# Advanced Algorithms and Data Structures

## Lecture# 02: Asymptotic Notations and Analysis

# Asymptotic Notations Properties

- Categorize algorithms based on asymptotic growth rate
  - e.g. linear, quadratic, exponential

- Ignore small constant and small inputs

- Estimate upper bound and lower bound on growth rate of time complexity function

- Describe running time of algorithm as n grows to $\infty$.

*Limitations*

- not always useful for analysis on fixed-size inputs.
- All results are for *sufficiently large* inputs.

# Asymptotic Notations

Asymptotic Notations **$\Theta$, O, $\Omega$, *o*, $\omega$**

- We use $\Theta$ to mean "order exactly", (Tight Bound)
- O to mean "order at most", (Tight Upper Bound)
- $\Omega$ to mean "order at least", (Tight Lower Bound)
- o to mean "upper bound",
- $\omega$ to mean "lower bound",

Define a ***set*** of functions which is in practice used to compare two function sizes.

# Big-Oh Notation (O)

If f, g: $N \rightarrow R^+$, then we can define Big-Oh as

For a given function $g(n) \geq 0$, denoted by $O(g(n))$ the set of functions,

$O(g(n)) = \{f(n)$: there exist positive constants $c$ and $n_o$ such that

$0 \leq f(n) \leq cg(n)$, for all $n \geq n_o\}$

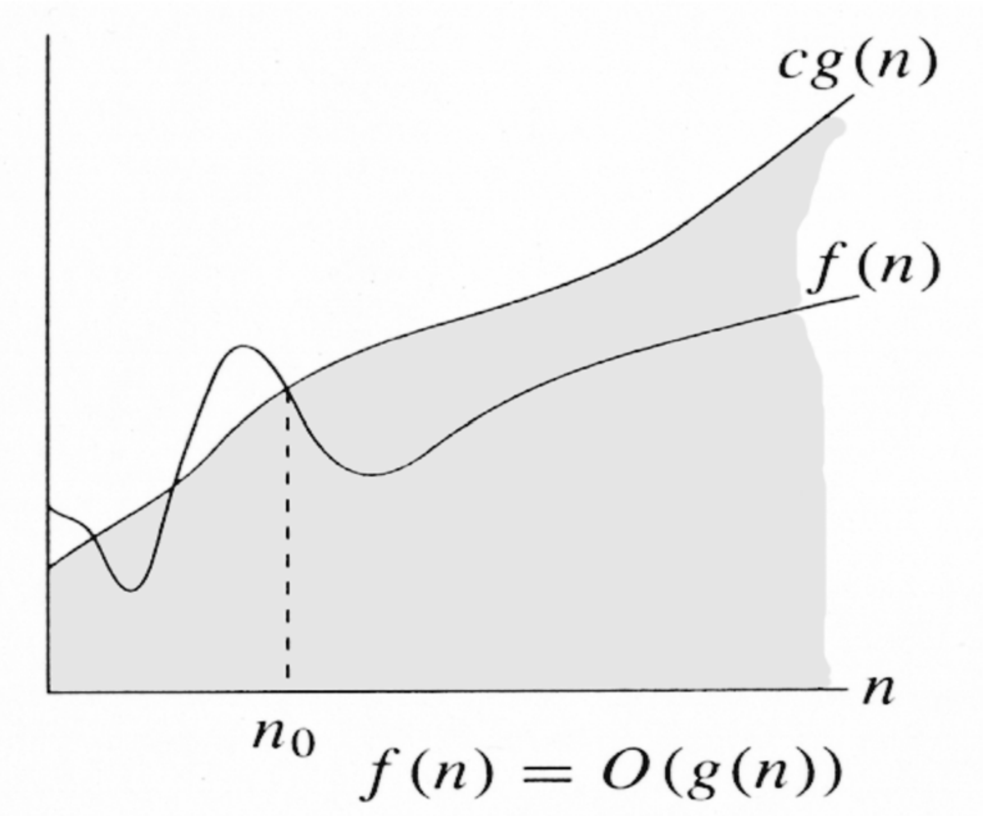
$f(n) = O(g(n))$ means function $g(n)$ is an asymptotically upper bound for $f(n)$.

