CS 2009
Design and Analysis of Algorithms

Waheed Ahmed
Email : waheedahmed@nu.edu.pk

RECURRENCE RELATIONS

Tool for describing runtimes of *recursive algorithms*

Recurrences and Running Time

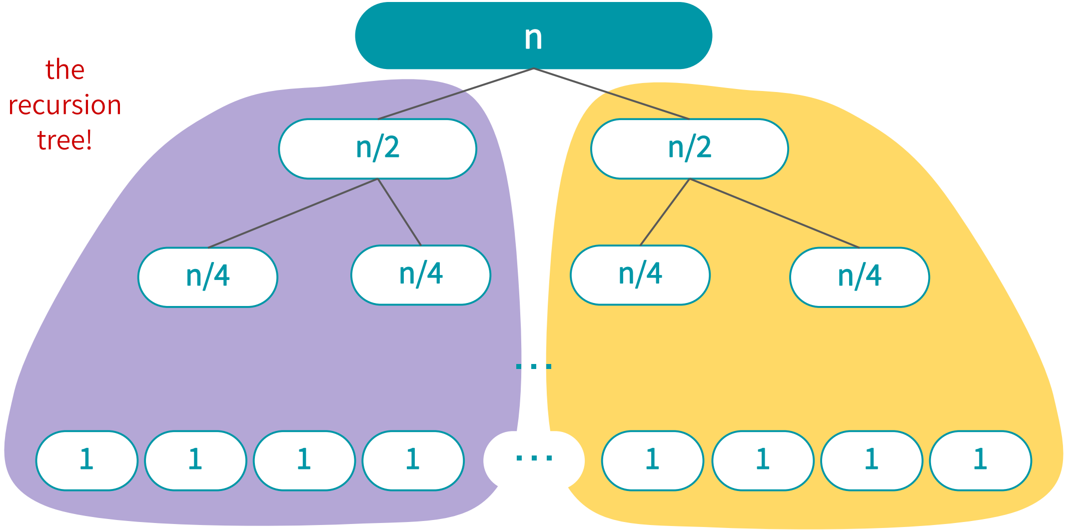
- An equation or inequality that describes a function in terms of its value on smaller inputs.

$$T(n) = T(n-1) + n$$

- Recurrences arise when an algorithm contains recursive calls to itself
- What is the actual running time of the algorithm?
- Need to solve the recurrence
    - Find an explicit formula of the expression
    - Bound the recurrence by an expression that involves n

RECURRENCE RELATIONS

To build the recurrence relation for MergeSort, we can think of its runtime as follows:



T(n) =

T(n/2)

+

T(n/2)

+

O(n)

RECURRENCE RELATIONS

To build the recurrence relation for MergeSort, we can think of its runtime as follows:

$$T(n) = \underbrace{\boxed{T(n/2)} + \boxed{T(n/2)}}_{\textit{since the subproblems are equal sizes, we can also write this as } \boldsymbol{2 \cdot T(n/2)}} + \boxed{O(n)}$$

This is a *recursive* definition for T(n), so we also need a BASE CASE:

$$T(1) = O(1)$$

$$T(n) = \begin{cases} O(1) & \text{if } n = 1 \\ 2\,T(n/2) + O(n) & \text{if } n > 1 \end{cases}$$

Methods for Solving Recurrences

- Recursion tree method
- Iterative Substitution method
- Guess and Test method
- Master method

RECURSION TREE

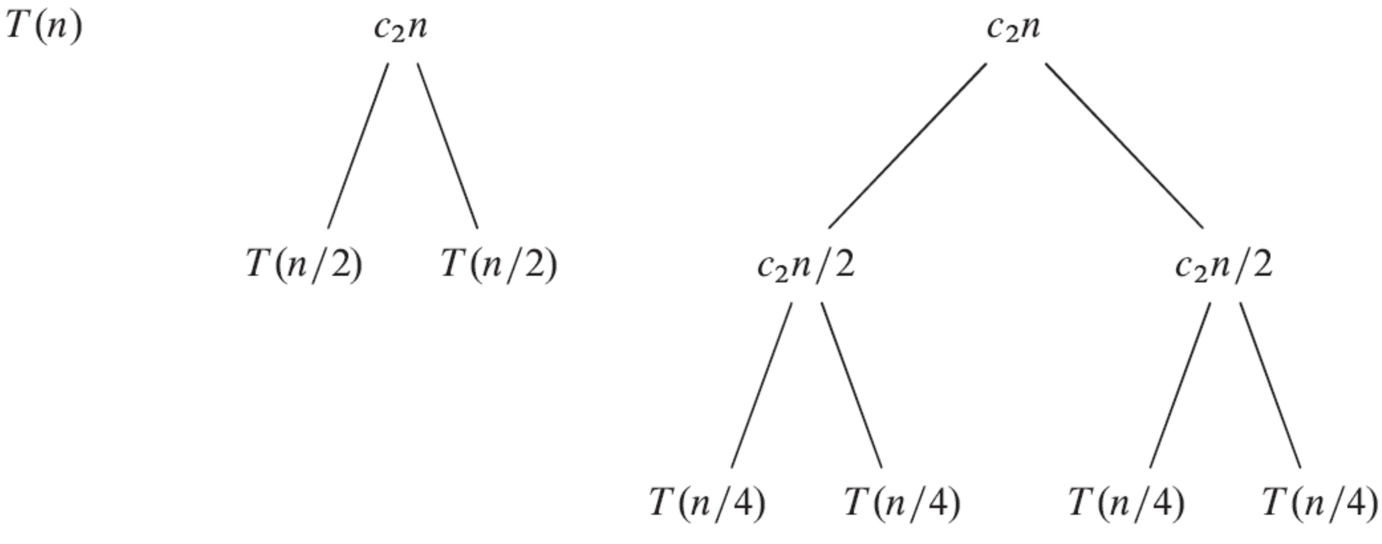
Recursion-tree Method

- ***Convert the recurrence into a tree:***
  1. Each node represents the cost incurred at various levels of recursion
  2. Sum up the costs of all levels

Solve the Recurrence

$$T(n) = \begin{cases} c_1 & \text{if } n = 1 \\ 2T(n/2) + c_2 n & \text{if } n > 1 \end{cases}$$

