Growth of Function

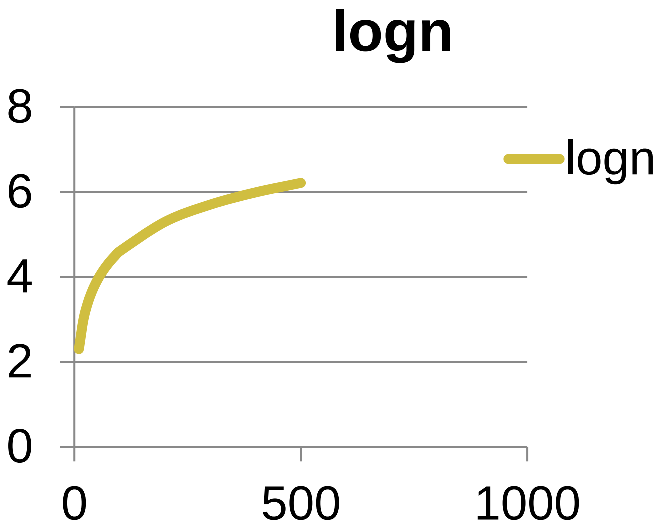
Analysis of Algorithm

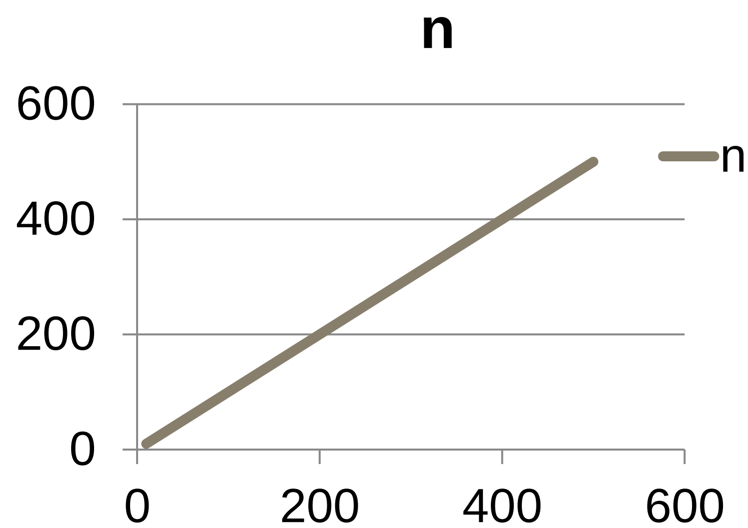
Faculty of Computing and Information Technology (FCIT)
University of the Punjab, Lahore, Pakistan.

