ADVANCED DATA STRUCTURES AND ALGORITHMS

(Theory and Practice)

Course Code:	22MCE12TL		**CIE Marks: 100**
Hrs/week	L:T:P 3:0:1 42L + 28P		**SEE Marks: 100**
Credits:	**4**		**SEE: 3 Hrs**

UNIT-1

Algorithmic Complexity Measures Methods for expressing and comparing complexity of algorithms: worst and average cases, lower bounds, and asymptotic analysis. **Abstract Data Type (ADT)** Specification and Design techniques

Elementary ADTs: Lists, Trees, Stacks, Queues, and Dynamic Sets. Sorting in Linear Time Lower bounds for sorting, Radix sort and Bucket sort

Asymptotic Analysis

Asymptotic analysis is ***the process of calculating the running time of an algorithm in mathematical units to find the program's limitations, or "run-time performance."***

The goal is to determine the best case, worst case and average case time required to execute a given task.

Asymptotic Notations ***allow us to analyze an algorithm's running time by identifying its behavior as the input size for the algorithm increases***

Upper bound & Lower bound

- **Upper bound:** The **maximum time** a program can take to produce outputs, expressed in terms of the size of the inputs (**worst-case** scenario).
- **Upper bound - the maximum guaranteed time**

- **Lower bound:** The **minimum time** a program will take to produce outputs, expressed in terms of the size of the inputs (**best-case scenario**).
- **Lower bound - the minimum guaranteed time.**

Asymptotic notation

Precisely calculating the actual steps is tedious and not generally useful

Different operations take different amounts of time. Even from run to run, things such as caching, etc. cause variations

We want to identify **categories** of algorithmic runtimes

For example…

$f_1(n)$ takes n^2 steps

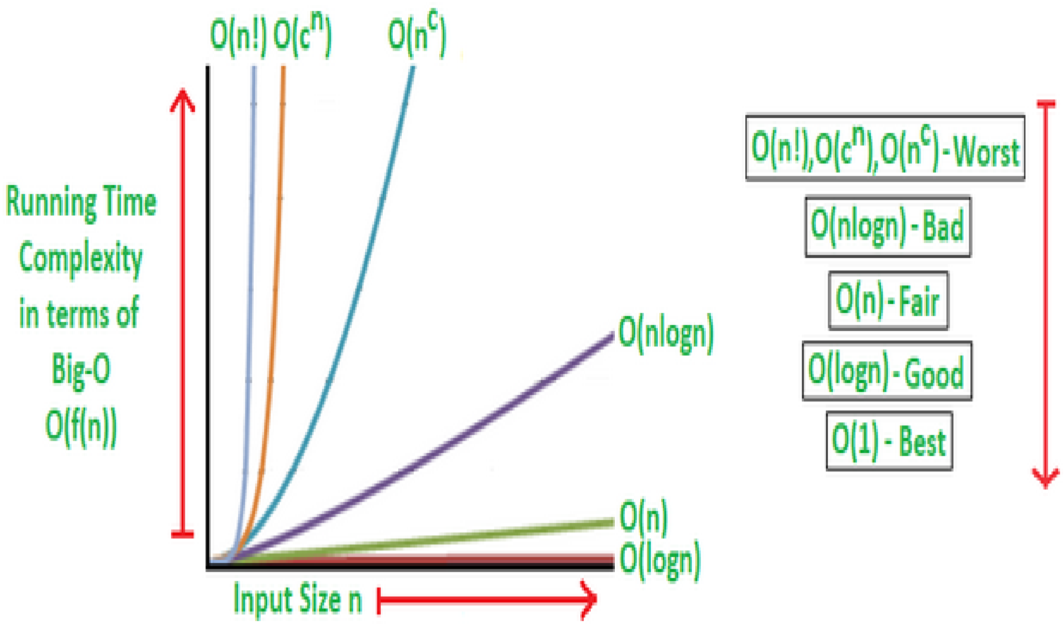
$f_2(n)$ takes $2n + 100$ steps

$f_3(n)$ takes $3n+1$ steps

Which algorithm is better?

Is the difference between f_2 and f_3 important/significant?

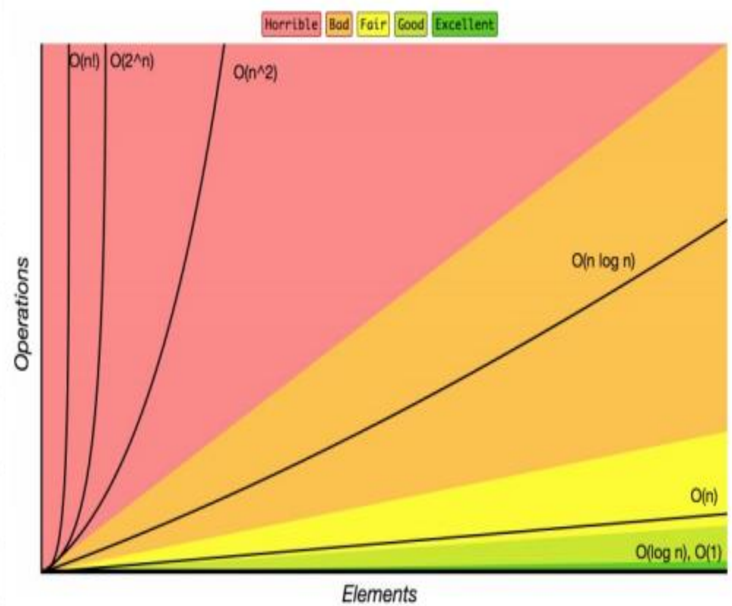
Runtime examples



notation	name
$O(1)$	constant
$O(\log(n))$	logarithmic
$O((\log(n))^c)$	polylogarithmic
$O(n)$	linear
$O(n^2)$	quadratic
$O(n^c)$	polynomial
$O(c^n)$	exponential

complexity class: A category of algorithm efficiency based on the algorithm's relationship to the input size N.