We may write f(n) = O(g(n)) OR $f(n) \in O(g(n))$

***Intuitively***:

Set of all functions whose *rate of growth* is the same as or lower than that of *g*(*n*). *f*(*n*) is bounded above by *g*(*n*) for all sufficiently large n

# Big-Oh Notation (O)




$f(n) \in O(g(n))$

$\exists\, c > 0, \exists\, n_0 > 0$ and $\forall n \geq n_0,\ 0 \leq f(n) \leq c.g(n)$

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

# Big-Oh Notation (O)

The idea behind the big-O notation is to establish an **upper boundary** for the growth of a function f(n) for large n.

This boundary is specified by a function g(n) that is usually much **simpler** than f(n).

We accept the constant C in the requirement
$f(n) \leq C \cdot g(n)$ whenever $n > n_0$,

We are only interested in large n, so it is OK if
$f(n) > C \cdot g(n)$ for $n \leq n_0$.

The relationship between f and g can be expressed by stating either that g(n) is an upper bound on the value of f(n) or that in the long run , f grows at most as fast as g.

# Example

- As a simple illustrative example, we show that the function $2n^2 + 5n + 6$ is $O(n^2)$.

- For all $n \geq 1$, it is the case that

$$2n^2 + 5n + 6 \leq 2n^2 + 5n^2 + 6n^2 = 13n^2$$

- Hence, we can take $c = 13$ and $n_o = 1$, and the definition is satisfied.

# Example

Prove that $2n^2 = O(n^3)$

Proof:

Assume that $f(n) = 2n^2$, and $g(n) = n^3$

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

Now we have to find the existence of c and $n_0$

$f(n) \leq c.g(n) \Rightarrow 2n^2 \leq c.n^3 \Rightarrow 2 \leq c.n$

if we take, $c = 1$ and $n_0 = 2$ OR

$c = 2$ and $n_0 = 1$ then

$2n^2 \leq c.n^3$

Hence $f(n) = O(g(n))$, $c = 1$ and $n_0 = 2$

# Example

Prove that $n^2 = O(n^2)$

Proof:

Assume that $f(n) = n^2$, and $g(n) = n^2$

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

Now we have to find the existence of c and $n_0$

$f(n) \le c.g(n) \Rightarrow n^2 \le c.n^2 \Rightarrow 1 \le c$

if we take, $c = 1$, $n_0 = 1$

Then

$n^2 \le c.n^2$ for $c = 1$ and $n \ge 1$

Hence, $n^2 = O(n^2)$, where $c = 1$ and $n_0 = 1$

# Example

Prove that $1000.n^2 + 1000.n = O(n^2)$

Proof:

Assume that $f(n) = 1000.n^2 + 1000.n$, and $g(n) = n^2$

We have to find existence of c and $n_0$ such that

$0 \le f(n) \le c.g(n)$ for all $n \geq n_0$

$1000.n^2 + 1000.n \le c.n^2$

for $c = 1001$, $1000.n^2 + 1000.n \le 1001.n^2$

$1000.n \le n^2 \Rightarrow n^2 - 1000.n \geq 0$

$n (n-1000) \geq 0$,

this true for $n \geq 1000$

Hence $f(n) = O(g(n))$ for $c = 1001$ and $n_0 = 1000$

# Example

Disprove that $n^3 \neq O(n^2)$

Proof:

On contrary we assume that there exist some positive constants c and $n_0$ such that

$0 \leq n^3 \leq c.n^2$ for all $n \geq n_0$

$n \leq c$

Since c is any fixed number and n is any arbitrary constant, therefore $n \leq c$ is not possible in general.

Hence our supposition is wrong and $n^3 \leq c.n^2$, for $n \geq n_0$ is not true for any combination of c and $n_0$.

Hence, $n^3 \neq O(n^2)$

# Example

Prove that $2n + 10 = O(n)$

Proof:

Assume that $f(n) = 2n + 10$, and $g(n) = n$

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

Now we have to find the existence of c and $n_0$

$f(n) \le c.g(n) \Rightarrow 2n + 10 \le c.n \Rightarrow (c - 2)\, n \ge 10 \Rightarrow n \ge 10/(c - 2)$

$c > 2$ for $n > 0$, we pick, $c = 3$, then $n_0 = 10$

Then

$2n + 10 \le c.n$ for $c = 3$ and $n \ge 10$

Hence, $2n + 10 = O(n)$, where $c = 3$ and $n_0 = 10$

# Example

Prove which of the following function is larger by order of growth?
$(1/3)^n$ or 17?

