# Design and Analysis of Algorithm

Lecture-2:

# Contents

# Asymptotic notations

When the input sizes are large enough to make only the order of growth of the running time relevant, we are interested in asymptotic efficiency of algorithms.

## There are mainly three asymptotic notations:

- Big-O notation
- Theta notation
- Omega notation

# Big-O notation (O)

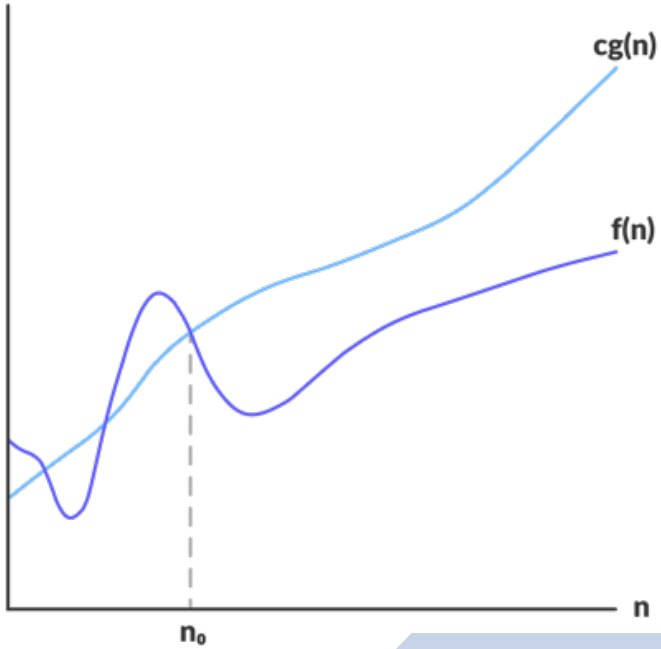
Big-O notation represents the upper bound of the running time of an algorithm. Thus, it gives the worst case complexity of an algorithm.

Let $f(n)$ and $g(n)$ be two $+ve$ function

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

$f(n)$ is order of $g(n)$ if and only if there exists two constant $c > 0$ && $n_0 > 0$

Such that $0 \leq f(n) \leq c\big(g(n)\big), \forall n, n \geq n_0$



# Omega notation (Ω)

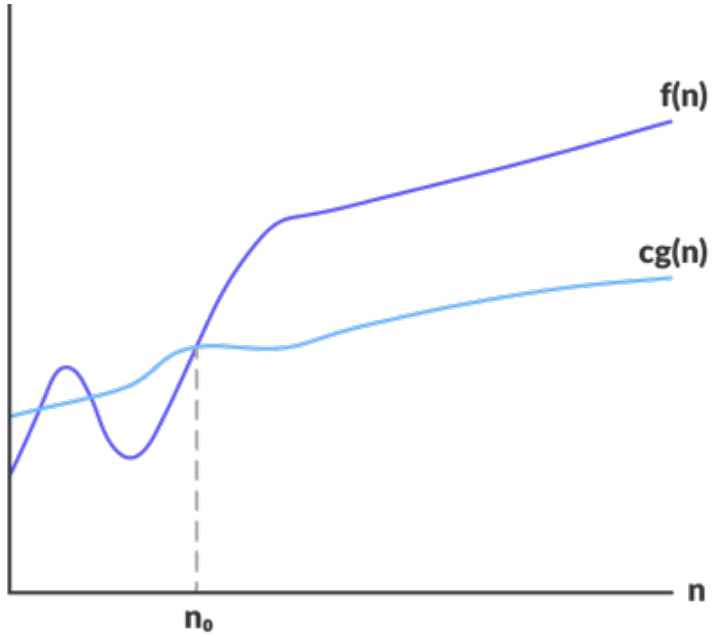
Omega notation represents the lower bound of the running time of an algorithm. Thus, it provides best case complexity of an algorithm.