Complexity Class	Big-O	Runtime if you double N	Example Algorithm
constant	$O(1)$	unchanged	Accessing an index of an array
logarithmic	$O(\log_2 N)$	increases slightly	Binary search
linear	$O(N)$	doubles	Looping over an array
log-linear	$O(N \log_2 N)$	slightly more than doubles	Merge sort algorithm
quadratic	$O(N^2)$	quadruples	Nested loops!
...	...	...	...
exponential	$O(2^N)$	multiplies drastically	Fibonacci with recursion



Big O: Upper bound

O(g(n)) is the set of functions:

$$O(g(n)) = \left\{ f(n) : \begin{array}{l} \text{there exists positive constants } c \text{ and } n_0 \text{ such that} \\ 0 \le f(n) \le cg(n) \text{ for all } n \ge n_0 \end{array} \right\}$$

Big O: Upper bound

O(g(n)) is the set of functions:

$$O(g(n)) = \left\{ f(n) : \begin{array}{l} \text{there exists positive constants } c \text{ and } n_0 \text{ such that} \\ 0 \le f(n) \le cg(n) \text{ for all } n \ge n_0 \end{array} \right\}$$

We can bound the function f(n) above by some constant factor of g(n)

Big O: Upper bound

O(g(n)) is the set of functions:

$$O(g(n)) = \left\{ f(n): \begin{array}{l} \text{there exists positive constants } c \text{ and } n_0 \text{ such that} \\ 0 \le f(n) \le cg(n) \text{ for all } n \ge n_0 \end{array} \right\}$$

We can bound the function f(n) above by some constant multiplied by g(n)

For some increasing range

Big O: Upper bound

O(g(n)) is the set of functions:

$$O(g(n)) = \left\{ f(n) : \begin{array}{l} \text{there exists positive constants } c \text{ and } n_0 \text{ such that} \\ 0 \le f(n) \le cg(n) \text{ for all } n \ge n_0 \end{array} \right\}$$

$$O(n^2) = \begin{array}{rcl} f_1(x) & = & 3n^2 \\ f_2(x) & = & 1/2n^2 + 100 \\ f_3(x) & = & n^2 + 5n + 40 \end{array}$$

Big O: Upper bound

O(g(n)) is the set of functions:

$$O(g(n)) = \left\{ f(n) : \begin{array}{l} \text{there exists positive constants } c \text{ and } n_0 \text{ such that} \\ 0 \le f(n) \le cg(n) \text{ for all } n \ge n_0 \end{array} \right\}$$

Generally, we're most interested in big O notation since it is an upper bound on the running time

Omega: Lower bound

Ω(g(n)) is the set of functions:

$$\Omega(g(n)) = \left\{ f(n) : \begin{array}{l} \text{there exists positive constants } c \text{ and } n_0 \text{ such that} \\ 0 \leq cg(n) \leq f(n) \text{ for all } n \geq n_0 \end{array} \right\}$$

Omega: Lower bound

$\Omega(g(n))$ is the set of functions:

$$\Omega(g(n)) = \left\{ f(n) : \begin{array}{l} \text{there exists positive constants } c \text{ and } n_0 \text{ such that} \\ 0 \le cg(n) \le f(n) \text{ for all } n \ge n_0 \end{array} \right\}$$

We can bound the function f(n) below by some constant factor of g(n)

Omega: Lower bound

Ω(g(n)) is the set of functions:

$$\mathrm{W}(g(n)) = \left\{ f(n) : \begin{array}{l} \text{there exists positive constants } c \text{ and } n_0 \text{ such that} \\ 0 \leq cg(n) \leq f(n) \text{ for all } n \geq n_0 \end{array} \right\}$$

$$\Omega(n^2) = \begin{array}{rcl} f_1(x) & = & 3n^2 \\ f_2(x) & = & 1/2n^2 + 100 \\ f_3(x) & = & n^2 + 5n + 40 \end{array}$$

Theta: Upper and lower bound

Θ(g(n)) is the set of functions:

$$\Theta(g(n)) = \left\{ f(n) : \begin{array}{l} \text{there exists positive constants } c_1, c_2 \text{ and } n_0 \text{ such that} \\ 0 \le c_1 g(n) \le f(n) \le c_2 g(n) \text{ for all } n \ge n_0 \end{array} \right\}$$

Theta: Upper and lower bound

Θ(g(n)) is the set of functions:

$$\Theta(g(n)) = \left\{ f(n) : \begin{array}{l} \text{there exists positive constants } c_1, c_2 \text{ and } n_0 \text{ such that} \\ 0 \le c_1 g(n) \le f(n) \le c_2 g(n) \text{ for all } n \ge n_0 \end{array} \right\}$$

*We can bound the function f(n) above **and** below by some constant factor of g(n) (though different constants)*

Theta: Upper and lower bound

Θ(g(n)) is the set of functions:

$$\Theta(g(n)) = \left\{ f(n) : \begin{array}{l} \text{there exists positive constants } c_1, c_2 \text{ and } n_0 \text{ such that} \\ 0 \le c_1 g(n) \le f(n) \le c_2 g(n) \text{ for all } n \ge n_0 \end{array} \right\}$$

Note: A function is theta bounded ***iff*** *it is big O bounded and Omega bounded*