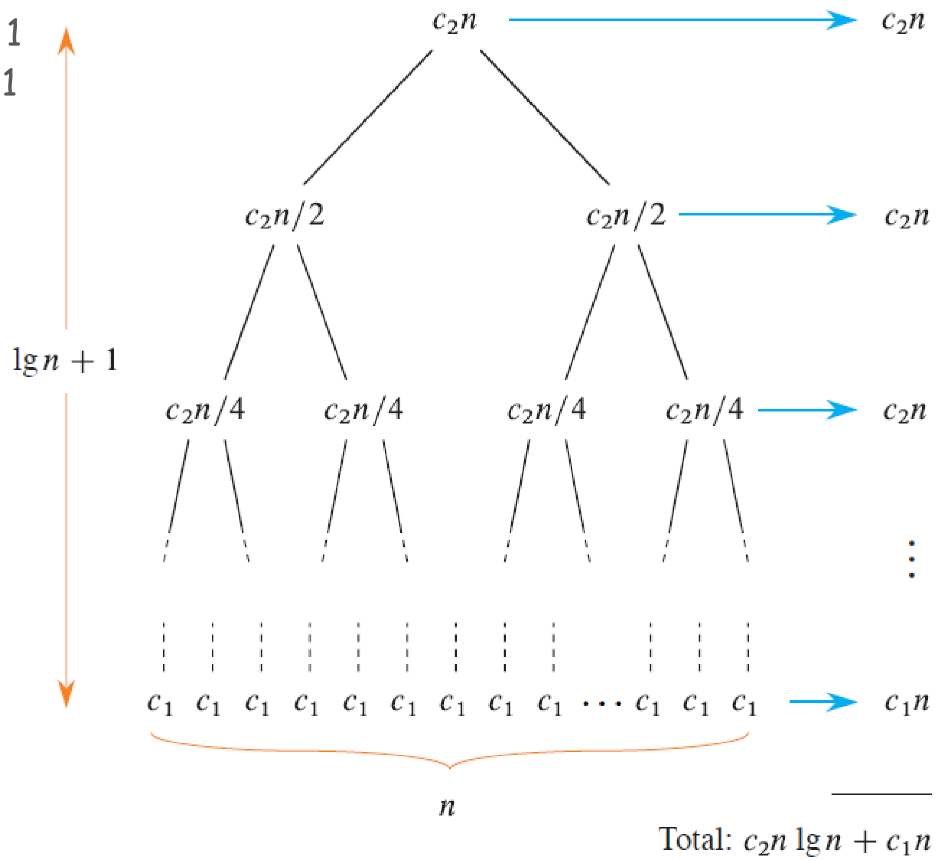
Recurrence Tree method



(a) (b) (c)

Solve the Recurrence (Recurrence Tree)

$$T(n) = \begin{cases} c_1 & \text{if } n = 1 \\ 2T(n/2) + c_2 n & \text{if } n > 1 \end{cases}$$

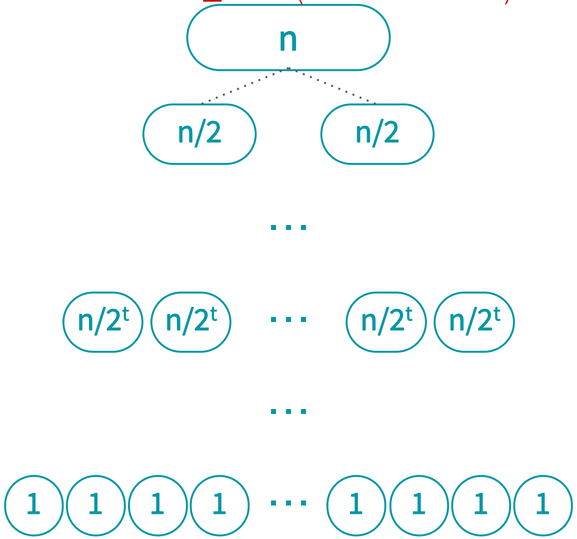


Number of levels calculation

- The array of size n = 4 has 3 levels
- The array of size n = 8 has 4 levels
- The array of size n = 16 has 5 levels
- The array of size n = 32 has 6 levels
- The array of size n = 2^5 has **5**+1 levels
- The array of size n = 2^k has **k**+1 levels
- $n = 2^k$ => $\log.n = \log.2^k$ => $\log.n = k.\log.2$
- $\log.n = k$

MERGESORT COMPLEXITY USING RECURSION TREE

If a subproblem is of size **n**, then the work done in that subproblem is **O(n)**.
⇒ **Work ≤ c · n** (c is a constant)



Level	# of Problems	Size of each Problem	Work done per Problem ≤	Total work on this level
0	1	n	$c \cdot n$	$O(n)$
1	2^1	$n/2$	$c \cdot (n/2)$	$2^1 \cdot c \cdot (n/2) = \mathbf{O(n)}$
. . .				
t	2^t	$n/2^t$	$c \cdot (n/2^t)$	$2^t \cdot c \cdot (n/2^t) = O(n)$
. . .				
$\log_2 n$	$2^{\log_2 n} = n$	1	$c \cdot (1)$	$n \cdot c \cdot (1) = \mathbf{O(n)}$

We have $(\log_2 n + 1)$ levels, each level has $O(n)$ work total ⇒ **O(n log n)** work overall!