Let $f(n)$ and $g(n)$ be two $+ve$ function

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

$f(n)$ is omega of $g(n)$ if and only if there exists two constant $c > 0$ && $n_0 > 0$

Such that $0 \leq c\big(g(n)\big) \leq f(n), \forall n, n \geq n_0$



# Theta notation ($\theta$)

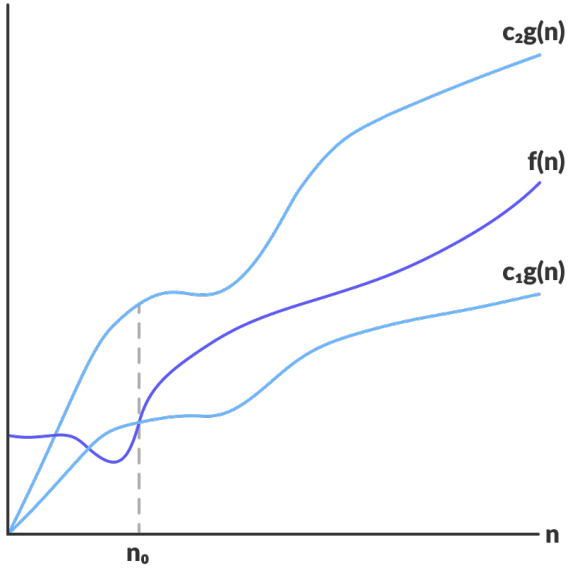
Theta notation encloses the function from above and below. Since it represents the upper and the lower bound of the running time of an algorithm, it is used for analysing the average case complexity of an algorithm

Let $f(n)$ and $g(n)$ be two $+ve$ function

$$f(n) = \theta\big(g(n)\big)$$

$f(n)$ is theta of $g(n)$ if and only if there exists three constant $c_1 > 0, c_2 > 0$ && $n_0 > 0$

Such that $0 \leq c_1\big(g(n)\big) \leq f(n) \leq c_2\big(g(n)\big), \forall n, n \geq n_0$



# Small-O notation (o)

Let $f(n)$ and $g(n)$ be two $+ve$ function

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

$f(n)$ is order of $g(n)$ if and only if there exists two constant $c > 0$ && $n_0 > 0$

Such that $0 \leq f(n) < c\big(g(n)\big), \forall n, n \geq n_0$

# Little omega notation ($\omega$)

Let $f(n)$ and $g(n)$ be two $+ve$ function

$$f(n) = \omega\big(g(n)\big)$$

$f(n)$ is omega of $g(n)$ if and only if there exists two constant $c > 0$ && $n_0 > 0$

Such that $0 \leq c\big(g(n)\big) < f(n), \forall n, n \geq n_0$

# Questions

(1) $f(n) = n \; and \; g(n) = n^2$

Which of the following is true
(a) $fn = O(g(n))$
(b) $fn = \Omega(g(n))$
(c) $fn = \theta(g(n))$

(a)

(2) $f(n) = n^2 \; and \; g(n) = 2^n$

Which of the following is true
(a) $fn = O(g(n))$
(b) $fn = \Omega(g(n))$
(c) $fn = \theta(g(n))$

(a)

(3) Let k and m are constants

(a) $(n + k)^m = \Theta(n^m)$,
(b) $2^{n+1} = O(2^n)$
(c) $2^{2n+1} = O(2^n)$

Which of the following is not correct.

(c)

# Properties of asymptotic notations

**Transitivity**

$$
\begin{array}{lllll}
f(n) = \Theta(g(n)) & \text{and} & g(n) = \Theta(h(n)) & \text{imply} & f(n) = \Theta(h(n)) , \\
f(n) = O(g(n)) & \text{and} & g(n) = O(h(n)) & \text{imply} & f(n) = O(h(n)) , \\
f(n) = \Omega(g(n)) & \text{and} & g(n) = \Omega(h(n)) & \text{imply} & f(n) = \Omega(h(n)) , \\
f(n) = o(g(n)) & \text{and} & g(n) = o(h(n)) & \text{imply} & f(n) = o(h(n)) , \\
f(n) = \omega(g(n)) & \text{and} & g(n) = \omega(h(n)) & \text{imply} & f(n) = \omega(h(n)) .
\end{array}
$$

**Reflexivity**

$$
\begin{aligned}
f(n) &= \Theta(f(n)) , \\
f(n) &= O(f(n)) , \\
f(n) &= \Omega(f(n)) .
\end{aligned}
$$