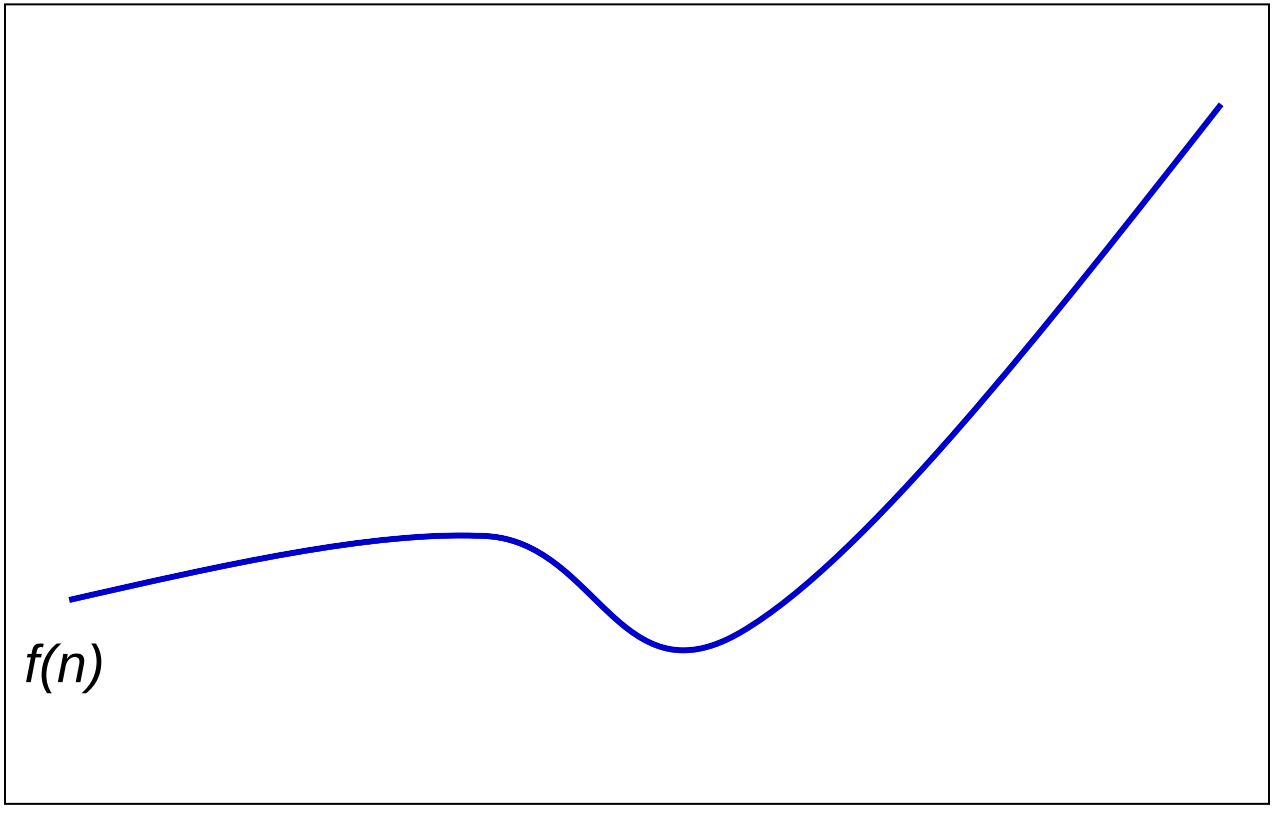
Theta: Upper and lower bound

Θ(g(n)) is the set of functions:

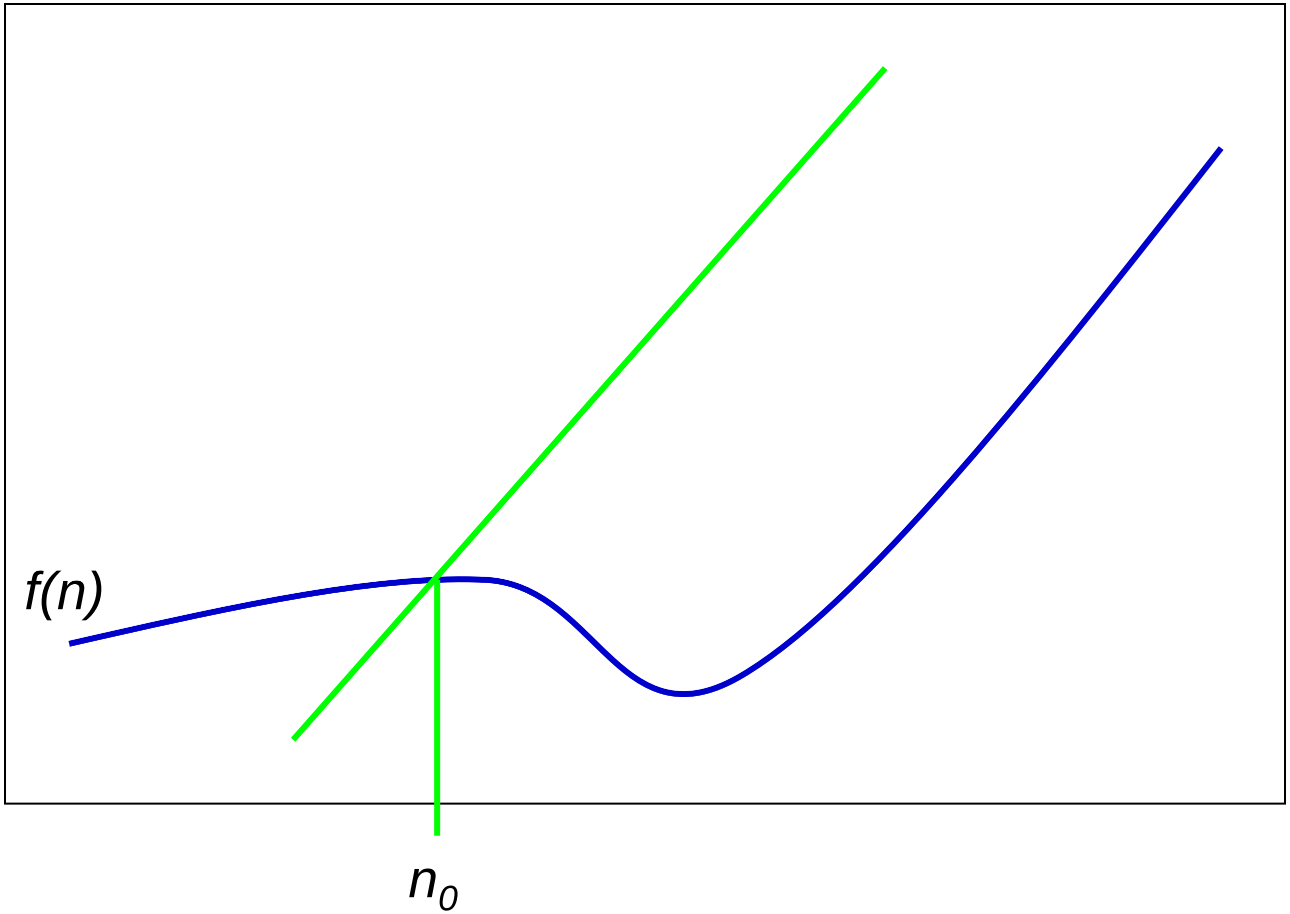
$$\Theta(g(n)) = \left\{ f(n) : \begin{array}{l} \text{there exists positive constants } c_1, c_2 \text{ and } n_0 \text{ such that} \\ 0 \le c_1 g(n) \le f(n) \le c_2 g(n) \text{ for all } n \ge n_0 \end{array} \right\}$$

$$\Theta(n^2) = \begin{array}{lcl} f_1(x) & = & 3n^2 \\ f_2(x) & = & 1/2n^2 + 100 \\ f_3(x) & = & n^2 + 5n + 40 \\ f_4(x) & = & 3n^2 + n\log n \end{array}$$