Example of recursion tree

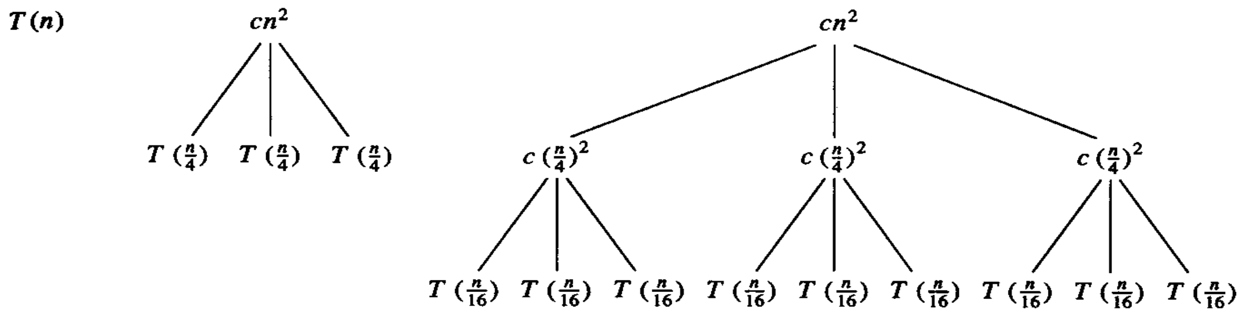
$$T(n) = 4\,T(n/4) + n + 50$$

Example of recursion tree

$$T(n) = 2\,T(n/2) + 3n^2 + 5n.$$

Example of recursion tree

$T(n) = 3\,T(n/4) + O(n^2)$.

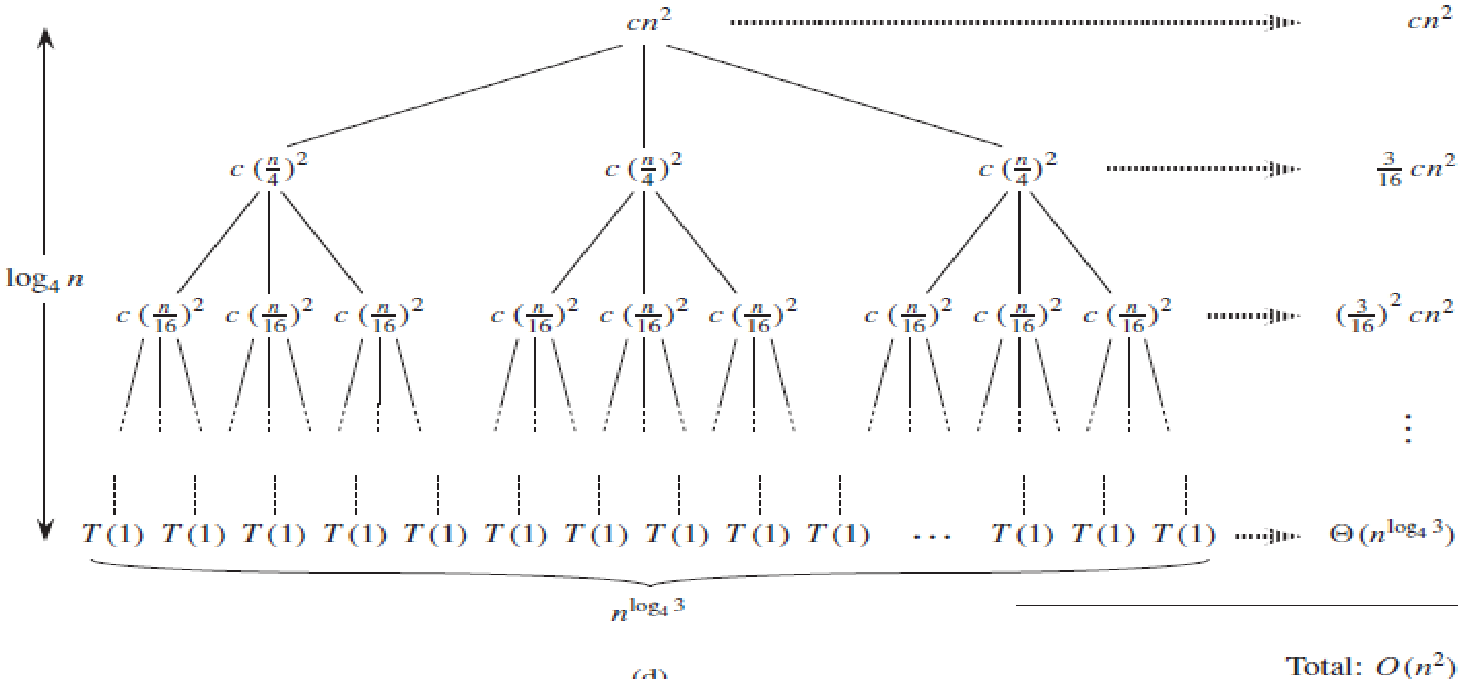


- Subproblem size at level i is: $n/4^i$
- Subproblem size hits 1 when $1 = n/4^i \Rightarrow i = \log_4 n$
- Cost of a node at level i = $c(n/4^i)^2$
- Number of nodes at level i = $3^i \Rightarrow$ last level has $3^{\log_4 n} = n^{\log_4 3}$ nodes

$$T(n) = \sum_{i=0}^{\log_4 n-1}\left(\frac{3}{16}\right)^i cn^2 + \Theta\left(n^{\log_4 3}\right) \le \sum_{i=0}^{\infty}\left(\frac{3}{16}\right)^i cn^2 + \Theta\left(n^{\log_4 3}\right) = \frac{1}{1-\frac{3}{16}}cn^2 + \Theta\left(n^{\log_4 3}\right) = O(n^2)$$

- Total cost: $\Rightarrow T(n) = O(n^2)$

Example of recursion tree $T(n) = 3\,T(n/4) + O(n^2)$.



$$T(n)=\sum_{i=0}^{\log_4 n-1}\left(\frac{3}{16}\right)^i cn^2+\Theta\left(n^{\log_4 3}\right)\le\sum_{i=0}^{\infty}\left(\frac{3}{16}\right)^i cn^2+\Theta\left(n^{\log_4 3}\right)=\frac{1}{1-\frac{3}{16}}cn^2+\Theta\left(n^{\log_4 3}\right)=O(n^2)$$

Geometric Series

$$1 + x + x^2 + \cdots + x^n = \frac{1 - x^{n+1}}{1 - x} \quad \text{for } x \neq 1$$

$$1 + x + x^2 + \cdots = \frac{1}{1 - x} \quad \text{for } |x| < 1$$

Geometric Series (Aside)