**Symmetry**

$f(n) = \Theta(g(n))$ if and only if $g(n) = \Theta(f(n))$

# Recursion

## Definition

*A function calling itself to solve a particular problem is called a recursion*

e.g. $fact(5) = 5 * fact(4)$
$= 4 * fact(3)$
$= 3 * fact(2)$
$= 2 * fact(1)$

# Properties of recursion

- Recursive step
- Termination condition
- There should be change in its parametric value

$$\text{e.g. } fact(n) = \begin{cases} 1 & if\ n = 1 \\ n * fact(n-1) & otherwise \end{cases}$$

# Complexity of recurrence relation

- Substitution Method
- Recursion tree method
- Master Method

# Substitution Method

Substitute the given method repeatedly until function is removed.

$$F(n) = 2F(n/2) + n$$
$$F(1)=1$$

# Substitution Method

$F(n) = 2F(n/2) + n$

$=2[2F(\frac{n}{2*2})+n/2] + n \quad \Rightarrow 2^2F\left(\frac{n}{2^2}\right) + 2 * \frac{n}{2} + n \quad \Rightarrow 2^2F\left(\frac{n}{2^2}\right) + 2n$

Now if F(n)=2F(n/2)+ n then what will be the value of $F(\frac{n}{2^2})$ ??

$2^2\left[2F\left(\frac{n}{2*2^2}\right) + \frac{n}{2^2}\right] + 2n \Rightarrow 2^3F\left(\frac{n}{2^3}\right) + 3n$

$2^kF\left(\frac{n}{2^k}\right) + kn$ When it will terminate ??

If $2^k = n \Rightarrow k = \log_2 n$

$2^{\log_2 n}F(1) + n.\log_2 n \Rightarrow n + n.\log_2 n \Rightarrow O(n.\log_2 n)$

# Question

$F(n) = F(n/2) + n$

F(1)=1

$$F(n) = F(n/2) + n$$

$$= F\left(\frac{n}{2\times 2}\right) + \frac{n}{2} + n \Rightarrow F\left(\frac{n}{2^2}\right) + \frac{n}{2} + n$$

$$= F\left(\frac{n}{2^3}\right) + \frac{n}{4} + \frac{n}{2} + n$$

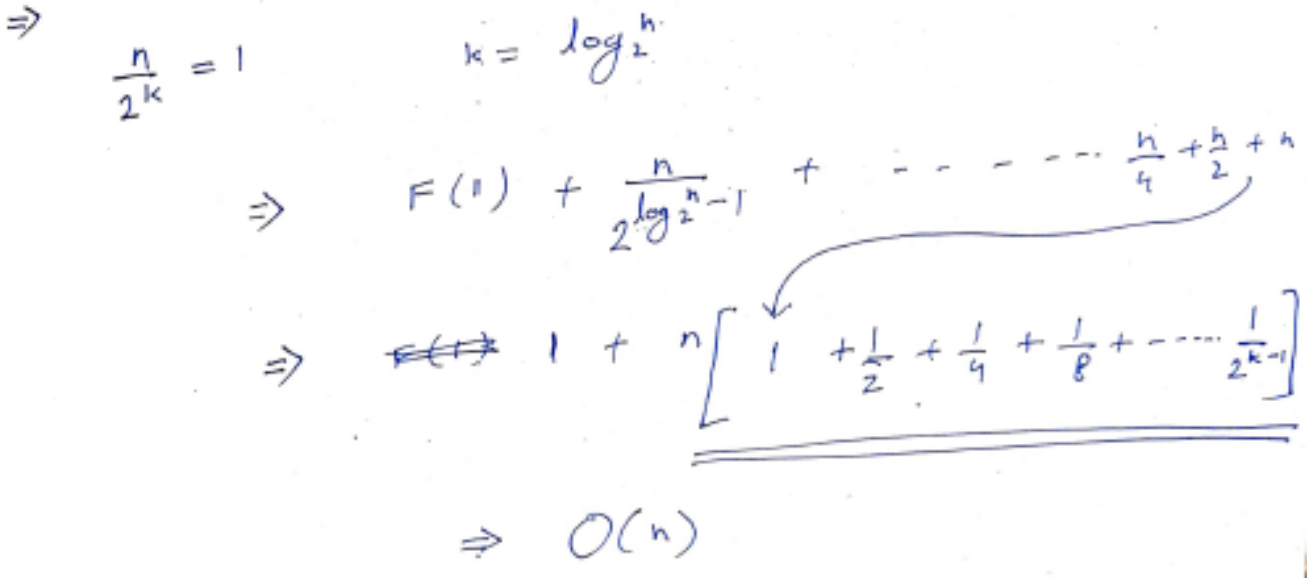
$$= F\left(\frac{n}{2^4}\right) + \frac{n}{8} + \frac{n}{4} + \frac{n}{2} + n$$

$$\vdots$$

$$= F\left(\frac{n}{2^k}\right) + \frac{n}{2^{k-1}} + \cdots\cdots\cdots \frac{n}{4} + \frac{n}{2} + n$$

