CS 203

Design & Analysis of Algorithms

Instructors: Dr. Ashish Phophalia
ashish_p@iiitvadodara.ac.in

TIMING ANALYSIS WITH STEP COUNT METHOD: INSERTION SORT

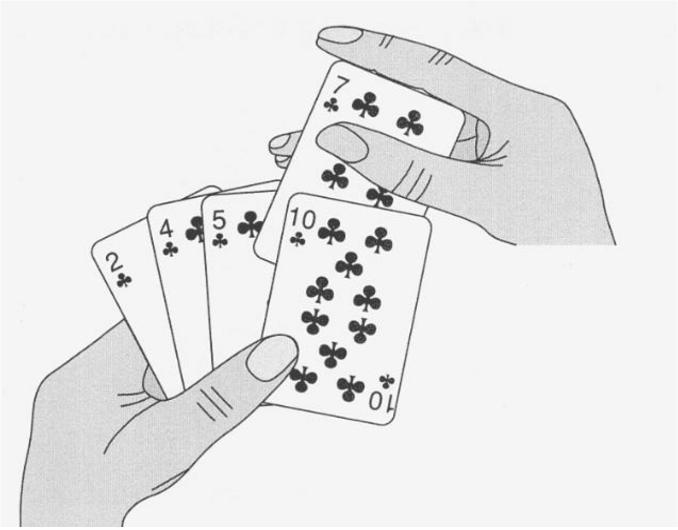
An example analysis of a sorting algorithm

- **Sorting**
  - Input:- $\langle a_1, \ a_2 \ ......, \ a_n \rangle$

Sorting Mechanism

  - Output:- A permutation of $\langle a_1, \ a_2 \ ......, \ a_n \rangle$ such that $a_i \leq a_{i+1}, 0 \leq i \leq n-1$

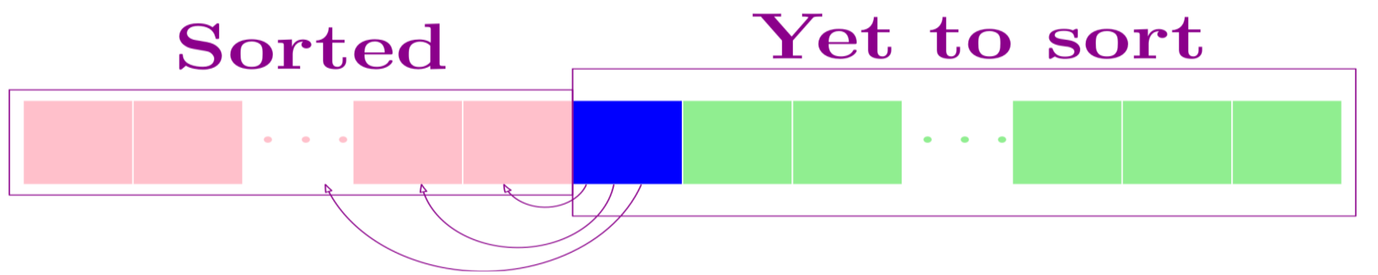
Idea



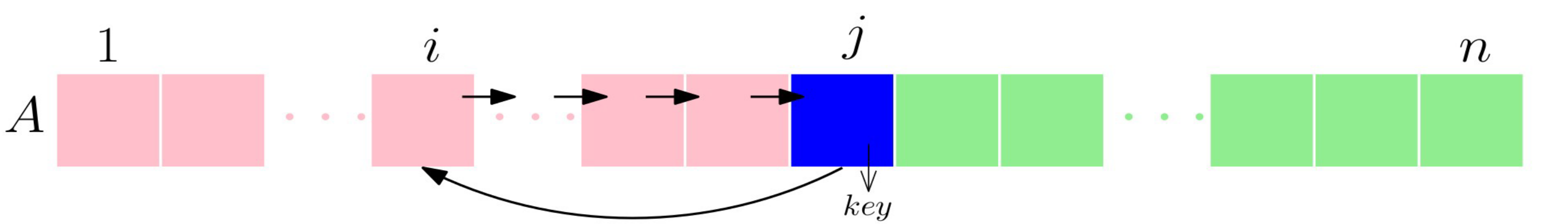
- Idea: like sorting a hand of playing cards
  - Start with an empty left hand and the cards facing down on the table.
  - Remove one card at a time from the table, and insert it into the correct position in the left hand
  - compare it with each of the cards already in the hand, from right to left
  - The cards held in the left hand are sorted