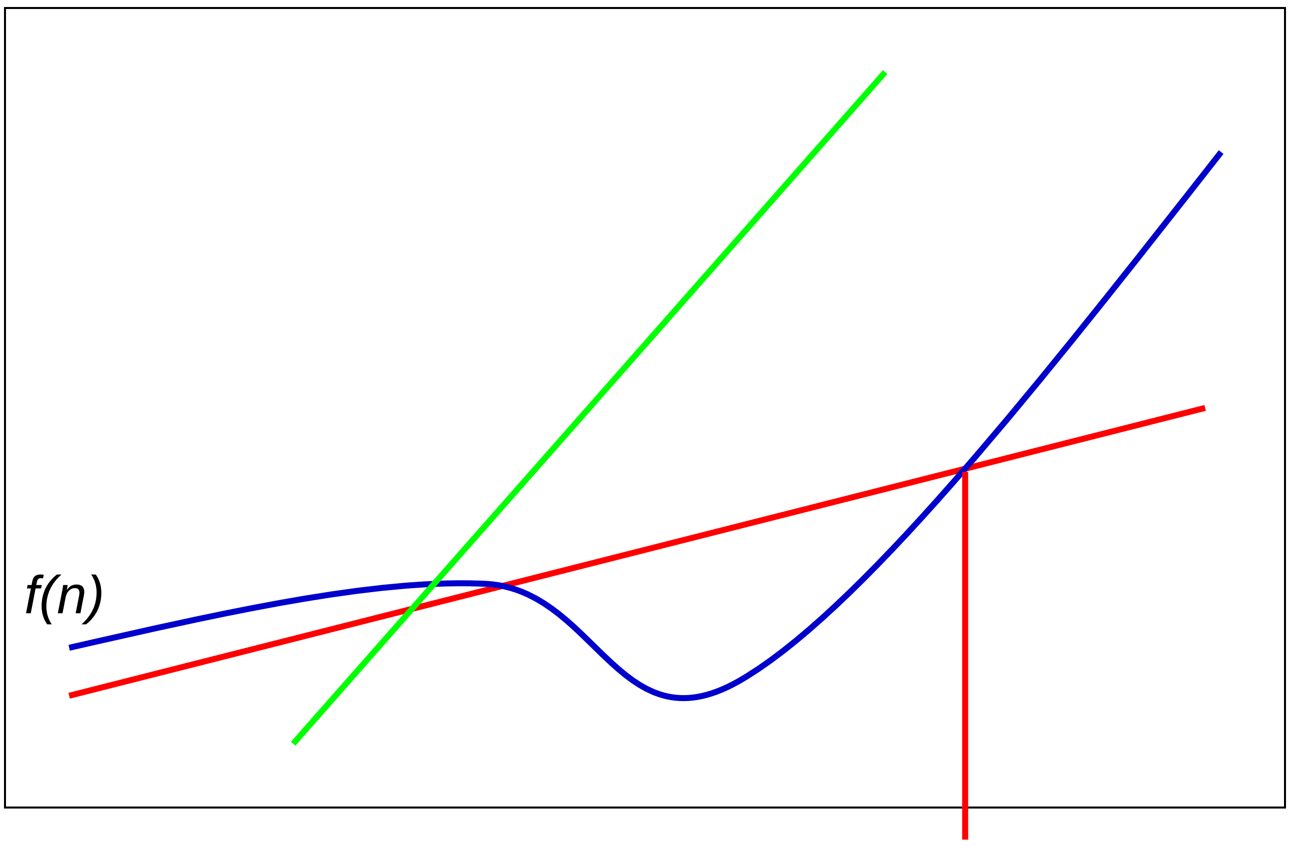
Visually



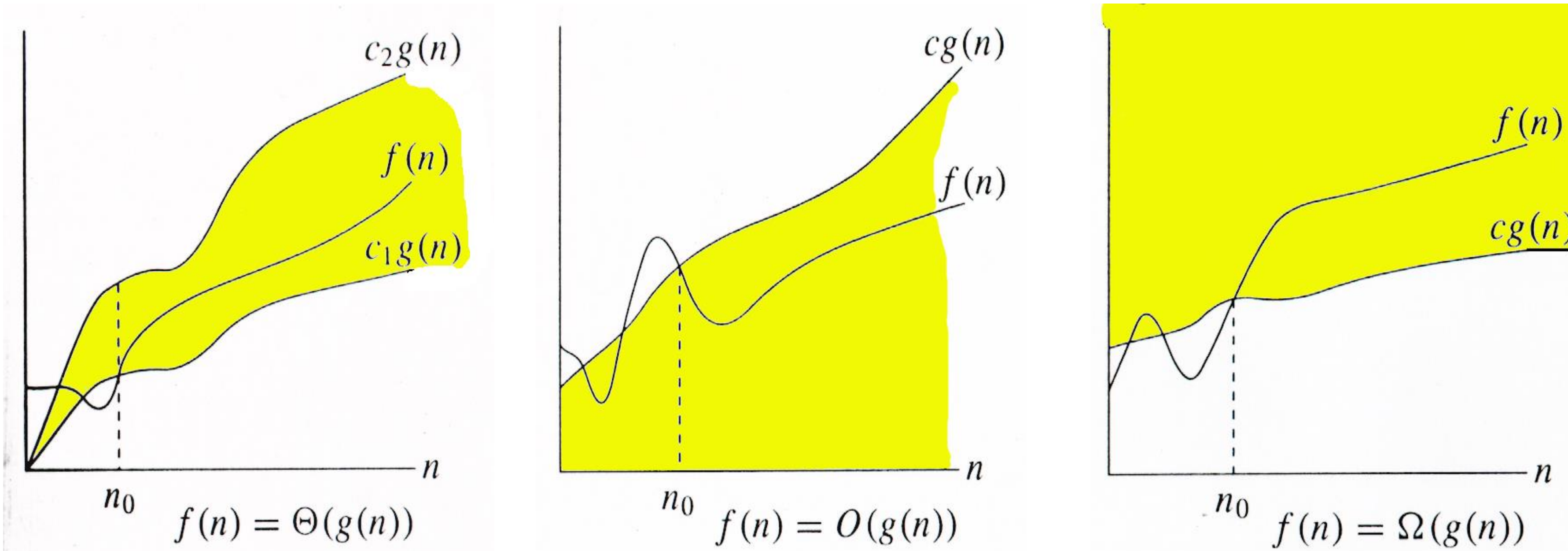
Visually: upper bound



Visually: lower bound



Relations Between Θ, *O*, Ω



Worst-case vs. Best-case vs. Average-case

worst-case: what is the worst the running time of the algorithm can be?

best-case: what is the best the running time of the algorithm can be?

average-case: given random data, what is the running time of the algorithm?

