Introduction to Algorithm

Solving Recurrences using

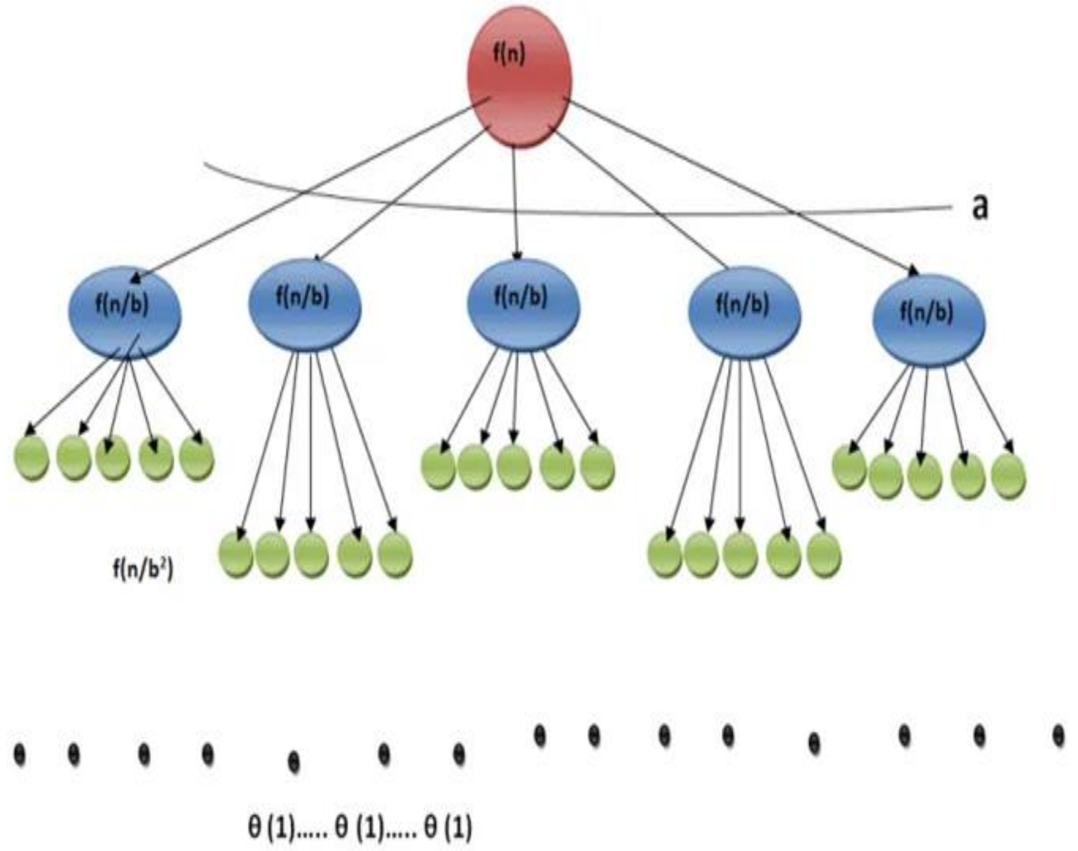
Recursion Tree Method

Recursion Tree Method

In this method, we draw a recurrence tree and calculate the time taken by every level of tree. Finally, we sum the work done at all levels. To draw the recurrence tree, we start from the given recurrence and keep drawing till we find a pattern among levels. The pattern is typically a arithmetic or geometric series.

- 1. Recursion Tree Method is a pictorial representation of an iteration method which is in the form of a tree where at each level nodes are expanded.
- 2. It is sometimes difficult to come up with a good guess. In Recursion tree, each root and child represents the cost of a single subproblem.
- 3. We sum the costs within each of the levels of the tree to obtain a set of pre-level costs and then sum all pre-level costs to determine the total cost of all levels of the recursion.
- 4. A Recursion Tree is best used to generate a good guess, which can be verified by the Substitution Method.

$T(n)=aT(n/b)+f(n)$ if $n>1$



Remember that n is the size of the original problem you can imagine the recursion continuing until only a single element remains. This would happen at T(1).

This means the subproblem size (where n is the original problem size) is n/b^i at the i th level.