IDEA

- Place (insert) the first (blue) unsorted element in the sorted (pink) subarray



- for *j* = 2 to n
  - place (insert) *A*[*j*] in the sorted subarray *A*[1:*j*-1]



EXAMPLE

- 13 34 6 57 63 7

Insertion Sort Analysis: Step Count Method

INSERTION-SORT(A)	*cost*	*times*
for $j = 2$ **to** $A.length$	c_1	n
$\quad key = A[j]$	c_2	$n - 1$
$\quad$// Insert $A[j]$ into the sorted sequence $A[1 \,..\, j-1]$.	0	$n - 1$
$\quad i = j - 1$	c_4	$n - 1$
$\quad$**while** $i > 0$ and $A[i] > key$	c_5	$\sum_{j=2}^{n} t_j$
$\qquad A[i+1] = A[i]$	c_6	$\sum_{j=2}^{n} (t_j - 1)$
$\qquad i = i - 1$	c_7	$\sum_{j=2}^{n} (t_j - 1)$
$\quad A[i+1] = key$	c_8	$n - 1$

t_j: # of times the while statement is executed at iteration j

INSERTION SORT ANALYSIS: STEP COUNT METHOD

- Best Case [Sorted]

$$
\begin{aligned}
T(n) &= c_1 n + c_2(n-1) + c_4(n-1) + c_5(n-1) + c_8(n-1) \\
&= (c_1 + c_2 + c_4 + c_5 + c_8)n - (c_2 + c_4 + c_5 + c_8)\,.
\end{aligned}
$$

- T(n) = dn+e

INSERTION SORT ANALYSIS: STEP COUNT METHOD

- Worst Case [Reverse Sorted]

$$
\begin{aligned}
T(n) &= c_1 n + c_2(n-1) + c_4(n-1) + c_5\left(\frac{n(n+1)}{2} - 1\right) \\
&\quad + c_6\left(\frac{n(n-1)}{2}\right) + c_7\left(\frac{n(n-1)}{2}\right) + c_8(n-1) \\
&= \left(\frac{c_5}{2} + \frac{c_6}{2} + \frac{c_7}{2}\right) n^2 + \left(c_1 + c_2 + c_4 + \frac{c_5}{2} - \frac{c_6}{2} - \frac{c_7}{2} + c_8\right) n \\
&\quad - (c_2 + c_4 + c_5 + c_8)\ .
\end{aligned}
$$

$$\sum_{j=2}^{n} j = \frac{n(n+1)}{2} - 1$$

and

$$\sum_{j=2}^{n} (j-1) = \frac{n(n-1)}{2}$$

- $T(n) = an^2 + bn+c$

Analysis of Algorithm

- In general, we are not so much interested in the time and space complexity for small inputs.

- For example, while the difference in time complexity between linear and binary search is meaningless for a sequence with $n = 10$, but it is significant for $n = 2^{30}$

EXAMPLE

Consider two algorithms A and B that solve the same class of problems.

- The time complexity of A is 5,000n, the one for B is $\lceil 1.1^n \rceil$ for an input with n elements.
- For n = 10, A requires 50,000 steps, but B only 3, so B seems to be superior to A.
- For n = 1000, however, A requires 5,000,000 steps, while B requires 2.5×10^{41392}

Input Size (n)	Algorithm A=(5000n)	Algorithm B = $\lceil 1.1^n \rceil$
10	50000	3
100	5×10^5	13,781
1000	5×10^6	2.5×10^{41}
1000 000	5×10^9	2.5×10^{41392}

Analysis of Algorithm

- During design we are interested to measure the (relative) performance of algorithms for sufficiently larger input size
- Try to approximate the growth of running time as input size increases
  - More specifically, $n \to \infty$

Asymptotic Analysis

Asymptotic Analysis

Why not precise computation time analysis?

- Need to implement
- Machine/Input/Programming Support specific

WHY NOT STEP COUNT METHOD?