Don't confuse this with O, Ω and Θ. The cases above are *situations*, asymptotic notation is about bounding particular situations

Proving bounds: find constants that satisfy inequalities

Show that $5n^2 - 15n + 100$ is $\Theta(n^2)$

Step 1: Prove $O(n^2)$ – Find constants c and n_0 such that
$5n^2 - 15n + 100 \leq cn^2$ for all n > n_0

$$cn^2 \geq 5n^2 - 15n + 100$$

$$c \geq 5 - 15/n + 100/n^2$$

Let n_0 =1 and c = 5 + 100 = 105.
100/n^2 only get smaller as n increases and we ignore -15/n since it only varies between -15 and 0

Proving bounds

Step 2: Prove $\Omega(n^2)$ – Find constants c and n_0 such that $5n^2 - 15n + 100 \geq cn^2$ for all n > n_0

$$cn^2 \leq 5n^2 - 15n + 100$$

$$c \leq 5 - 15/n + 100/n^2$$

Let n_0 =4 and c = 5 – 15/4 = 1.25 (or anything less than 1.25). 15/n is always decreasing and we ignore $100/n^2$ since it is always between 0 and 100.

Bounds

Is $5n^2 \ O(n)$? ***No***

How would we prove it?

$$O(g(n)) = \left\{ f(n): \begin{array}{l} \text{there exists positive constants } c \text{ and } n_0 \text{ such that} \\ 0 \le f(n) \le cg(n) \text{ for all } n \ge n_0 \end{array} \right\}$$

Disproving bounds

Is $5n^2 \;\; O(n)$?

$$O(g(n)) = \left\{ f(n) : \begin{array}{l} \text{there exists positive constants } c \text{ and } n_0 \text{ such that} \\ 0 \le f(n) \le cg(n) \text{ for all } n \ge n_0 \end{array} \right\}$$

***Assume it's true*.**

That means there exists some c and n_0 such that

$$5n^2 \le cn \text{ for } n > n_0$$

$$5n \le c$$ contradiction!

Some rules of thumb

Multiplicative constants can be omitted

- $14n^2$ becomes n^2
- 7 log n become log n

Lower order functions can be omitted

- $n + 5$ becomes n
- $n^2 + n$ becomes n^2

n^a dominates n^b if $a > b$

- n^2 dominates n, so n^2+n becomes n^2
- $n^{1.5}$ dominates $n^{1.4}$

Some rules of thumb

a^n dominates b^n if $a > b$

- 3^n dominates 2^n

Any exponential dominates any polynomial

- 3^n dominates n^5
- 2^n dominates n^c

Any polynomial dominates any logarithm

- n dominates $\log n$ or $\log \log n$
- n^2 dominates $n \log n$
- $n^{1/2}$ dominates $\log n$

Do **not** omit lower order terms of different variables ($n^2 + m$) does not become n^2

Big O

$n^2 + n \log n + 50$

$2^n - 15n^2 + n^3 \log n$

$n^{\log n} + n^2 + 15n^3$

$n^5 + n! + n^n$

Some examples

- O(1) – constant. Fixed amount of work, regardless of the input size
  - add two 32 bit numbers
  - determine if a number is even or odd
  - sum the first 20 elements of an array
  - delete an element from a doubly linked list
- O(log *n*) – logarithmic. At each iteration, discards some portion of the input (i.e. half)
  - binary search

Some examples

- $O(n)$ – linear. Do a constant amount of work on each element of the input
  - find an item in a linked list
  - determine the largest element in an array
- $O(n \log n)$ log-linear. Divide and conquer algorithms with a linear amount of work to recombine
  - Sort a list of number with MergeSort
  - FFT

Some examples

- $O(n^2)$ – quadratic. Double nested loops that iterate over the data
  - Insertion sort
- $O(2^n)$ – exponential
  - Enumerate all possible subsets
  - Traveling salesman using dynamic programming
- $O(n!)$
  - Enumerate all permutations
  - determinant of a matrix with expansion by minors