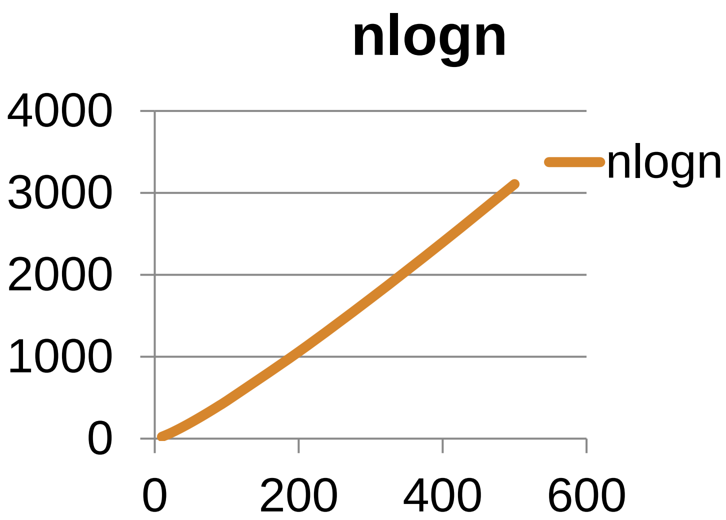
Credit

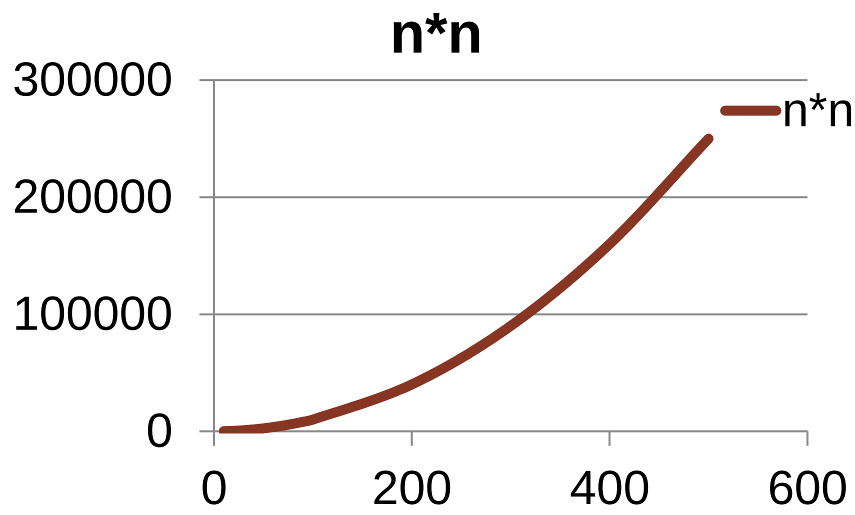
- These notes contain material from Chapter 3 of Cormen, Leiserson, Rivest, and Stein (3rd Edition).
- "*Design and Analysis Part 1*" by Tim Roughgarden, Stanford University, available at coursera.
- "Algorithms Part 1" by Kevin Wayne and Robert Sedgewick, Princeton University, available at coursera.

Growth of Function









Cost of Basic Operations

operation	example	nanoseconds †
variable declaration	`int a`	c_1
assignment statement	`a = b`	c_2
integer compare	`a < b`	c_3
array element access	`a[i]`	c_4
array length	`a.length`	c_5
1D array allocation	`new int[N]`	$c_6 N$
2D array allocation	`new int[N][N]`	$c_7 N^2$
string length	`s.length()`	c_8
substring extraction	`s.substring(N/2, N)`	c_9
string concatenation	`s + t`	$c_{10} N$

Asymptotic Analysis

High Level Idea

- Suppress constant factors and lower-order terms
  - Constant factors; too system dependent
  - Lower-order terms: irrelevant for large inputs

- Example: an^2+bn+c is just $O(n^2)$

Asymptotic Analysis (Cont.)

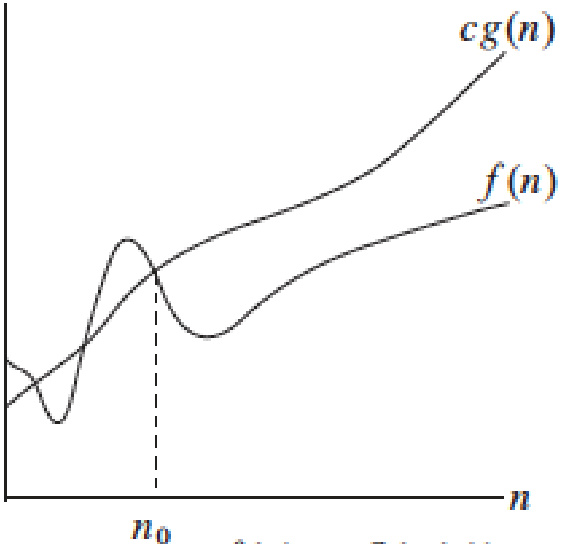
- Lets assume an algorithm can be represented as f(n); where n is the input size. We need to calculate the running time of f(n).

- We define another function lets call it g(n) which represents the running time of the algorithm.

- Now three inequalities are possible
  1. $f(n) < g(n)$
  2. $f(n) > g(n)$
  3. $f(n) = g(n)$

Big-Oh (O)

- We use Big-Oh to represent the worst case running time of an algorithm.
- Upper bound of f(n)
- We define Big-Oh as:



$f(n) = O(g(n))$

$$O(g(n)) = \{ f(n) : \text{there exist positive constants } c \text{ and } n_0 \text{ such that } 0 \le f(n) \le cg(n) \text{ for all } n \ge n_0 \} .$$

Big-Oh (O)

Let $f, g : \mathbb{N} \longrightarrow \mathbb{R}^+$ be functions. Define the set

$$O(g(n)) \;\; := \;\; \{\, f : \mathbb{N} \longrightarrow \mathbb{R}^+ : \exists n_0 \in \mathbb{N}^+ \,.\, \exists c \in \mathbb{R}^+ \,.\, \forall n \,. \\ n \geq n_0 \;\rightarrow\; f(n) \leq c \cdot g(n) \,\}$$

In words, $f \in O(g)$ if there exist a positive integer n_0 and a positive real c such that $f(n) \leq c \cdot g(n)$ for all $n \geq n_0$.

