CS2x1:Data Structures and Algorithms

Koteswararao Kondepu

k.kondepu@iitdh.ac.in

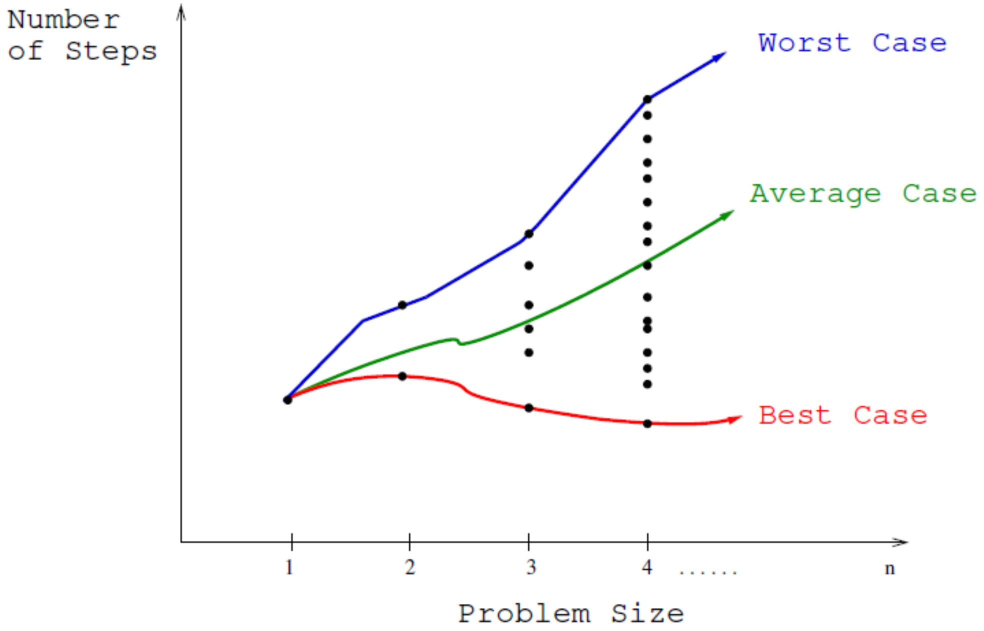
Recap (1)

- Introduction to algorithms
- Expression of algorithms
- Program
- Efficiency of algorithms
    - Running Time
    - Memory consumed
- Different time complexities

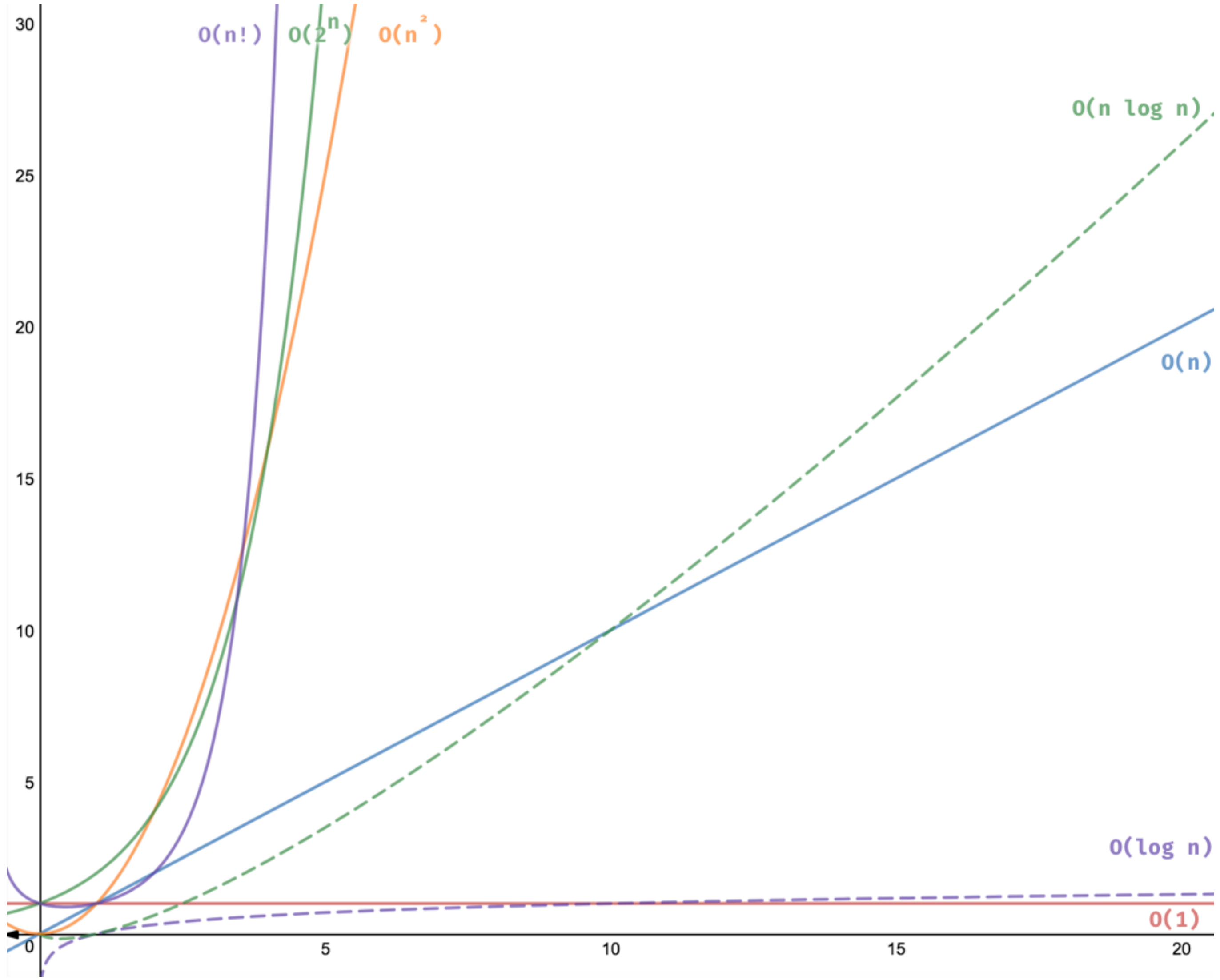
Recap (2)