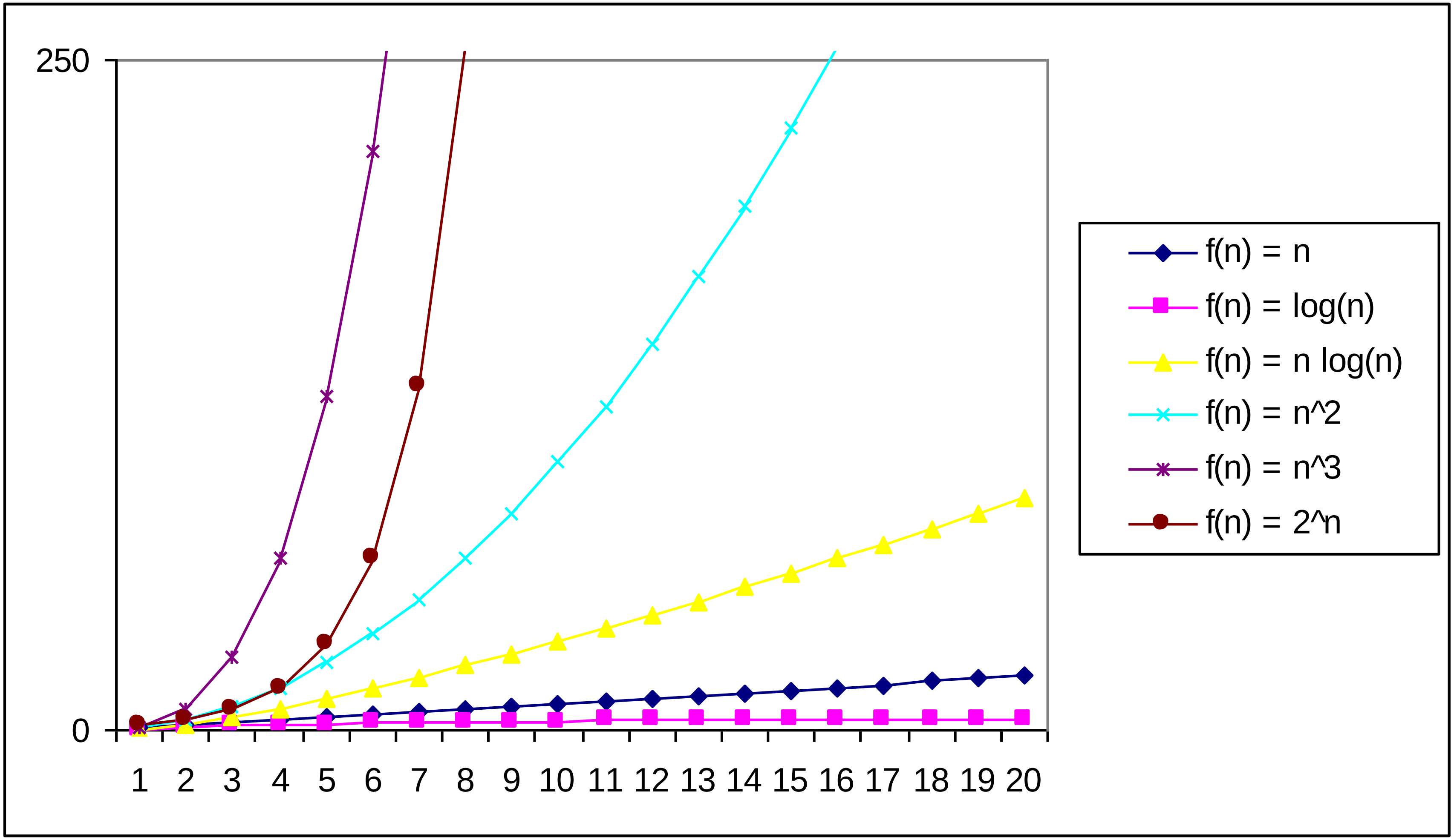
A Common Misunderstanding

Confusing worst case with upper bound.

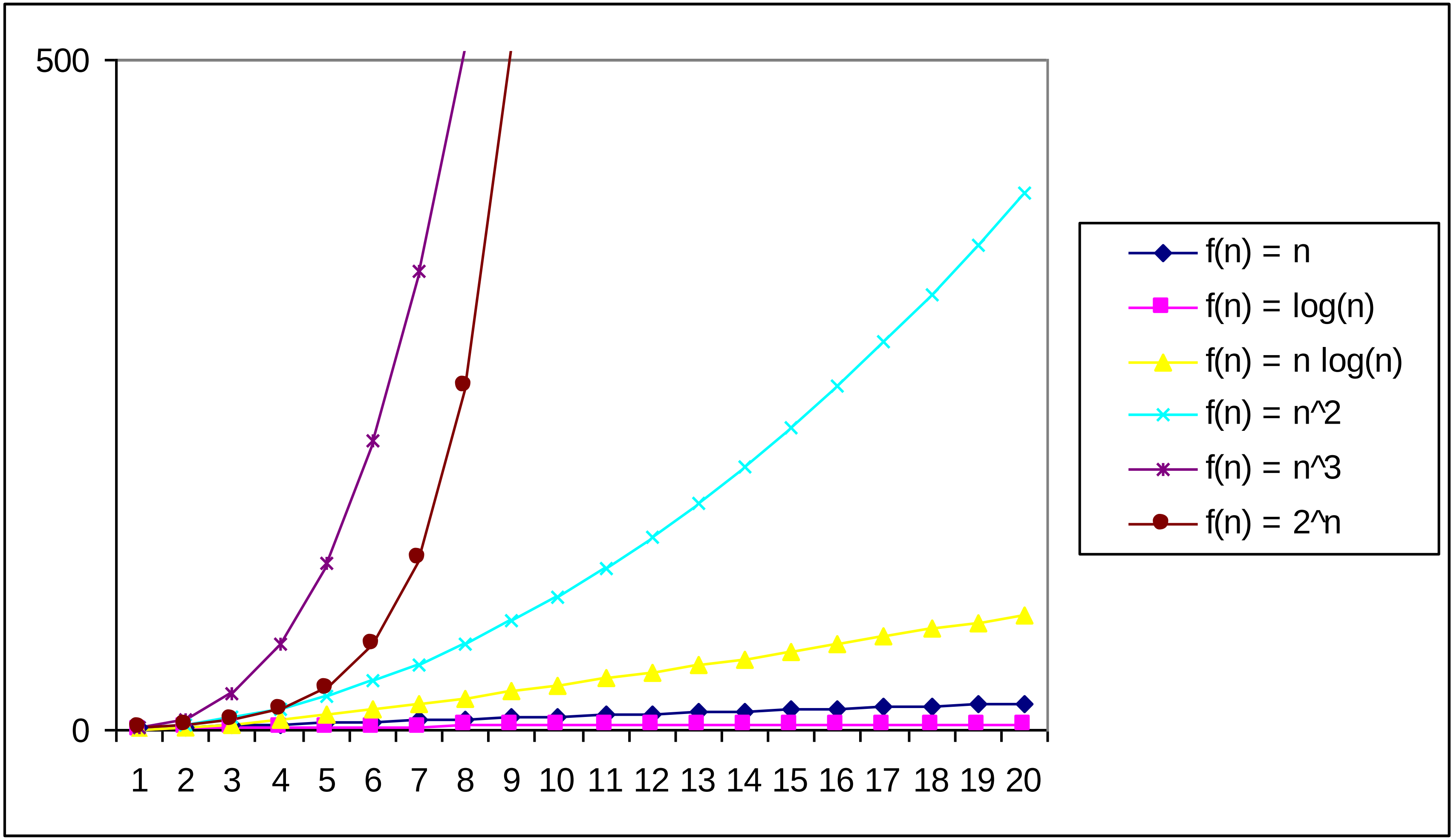
Upper bound refers to a growth rate.

Worst case refers to the worst input from among the choices for possible inputs of a given size.