-

$$1 + x + x^2 + \ ... + \ x^n = \frac{1 - \ x^{n+1}}{1 - x} \ for\ x \ \neq 1$$

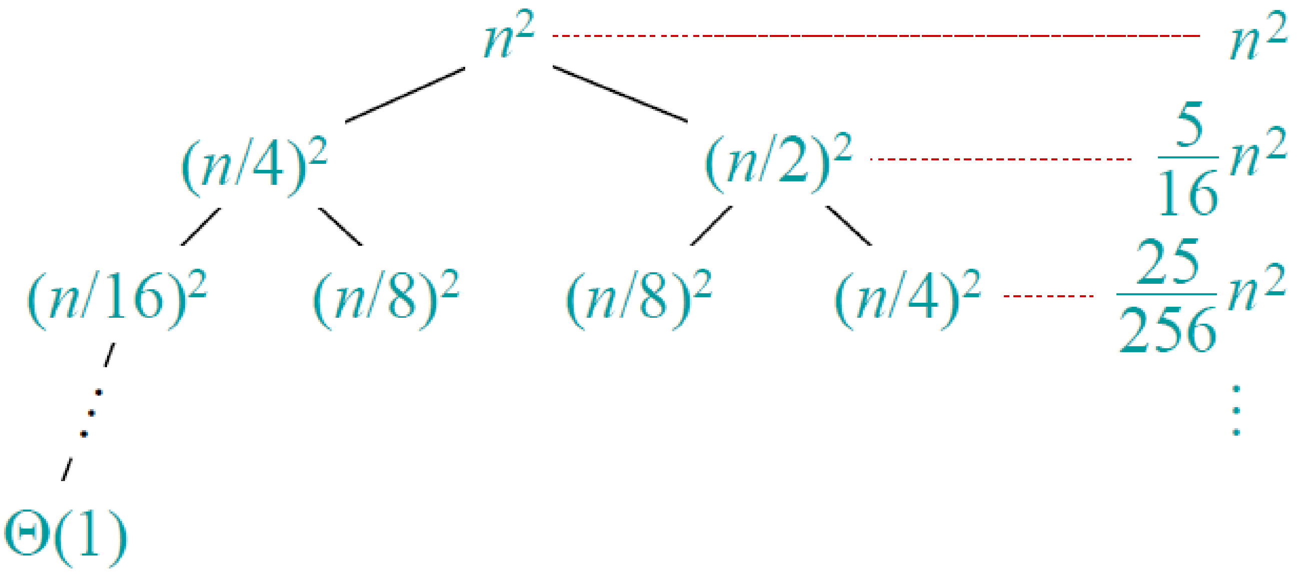
OR

$$1 + x + x^2 + \ ... + \ x^n = \frac{x^{n+1} \ - 1}{x \ - 1} \ for\ x \ \neq 1$$

$$1 + x + x^2 + \ ... + \ x^n = \frac{1}{1 - x} \ for\ x < 1$$

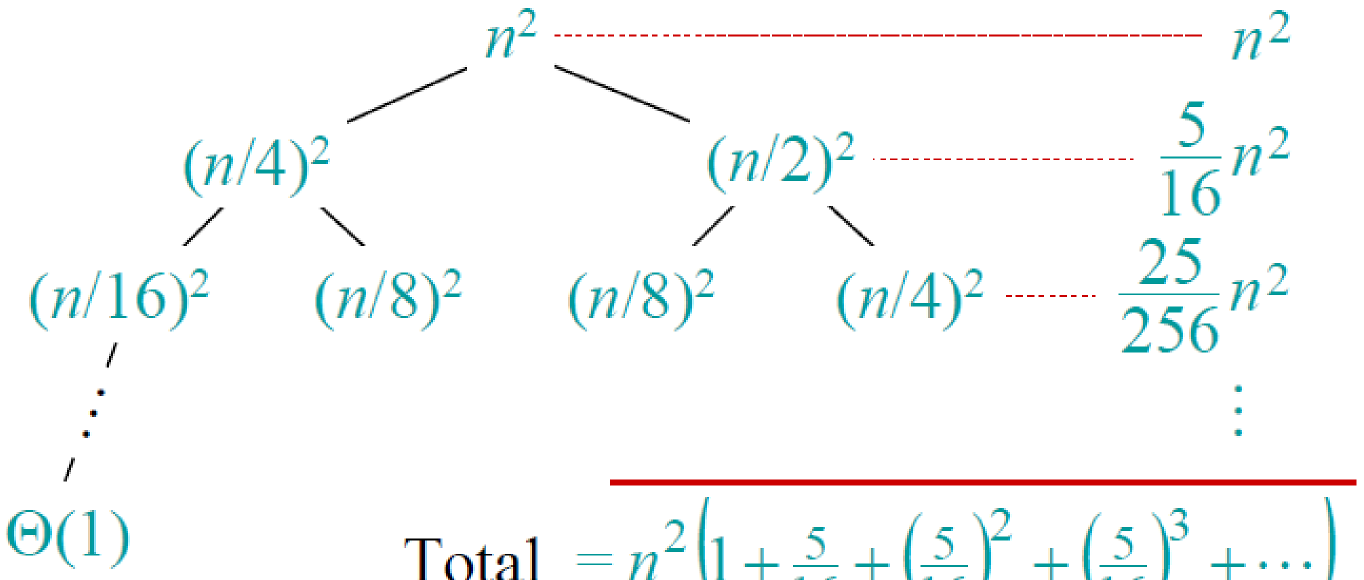
Example of recursion tree

$T(n) = T(n/4) + T(n/2) + O(n^2).$



Example of recursion tree

$T(n) = T(n/4) + T(n/2) + O(n^2).$



$$\text{Total} = n^2\left(1 + \frac{5}{16} + \left(\frac{5}{16}\right)^2 + \left(\frac{5}{16}\right)^3 + \cdots\right)$$
$$= \Theta(n^2)$$