- Let's check if

  $(1/3)^n = O(17)$

  $(1/3)^n \leq c.17$, which is true for $c=1, n_0 = 1$
- Let's check if

  $17 = O((1/3)^n)$

  $17 \leq c. (1/3)^n$, which is true for $c > 17. 3^n$
- And hence can't be bounded for large n.
- That's why $(1/3)^n$ is less in growth rate then 17.

# Example

Prove or disprove $2^{2n} = O(2^n)$?

- To prove above argument we have to show
  - $2^{2n} \leq C \cdot 2^n$
  - $2^n 2^n \leq C \cdot 2^n$

- This inequality holds only when
- $C \geq 2^n$
- which makes C to be **non-constant**.

- Hence we can't bound $2^{2n}$ by $O(2^n)$

# Example

Prove that : $8n^2 + 2n - 3 = O(n^2)$

Proof:

Need $c > 0$ and $n_0 \geq 1$ such that

$8n^2 + 2n - 3 \leq c.n^2$ for $n \geq n_0$

Consider the reasoning:

$f(n) = 8n^2 + 2n - 3 \leq 8n^2 + 2n \leq 8n^2 + 2n^2 = 10n^2$

Hence, $8n^2 + 2n - 3 = O(n^2)$, where $c = 10$ and $n_0 = 1$

# Example

Can you bound $3^n = O(2^n)$ ?

To prove above argument we have to show

$3^n \leq C. 2^n$

$3^n \leq C. 2^n$

$3^n \leq (3/2)^n\ 2^n$

This inequality holds only when $C \geq (3/2)^n$, which makes C to be non-constant.

Hence we can't bound $3^n$ by $O(2^n)$

# Example

Which of the following function is larger by order of growth? N log N or $N^{1.5}$?

Note that **g(N)** $= N^{1.5} = N \bullet N^{0.5}$

**Hence, between f(N) and g(N)**, we only need to compare growth rate of **log(N)** and $N^{0.5}$

Equivalently, we can compare growth rate of $\log^2 N$ with **N**

Now, we can refer to the previously state result to figure out whether **f(N)** or **g(N)** grows faster!

# Big-Omega Notation ($\Omega$)

If f, g: N $\rightarrow$ $R^+$, then we can define Big-Omega as

For a given function $g(n)$ denote by $\Omega(g(n))$ the set of functions,

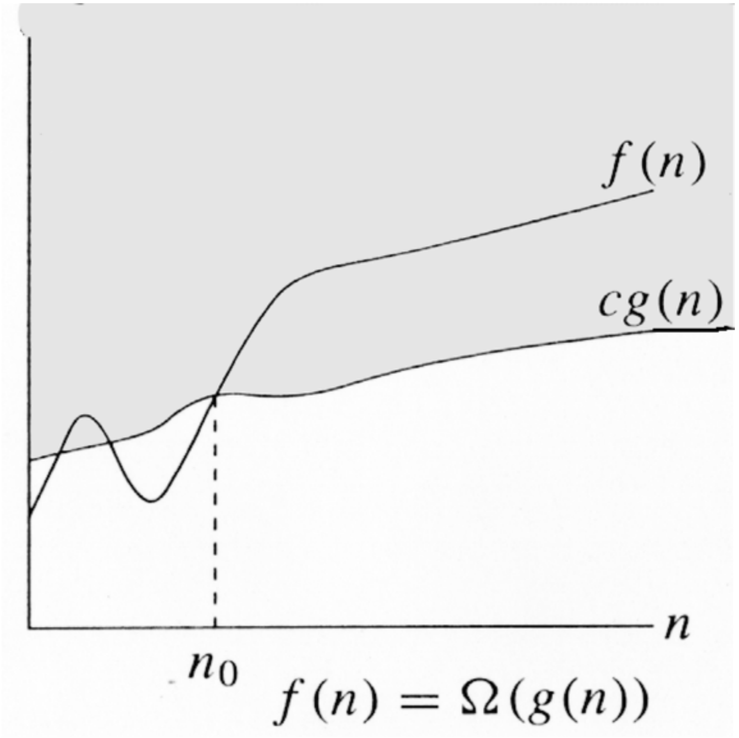
$\Omega(g(n)) = \{f(n)$: there exist positive constants $c$ and $n_o$ such that $0 \le cg(n) \le f(n)$ for all $n \ge n_o\}$

$f(n) = \Omega(g(n))$, means that function $g(n)$ is an asymptotically lower bound for $f(n)$.

