can still say "Worst-case running time is $\Omega(f(n))$"
  - Means worst-case running time is given by some unspecified function $g(n) \in \Omega(f(n))$.

Example

- ***Insertion sort*** takes $\Theta(n^2)$ in the worst case, so sorting (as a *problem*) is $O(n^2)$. **Why?**
- Any sort algorithm must look at each item, so sorting is $\Omega(n)$.
- In fact, using (e.g.) merge sort, sorting is $\Theta(n \lg n)$ in the worst case.
  - Later, we will prove that we cannot hope that any comparison sort to do better in the worst case.

Asymptotic Notation in Equations

- Can use asymptotic notation in equations to replace expressions containing lower-order terms.
- For example,

  $4n^3 + 3n^2 + 2n + 1 = 4n^3 + 3n^2 + \Theta(n)$

  $= 4n^3 + \Theta(n^2) = \Theta(n^3)$. **How to interpret?**
- In equations, $\Theta(f(n))$ always stands for an ***anonymous function*** $g(n) \in \Theta(f(n))$
  - In the example above, $\Theta(n^2)$ stands for $3n^2 + 2n + 1$.

o-notation

For a given function $g(n)$, the set little-*o*:

$$o(g(n)) = \{f(n): \forall\, c > 0, \exists\, n_0 > 0 \text{ such that } \forall\, n \geq n_0, \text{ we have } 0 \leq f(n) < cg(n)\}.$$

$f(n)$ becomes insignificant relative to $g(n)$ as n approaches infinity:

$$\lim_{n \to \infty} [f(n) / g(n)] = 0$$

$g(n)$ is an ***upper bound*** for $f(n)$ that is not asymptotically tight.

Observe the difference in this definition from previous ones. **Why?**

For a given function $g(n)$, the set little-omega: ω -notation

$\omega(g(n)) = \{f(n)$: $\forall\ c > 0$, $\exists\ n_0 > 0$ such that
$\forall\ n \geq n_0$, we have $0 \leq cg(n) < f(n)\}$.

$f(n)$ becomes arbitrarily large relative to $g(n)$ as n approaches infinity:

$$\lim_{n \to \infty} [f(n) / g(n)] = \infty.$$

$g(n)$ is a ***lower bound*** for $f(n)$ that is not asymptotically tight.

Asymptotic Analysis and Limits

If $\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$, then $f(n) = o(g(n))$.

If $\lim_{n\to\infty} \frac{f(n)}{g(n)} = c$, for some constant $c > 0$, then $f(n) = \Theta(g(n))$.

If $\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$, then $a^{f(n)} = o(a^{g(n)})$, for any $a > 1$.

$f(n) = o(g(n)) \Longrightarrow a^{f(n)} = o(a^{g(n)})$, for any $a > 1$.

$f(n) = \Theta(g(n)) \not\Longrightarrow a^{f(n)} = \Theta(a^{g(n)})$

Limits

- $\lim_{n\to\infty}[f(n) / g(n)] = 0 \Rightarrow f(n) \in o(g(n))$
- $\lim_{n\to\infty}[f(n) / g(n)] < \infty \Rightarrow f(n) \in O(g(n))$
- $0 < \lim_{n\to\infty}[f(n) / g(n)] < \infty \Rightarrow f(n) \in \Theta(g(n))$
- $0 < \lim_{n\to\infty}[f(n) / g(n)] \Rightarrow f(n) \in \Omega(g(n))$
- $\lim_{n\to\infty}[f(n) / g(n)] = \infty \Rightarrow f(n) \in \omega(g(n))$
- $\lim_{n\to\infty}[f(n) / g(n)]$ undefined $\Rightarrow$ can't say

Properties

- **Transitivity**

$$f(n) = \Theta(g(n)) \ \& \ g(n) = \Theta(h(n)) \Rightarrow f(n) = \Theta(h(n))$$
$$f(n) = O(g(n)) \ \& \ g(n) = O(h(n)) \Rightarrow f(n) = O(h(n))$$
$$f(n) = \Omega(g(n)) \ \& \ g(n) = \Omega(h(n)) \Rightarrow f(n) = \Omega(h(n))$$
$$f(n) = o\ (g(n)) \ \& \ g(n) = o\ (h(n)) \Rightarrow f(n) = o\ (h(n))$$
$$f(n) = \omega(g(n)) \ \& \ g(n) = \omega(h(n)) \Rightarrow f(n) = \omega(h(n))$$

- **Reflexivity**

$$f(n) = \Theta(f(n))$$
$$f(n) = O(f(n))$$
$$f(n) = \Omega(f(n))$$

Properties

- **Symmetry**

  $f(n) = \Theta(g(n))$ *iff* $g(n) = \Theta(f(n))$

- **Complementarity**

  $f(n) = O(g(n))$ *iff* $g(n) = \Omega(f(n))$

  $f(n) = o(g(n))$ *iff* $g(n) = \omega((f(n))$

Standard Notation and Common Functions

- ***Floors and ceilings***

  For any real number x, the greatest integer less than or equal to x is denoted by $\lfloor x \rfloor$.