geometric series

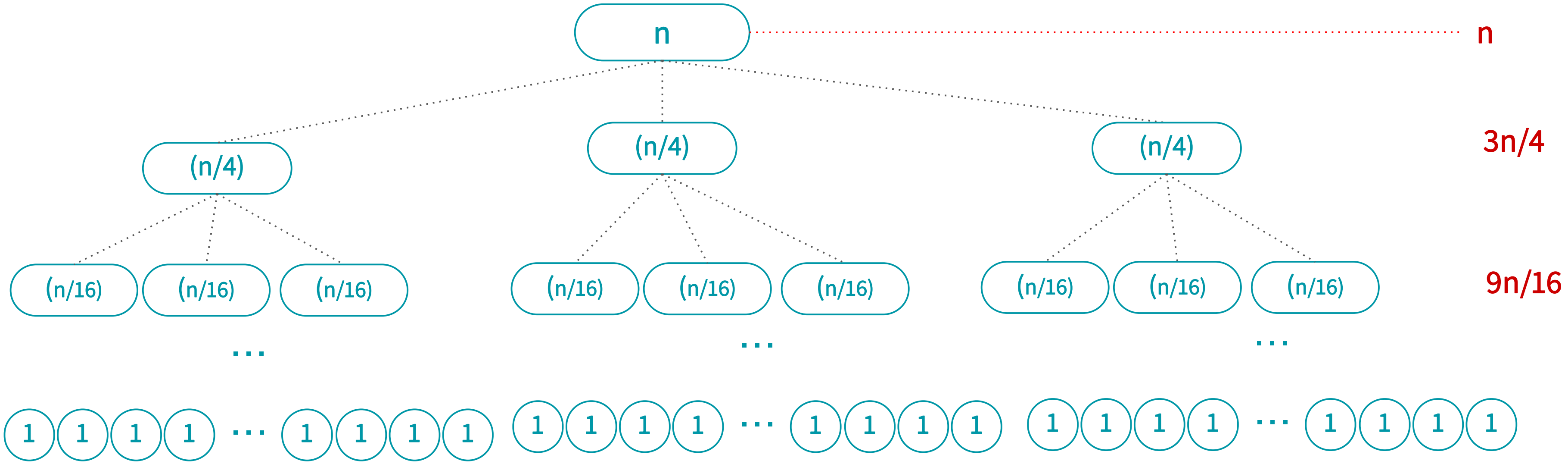
Example of recursion tree

$T(n) = 3\ T(n/4) + O(n)$ (H. W)

Draw recurrence tree

Example of recursion tree

$$T(n) = 3\,T(n/4) + O(n) \text{ (H. W)}$$



ITERATIVE Substitution METHOD

Iterative Substitution Method

- ***Convert the recurrence into a summation and try to bound it using known series***

1. Iterate the recurrence until the initial condition is reached.
2. Use back-substitution to express the recurrence in terms of *n* and the initial (boundary) condition.

Substitution (Example # 1)

$$T(n) = \begin{cases} c & \text{if } n = 1 \\ 2T(n/2) + cn & \text{if } n > 1 \end{cases}$$

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

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

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

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

$T(n/4) = n/4 + 2T(n/8)$

For $2^k = n$ => $k = \log n$

By putting 'log' both sides, and

Using: $\log a^b = b.\log a$

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

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

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

$T(n) = n + 2n/2 + 4(n/4 + 2T(n/8))$

$T(n) = n + n + n + 8T(n/8)$

$T(n) = 3n + 8(n/8 + 2T(n/16))$

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

.....

$T(n) = kn + 2^k.T(n/2^k)$

$T(n) = n.\log n + n.T(1)$

$T(n) = O(n\log n)$

Substitution (Example # 2)

$$T(n) = \begin{cases} 1 & \text{if } n = 1 \\ T(n/2) + n & \text{if } n > 1 \end{cases}$$

For $2^k = n$ => $k = \log n$

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

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

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

$T(n) = n + n/2 + n/4 + T(n/8))$

$T(n) = n + n/2 + n/4 + T(n/8)$

$T(n) = n\ (1 + 1/2 + 1/\ 4\ ) + T(n/2^3)$

.....

$T(n) = n(1 + ½ + ¼ \ldots\ldots + 1/\ 2^k) + T(n/2^k)$

$T(n) = n(1+ ½ + ¼ + \ldots.. + 1/n) + T(1)$

$T(n) = n.(1+1) + 1$

$T(n) = \text{Theta}(n)$

Solve the Recurrence

$$T(n) = \begin{cases} 1 & \text{if } n = 1 \\ 4T(n/2) + n & \text{if } n > 1 \end{cases}$$

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

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

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

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

$T(n/4) = n/4 + 4T(n/8)$

======================

$2^0 + 2^1 + 2^2 + 2^3 + 2^4 + \dots + 2^k$

$= \frac{2^{k+1} - 1}{2 - 1}$

======================

For $k = \log n \Rightarrow n = 2^k$

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

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

$T(n) = n + 4n/2 + 16T(n/4)$

$T(n) = n + 2n + 4^2(n/4 + 4T(n/8))$

$T(n) = n + 2n + 4n + 4^3\, T(n/8)$

$T(n) = n + 2n + 4 + 4^3(n/8 + 4T(n/16))$

$T(n) = n + 2n + 4n + 8n + 4^4\, T(n/2^4)$

$T(n) = n\,(1 + 2 + 4 + 8) + 4^4 T(n/ 2^4)$

...

$T(n) = n.\ \frac{2^{k+1} - 1}{2 - 1} + 4^k.T(n/2^k)$

$T(n) = n.\ (2^{\log n + 1} - 1) + 4^{\log n}\ .\ T(1)$

$T(n) = O(n^2)$

Substitution (Example # 4)

$$T(n) = \begin{cases} 1 & \text{if } n = 1 \\ T(n/2) + 1 & \text{if } n > 1 \end{cases}$$

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

$T(n) = 1 + (1+T(n/4))$

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

$T(n) = 3 + T(n/8)$

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

…..

$T(n) = k + T(n/2^k)$

For $k = \log n$, as: $n = 2^k$

$T(n) = \log n + T(1)$

$T(n) = 1 + \log(n)$

i.e. $T(n) = \Theta(\log n)$ — Theta (logn)

Substitution (Example # 5)

$T(n) = 1 + T(n-1)$

$T(n) = 1 + (1+T(n-2))$

$T(n) = 2 + T(n-2)$

$T(n) = 3 + T(n-3)$

$T(n) = 4 + T(n-4)$

…..