At the boundary, the subproblem size is 1

n/b^i = 1

log(n/b^i) = log(1)

log(n) - log(b^i) = $\underline{0}$

log(n) = log(b^i)

$\log_b(n) = i$

Since the height of the tree is the level where the boundary condition is met, the tree has height log(n).

H is the height of the tree, then **$H = \log_b n$**.

The problem size reduces by **a factor b** at each level. So it would take $\log_b n$ levels to reduce it to a problem of size 1 and it cannot be divided further.

Recurrences

Lets say we have a problem of size **n** to be solved. At each step the problem can be divided into **a** sub problems and each sub problem is of smaller size, where the size is reduced by a factor of **b**.

The above statement in simple words means that a problem of size **n** can be divided into **a** sub problems of relatively smaller sizes **n/b**.

At the end when we have divided the problems multiple times, each sub problem would be so small that it can be solved in constant time.

what can we say about the height of the tree?

What is the number of leaves in the tree?

The relationship between height of a tree and the number of leaf nodes is as follows:
number of leaves = $\text{branching\_factor}^{\text{height}}$.
Hence the number of leaves would be $\mathbf{a^H}$. Replacing H in this equation, we would get the number of leaves = $\mathbf{a^{\log_b n}}$.
By the following properties of logarithms we can rearrange the formula. $N^{\log_k M} = M^{\log_k N}$. Therefore the number of leaves = $\mathbf{n^{\log_b a}}$.

Recurrences Tree Method

T(n) = $\Theta(1)$ if n==1
aT(n/b)+f(n) if n>1

Where a>=1 and b>1 be constants and let f(n) be a nonnegative function defined on exact power of b