# Question

$$F(n) = F(n/2) + n$$



# Question

$F(n) = 2F(n/2) + 2$

F(1)=1

$$F(n) = 2(F(n/2)) + 2$$

$$\Rightarrow 2\left[2F\left(\frac{n}{4}\right) + 2\right] + 2 \quad \Rightarrow 2^2 F\left(\frac{n}{2^2}\right) + 2^2 + 2$$

$$= 2^3 F\left(\frac{n}{2^3}\right) + 2^3 + 2^2 + 2$$

$$\vdots$$

$$\Rightarrow 2^k F\left(\frac{n}{2^k}\right) + 2^k + 2^{k-1} \quad \cdots\cdots 2$$

# Question

$$F(n) = 2F(n/2) + 2$$

$\frac{n}{2^k} = 1 \qquad k = \log_2^n$

$\Rightarrow 2^{\log_2^n} + 2^{\log_2^n} + \cdots\cdots 2^3 + 2^2 + 2$

$\Rightarrow n + 2\left[1 + 2 + 2^2 + 2^3 + \cdots 2^{\log_2^n - 1}\right]$

Series

$1 + 2 + 2^2 + 2^3 + \cdots\cdots 2^{t-1} = 2^t - 1$

$\Downarrow$

$n + 2\left[2^{\log_2^n} - 1\right] = 3n - 2$

$= O(n)$

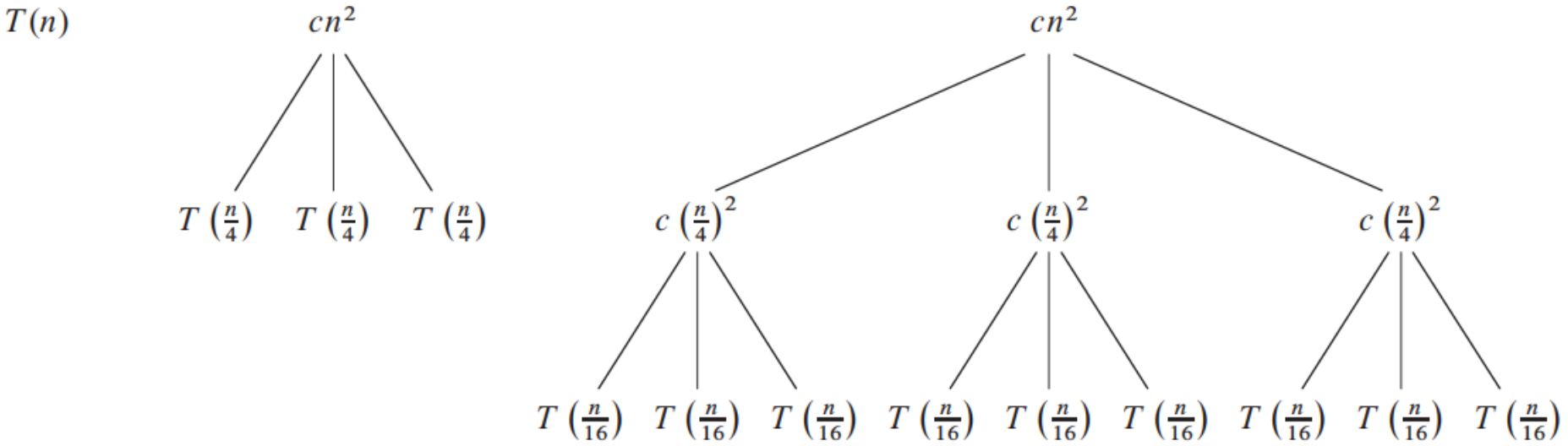
# The recursion-tree method

In a recursion tree, each node represents the cost of a single sub problem somewhere in the set of recursive function invocations.