$T(n) = k + T(n-k)$

For $k = (n-1)$

$T(n) = n – 1 + T(1)$

$T(n) = n$

i.e. $T(n) = \text{Theta}(n)$

For $n – k = 1$
So, $k = n-1$

Substitution (Example # 6)

$T(n) = n + T(n-1)$

$T(n) = n + (n-1+T(n-2))$

$T(n) = 2n – 1 + T(n-2)$

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

$T(n) = 3n -3 + T(n-3)$

$T(n) = 3n -3 + n-3 + T(n-4))$

$T(n) = 4n – 6 + T(n-4)$

…..

$T(n) = kn – c + T(n-k)$

$T(n) = (n – 1).n - n – c + T(1)$

$T(n) = n^2 - n – c + T(1)$

i.e. $T(n) = Theta(n^2)$

For $n – k = 1$

So, $k = n-1$

Substitution (Example # 7)

$T(n) = 2T(n-1)$

$T(n) = 2(2T(n-2))$

$T(n) = 4T(n-2)$

$T(n) = 4(2T(n-3))$

$T(n) = 8T(n-3)$

$T(n) = 8(2T(n-4))$

$T(n) = 16T(n-4)$

$T(n) = 2^4T(n-4)$

……..

$T(n) = 2^k.T(n-k)$

$T(n) = 2^{(n-1)}.T(1)$

$T(n) = O(2^n)$

For $n – k = 1$

So, $k = n-1$

THE MASTER METHOD

A formula for solving many recurrence relations!

(For some recurrence relations, as we'll see later, it won't work)

Master's Method: Solving Recurrences

Theorem (Master Theorem)

Let $T(n)$ *be a monotonically increasing function that satisfies*

$$\begin{aligned} T(n) &= aT(\tfrac{n}{b}) + f(n) \\ T(1) &= c \end{aligned}$$

where $a \geq 1, b \geq 2, c > 0$*. If* $f(n) \in \Theta(n^d)$ *where* $d \geq 0$*, then*

$$T(n) = \begin{cases} \Theta(n^d) & \text{if } a < b^d \\ \Theta(n^d \log n) & \text{if } a = b^d \\ \Theta(n^{\log_b a}) & \text{if } a > b^d \end{cases}$$

MASTER THEOREM LIMITATIONS

You cannot use the Master Theorem if

- T(n) is not monotone, ex: T(n) = sin n
- f(n) is not a polynomial, ex: T(n) = 2T(n/ 2) + 2^n
- b cannot be expressed as a constant, ex: T(n) = T($\sqrt{n}$) , T(n) = T(n/2) + T(n/4) + O(n)

Note here, that the Master Theorem does not solve a recurrence relation.

MASTER THEOREM EXAMPLES

$$T(n) = \begin{cases} O(n^d) & \text{if } a < b^d \\ O(n^d \log n) & \text{if } a = b^d \\ O(n^{\log_b(a)}) & \text{if } a > b^d \end{cases}$$

a: # of subproblems (branching factor)
b: factor by which input size shrinks (shrinking factor)
d: need to do $O(n^d)$ work to create subproblems + "merge" soluti

$$T(n) = 4 \cdot T(n/2) + O(n)$$

a =
b =
d =

$$T(n) = 2 \cdot T(n/2) + 1/3\, n^2 + n$$

a =
b =
d =

$$T(n) = 2 \cdot T(n/2) + O(n)$$

a =
b =
d =

MASTER THEOREM EXAMPLES

$$T(n) = \begin{cases} O(n^d) & \text{if } a < b^d \\ O(n^d \log n) & \text{if } a = b^d \\ O(n^{\log_b(a)}) & \text{if } a > b^d \end{cases}$$

a: # of subproblems (branching factor)
b: factor by which input size shrinks (shrinking factor)
d: need to do $O(n^d)$ work to create subproblems + "merge" soluti

USELESS DIVIDE & CONQUER MULTIPLICATION

$T(n) = 4 \cdot T(n/2) + O(n)$

$T(n) = O(n^{\log_2 4}) = O(n^2)$

$a = 4$
$b = 2$
$d = 1$

$a > b^d$

$T(n) = 2 \cdot T(n/2) + 1/3\, n^2 + n$

$T(n) = O(n^2)$

$a = 2$
$b = 2$
$d = 2$

$a < b^d$

$T(n) = 2 \cdot T(n/2) + O(n)$

$T(n) = O(n \log n)$

$a = 2$
$b = 2$
$d = 1$

$a = b^d$

MASTER THEOREM EXAMPLES

- Example 4:

Let $T(n) = 2T\left(\frac{n}{4}\right) + \sqrt{n} + 42$. What are the parameters?

$$\begin{aligned} a &= \\ b &= \\ d &= \end{aligned}$$

Therefore which condition?

MASTER THEOREM EXAMPLES

- Example 4: Solution

Let $T(n) = 2T\left(\frac{n}{4}\right) + \sqrt{n} + 42$. What are the parameters?

$$\begin{aligned} a &= 2 \\ b &= 4 \\ d &= \frac{1}{2} \end{aligned}$$

Therefore which condition?

Since $2 = 4^{\frac{1}{2}}$, case 2 applies.

Thus we conclude that

$$T(n) \in \Theta(n^d \log n) = \Theta(\sqrt{n} \log n)$$

MASTER THEOREM 4th Case

- Fourth Condition:

Recall that we cannot use the Master Theorem if $f(n)$ (the non-recursive cost) is not polynomial.

There is a limited 4-th condition of the Master Theorem that allows us to consider polylogarithmic functions.

Corollary

If $f(n) \in \Theta(n^{\log_b a} \log^k n)$ *for some* $k \geq 0$ *then*

$$T(n) \in \Theta(n^{\log_b a} \log^{k+1} n)$$

This final condition is fairly limited and we present it merely for completeness.