i : the last level number

Assume n=b^i

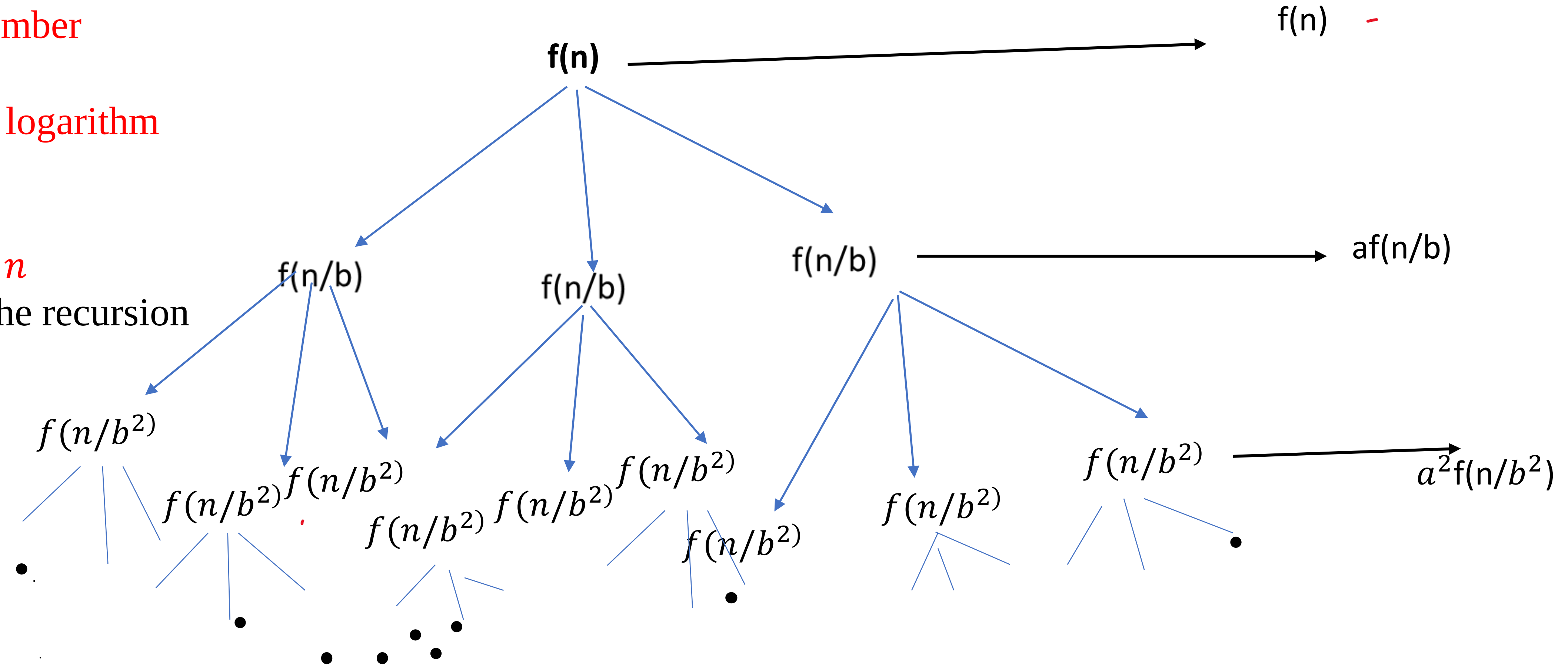
Taking both sides logarithm

log n=log b^i

= i log b

i=logn/logb=$\log_b n$

i is the height of the recursion tree.



Recurrences Tree Method

Since the recursion tree is an complete a-ary tree (every internal node has exactly a children) hence number of leaf nodes= number of nodes in the last i.e. ith level

$$= a^i = a^{\log_b n} = n^{\log_b a}$$

Now T(n) can be represented as the sum of total cost in each level which is as follows:

$$T(n) = n^{\log_b a} + \underline{\sum_{i=0}^{\log_b n - 1} a^i f(n/b^i)}$$

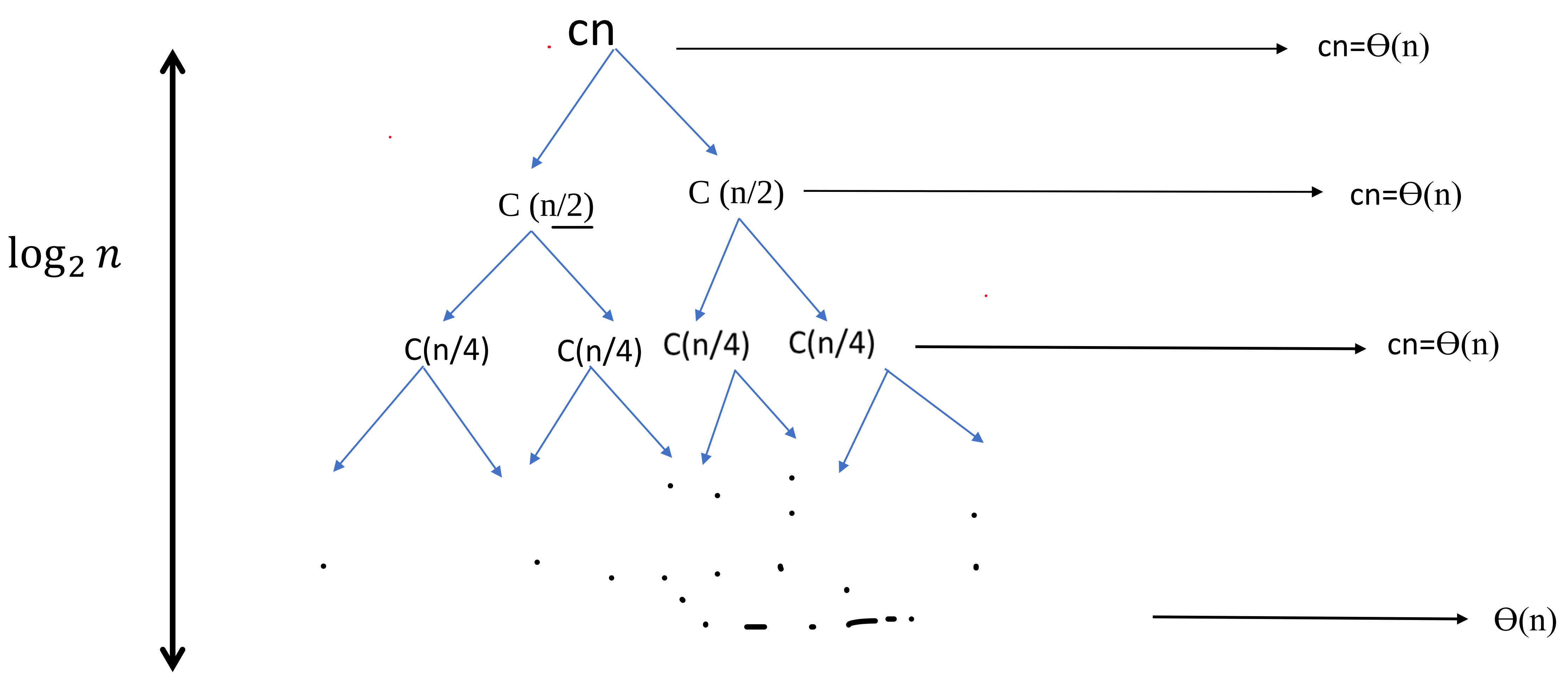
$\downarrow$

Describes the cost of the dividing and combining

The recursion –tree is a complete a-ary tree with $n^{\log_b a}$ leaves and height $\log_b n$