  For any real number x, the least integer greater than or equal to x is denoted by $\lceil x \rceil$.

  For all real numbers x,
  $x-1 < \lfloor x \rfloor \leq x \leq \lceil x \rceil < x+1$.

  Both functions are monotonically increasing.

Monotonicity

- $f(n)$ is
  - **monotonically increasing** if $m \leq n \Rightarrow f(m) \leq f(n)$.
  - **monotonically decreasing** if $m \geq n \Rightarrow f(m) \geq f(n)$.
  - **strictly increasing** if $m < n \Rightarrow f(m) < f(n)$.
  - **strictly decreasing** if $m > n \Rightarrow f(m) > f(n)$.

Standard Notation and Common Functions

- ***Exponentials***

  For all n and $a \geq 1$, the function a^n is the exponential function with base a and is monotonically increasing.

- ***Logarithms***

  Textbook adopts the following convention

  $\lg n = \log_2 n$ (binary logarithm),
  $\ln n = \log_e n$ (natural logarithm),
  $\lg^k n = (\lg n)^k$ (exponentiation),
  $\lg \lg n = \lg(\lg n)$ (composition),
  $\lg n + k = (\lg n) + k$ (precedence of lg).

a^i

Exponentials

- **Useful Identities:**

$$a^{-1} = \frac{1}{a}$$

$$(a^m)^n = a^{mn}$$

$$a^m a^n = a^{m+n}$$

- **Exponentials and polynomials**

$$\lim_{n\to\infty} \frac{n^b}{a^n} = 0$$

$$\Rightarrow n^b = o(a^n)$$

Logarithms

$x = \log_b a$ is the exponent for $a = b^x$.

Natural log: $\ln a = \log_e a$

Binary log: $\lg a = \log_2 a$

$\lg^2 a = (\lg a)^2$

$\lg \lg a = \lg (\lg a)$

$$a = b^{\log_b a}$$

$$\log_c (ab) = \log_c a + \log_c b$$

$$\log_b a^n = n \log_b a$$

$$\log_b a = \frac{\log_c a}{\log_c b}$$

$$\log_b (1/a) = -\log_b a$$

$$\log_b a = \frac{1}{\log_a b}$$

$$a^{\log_b c} = c^{\log_b a}$$

Logarithms and exponentials – Bases

- If the base of a logarithm is changed from one constant to another, the value is altered by a constant factor.
  - **Ex:** $\log_{10} n * \mathbf{\log_2 10} = \log_2 n$.
  - Base of logarithm is not an issue in asymptotic notation.
- Exponentials with different bases differ by a exponential factor (not a constant factor).
  - **Ex:** $2^n = \mathbf{(2/3)^n} * 3^n$.

Polylogarithms

- **For $a \geq 0$, $b > 0$,** $\lim_{n \to \infty} (\lg^a n / n^b) = 0$, so $\lg^a n = o(n^b)$, and $n^b = \omega(\lg^a n)$
  - Prove using L'Hopital's rule repeatedly

- $\lg(n!) = \Theta(n \lg n)$
  - Prove using Stirling's approximation (in the text) for $\lg(n!)$.

Standard Notation and Common Functions

- ***Important relationships***

  For all real constants a and b such that $a>1$,

  $$n^b = o(a^n)$$

  that is, any exponential function with a base strictly greater than unity grows faster than any polynomial function.

  For all real constants a and b such that $a>0$,

  $$\lg^b n = o(n^a)$$

  that is, any positive polynomial function grows faster than any polylogarithmic function.

Exercise

Express functions in A in asymptotic notation using functions in B.