- **Algorithm**: A sequence of computational steps that transform the ***input*** into the **output**
- The worst-case complexity of an algorithm → the maximum number of steps taken on any instance of size *n.*
- The average-case complexity of an algorithm → the average number of steps taken on any instance of size *n.*
- The best-case complexity of an algorithm → the minimum number of steps taken on any instance of size *n.*

Function → Time vs. Size



Algorithm Time Complexities



Exercise: Algorithm Introduction

Suppose a stack is to be implemented with a *linked list instead of an array*. What would be the effect on the time complexity of the *push and pop* operations of the stack implemented using linked list [Assumption: Stack is implemented efficiently] ?

A. O(1) for insertion and O(n) for deletion

B. O(1) for insertion and O(1) for deletion

C. O(n) for insertion and O(1) for deletion

D. O(n) for insertion and O(n) for deletion

insert_begin () and delete_begin ()
insert_begin () and delete_end ()

insert_end and delete_begin()
insert_end and delete_end ()

Asymptotic Analysis

- Analysis of <u>*Worst, Average and Best cases*</u> are difficult if there is no precise function exists.
- Asymptotic Goal: to simply the analysis of running time → may be affected by specific implementation and hardware
- *Worst, Average and Best cases* → *Upper bound and lower bounds*
- Required some kind of syntax to represent *Upper and lower bounds*

Asymptotic Notations	Symbol
<u>Worst</u>-case analysis	big-Oh → O–Notation
<u>Average</u>-case analysis	big-Theta → Θ–Notation
<u>Best</u>-case analysis	big-Omega → Ω–Notation