- Consider Linear Search O(an) and Binary Search (b log n).
- We run the Linear Search on a fast computer **A** and Binary Search on a slow computer **B**.
- Let's say the constant for **A** is 0.2 and constant for **B** is 1000.

n	0.2*n	1000 log n
10	2 sec	~38 min
100	20 sec	~1 hr
10^6	5.5 hr	~4 hr
10^9	6.3 years	~5 hr

- Lower order terms and constant terms does not impact much for sufficiently large input
- Overhead of considering all the terms

Concepts of order of growth and Asymptotic Notations are essential to understand.

ORDER OF GROWTH

- **Focus on the dominating terms**
  - Ignore lower order terms:- Does not matter much for significantly large input
  - Ignore constant multiplier:- Exact value differs by a constant factor

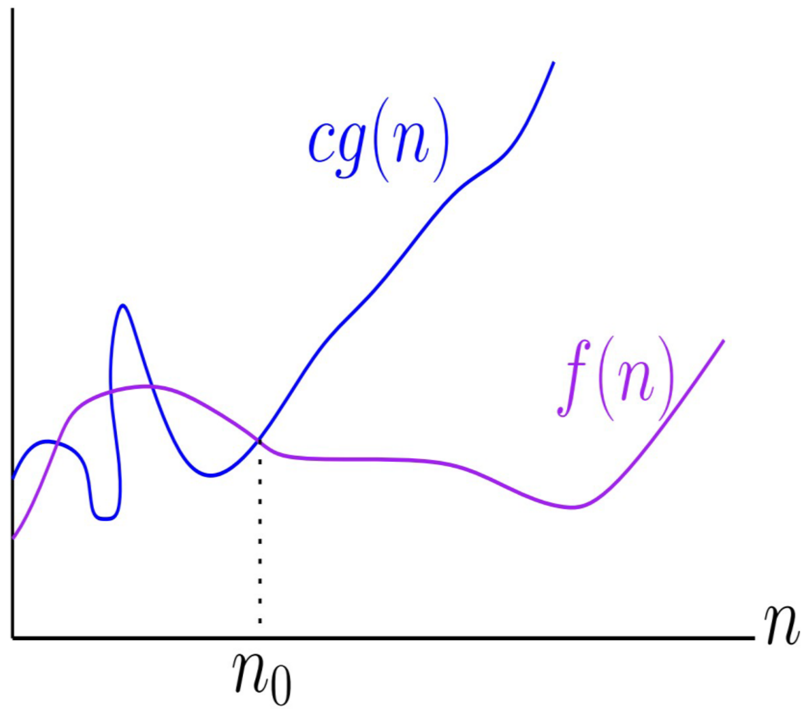
- **For insertion sort: *an^2 + bn+c***
  - Ignore lower order terms=> ***an^2***
  - Ignore constant multiplier=> ***n^2***
- Meaningful (but inexact) analysis
- Specifically, worst-case running time (***$an^2 + bn+c$***) is not equal to ***n^2***, rather it grows like ***n^2***
- Running time is ***n^2*** captures the notion that the order of growth is ***n^2***
- Efficient way of analyzing (in fact, comparing the relative) performance of an algorithm

Asymptotic Analysis

- Considering the order of growth for the larger input, we are studying the ***asymptotic*** efficiency of algorithms.
- It measure of the efficiency of algorithms that don't depend on machine-specific constants and doesn't require algorithms to be implemented and time taken by programs to be compared.
- How the running time of an algorithm increases with the size of the input.

Asymptotic Notations: O (Big Oh)

- Asymptotic Upper Bound => Asymptotic "less than or equal to"
  - $f(n) = O(g(n)) \Rightarrow f(n)$ "$\leq$" $g(n)$
- $O(g(n)) = \{f(n)$: there exist **positive** constants c and n0such that $\mathbf{0 \leq f(n) \leq cg(n)}$ for all $\mathbf{n \geq n0}\}$