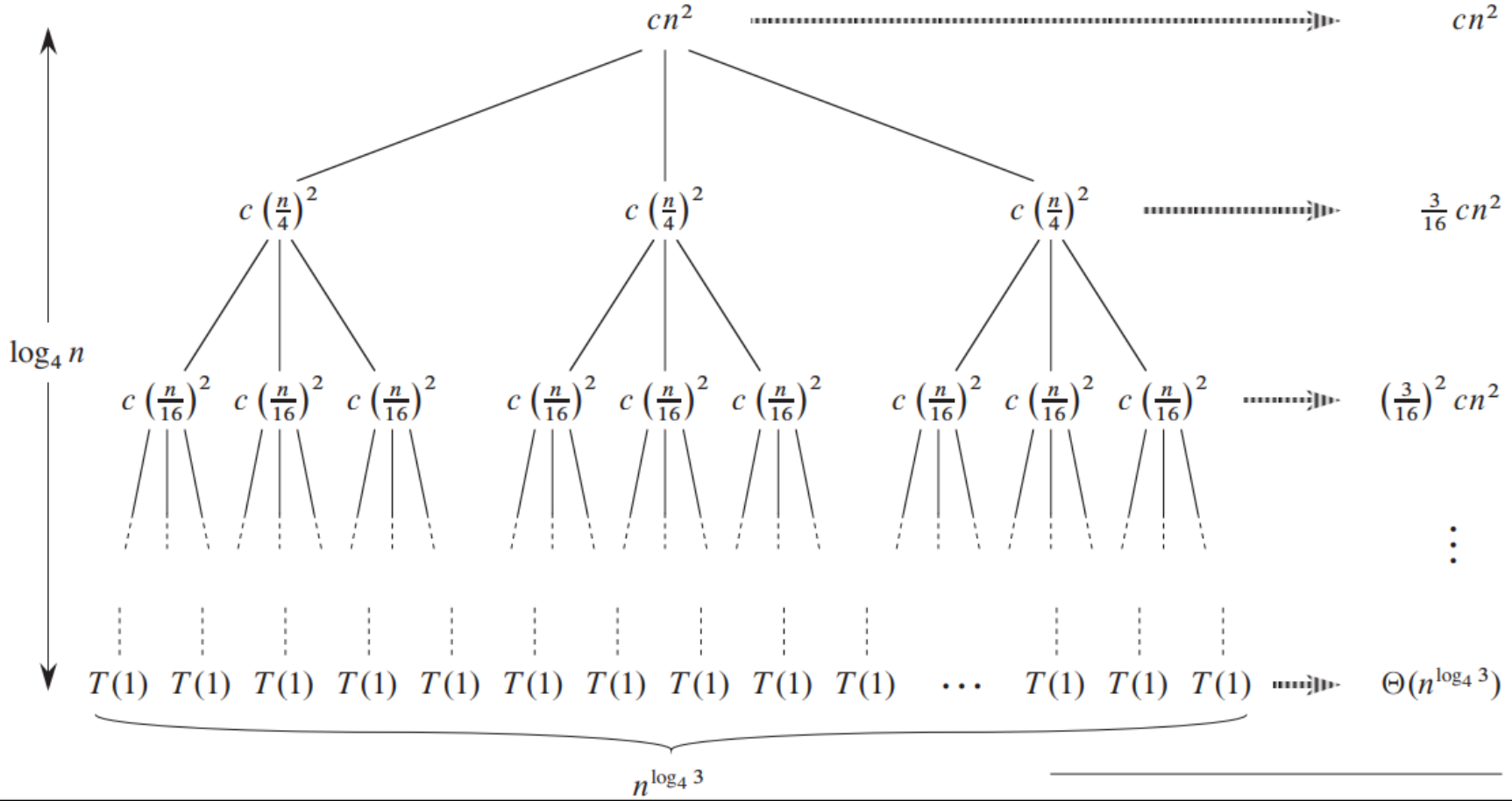
$$T(n) = 3.T\left(\frac{n}{4}\right) + cn^2$$