We may write f(n) = $\Omega$(g(n)) OR f(n) $\in$ $\Omega$(g(n))

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

# Big-Omega Notation (Ω)




**$f(n) \in \Omega(g(n))$**

**$\exists\ c > 0, \exists\ n_0 > 0\ , \forall n \geq n_0, f(n) \geq c.g(n)$**

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

Note the duality rule: $t(n) \in \Omega(f(n)) \equiv f(n) \in O(t(n))$

# Example

Prove that $3n + 2 = \Omega(n)$

Proof:

Assume that $f(n) = 3n + 2$ , and $g(n) = n$

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

We have to find the existence of c and $n_0$ such that

$c.g(n) \leq f(n)$ for all $n \geq n_0$

$c.n \leq 3n + 2$

At R.H.S a positive term is being added to 3n, which will make L.H.S ≤ R.H.S for all values of n, when c = 3.

Hence $f(n) = \Omega(g(n))$, for c = 3 and $n_0 = 1$

# Example

Prove that $5.n^2 = \Omega(n)$

Proof:

Assume that $f(n) = 5.n^2$ , and $g(n) = n$

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

We have to find the existence of c and $n_0$ such that

$c.g(n) \leq f(n)$ for all $n \geq n_0$

$c.n \leq 5.n^2 \Rightarrow c \leq 5.n$

if we take, $c = 5$ and $n_0 = 1$ then

$c.n \leq 5.n^2$ for all $n \geq n_0$

Hence $f(n) = \Omega(g(n))$, for $c = 5$ and $n_0 = 1$

# Example

Prove that $5n^2 + 2n - 3 = \Omega(n^2)$

Proof:

Assume that $f(n) = 5n^2 + 2n - 3$, and $g(n) = n^2$

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

We have to find the existence of c and $n_0$ such that

$c.g(n) \leq f(n)$ for all $n \geq n_0$

$c.n^2 \leq 5.n^2 + 2n - 3$

We can take c = 5, given that 2n-3 is always positive.

2n-3 is always positive for $n \geq 2$. Therefore $n_0 = 2$.

And hence $f(n) = \Omega(g(n))$, for c = 5 and $n_0 = 2$

# Example

Prove that 100.n + 5 = $\Omega(n^2)$

Proof:

Let f(n) = 100.n + 5, and g(n) = $n^2$

Assume that f(n) = $\Omega$(g(n)) ?

Now if f(n) = $\Omega$(g(n)) then there exist c and $n_0$ such that

c.g(n) ≤ f(n) for all $n \geq n_0$

$c.n^2 \leq 100.n + 5$

For the above inequality to hold $\lim_{n\to\infty} \frac{f(n)}{g(n)} = \infty$ meaning f(n) grows faster than g(n).

But $\lim_{n\to\infty} \frac{100n+5}{n^2} = 0 \neq \infty$ which means g(n) is growing faster than f(n)

And hence f(n) ≠ $\Omega$(g(n))