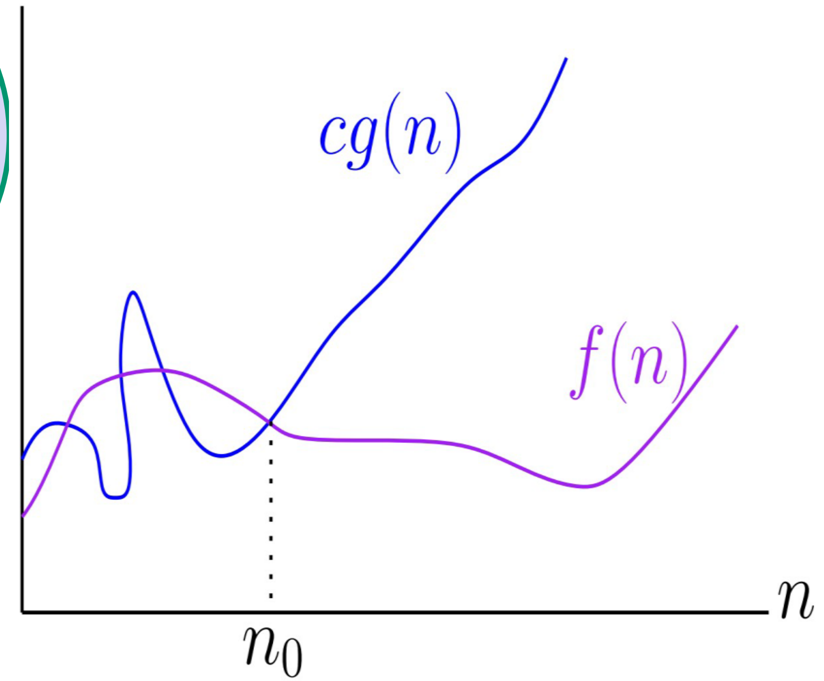
g(n) is an asymptotic upper bound of f (n)

O (Big Oh) Notation

$O(n^2)$
$2n^2-3n+4$
$3n+5$
30

$O(1)$
20
1000
10

$O(n)$
$3n-2$
$5n$
10



f grows not faster than g
f does not exceed g

- $3n^2 = O(n^3)$:
  - $3n^2 \le cn^3 \Rightarrow 3 \le cn \Rightarrow c = 1$ and $n0 = 3$

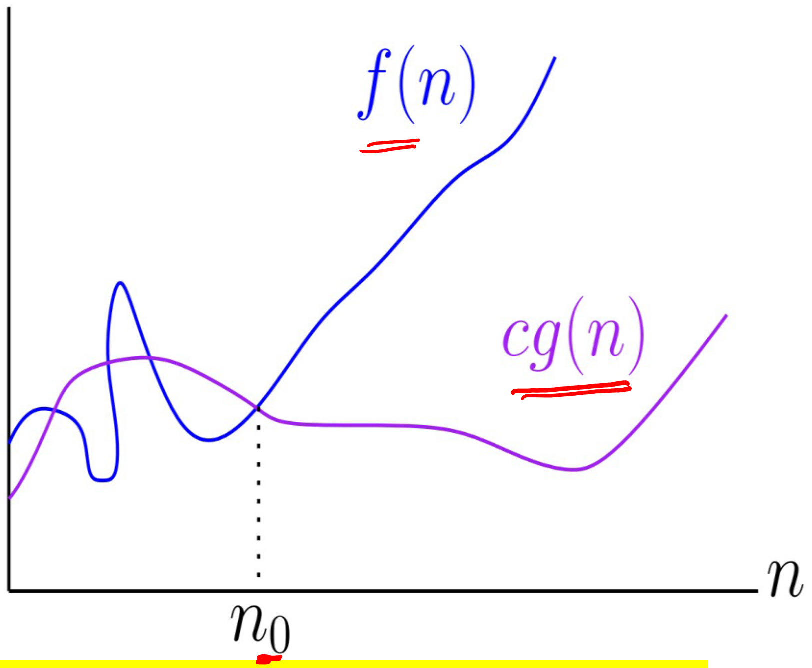
    (Also $c = 3$ and $n0 = 1$ or $c = 3.5$ and $n0 = 1$)
- $n^2 = O(n^2)$:
  - $n^2 \le cn^2 \Rightarrow c \ge 1 \Rightarrow c = 1$ and $n0 = 1$
- $150n^2+200n = O(n^2)$:
  - $150n^2+200n \le 150n^2 + n^2 = 151n^2$ (for $n \ge 200$)

  $\Rightarrow c=151$ and $n0 = 200$
- $3n = O(n^2)$:
  - $3n \le cn^2 \Rightarrow cn \ge 3 \Rightarrow c = 1$ and $n0 = 3$
    -