# Solution

$$T(n) = 3.T\left(\frac{n}{4}\right) + cn^2$$



# Solution


$cn^2$
$cn^2$
$c\left(\frac{n}{4}\right)^2$
$c\left(\frac{n}{4}\right)^2$
$c\left(\frac{n}{4}\right)^2$
$\frac{3}{16}cn^2$
$\log_4 n$
$c\left(\frac{n}{16}\right)^2$
$c\left(\frac{n}{16}\right)^2$
$c\left(\frac{n}{16}\right)^2$
$c\left(\frac{n}{16}\right)^2$
$c\left(\frac{n}{16}\right)^2$
$c\left(\frac{n}{16}\right)^2$
$c\left(\frac{n}{16}\right)^2$
$c\left(\frac{n}{16}\right)^2$
$c\left(\frac{n}{16}\right)^2$
$\left(\frac{3}{16}\right)^2 cn^2$
$T(1)$ $T(1)$ $T(1)$ $T(1)$ $T(1)$ $T(1)$ $T(1)$ $T(1)$ $T(1)$ $T(1)$ $\cdots$ $T(1)$ $T(1)$ $T(1)$
$\Theta(n^{\log_4 3})$
$n^{\log_4 3}$

# Master Theorem

The master method provides a “cookbook” method for solving recurrences of the form

$T(n) = a\,T(n/b) + f(n)$ , where, $a \geq 1$ and $b > 1$

1. if $f(n) = O(\,n^{\log_b a - \epsilon}\,)$ where $\epsilon > 0$
   then $T(n) = \theta(\,n^{\log_b a}\,)$
2. if $f(n) = \theta\big(\,n^{\log_b a}\,\big)$
   then $T(n) = \theta(\,n^{\log_b a}\,\lg(n)\,)$
3. if $f(n) = \Omega(\,n^{\log_b a + \epsilon}\,)$
   if $af\left(\frac{n}{b}\right) \leq cf(n)$ for some c<1
   then $T(n) = \theta(f(n))$

# Numerical on Master Theorem

$T(n) = 9T(n/3) + n$

Step 1: $a = 9 \quad and \quad b = 3 \quad and \quad f(n) = n$
Step 2: compute $\log_b a \Rightarrow \log_3 9$
Step 3: compute $n^{\log_b a} \Rightarrow n^{\log_3 9} \Rightarrow n^2$
Step 4: compare $f(n)$ with $n^{\log_b a} \Rightarrow n < n^2$
Step 5: Choose the case: Case 1
Step 6: $T(n) = \theta(n^2)$

$T(n) = T(2n/3) + 1$

Step 1: $a = 1 \quad and \quad b = 3/2 \quad and \quad f(n) = 1$
Step 2: compute $\log_b a \Rightarrow \log_{3/2} 1$
Step 3: compute $n^{\log_b a} \Rightarrow n^{\log_{3/2} 1} \Rightarrow 1$
Step 4: compare $f(n)$ with $n^{\log_b a} \Rightarrow 1 = 1$
Step 5: Choose the case: Case 2
Step 6: $T(n) = \theta(\log(n))$

$T(n) = 3T(n/4) + nlog(n)$

Step 1: $a = 3 \quad and \quad b = 4 \quad and \quad f(n) = n\ log\ (n)$
Step 2: compute $\log_b a \Rightarrow \log_4 3$
Step 3: compute $n^{\log_b a} \Rightarrow n^{\log_4 3} \Rightarrow n^{0.793}$
Step 4: compare $f(n)$ with $n^{\log_b a} \Rightarrow nlog(n) > n^{.793}$
Step 5: Choose the case: Case 3
Step 6: $af\left(\frac{n}{b}\right) = 3\left(\frac{n}{4}\right)\lg\left(\frac{n}{4}\right) \leq \frac{3}{4}nlog(n) = cf(n); c = 3/4$

$T(n) = \theta(nlog(n))$

# Numerical on Master Theorem

(a) $T(n) = T(n/2) + n$

**Answer:** T(n) =$\theta(n)$

(b) $T(n) = 2T(n/2) + 2$

**Answer:** T(n) =$\theta(n)$

(c) $T(n) = 4T(n/2) + n$

**Answer:** T(n) =$\theta(n^2)$

(d) $T(n) = 8T(n/2) + n^2$

**Answer:** T(n)=$\theta(n^3)$

Assignment Question:

(1) $T(n) = 2T(\sqrt{n}) + c$
(2) $T(n) = T(\sqrt{n}) + \log(n)$
(3) $T(n) = T(\sqrt{n}) + c$