MASTER THEOREM

- $T(n) = 2 \cdot T(n/2) + n \log n$

$a = ?$
$b = ?$
$d = ?$

MASTER THEOREM

- $T(n) = 2 \cdot T(n/2) + n \log n$

$a = 2$
$b = 2$
$d =$ not applicable
$f(n)$ is not polynomial

MASTER THEOREM 4th Case

Say that we have the following recurrence relation:

$$T(n) = 2T\left(\frac{n}{2}\right) + n \log n$$

Clearly, $a = 2, b = 2$ but $f(n)$ is not a polynomial. However,

$$f(n) \in \Theta(n \log n)$$

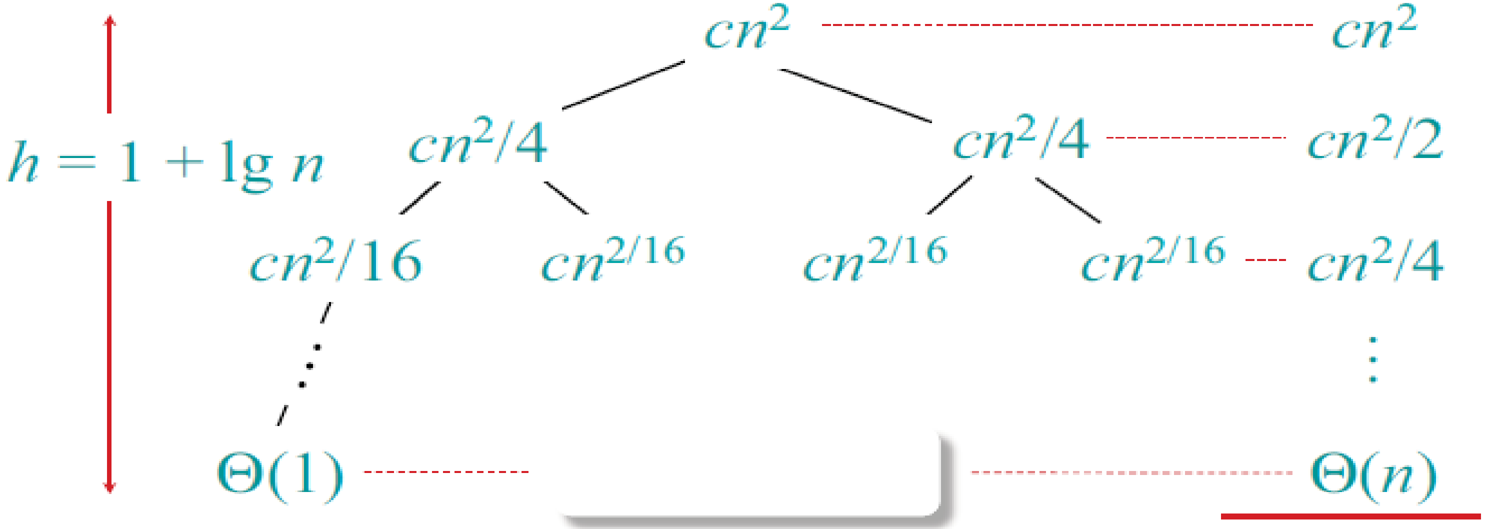
for $k = 1$, therefore, by the 4-th case of the Master Theorem we can say that

$$T(n) \in \Theta(n \log^2 n)$$

MASTER THEOREM: $2T(n/2) + O(n^2)$ Case 1 $a < b^d$

MASTER THEOREM: $2T(n/2) + O(n^2)$ Case 1 $a < b^d$

Solve $T(n) = 2T(n/2) + cn^2$, $c > 0$ is constant.



Note that $1 + ½ + ¼ + \ldots < 2$

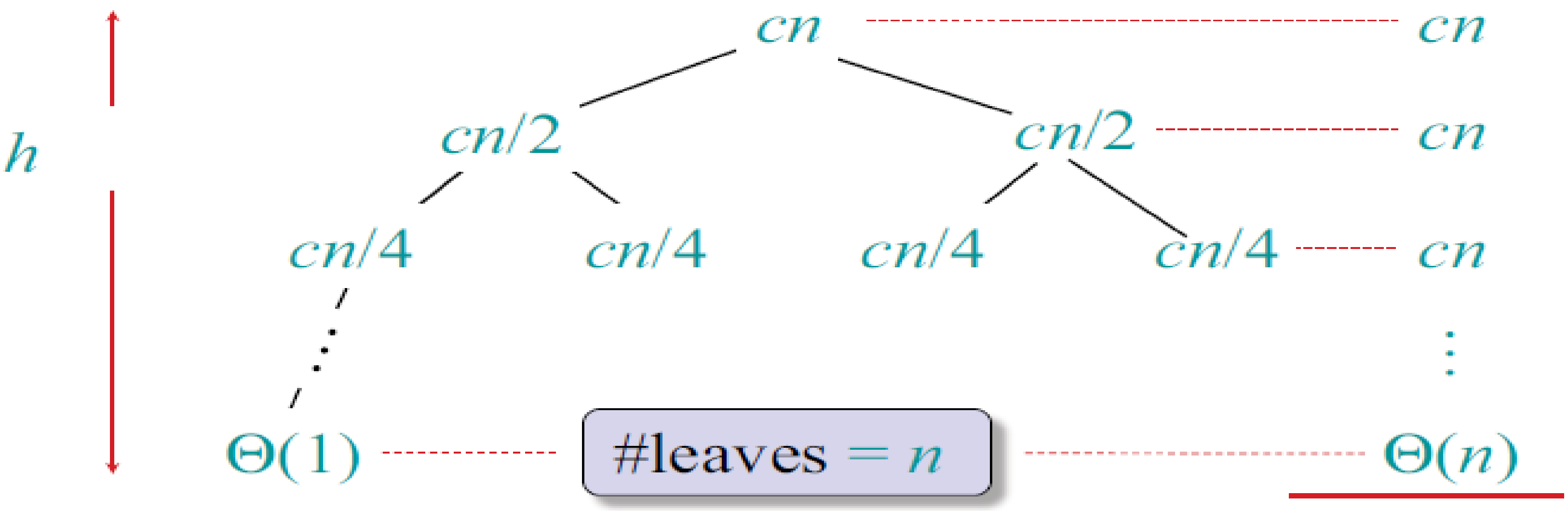
Total $= \Theta(n^2)$

All the work done at the root

MASTER THEOREM: $2T(n/2) + O(n)$ Case 2 $a = b^d$

MASTER THEOREM: 2T(n/2) + O(n) Case 2 $a = b^d$

Solve $T(n) = 2T(n/2) + cn$, where $c > 0$ is constant.