- **no unique pair of *n0* and *c***
- **To prove upper bound, find some *n0* and *c***

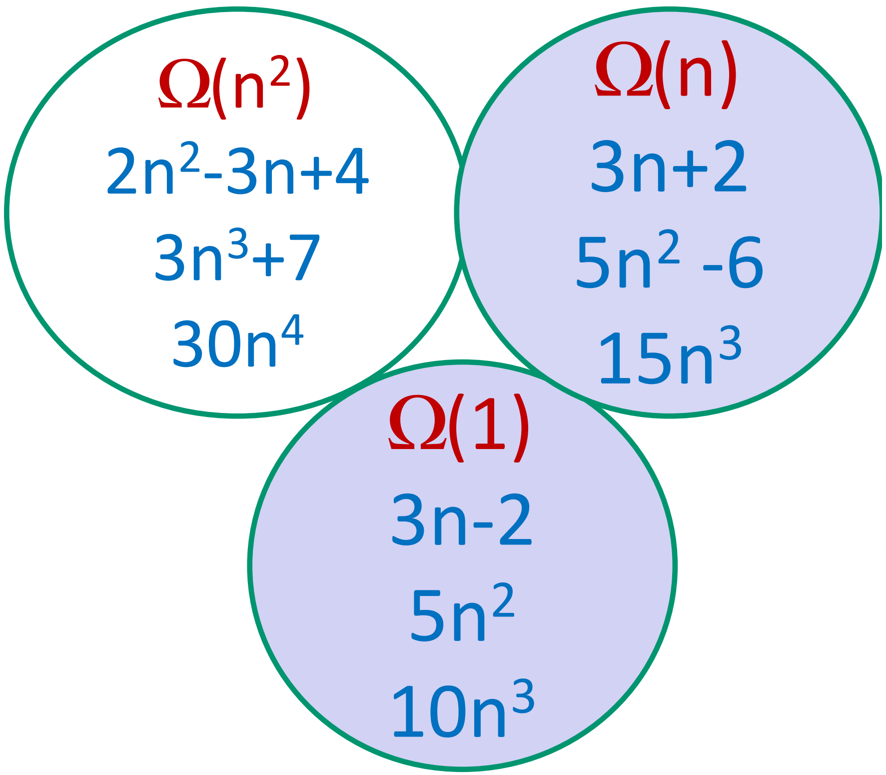
ASYMPTOTIC NOTATIONS: Ω (BIG OMEGA)

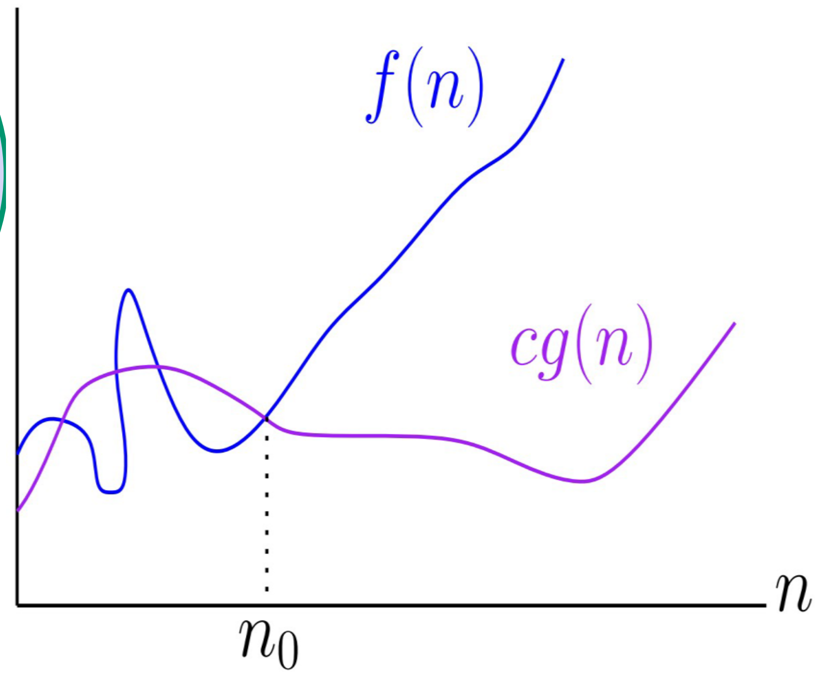
- Asymptotic Lower Bound
- Asymptotic "greater than or equal to" f(n) = Ω(g(n))
- => f(n) "≥" g(n)
- Ω(g(n)) is a set of functions that are asymptotically "greater than" or "equal to" g(n)
- **Ω(g(n))** = {f(n) : there exist **positive constants c** and n0 ~~such~~ that **0 ≤ cg(n) ≤ f (n)** for all **n≥n0**}



g(n) is an asymptotic lower bound of f (n)