Asymptotic Notation: big-Oh → O

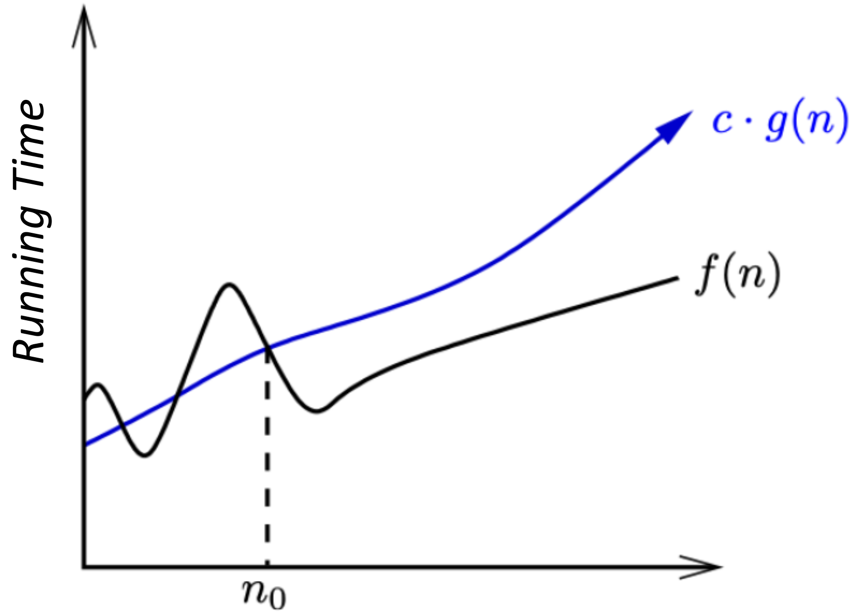
Asymptotic Notation: big-Oh → O

- Asymptotic *__tight Upper bound__* of the given function
- **Definition**: $f(n) = O\ (g(n))$, if there are positive constants c and n_0 such that $0 \leq f(n) \leq$ c. $g(n)$ for all $n \geq n_0$; $c > 0$
- $g(n)$ is an asymptotic *__tight upper bound__* for $f\ (n)$
- *Used to describe worst-case running time or upper bound for algorithmic problems*

Example: Find the upper bound of $f\ (n) = 3n+8$

$f(n) = O\ (g(n))$
$f(n) \leq$ c. $g(n)$

$3n+8 \leq$c. n



Exercise: big-Oh Notation

- **Definition**: $f(n) = O(g(n))$, if there are positive constants c and n_0 such that $0 \le f(n) \le c.\,g(n)$ for all $n \ge n_0$; $c > 0$

- *Find the upper bound of $f(n) = 4n$ and $g(n) = n^2$*

$$f(n) = O(g(n))$$
$$4n \le c.\,n^2$$

c =

$n_0 \ge$

c =

$n_0 \ge$

- *Find the upper bound of $f(n) = 2n^3 - 2n^2$ and $g(n) = n^3$*

$$f(n) = O(g(n))$$
$$2n^3 - 2n^2 \le c.\,n^3$$

Exercise: big-Oh Notation

- **Definition**: $f(n) = O(g(n))$, if there are positive constants c and n_0 such that $0 \leq f(n) \leq c.\, g(n)$ for all $n \geq n_0$; $c > 0$

- *Find the upper bound of $f(n) = 7n-2$ and $g(n) = n$*

$$f(n) = O(g(n))$$
$$7n-2 \leq c.\, n$$

c =

$n_0 \geq$

- *Find the upper bound of $f(n) = 963$*

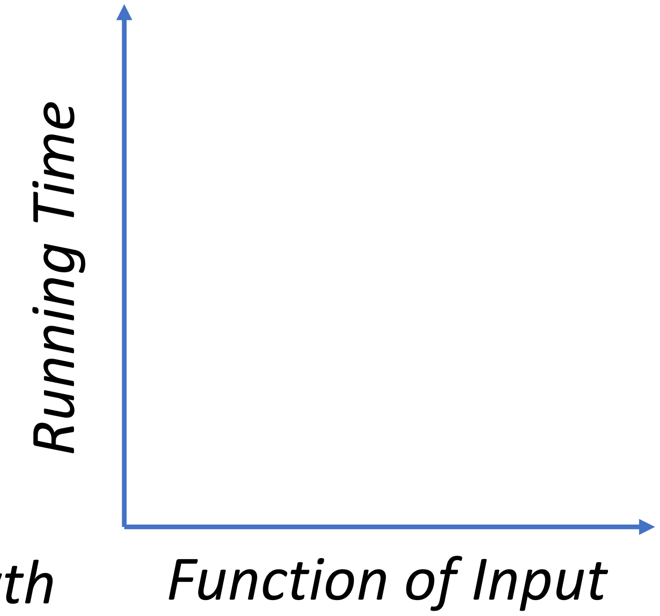
c =

$n_0 \geq$

What is Rate of Growth?