Informally $O(g)$ is the set of functions that are bounded above by g, ignoring constant factors, and ignoring a finite number of exceptions.

If $f \in O(g)$, then we say that "g is an ***asymptotic upper bound*** for f"

Big-Oh (O)

$$O(g(n)) \quad := \quad \{\, f : \mathbb{N} \longrightarrow \mathbb{R}^+ : \exists n_0 \in \mathbb{N}^+ \,.\, \exists c \in \mathbb{R}^+ \,.\, \forall n \,.\, n \geq n_0 \;\rightarrow\; f(n) \leq c \cdot g(n) \,\}$$

1. $3^{98} \in O(1)$ [regarding 3^{98} and 1 as (constant) functions of n].
   Take $n_0 = 1$ and $c = 3^{98}$.
2. $5n^2 + 9 \in O(n^2)$.
   Take $n_0 = 3$ and $c = 6$. Then for for all $n \geq n_0$, we have $9 \leq n^2$, and so $5n^2 + 9 \leq 5n^2 + n^2 = 6n^2 = cn^2$.
3. Take $g(n) = n^2$ and $f(n) = 7n^2 + 3n + 11$. Then $f \in O(g)$.
4. Some more functions in $O(n^2)$:
   $1000n^2$, n, $n^{1.9999}$, $n^2 / \lg\lg\lg n$ and 6.

Big-Oh (O)

Lemma 1. *Let* $f, g, h : \mathbb{N} \longrightarrow \mathbb{R}^+$. *Then:*

1. *For every constant* $c > 0$, *if* $f \in O(g)$ *then* $c\,f \in O(g)$.
2. *For every constant* $c > 0$, *if* $f \in O(g)$ *then* $f \in O(c\,g)$.
3. *If* $f_1 \in O(g_1)$ *and* $f_2 \in O(g_2)$ *then* $f_1 + f_2 \in O(g_1 + g_2)$.
4. *If* $f_1 \in O(g_1)$ *and* $f_2 \in O(g_2)$ *then* $f_1 + f_2 \in O(\max(g_1, g_2))$.
5. *If* $f_1 \in O(g_1)$ *and* $f_2 \in O(g_2)$ *then* $f_1 \cdot f_2 \in O(g_1 \cdot g_2)$.
6. *If* $f \in O(g)$ *and* $g \in O(h)$ *then* $f \in O(h)$.
7. *Every polynomial of degree* $l \geq 0$ *is in* $O(n^l)$.
8. *For any* $c > 0$ *in* $\mathbb{R}$, *we have* $\lg(n^c) \in O(\lg(n))$.
9. *For every constant* $c, d > 0$, *we have* $\lg^c(n) \in O(n^d)$.
10. *For every constant* $c > 0$ *and* $d > 1$, *we have* $n^c \in O(d^n)$.
11. *For every constant* $0 \leq c \leq d$, *we have* $n^c \in O(n^d)$.

Big-Oh (O)

Example. Show that

$$57n^3 + 4n^2 \cdot \lg^5(n) + 17n + 498 \in O(n^3)$$

by appealing to Lemma 1.

$$\begin{array}{rcll}
\lg^5(n) & \in & O(n) & \because 9 \\
4n^2 \cdot \lg^5(n) & \in & O(4n^3) & \because 5 \\
57n^3 + 4n^2 \cdot \lg^5(n) + 17n + 498 & \in & O(57n^3 + 4n^3 + 17n + 498) & \because 3 \\
57n^3 + 4n^3 + 17n + 498 & \in & O(n^3) & \because 7 \\
57n^3 + 4n^2 \cdot \lg^5(n) + 17n + 498 & \in & O(n^3) & \because 6
\end{array}$$

Big-Oh (O)

Example: $n^2 + n = O(n^3)$

Proof:

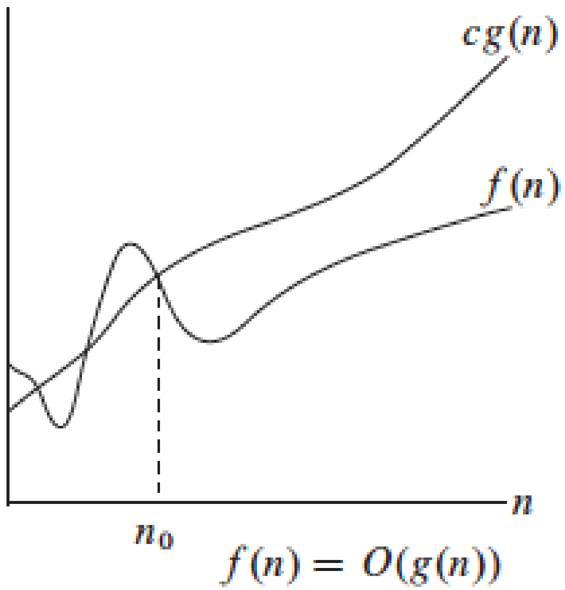
- Here, we have $f(n) = n^2 + n$, and $g(n) = n^3$
- Notice that if $n \geq 1$, $n \leq n^3$ is clear.
- Also, notice that if $n \geq 1$, $n^2 \leq n^3$ is clear.
- **Side Note:** In general, if $a \leq b$, then $n^a \leq n^b$ whenever $n \geq 1$. This fact is used often in these types of proofs.
- Therefore,

$$n^2 + n \leq n^3 + n^3 = 2n^3$$

- We have just shown that

$$n^2 + n \leq 2n^3 \text{ for all } n \geq 1$$

- Thus, we have shown that $n^2 + n = O(n^3)$ (by definition of Big-O, with $n_0 = 1$, and $c = 2$.)



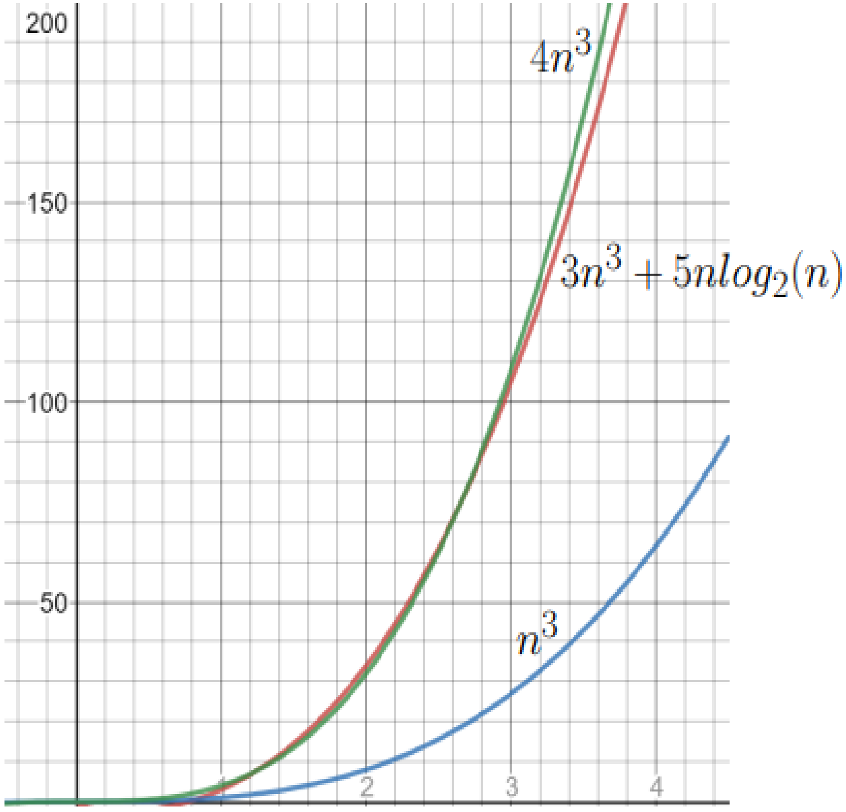
$O(g(n)) = \{f(n) :$ there exist positive constants c and n_0 such that $0 \leq f(n) \leq cg(n)$ for all $n \geq n_0\}$.

Lets also verify this example for O(n^2)

Big-Oh (O)