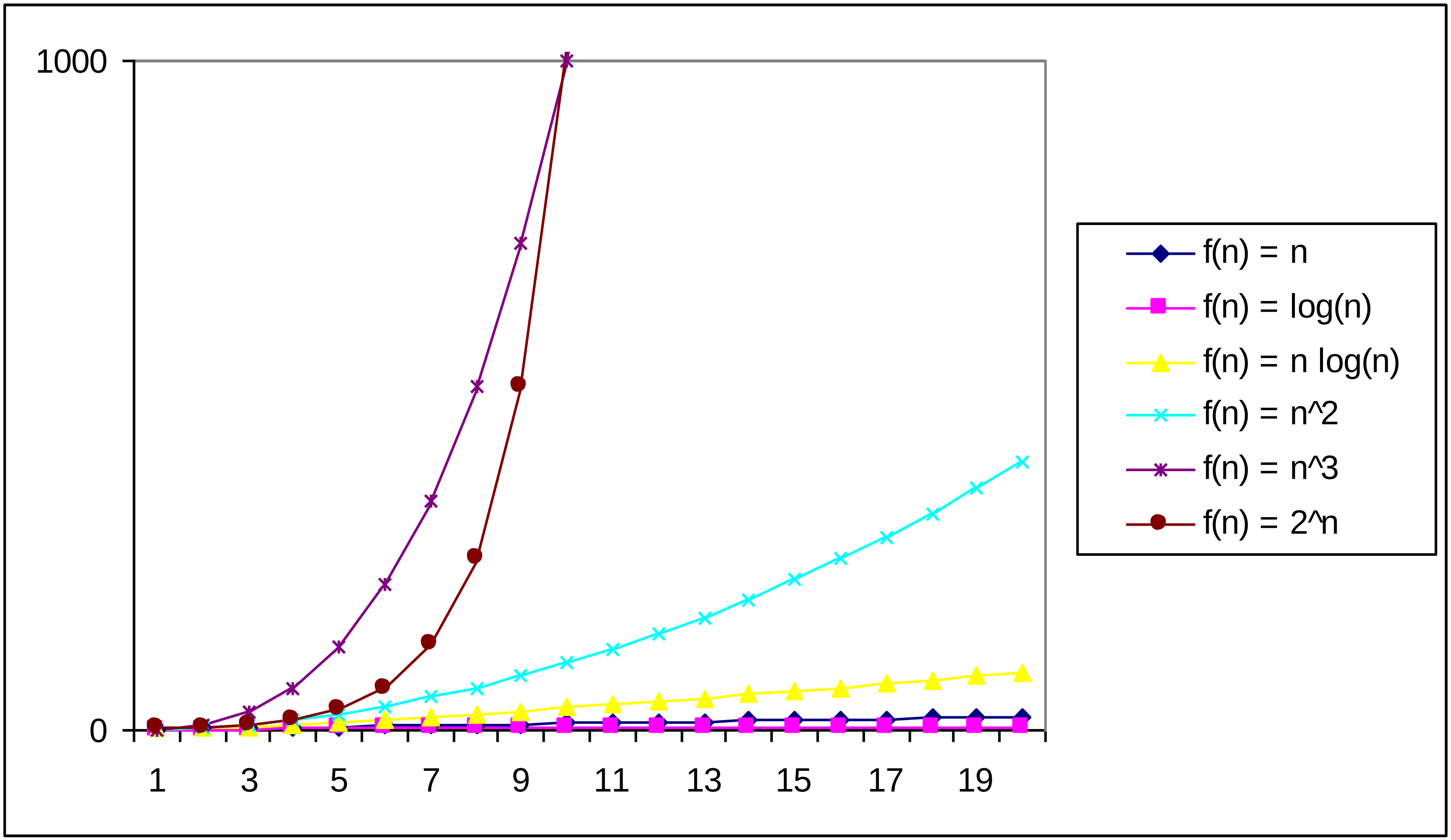
Practical Complexity



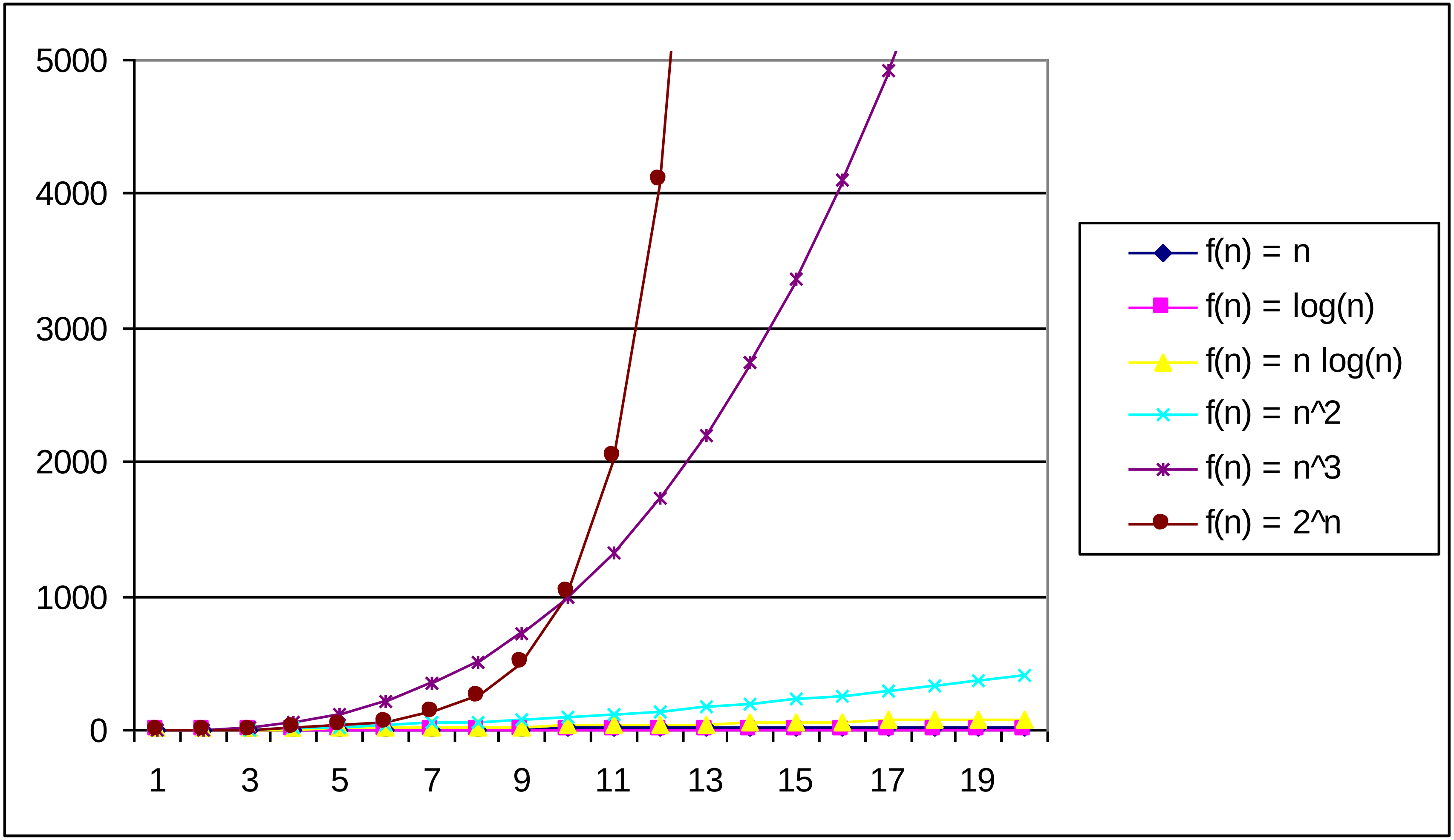
Practical Complexity



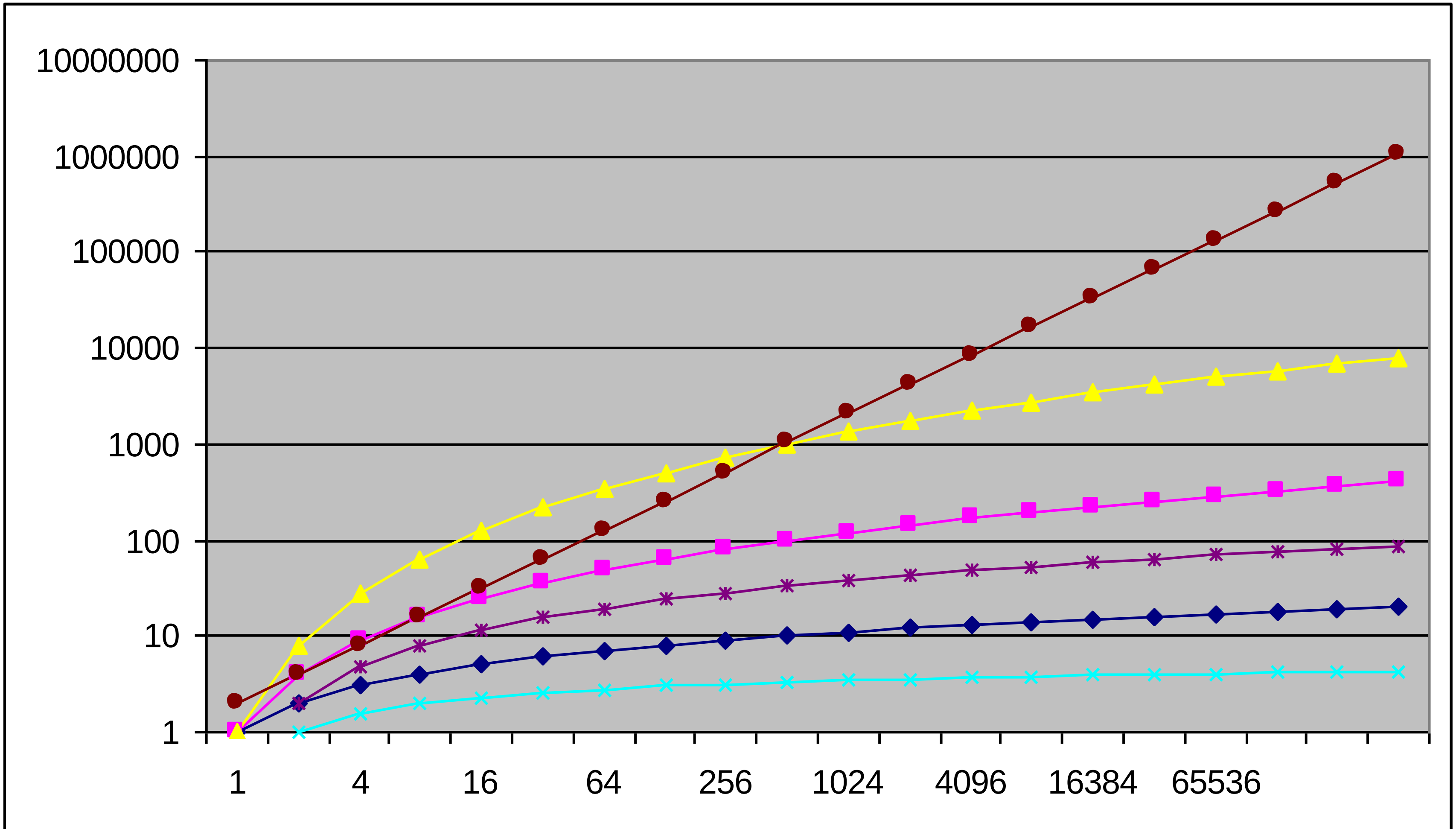
Practical Complexity



Practical Complexity



Practical Complexity



Comparison of Functions

$$f \leftrightarrow g \approx a \leftrightarrow b$$

$$f(n) = O(g(n)) \approx a \leq b$$

$$f(n) = \Omega(g(n)) \approx a \geq b$$

$$f(n) = \Theta(g(n)) \approx a = b$$

$$f(n) = o(g(n)) \approx a < b$$

$$f(n) = \omega(g(n)) \approx a > b$$

Summations – Review

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 $|x| < 1$, $\displaystyle \sum_{k=0}^{\infty} x^k = \frac{1}{1-x}$

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 $|x| < 1$,

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

Summation

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

$$\sum_{i=1}^{n} \frac{1}{2^i} = 1 - \frac{1}{2^n},$$

$$\sum_{i=0}^{n} 2^i = 2^{n+1} - 1.$$

$$\sum_{i=1}^{n} \frac{i}{2^i} = 2 - \frac{n+2}{2^n}.$$