Recursion Tree Method

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



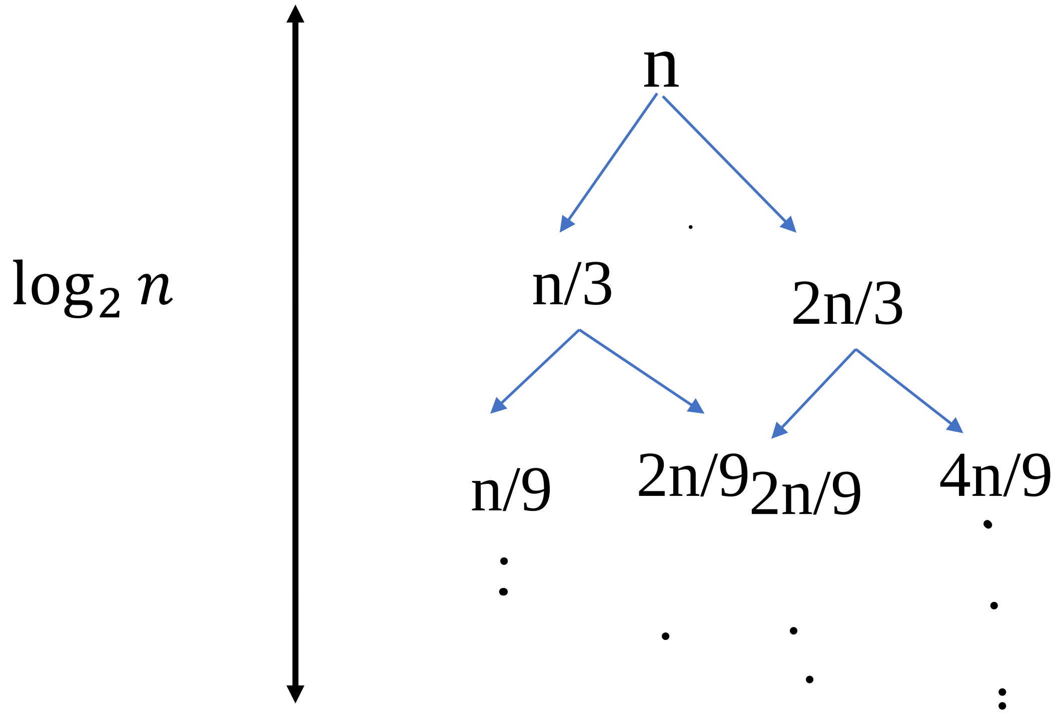
Recurrences Tree Method

Height of this recursion tree is $\log_2 2 = n = \Theta(n)$
Since the cost in each level (including the last level) is Θ(n) hence T(n) can be computed as:
T(n)=(cost in each level) *(height of the recursion tree)

$$= \Theta(n) * \log_2 n = \Theta(n \log_2 n)$$

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

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



Recurrences Tree Method

Height of this recursion tree is $\log_{3/2} n$ .Since the cost in each level is Θ(n). Hence T(n) can be computed as:

T(n)=(cost in each level)*(height of the recursion tree)