- The rate at which the running time increases as a function of input is called ***Rate of Growth***

- $f(n) = n^4 + 2n^2 + 100n + 500$



n^4 is the highest rate of growth

- DO NOT Ignore the constant that is not multipliers:

  n^4 is $O(n^3)$ → *TRUE/FALSE?*

  4^n is $O(2^n)$ → *TRUE/FALSE?*

Big-Oh: Growth Rate

- The big-Oh notation gives a ***tight upper bound*** on the growth rate of a function

- *f(n) is O(g(n))* → *the rate of f(n) is no more than the growth of the g(n)*

- *g(n) is O(f(n))* → *the rate of g(n) is no more than the growth of the f(n)*

- *g (n) growth is more* → *select the correct option from the below:*

  (i) f(n) is O (g(n)) *(ii) g (n) is O (f(n))* *Ans:*

- *f (n) growth is more* → *select the correct option from the below:*

  (i) f(n) is O (g(n)) *(ii) g (n) is O (f(n))* *Ans:*

Big-Oh: Rules

- Drop lower order terms and constant factors

  Example: 100 n log n → O (n log n)

  7n – 2 → O (n)

- *Use the smallest possible class of functions*

  Example: 3n is O (n) instead of 3n is O (n^2)

- *Use the simplest expression of the class*

  Example: 3n+2 is O (n) instead of 3n+2 is O (3n)

- Comparing asymptotic running time
  - ✓ an algorithm that runs $O(n)$ is better than one that runs $O(n^2)$
  - ✓ functions hierarchy:

$$1 < \log\log n < \sqrt{\log n} < \log^2 n < 2^{\log n} < n < \log(n!) < n\log n < n^2 < 2^n < 4^n < n! < 2^{2^n}$$

Asymptotic Notation: big-Omega → Ω

Asymptotic Notation: big-Omega → Ω

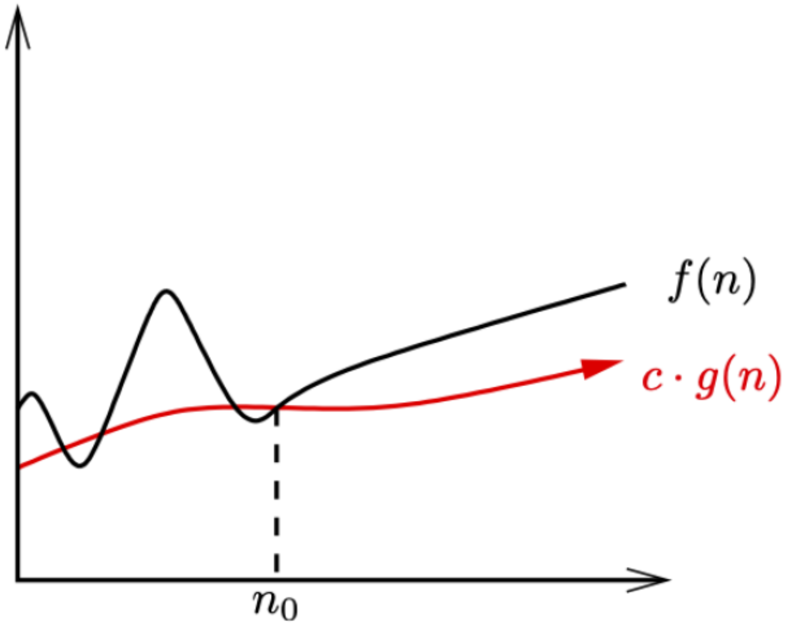
- Asymptotic <u>*tight lower bound*</u> of the given function
- **Definition**: $f(n) = \Omega(g(n))$, if there are positive constants c and n_0 such that $f(n) \geq c.\, g(n)$ for all $n \geq n_0$; $c > 0$; $n_0 \geq 1$
- $g(n)$ is an asymptotic <u>*tight lower bound*</u> for $f(n)$
- *Used to describe best-case running time or lower bound for algorithmic problems*

  Example: *Find the lower bound of $f(n) = 3n+8$*

$$f(n) = \Omega(g(n))$$
$$f(n) \geq c.\, g(n)$$

$$3n+8 \geq c.\, n$$
$$3n+8 \rightarrow \Omega(n)$$

$c = 1$; $n_0 \geq 1$



- **Select the closest lower-bound for the given f(n) = 3n+8:**

  (i) $\Omega(n)$ (ii) $\Omega(\log n)$ (iii) $\Omega(\log\log n)$ (iv) $\Omega(n \log n)$