# Theta Notation ($\Theta$)

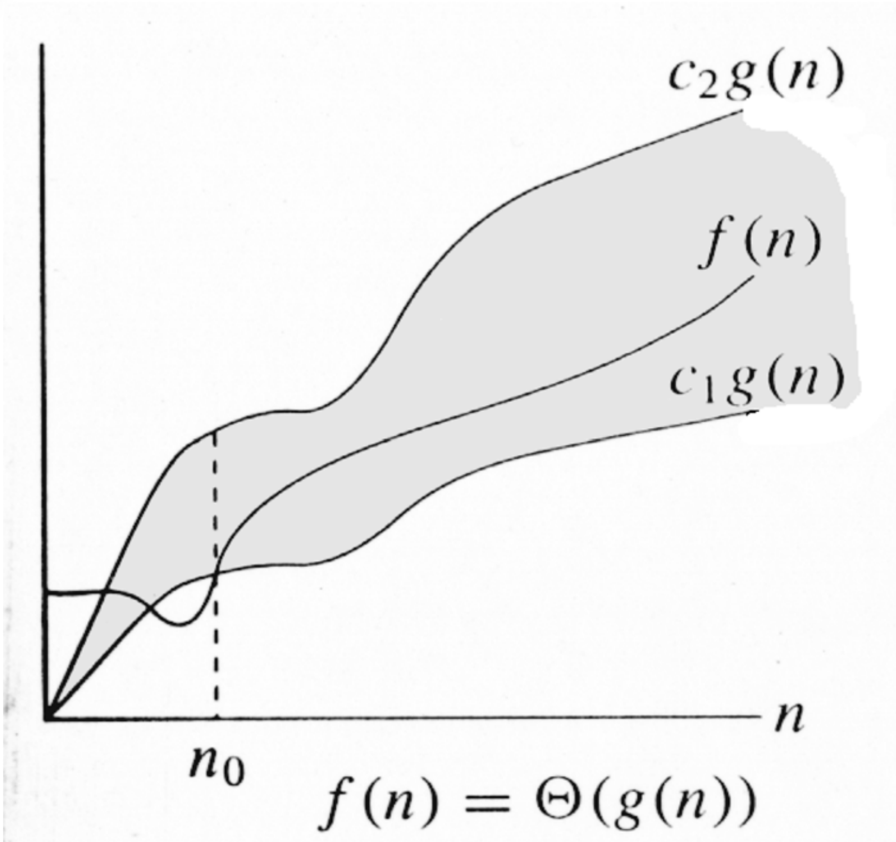
If f, g: N $\rightarrow$ $R^+$, then we can define Big-Theta as

For a given function $g(n)$ denoted by $\Theta(g(n))$ the set of functions,
$\Theta(g(n)) = \{f(n)$: there exist positive constants $c_1, c_2$ and $n_o$ such that
$0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n)$ for all $n \geq n_o\}$
$f(n) = \Theta(g(n))$ means function $f(n)$ is equal to $g(n)$ to within a constant factor, and $g(n)$ is an asymptotically tight bound for $f(n)$.

We may write f(n) = $\Theta$(g(n)) OR f(n) $\in$ $\Theta$(g(n))

***Intuitively***: Set of all functions that have same *rate of growth* as *g*(*n*).
When a problem is $\Theta$(*n*), this represents both an upper and lower bound i.e. it is O(*n*) **and** $\Omega$(*n*) (no algorithmic gap)

# Theta Notation ($\Theta$)




**$f(n) \in \Theta(g(n))$**

**$\exists\ c_1 > 0,\ c_2 > 0,\ \exists\ n_0 > 0,\ \forall\ n \geq n_0,\ c_2 . g(n) \leq f(n) \leq c_1 . g(n)$**

***We say that g(*n*) is an asymptotically tight bound for f(*n*).***

# Example

Prove that $½.n^2 – ½.n = \Theta(n^2)$

Proof:

Assume that $f(n) = ½.n^2 - ½.n$, and $g(n) = n^2$

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

We have to find the existence of $c_1$, $c_2$ and $n_0$ such that

$c_1.g(n) \le f(n) \le c_2.g(n)$ for all $n \ge n_0$

Since, $½ n^2 - ½ n \le ½ n^2$ then $c_2 = ½$ , $\forall n \ge 0$ and

Since $½ n$ is subtracted from $½ n^2$ , $c_1$ must be less than ½,

Assuming $c_1 = ¼$ ➔ $¼ n^2 \le ½ n^2 - ½ n$ ➔ $\forall n \ge 2$

$c_1.g(n) \le f(n) \le c_2.g(n)$ $\forall n \ge 2$, $c_1 = ¼$, $c_2 = ½$

Hence $f(n) = \Theta(g(n)) \Rightarrow ½.n^2 – ½.n = \Theta(n^2)$

# Example

Prove that $2.n^2 + 3.n + 6 \neq \Theta(n^3)$

Proof: Let $f(n) = 2.n^2 + 3.n + 6$, and $g(n) = n^3$

we have to show that $f(n) \neq \Theta(g(n))$

On contrary assume that $f(n) \in \Theta(g(n))$ i.e. there exist some positive constants $c_1$, $c_2$ and $n_0$ such that:

$c_1.g(n) \leq f(n) \leq c_2.g(n)$

Solve for $c_2$:

$f(n) \leq c_2.g(n) \Rightarrow 2n^2 + 3n + 6 \leq 2n^2 + 3n^2 + 6n^2 \leq c_2n^3 \Rightarrow c = 11$ and $n_0 = 1$