A	B	
$5n^2 + 100n$	$3n^2 + 2$	$A \in \Theta(B)$

$A \in \Theta(n^2)$, $n^2 \in \Theta(B) \Rightarrow A \in \Theta(B)$

$\log_3(n^2)$	$\log_2(n^3)$	$A \in \Theta(B)$

$\log_b a = \log_c a / \log_c b$; $A = 2\lg n / \lg 3$, $B = 3\lg n$, $A/B = 2/(3\lg 3)$

$n^{\lg 4}$	$3^{\lg n}$	$A \in \omega(B)$

$a^{\log b} = b^{\log a}$; $B = 3^{\lg n} = n^{\lg 3}$; $A/B = n^{\lg(4/3)} \to \infty$ as $n \to \infty$

$\lg^2 n$	$n^{1/2}$	$A \in o(B)$

$\lim_{n\to\infty} (\lg^a n / n^b) = 0$ (here $a = 2$ and $b = 1/2$) $\Rightarrow A \in o(B)$

Space Complexity

Space Overhead

- The total storage space can be determined at two different times:-

a. Compile time
   - Determine the storage size for each variable only

b. Run time
   - Determine space required for storing the program **(Code Space)**

   - Determine space required for constants, variables and dynamic memory allocation (if any). **(Data Space)**

   - Determine memory required to save addresses while making nested recursive function calls. **(Stack Space)**

Summations – Review

Review on Summations

- Why do we need summation formulas?

  For computing the running times of iterative constructs (loops).

  Example: Maximum Subvector

  Given an array $A[1\ldots n]$ of numeric values (can be positive, zero, and negative) determine the subvector $A[i\ldots j]$ $(1 \le i \le j \le n)$ whose sum of elements is maximum over all subvectors.

1	-2	2	2

Review on Summations

```
MaxSubvector(A, n)
        maxsum ← 0;
        for i ← 1 to n
            do for j = i to n
                        sum ← 0
                        for k ← i to j
                              do sum += A[k]
                        maxsum ← max(sum, maxsum)
        return maxsum
```

- $T(n) = \sum_{i=1}^{n} \sum_{j=i}^{n} \sum_{k=i}^{j} 1$
- NOTE: This is not a simplified solution. What *is* the final answer?

Review on Summations

- **Constant Series:** For integers a and b, $a \le b$,

$$\sum_{i=a}^{b} 1 = b - a + 1$$

- **Linear Series (Arithmetic Series):** For $n \ge 0$,

$$\sum_{i=1}^{n} i = 1 + 2 + \cdots + n = \frac{n(n+1)}{2}$$

- **Quadratic Series:** For $n \ge 0$,

$$\sum_{i=1}^{n} i^2 = 1^2 + 2^2 + \cdots + n^2 = \frac{n(n+1)(2n+1)}{6}$$

Review on Summations

- **Cubic Series:** For $n \geq 0$,

$$\sum_{i=1}^{n} i^3 = 1^3 + 2^3 + \cdots + n^3 = \frac{n^2(n+1)^2}{4}$$

- **Geometric Series:** For real $x \neq 1$,

$$\sum_{k=0}^{n} x^k = 1 + x + x^2 + \cdots + x^n = \frac{x^{n+1} - 1}{x - 1}$$

For $|x| < 1$,

$$\sum_{k=0}^{\infty} x^k = \frac{1}{1 - x}$$

Review on Summations

- **Linear-Geometric Series:** For $n \geq 0$, real $c \neq 1$,

$$\sum_{i=1}^{n} ic^i = c + 2c^2 + \cdots + nc^n = \frac{-(n+1)c^{n+1} + nc^{n+2} + c}{(c-1)^2}$$

- **Harmonic Series:** nth harmonic number, $n \in \mathrm{I}^+$,

$$H_n = 1 + \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{n}$$

$$= \sum_{k=1}^{n} \frac{1}{k} = \ln(n) + O(1)$$

Review on Summations

- **Telescoping Series:**

$$\sum_{k=1}^{n} a_k - a_{k-1} = a_n - a_0$$

- **Differentiating Series:** For $|x| < 1$,

$$\sum_{k=0}^{\infty} kx^k = \frac{x}{(1-x)^2}$$

Review on Summations

- **Approximation by integrals:**
  - For monotonically increasing $f(n)$

$$\int_{m-1}^{n} f(x)dx \le \sum_{k=m}^{n} f(k) \le \int_{m}^{n+1} f(x)dx$$

  - For monotonically decreasing $f(n)$

$$\int_{m}^{n+1} f(x)dx \le \sum_{k=m}^{n} f(k) \le \int_{m-1}^{n} f(x)dx$$

- **How?**

Review on Summations

- ***n*th harmonic number**

$$\sum_{k=1}^{n} \frac{1}{k} \geq \int_{1}^{n+1} \frac{dx}{x} = \ln(n+1)$$

$$\sum_{k=2}^{n} \frac{1}{k} \leq \int_{1}^{n} \frac{dx}{x} = \ln n$$

$$\Rightarrow \sum_{k=1}^{n} \frac{1}{k} \leq \ln n + 1$$