$= \Theta(n) * \log_{3/2} n = \Theta(n \log_{3/2} n)$

Recursion Tree Method

Steps: 1 . solving recurrences expanding the recurrence into a tree

Step:2. summing the cost at each level

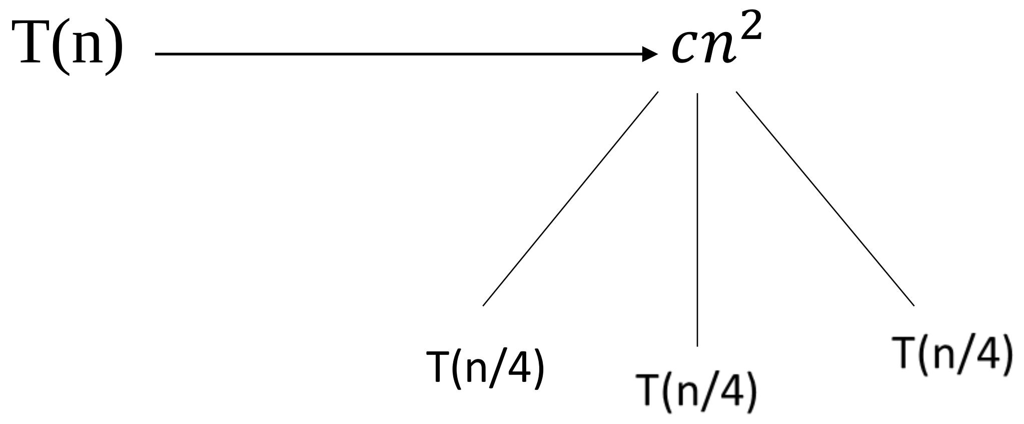
Step:3. applying the substitution method

Example:

Consider the recurrence relation T(n) = 3T(n/4) + cn^2 for some constant c

We assume that n is an exact power of 4.

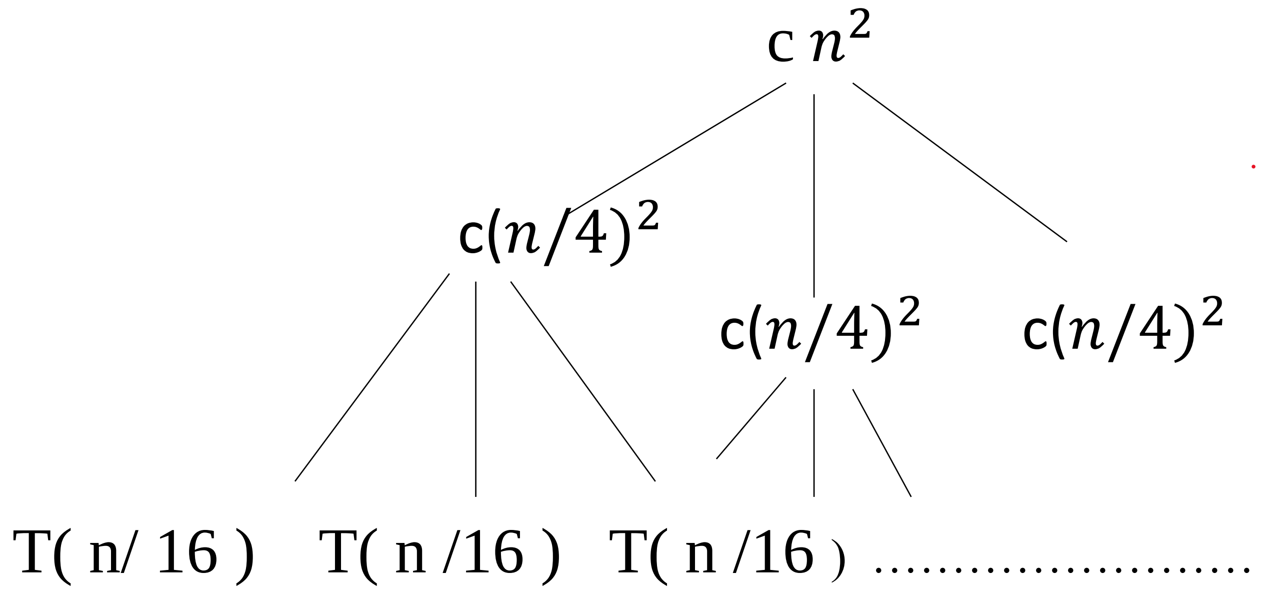
In the recursion-tree method we expand T(n) into a tree:



Recursion Tree Method

- we expand T(n/4)