Solve for $c_1$:

$c_1.g(n) \leq f(n) \Rightarrow c_1n^3 \leq 2n^2 + 3n + 6 \Rightarrow c_1n^3 \leq 2n^2 \leq 2n^2 + 3n + 6$

$c_1.n \leq 2$, for large n this is not possible

Hence $f(n) \neq \Theta(g(n)) \Rightarrow 2.n^2 + 3.n + 6 \neq \Theta(n^3)$

# Example

Prove that $\frac{1}{2}.n^2 - 3.n = \Theta(n^2)$

Proof

Let $f(n) = \frac{1}{2}.n^2 - 3.n$, and $g(n) = n^2$ $\quad f(n) = \Theta(g(n))$?

We have to find the existence of $c_1$, $c_2$ and $n_0$ such that

$c_1.g(n) \leq f(n) \leq c_2.g(n) \qquad \forall\, n \geq n_0$

$c_1.\, n^2 \leq \frac{1}{2}.n^2 - 3.n \leq c_2.\, n^2$

Since, $\frac{1}{2}\, n^2 - 3\, n \leq \frac{1}{2}\, n^2 \qquad \forall n \geq 1$ if $c_2 = \frac{1}{2}$ and

$\frac{1}{2}\, n^2 - 3\, n \geq 1/4\, n^2$ ( $\forall n \geq 7$ ), $c_1 = 1/14$

$c_1.g(n) \leq f(n) \leq c_2.g(n) \qquad \forall n \geq 6$, $c_1 = 1/14$, $c_2 = \frac{1}{2}$

Hence $f(n) = \Theta(g(n)) \Rightarrow \frac{1}{2}.n^2 - 3.n = \Theta(n^2)$

# Little-Oh Notation

o-notation is used to denote a upper bound that is not asymptotically tight.

For a given function $g(n) \geq 0$, denoted by $o(g(n))$ the set of functions,

$$o(g(n)) = \begin{cases} f(n): \text{for any positive constants } c, \text{ there exists a constant } n_o \\ \text{such that } 0 \leq f(n) < cg(n) \text{ for all } n \geq n_o \end{cases}\Bigg\}$$

f(n) becomes insignificant relative to g(n) as n approaches infinity. g(n) is an upper bound for f(n), not asymptotically tight

$$\text{e.g., } 2\text{n} = o(n^2) \text{ but } 2n^2 \neq o(n^2).. \qquad \lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$$

# Example

Prove that $2n^2 = o(n^3)$

Proof:

Assume that $f(n) = 2n^2$ , and $g(n) = n^3$

$f(n) = o(g(n))$ ?

Now we have to find the existence $n_0$ for any c

$f(n) < c.g(n)$ this is true

$2n^2 < c.n^3 \rightarrow 2 < c.n$

This is true for any c, because for any arbitrary c we can choose $n_0$ such that the above inequality holds.

Hence $f(n) = o(g(n))$

# Example

Prove that $n^2 \neq o(n^2)$

Proof:

Assume that $f(n) = n^2$ , and $g(n) = n^2$

Now we have to show that $f(n) \neq o(g(n))$

Since

$f(n) < c.g(n) \Rightarrow n^2 < c.n^2 \Rightarrow 1 \leq c$,

In our definition of small o, it was required to prove for any c but here there is a constraint over c .

Hence, $n^2 \neq o(n^2)$, where $c = 1$ and $n_0 = 1$

# Little-Omega Notation

Little-$\omega$ notation is used to denote a lower bound that is not asymptotically tight.

For a given function $g(n)$, denote by $\omega(g(n))$ the set of all functions.

$\omega(g(n)) = \{f(n)$: for any positive constants $c$, there exists a constant $n_o$ such that $0 \leq cg(n) < f(n)$ for all $n \geq n_o\}$

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

$$\text{e.g.,} \frac{n^2}{2} = \omega(n) \text{ but } \frac{n^2}{2} \neq \omega(n^2).. \qquad \lim_{n \to \infty} \frac{f(n)}{g(n)} = \infty$$

# Example

Prove that $5.n^2 = \omega(n)$

Proof:

Assume that $f(n) = 5.n^2$ , and $g(n) = n$

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

We have to prove that for any c there exists $n_0$ such that
$c.g(n) < f(n)$ for all $n \geq n_0$

$c.n < 5.n^2 \Rightarrow c < 5.n$

This is true for any c, because for any arbitrary c e.g. c = 1000000, we can choose $n_0$ = 1000000/5 = 200000 and the above inequality does hold.