$$3n^3 + 5n\log n = O(n^3)$$

▷ $c = 4$ and $n_0 = 3$

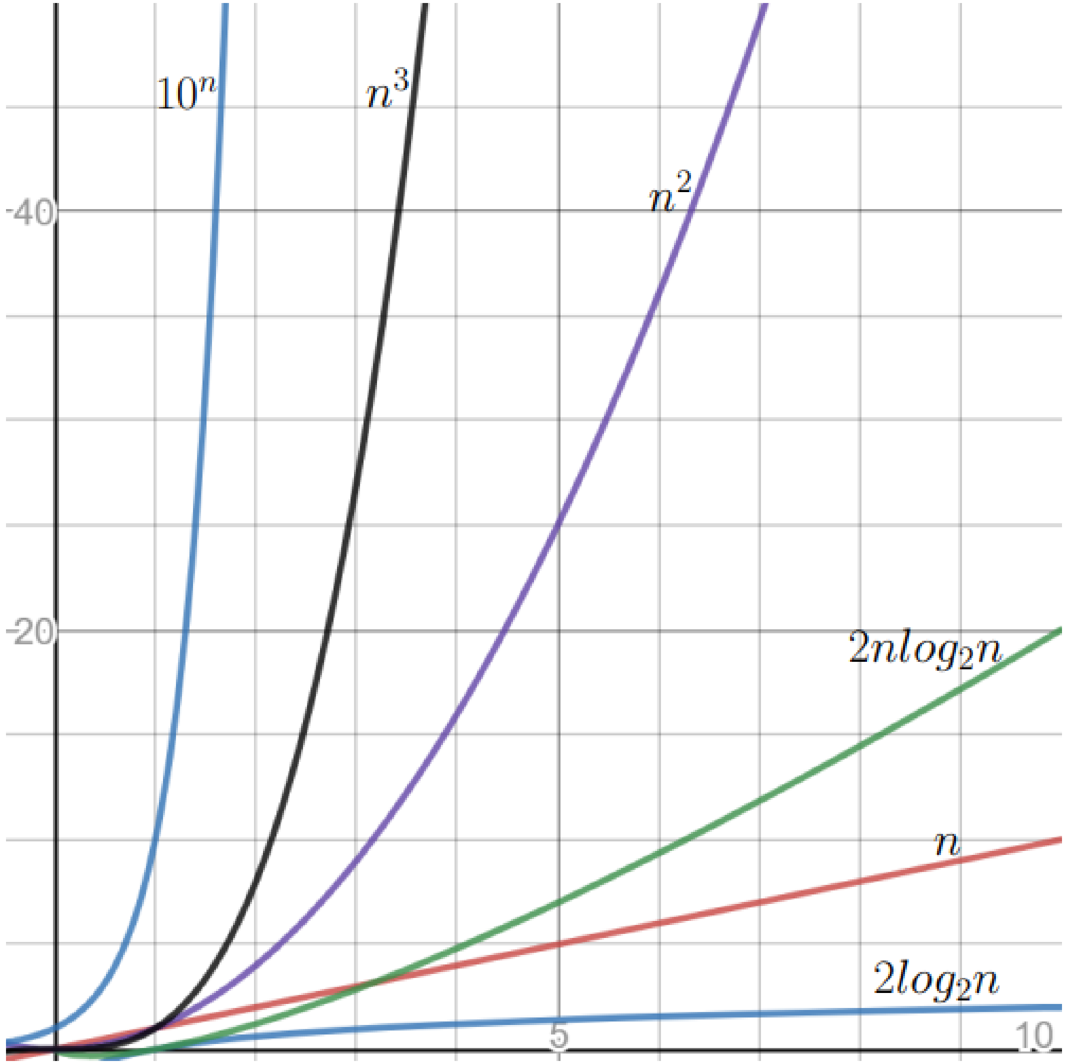


Asymptotic-Complexity Classes

Class Name	**Class Symbol**	**Example**
Constant	$O(1)$	Comparison of two integers
Logarithmic	$O(log(n))$	Binary Search, Exponentiation
Linear	$O(n)$	Linear Search
Log-Linear	$On(log(n))$	Merge Sort
Quadratic	$O(n^2)$	Integer multiplications
Cubic	$O(n^3)$	Matrix multiplication
Polynomial	$O(n^a),\ a \in \mathbb{R}$	
Exponential	$O(a^n),\ a \in \mathbb{R}$	Print all subsets
Factorial	$O(n!)$	Print all permutations