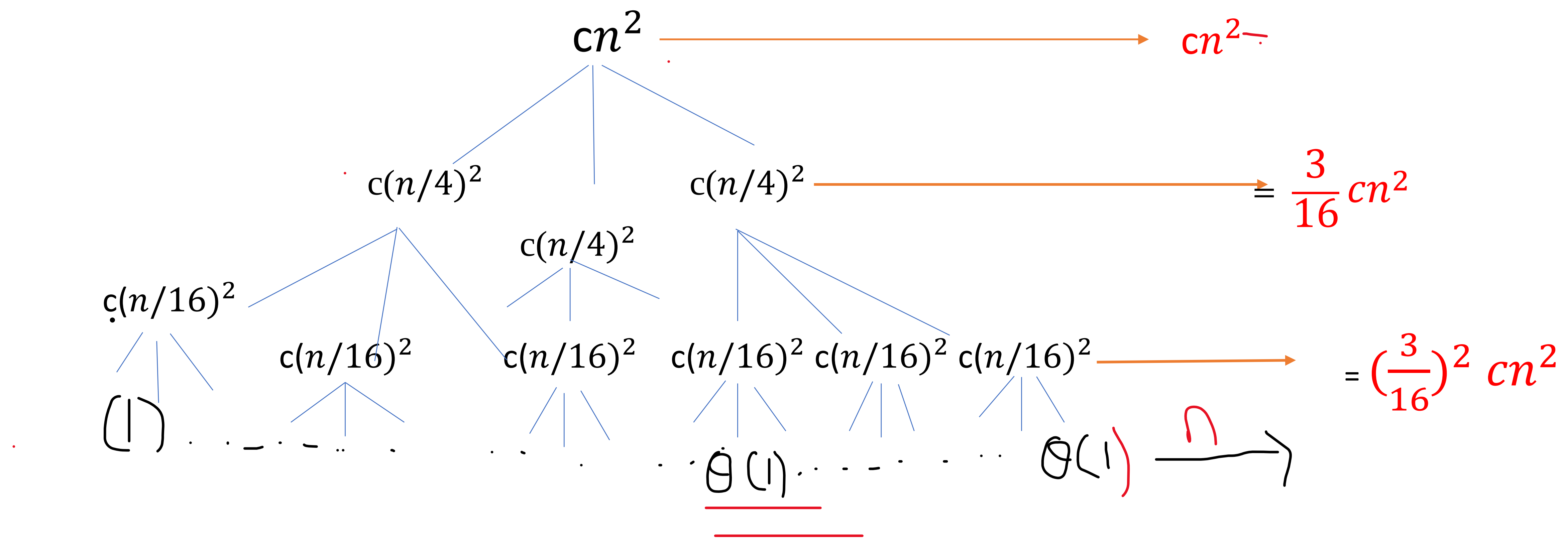
Applying T(n) = 3T(n/4) + c n^2 to T(n/4) leads to T(n/4) = 3T(n/16) + c$(n/4)^2$, expanding the leaves:



we expand T(n/ 16)

Applying T(n) = 3T(n/4) + c n^2 to T(n/16) leads to T(n/16) = 3T(n/64) + c(n/16)2, expanding the leaves:

Recursion Tree Method



2.summing the cost at each level

We sum the cost at each level of the tree:

T(n) = cn2 + (3 /16)cn2 + (3 /16)2 cn2 + ⋯ = cn2(1 + 3 /16 + (3 /16)2 + ⋯)

if n = 16, or the tree has depth at least 2 if n ≥ 16 =4^2.

Recursion Tree Method

- For n = 4^k, k = log₄(n),
- we have: T(n) =$cn^2 \sum_{i=0}^{logn} (3/16)^i$

T(n) = cn2 + (3 /16)cn2 + (3 /16)2 cn2 + ···
= cn2(1 + 3 /16 + (3 /16)2 + ···)

T(n) =$cn^2 \sum_{i=0}^{logn-1} (3/16)^i + \Theta(n^{\log_4 n})$ $Replace\ \log_4 n\ with$ ∞

$<= \Theta(n^{\log_4 3}) + cn^2 \sum_{i=0}^{\infty} (3/16)^i$

$= \Theta(n^{\log_4 3}) + cn^2 (\frac{1}{1-3/16})$

$= \Theta(n^{\log_4 3}) + 16/3\ cn^2 = O(\max(n^{\log_4 3}, n^2)) = O(n^2)$

Appendix: geometric series∞

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

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

$\sum_{i=0}^{\infty} x^i$ =1/1-x

Question:

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

Solve $T(n) = T(n/5) + T(4n/5) + n$