And hence $f(n) = \omega(g(n))$

# Example

Prove that $5.n + 10 \neq \omega(n)$

Proof:

Assume that $f(n) = 5.n + 10$, and $g(n) = n$

$f(n) \neq \Omega(g(n))$ ?

We have to find the existence $n_0$ for any c, such that

$c.g(n) < f(n)$ for all $n \geq n_0$

$c.n < 5.n + 10$, if we take $c = 16$ then

$16.n < 5.n + 10 \Rightarrow 11.n < 10$ is not true for any positive integer.

Hence $f(n) \neq \omega(g(n))$

# Example

Prove that $100.n \neq \omega(n^2)$

Proof:

Let $f(n) = 100.n$, and $g(n) = n^2$

Assume that $f(n) = \omega(g(n))$

Now if $f(n) = \omega(g(n))$ then there $n_0$ for any c such that

$c.g(n) < f(n)$ for all $n \geq n_0$

$c.n^2 < 100.n \Rightarrow c.n < 100$

If we take $c = 100$, $n < 1$, not possible

Hence $f(n) \neq \omega(g(n))$ i.e. $100.n \neq \omega(n^2)$

# Asymptotic Functions Summary

If f(n) = $\Theta$(g(n)) we say that f(n) and g(n) grow at the same rate, asymptotically

If f(n) = O(g(n)) and f(n) $\neq$ $\Omega$(g(n)), then we say that f(n) is asymptotically slower growing than g(n).

If f(n) = $\Omega$(g(n)) and f(n) $\neq$ O(g(n)), then we say that f(n) is asymptotically faster growing than g(n).

# Usefulness of Notations

It is not always possible to determine behaviour of an algorithm using $\Theta$ -notation.

For example, given a problem with n inputs, we may have an algorithm to solve it in $a.n^2$ time when n is even and $c.n$ time when n is odd. OR

We may prove that an algorithm never uses more than $e.n^2$ time and never less than $f.n$ time.

In either case we can neither claim $\Theta(n)$ nor $\Theta(n^2)$ to be the order of the time usage of the algorithm.

Big O and $\Omega$ notation will allow us to give at least partial information

# Usefulness of Notations

To express the efficiency of our algorithms which of the three notations should we use?

As computer scientist we generally like to express our algorithms as **big O** since we would like to know the upper bounds of our algorithms.

Why?

If we know the worse case then we can aim to improve it and/or avoid it.

# Usefulness of Notations

Even though it is correct to say "$7n - 3$ is $O(n^3)$", a better statement is "$7n - 3$ is $O(n)$", that is, one should make the approximation as tight as possible

Simple Rule:

Drop lower order terms and constant factors

**$7n-3$ is $O(n)$**

**$8n^2 \log n + 5n^2 + n$ is $O(n^2 \log n)$**

Strictly speaking this use of the equals sign is incorrect

- the relationship is a set inclusion, not an equality
- $f(n) \in O(g(n))$ is better

# Big Oh Does Not Tell the Whole Story

Question?

- If two algorithms A and B have the same asymptotic complexity, say $O(n^2)$, will the execution time of the two algorithms always be same?
- How to select between the two algorithms having the same asymptotic performance?

Answer:

- They may not be the same. There is this small matter of the constant of proportionality.
- Suppose that A does ten operations for each data item, but algorithm B only does three.
- It is reasonable to expect B to be faster than A even though both have the same asymptotic performance. The reason is that asymptotic analysis ignores constants of proportionality.

# Big Oh Does Not Tell the Whole Story

```
Algorithm_A {
    set up the algorithm; /*taking 50 time units*/
    read in n elements into array A; /* 3 units per element */
    for (i = 0; i < n; i++) {
        do operation1 on A[i]; /* takes 10 units */
        do operation2 on A[i]; /* takes 5 units */
        do operation3 on A[i]; /* takes 15 units */
    }
}
```

**TA(n) = 50 + 3n + (10 + 5 + 15)*n**
**= 50 + 33*n**

```
Algorithm_B {
    set up the algorithm; /*taking 200 time units*/
    read in n elements into array A; /* 3 units per element */
    for (i = 0; i < n; i++) {
        do operation1 on A[i]; /* takes 10 units */
        do operation2 on A[i]; /* takes 5 units */
    }
}
```