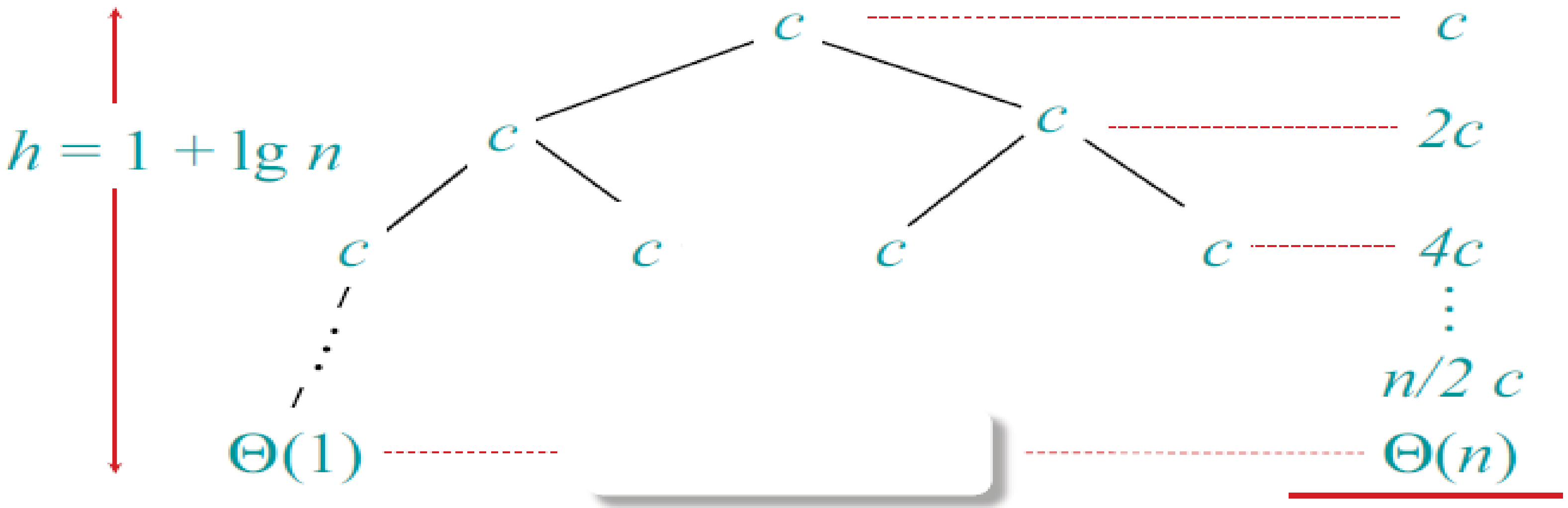
Total $= \Theta(n \lg n)$

Equal amount of work done at each level

MASTER THEOREM: $2T(n/2) + O(1)$ Case 3 $a > b^d$

MASTER THEOREM: 2T(n/2) + O(1) Case 3 $a > b^d$

Solve $T(n) = 2T(n/2) + c$, where $c > 0$ is constant.



Note that $1 + ½ + ¼ + \ldots < 2$

Total $= \Theta(n)$

All the work done at the leaves

MASTER THEOREM 3 Cases Summary

- The three cases of the master theorem correspond to three different costs which might be dominant as a function of *a*, *b*, and $f(n)$:

- Case 1 ($a < b^d$) : ***Too expensive at root*** – Since $f(n)$ costs grow rapidly enough with *n*, then the cost of the root evaluation may dominate. So, the total running time is $O\left(f(n)\right) or$ B.

MASTER THEOREM 3 Cases Summary

- The three cases of the master theorem correspond to three different costs which might be dominant as a function of a, b, and $f(n)$:

- Case 2 **($a = b^d$) :** ***Equal work per level*** – the leaves grow at the same rate as f, so the same order of work is done at every level of the tree. The tree has O(log n) levels, times the work done on one level, yielding T(n) is $O(n^d \log n)$.

MASTER THEOREM 3 Cases Summary

- The three cases of the master theorem correspond to three different costs which might be dominant as a function of *a*, *b*, and $f(n)$:

- Case 3 **(a > b^d)** **:** ***Too many leaves*** – *Since the leaves grow faster than f, asymptotically all of the work is done at the leaves, so the total running time is* $O(n^{\log_b(a)})$.

MASTER THEOREM MORE EXAMPLES

$T(n) = 4 \cdot T(n) + n^2 + n + 25$

a =
b =
d =

$T(n) = 4 \cdot T(n/2) + 2^n$

a =
b =
d =

$T(n) = T(n/2) + \sqrt{n}$

a =
b =
d =

$T(n) = T(n/2) + T(n/4) + \sqrt{n}$

a =
b =
d =

$T(n) = 2 \cdot T(n/2) + 2n \log n$

a =
b =
d =

MASTER THEOREM MORE EXAMPLES

$T(n) = 4 \cdot T(n) + n^2 + n + 25$

Master Theorem Not Applicable as b < 2

a = 4
b = 1
d = 1

$T(n) = 4 \cdot T(n/2) + 2^n$

Master Theorem Not Applicable as f(n) is exponent

a = 4
b = 2
f(n) = exp

$T(n) = T(n/2) + \sqrt{n}$

$$T(n) = O(\sqrt{n}\,\log n)$$

a = 1
b = 2
d = 1/2

$T(n) = T(n/2) + T(n/4) + \sqrt{n}$

Master Theorem Not Applicable as b is not expressed as constant

a = 2
b = 2 ,4
d = 1/2

$T(n) = 2 \cdot T(n/2) + 2n \log n \rightarrow 2 \cdot T(n/2) + n \log^2 n$

$$T(n) = O(n \log^3 n)$$

a = 2
b = 2
k = 2

Home Work: Solve the following recurrences using **Master's Method.**

$$T(n) = 4\ T(n/2) + 2n^2 + 25$$

$$T(n) = T(n/3) + 50$$

$$T(n) = 8T(n/2) + \sqrt{n}$$

$$T(n) = 2\ T(n/2) + n * 2^5$$

$$T(n) = 9 \cdot T(n/2) + n + 5$$