$$n! \gg 2^n \gg n^3 \gg n^2 \gg nlogn \gg n \gg logn \gg 1$$

Growth Rates of Functions



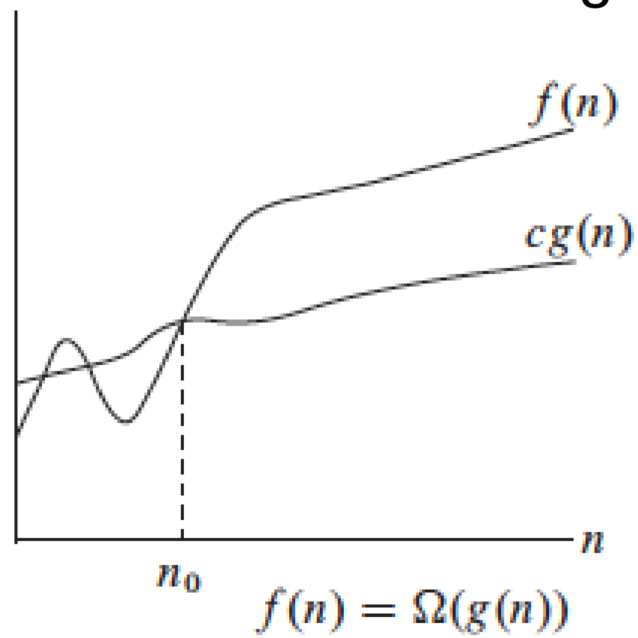
Growth Rates of Functions

Runtimes of algorithms of different runtime for input size n (on 1GHz PC). Assume that each operation takes 1 ns

n	$O(\log n)$	$O(n)$	$O(n \log n)$	$O(n^2)$	$O(2^n)$	$O(n!)$
10	$0.003\mu s$	$0.01\mu s$	$0.033\mu s$	$0.1\mu s$	$1\mu s$	$3.63ms$
20	$0.004\mu s$	$0.02\mu s$	$0.086\mu s$	$0.4\mu s$	$1ms$	77.1 *yrs*
30	$0.005\mu s$	$0.03\mu s$	$0.147\mu s$	$0.9\mu s$	$1sec$	$8 \cdot 10^{15}$ *yrs*
40	$0.005\mu s$	$0.04\mu s$	$0.213\mu s$	$1.6\mu s$	$18.3min$	very long
50	$0.006\mu s$	$0.05\mu s$	$0.282\mu s$	$2.5\mu s$	13 *days*	very long
100	$0.007\mu s$	$0.10\mu s$	$0.644\mu s$	$10\mu s$	$4 \cdot 10^{13}$ *yrs*	very long
10^3	$0.010\mu s$	$1.00\mu s$	$9.966\mu s$	$1ms$	very long	very long
10^4	$0.013\mu s$	$10\mu s$	$130\mu s$	$100ms$	very long	very long
10^5	$0.017\mu s$	$0.10ms$	$1.67ms$	$10sec$	very long	very long
10^6	$0.020\mu s$	$1ms$	$19.93ms$	$16.7min$	very long	very long
10^7	$0.023\mu s$	$0.01sec$	$0.23sec$	1.16 *days*	very long	very long
10^8	$0.027\mu s$	$0.10sec$	$2.66sec$	115.7 *days*	very long	very long
10^9	$0.030\mu s$	$1sec$	$29.90sec$	31.7 *yrs*	very long	very long

Big-Omega (Ω)

- We use Big-Omega to represent the best case running time of an algorithm
- Lower bound of f(n)
- We define Big-Omega as:



$$\Omega(g(n)) = \{ f(n) : \text{there exist positive constants } c \text{ and } n_0 \text{ such that } 0 \le cg(n) \le f(n) \text{ for all } n \ge n_0 \} .$$

Big-Omega (Ω)

Example: $n^3 + 4n^2 = \Omega(n^2)$

Proof:

- Here, we have $f(n) = n^3 + 4n^2$, and $g(n) = n^2$
- It is not too hard to see that if $n \geq 0$,

$$n^3 \leq n^3 + 4n^2$$

- We have already seen that if $n \geq 1$,

$$n^2 \leq n^3$$

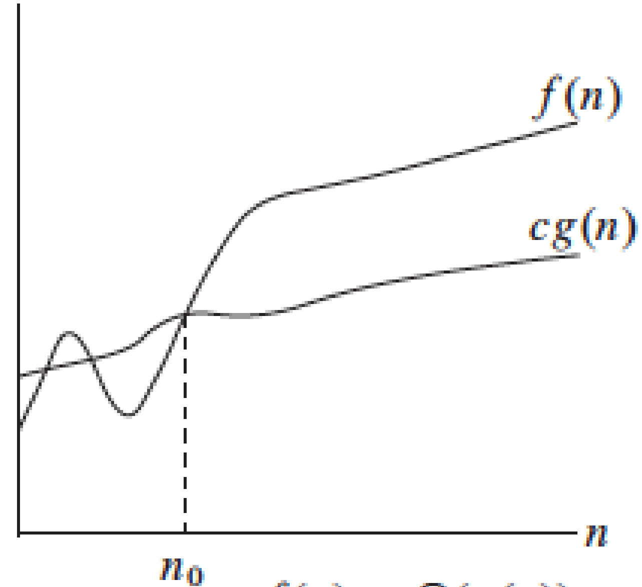
- Thus when $n \geq 1$,

$$n^2 \leq n^3 \leq n^3 + 4n^2$$

- Therefore,

$$1n^2 \leq n^3 + 4n^2 \text{ for all } n \geq 1$$

- Thus, we have shown that $n^3 + 4n^2 = \Omega(n^2)$ (by definition of Big-Ω, with $n_0 = 1$, and $c = 1$.)