Ω (BIG OMEGA) NOTATION



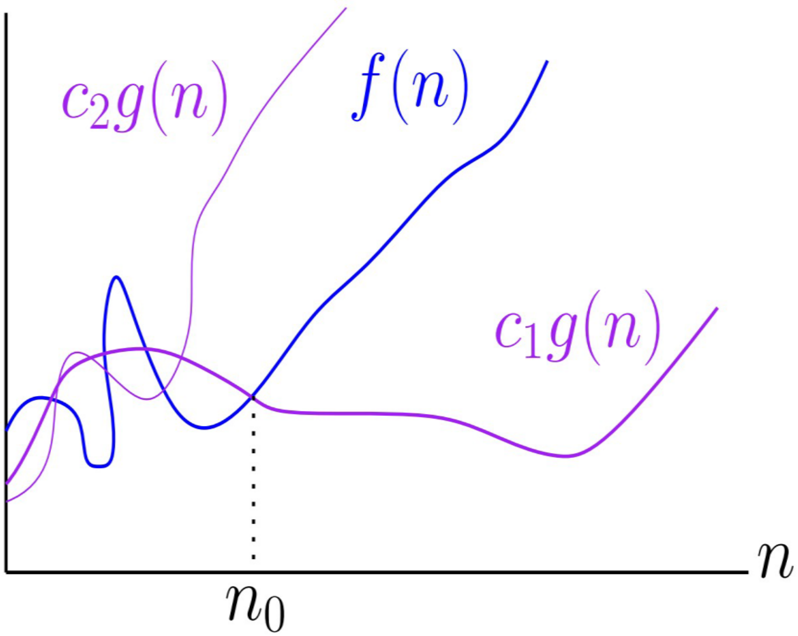


f grows no lower than g
f grows at least as fast as g

- $3n^2 = \Omega(n^2)$: $3n^2 \geq cn^2 \Rightarrow 3 \geq c \Rightarrow c = 1$ and $n0 = 1$
- $2n^2 = \Omega(n)$: $2n^2 \geq cn \Rightarrow c \leq 2 \Rightarrow c = 1$ and $n0 = 1$
- $150n^2+200n = \Omega(n^3)$:
  - $150n^2+200n \leq 150n^2+200n^2 \leq 350n^2$
  - $cn^3 \leq 150n^2+200n \leq 350n^2 \Rightarrow n \leq 350/c$
  - (n cannot be smaller than a constant)

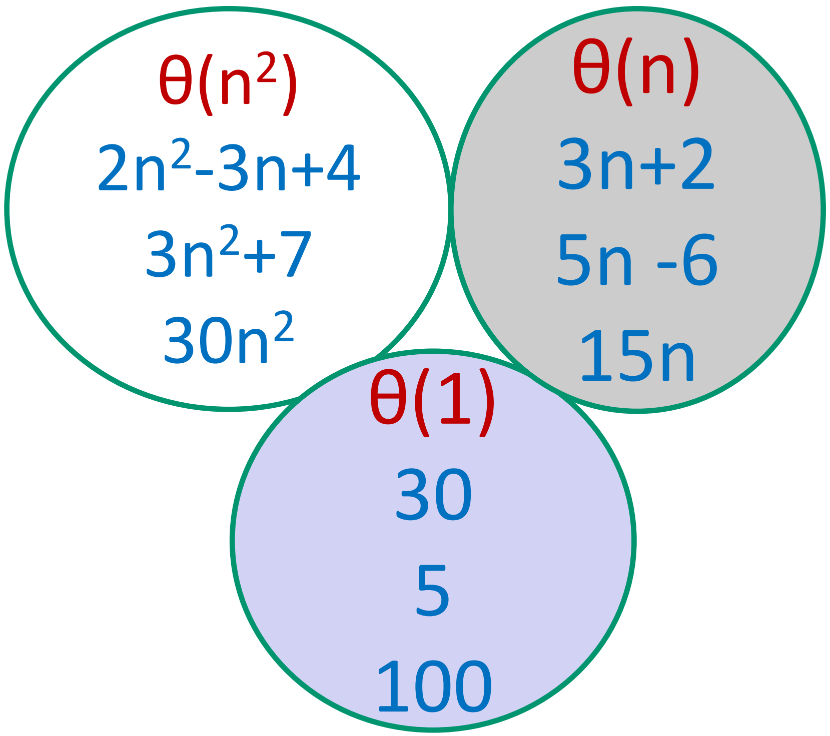
ASYMPTOTIC NOTATIONS: Θ (THETA)