Exercise: big-Omega → Ω

- *Find the lower bound for* $f(n) = 3n+8$ *and* $g(n) = n^2$

$$f(n) = \Omega(g(n))$$
$$3n+8 \geq c.\ n^2$$

c =

$n_0 \geq$

- *Find the lower bound for* $f(n) = 5n^2$ *and* $g(n) = n^2$

$$f(n) = \Omega(g(n))$$
$$5n^2 \geq c.\ n^2$$

c =

$n_0 \geq$

- *Find the lower bound for* $f(n) = 5n^2$ *and* $g(n) = n$

c =

$n_0 \geq$

$$f(n) = \Omega(g(n))$$
$$5n^2 \geq c.\ n$$

Asymptotic Notation: big-Theta → Θ

Asymptotic Notation: big-Theta → Θ

- **Definition**: $f(n) = \Theta(g(n))$, if there are positive constants c and n_0 such that $0 \le c_1.\ g(n) \le f(n) \le c_2.\ g(n)$, $f(n) \ge c.\ g(n)$ for all $n \ge n_0$; $c_1, c_2 > 0$; $n_0 \ge 1$
- $g(n)$ is an asymptotic <u>*tight bound*</u> for $f(n)$
- *Used to describe <u>average-case running time</u>*

Example#1: Find the Θ bound of $f(n) = 3n+1$

$f(n) = O(g(n))$	$f(n) = \Omega(g(n))$
$f(n) \le c_2.\ g(n)$	$c_1.\ g(n) \le f(n)$
$3n+1 \le c_2.\ n$	$c_1.\ n \le 3n+1$

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

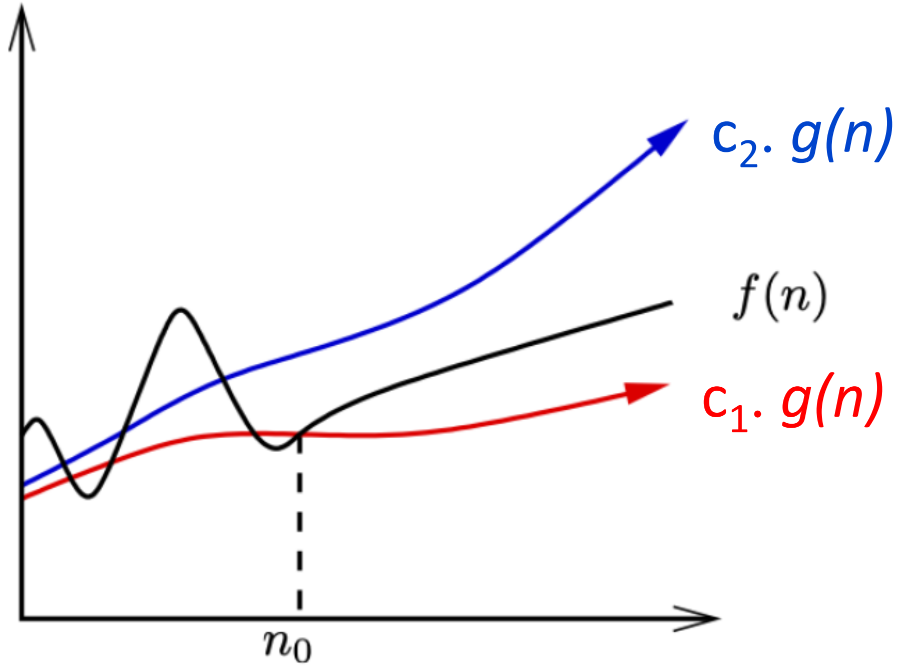
$c_1.\ n \le 3n+1 \le c_2.\ n$

Example#2: Find the Θ bound of $f(n) = \frac{n^2}{2} - \frac{n}{2}$

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

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

$c_1.\ n^2 \le \frac{n^2}{2} - \frac{n}{2} \le c_2.\ n^2$



f(n) is Θ(g(n)) if and only if f(n) is O(g(n)) and f(n) is Ω(g(n))

Math: Need to review

- properties of logarithms:

  $\log_b(xy) = \log_b x + \log_b y$

  $\log_b (x/y) = \log_b x - \log_b y$

  $\log_b xa = a\log_b x$

  $\log_b a = \log_x a/\log_x b$

- properties of exponentials:

  $a^{(b+c)} = a^b a^{c}$

  $a^{bc} = (a^b)^c$

  $a^b /a^c = a^{(b-c)}$

  $b = a^{\log_a b}$

  $b^c = a^{c*\log_a b}$

- Floor: $\lfloor x \rfloor$ = the largest integer ≤ x
- Ceil: $\lceil x \rceil$ = the smallest integer ≥ x

Exercise: Find a number from the given array

a:	10	20	30	40	50	60
Index	0	1	2	3	4	5

```
for (i = 0; i < n; i++) {
    if (a[i] == findNum)  {
      printf("\n%d is present at location %d\n", findNum, i+1);
        break; }
  }
```

What is the time complexity for finding element 10 from the given list?

(i) O (1) (ii) Θ (1) (iii) Ω (1)

What is the time complexity for finding element 40 from the given list?

(i) O (n) (ii) Θ (n) (iii) Ω (n)

What is the time complexity for finding element 70 from the given list?

(i) O (n) (ii) Θ (n) (iii) Ω (n)

Asymptotic Analysis: Loops (1)

- ```
  for(i=1;i <=n; i++)
          s = s+1;  //statement
  ```

  ```
  k=0
  ```
- ```
  for(i=1; k <=n; i++)
          k = k+i;  //statement
  ```

- ```
  for(i=1;i<=n; i = i * 2)
      printf ("CS2x1"); //statement
  ```
  ```

Asymptotic Analysis: Loops (2)

- ```
  for(i=1;i<=n;i++)
      for(j=1;j<=n;j++)
              s = s + 1; //statement
  ```

- ```
  for(i=1; i<=n; i++)
      for(j=1; j <= i; j++)
              s= s+1; //statement
  ```

- ```
  for (i=1; i<n; i++)
     m=m+2;
  for(k=0; k<n; k++)
     for(j=1; j <= n; j++)
         s= s+1; //statement
  ```
  ```

Asymptotic Analysis: Loops (3)

- ```
  k=0;
  for (i=1; i<n; i=i*2)
         k++;
   for(j=1; j<k; j=j*2)
          s= s+1; //statement
  ```

- ```
  for (i=1; i<n; i++)
   for(j=1; j<n; j=j*2)
          s= s+1; //statement
  ```

Asymptotic Analysis: Loops (4)

- ```
  void fun (int n) {
    int i=1, s=1;
    while (s<=n) {
        i++;
        s=s+i;
     }
  }
  ```
  ```

Asymptotic Notation: little-Oh → o

- **Definition**: $f(n) = o\,(g(n))$, if there are positive constants c and n_0 such that $0 \leq f(n) <$ c. $g(n)$ for all $n \geq n_0$; $c > 0$; $n_0 > 0$

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

  - $f(n) = \text{o}\,(g(n))$
    - ✓ $2n = o\,(n^2)$
    - ✓ $2n^2 \neq o\,(n^2)$

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

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

Asymptotic Notation: little-Omega → ω

- **Definition**: $f(n) = \omega(g(n))$, if there are positive constants c and n_0 such that $0 \leq c.\ g(n) < f(n)$ for all $n \geq n_0$; $c > 0$; $n_0 > 0$

- *g(n) is an <u>lower bound for</u> f(n) that is <u>not asymptotically tight</u>.*

  - $f(n) = \omega(g(n))$
    - ✓ $n^2/2 = \omega(n)$
    - ✓ $n^2/2 \neq \omega(n^2)$

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

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

Notation Properties

- **Transitivity**
  - ✓ $f(n) = \Theta(g(n))$ & $g(n) = \Theta(h(n)) \Rightarrow f(n) = \Theta(h(n))$
  - ✓ $f(n) = O(g(n))$ & $g(n) = O(h(n)) \Rightarrow f(n) = O(h(n))$
  - ✓ $f(n) = \Omega(g(n))$ & $g(n) = \Omega(h(n)) \Rightarrow f(n) = \Omega(h(n))$

- **Reflexivity**
  - ✓ $f(n) = \Theta(f(n))$; $f(n) = O(f(n))$; $f(n) = \Omega(f(n))$

- **Symmetry**
  - ✓ $f(n) = \Theta(g(n))$ iff $g(n) = \Theta(f(n))$

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

thank you!

email:
k.kondepu@iitdh.ac.in

NEXT Class: 11/05/2023