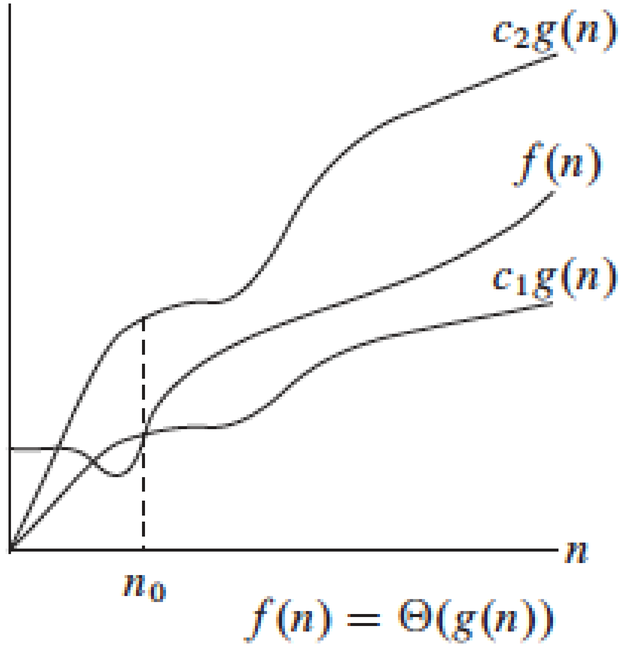


$f(n) = \Omega(g(n))$

$\Omega(g(n)) = \{f(n) :$ there exist positive constants c and n_0 such that $0 \leq cg(n) \leq f(n)$ for all $n \geq n_0\}$.

Big-Theta (Θ)

- Big-Theta represents the range; upper and lower
- Tight bound of f(n)
- We define Big-Theta as:



$$\Theta(g(n)) = \{ f(n) : \text{there exist positive constants } c_1, c_2, \text{ and } n_0 \text{ such that } 0 \le c_1 g(n) \le f(n) \le c_2 g(n) \text{ for all } n \ge n_0 \} .$$ [1]

Big-Theta (Θ)

Example: $n^2 + 5n + 7 = \Theta(n^2)$

Proof:

- When $n \geq 1$,

$$n^2 + 5n + 7 \leq n^2 + 5n^2 + 7n^2 \leq 13n^2$$

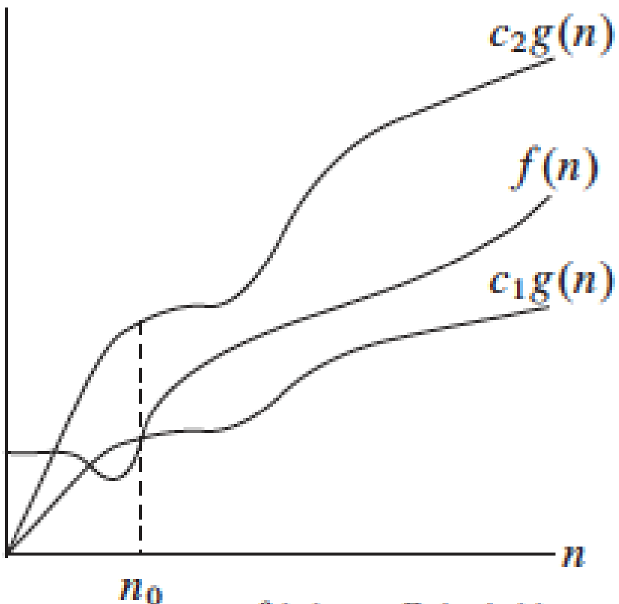
- When $n \geq 0$,

$$n^2 \leq n^2 + 5n + 7$$

- Thus, when $n \geq 1$

$$1n^2 \leq n^2 + 5n + 7 \leq 13n^2$$

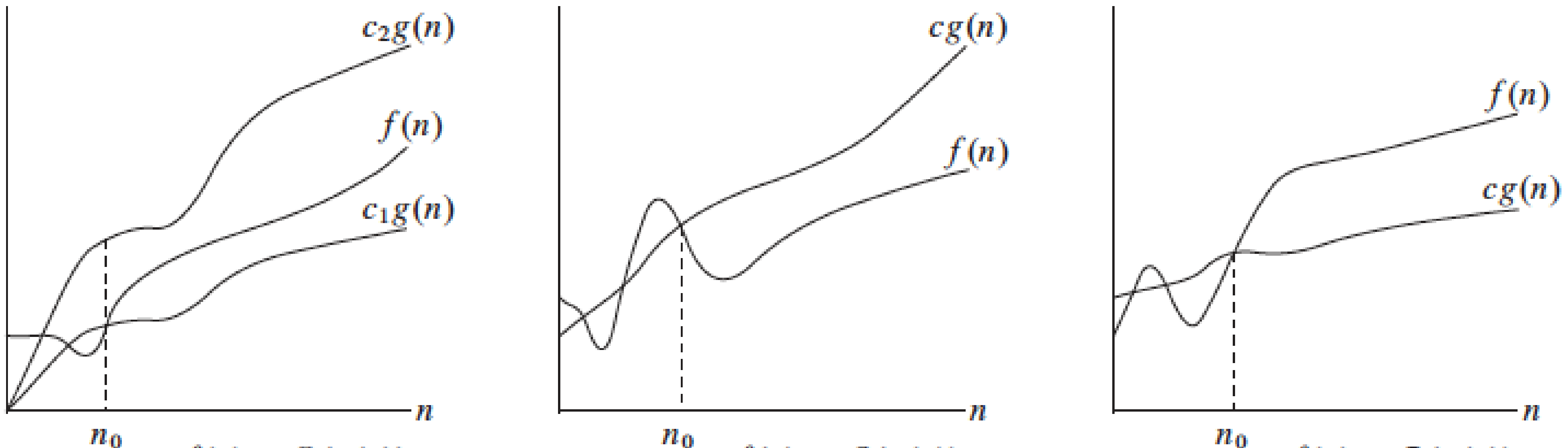
Thus, we have shown that $n^2 + 5n + 7 = \Theta(n^2)$ (by definition of Big-Θ, with $n_0 = 1$, $c_1 = 1$, and $c_2 = 13$.)



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

$\Theta(g(n)) = \{ f(n) :$ there exist positive constants c_1, c_2, and n_0 such that $0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n)$ for all $n \geq n_0 \}$.[1]

Θ, O, and Ω Comparison



$f(n) = \Theta(g(n))$ (a) $f(n) = O(g(n))$ (b) $f(n) = \Omega(g(n))$ (c)

a. $\Theta(g(n)) = \{f(n) :$ there exist positive constants c_1, c_2, and n_0 such that $0 \le c_1 g(n) \le f(n) \le c_2 g(n)$ for all $n \ge n_0\}$.[1]

c. $\Omega(g(n)) = \{f(n) :$ there exist positive constants c and n_0 such that $0 \le cg(n) \le f(n)$ for all $n \ge n_0\}$.

b. $O(g(n)) = \{f(n) :$ there exist positive constants c and n_0 such that $0 \le f(n) \le cg(n)$ for all $n \ge n_0\}$.

3-SUM Problem

Given a set of integer array of size n, identify three elements that sum to zero.

3-SUM Problem (Cont.)

```
sort(S);
for i=0 to n-3 do
   a = S[i];
   k = i+1;
   l = n-1;
   while (k<l) do
      b = S[k];
      c = S[l];
      if (a+b+c == 0) then
         output a, b, c;
         exit;
      else if (a+b+c > 0) then
         l = l - 1;
      else
         k = k + 1;
      end
   end
end
```

Dry run it with following array

-25 -10 -7 -3 2 4 8 10

```
-25 -10 -7 -3 2 4 8 10   (a+b+c==-25)
-25 -10 -7 -3 2 4 8 10   (a+b+c==-22)
. . .
-25 -10 -7 -3 2 4 8 10   (a+b+c==-7)
-25 -10 -7 -3 2 4 8 10   (a+b+c==-7)
-25 -10 -7 -3 2 4 8 10   (a+b+c==-3)
-25 -10 -7 -3 2 4 8 10   (a+b+c==2)
-25 -10 -7 -3 2 4 8 10   (a+b+c==0)
```

Source: Wikipedia.com

3-SUM Problem (Cont.)

Try this example:

8 ,4 , -10 , -7 , -3 , 2 , 10 , -25

Source: Wikipedia.com

3-SUM Problem (Cont.)

- Lets try to analyse the running time complexity of 3-SUM problem!