**TB(n) = 200 + 3n + (10 + 5)*n**
**= 200 + 18*n**

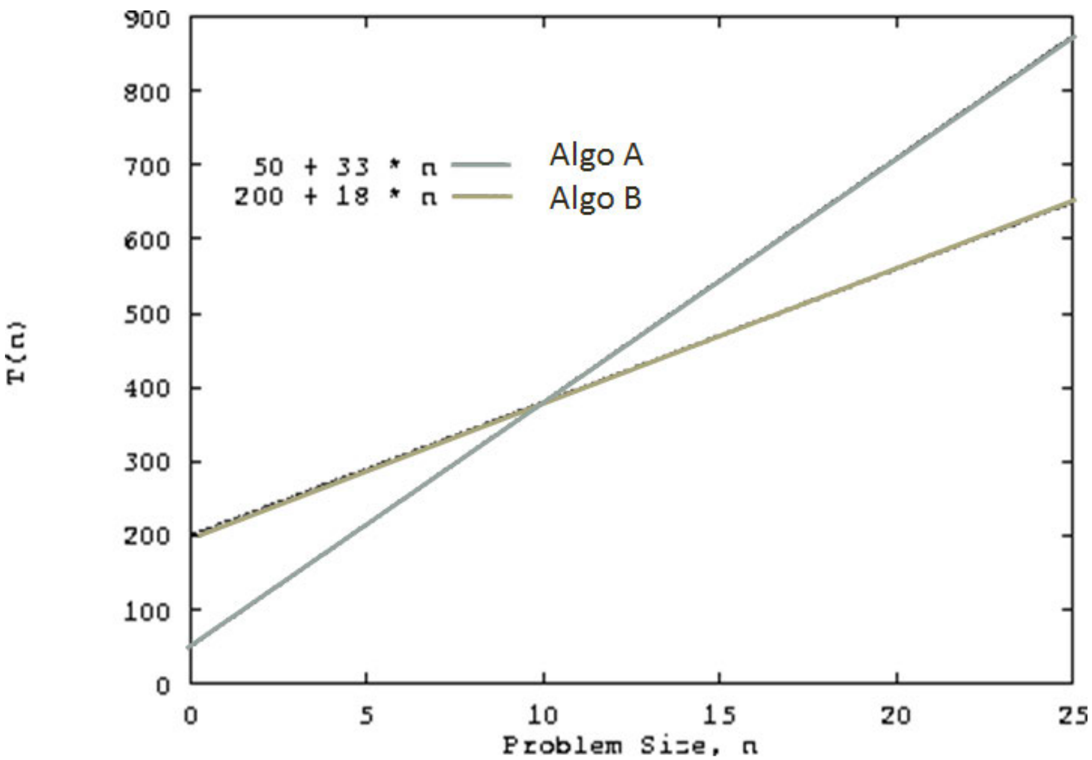
# Big Oh Does Not Tell the Whole Story

Both algorithms have time complexity O(n).

Algorithm A sets up faster than B, but does more operations on the data

Algorithm A is the better choice for small values of n.

For values of n > 10, algorithm B is the better choice



# A Misconception

A common misconception is that worst case running time is somehow defined by big-Oh, and that best case is defined by big-Omega.

There is no formal relationship like this.

However, worst case and big-Oh are commonly used together, because they are both techniques for finding an upper bound on running time.

# Summary

- $f(n) = O(g(n))$ if there exists positive constants $n_0$ and c such that $f(n) \leq c\, g(n)$ for all $n \geq n_0$.
- $f(n) = \Omega\ (g(n))$ if there exists positive constants $n_0$ and c such that $f(n) \geq c\, g(n)$ for all $n \geq n_0$.
- $f(n) = \Theta\ (g(n))$ if there exists positive constants $n_0$, $c_1$ and $c_2$ such that $c_1\, g(n) \leq f(n) \leq c_2\, g(n)$ for all $n \geq n_0$.
- $f(n) = o(g(n))$ if for any positive constant c there exists $n_0$ such that $f(n) > c\, g(n)$ for all $n \geq n_0$.
- $f(n) = \omega\ (g(n))$ if for any positive constant c there exists $n_0$ such that $f(n) < c\, g(n)$ for all $n \geq n_0$.