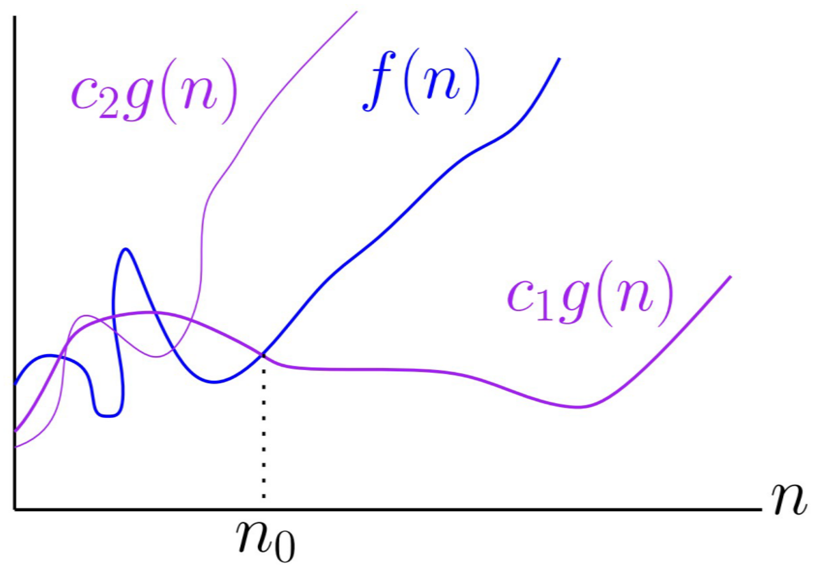
- Asymptotic Tight Bound
- Asymptotic "equal to"
- $f(n) = \Theta(g(n)) \Rightarrow f(n)$ "=" $g(n)$
- $\Theta(g(n))$ is a set of functions that are asymptotically "equal to" $g(n)$
- $f(n) \in \Theta(g(n))$, however, we say $f(n) = \Theta(g(n))$
- $\Theta(g(n)) = \{f(n)$: there exist **positive** constants c1, c2 and n0 such that **$0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n)$** for all **$n \geq n_0$**$\}$



g(n) is an asymptotic tight bound for f (n)

θ (Theta) Notation





f grows at the same rate as g

- $3/2\, n^2 = \Theta(n^2)$:

  1) $3/2\, n^2 \geq c1\, n^2$

  $\Rightarrow 3/2 \geq c1$

  $\Rightarrow c1 = 1$ and $n0 = 1$

  2) $3/2\, n^2 \leq c2\, n^2$

  $\Rightarrow 3/2 \leq c2$

  $\Rightarrow c1 = 2$ and $n0 = 1$

- $n \neq \Theta(n^2)$:

  $\Rightarrow c1\, n^2 \leq n \leq c2\, n^2$

  $\Rightarrow n \leq 1/c1$ and $n \geq 1/c2$

- $4n^3 \neq \Theta(n^2)$:

  $\Rightarrow c1\, n^2 \leq 4\, n^3 \leq c2\, n^2$

  $\Rightarrow$ only true for: $n \leq c2\, /4$

- $n \neq \Theta(\log n)$:

  $\Rightarrow c1 \log n \leq n \leq c2 \log n$

  $\Rightarrow c2 \geq n/\log n$

  $\Rightarrow c2 \geq n$

  $\Rightarrow$ n cannot be smaller than a constant