Recurrence Relations
Solution of Recurrence Relations
Substitution Method

Recurrence

- Recurrence relations often arise in calculating the time and space complexity of algorithms. Any problem can be solved either by writing **recursive algorithm** or by writing **non-recursive algorithm**.
- A recursive algorithm is one which makes a recursive call to itself with smaller inputs. We often use a recurrence relation to describe the running time of a recursive algorithm.

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 Examples

$$s(n) = \begin{cases} 0 & n = 0 \\ c + s(n-1) & n > 0 \end{cases}$$

$$s(n) = \begin{cases} 0 & n = 0 \\ n + s(n-1) & n > 0 \end{cases}$$

$$T(n) = \begin{cases} c & n = 1 \\ 2T\left(\frac{n}{2}\right) + c & n > 1 \end{cases}$$

$$T(n) = \begin{cases} c & n = 1 \\ aT\left(\frac{n}{b}\right) + cn & n > 1 \end{cases}$$

Example Recurrences

- $T(n) = T(n-1) + n$ $\Theta(n^2)$
  - Recursive algorithm that loops through the input to eliminate one item
- $T(n) = T(n/2) + c$ $\Theta(\lg n)$
  - Recursive algorithm that halves the input in one step
- $T(n) = T(n/2) + n$ $\Theta(n)$
  - Recursive algorithm that halves the input but must examine every item in the input
- $T(n) = 2T(n/2) + 1$ $\Theta(n)$
  - Recursive algorithm that splits the input into 2 halves and does a constant amount of other work

Recurrence Relation

- To solve a Recurrence Relation means to obtain a function defined on the natural numbers that satisfy the recurrence.

- The Expression

$$T(n) = \begin{cases} c & n = 1 \\ 2T\left(\frac{n}{2}\right) + cn & n > 1 \end{cases}$$

- is a *recurrence*.
  - Recurrence: an equation that describes a function in terms of its value on smaller functions
- **For Example**, the Worst Case Running Time T(n) of the MERGE SORT Procedures is described by the recurrence.

T (n) = θ (1) if n=1
2T(n/2) + θ (n) if n>1

- Like all recursive functions, a recurrence also consists of two steps:
  1. **Basic step**: Here we have one or more constant values which are used to terminate recurrence. It is also known as **initial conditions** or **base conditions.**
  2. **Recursive steps:** This step is used to find new terms from the existing (preceding) terms. Thus in this step the recurrence compute next sequence from the k preceding values

$$f_{n-1}, f_{n-2}, \cdots f_{n-k}$$

This formula is called a **recurrence relation (or recursive formula).** This formula refers to itself, and the argument of the formula must be on smaller values (close to the base value).

Hence a recurrence has one or more initial conditions and a recursive formula, known as **recurrence relation.**

Solution of Recurrence Relations

There are four methods for solving Recurrence:

- Substitution Method
- Iteration Method
- Recursion Tree Method
- Master Method

Substitution Method

- The Substitution Method Consists of two main steps:
  1. Guess the Solution.
  2. Use the mathematical induction to find the boundary condition and shows that the guess is correct.

The substitution method can be used to establish either upper or lower bounds on a recurrence.

Examples:

$T(n) = 2T(n/2) + \Theta(n)$ ⍰ $T(n) = \Theta(n \lg n)$

$T(n) = 2T(\lfloor n/2 \rfloor) + n$ ⍰ ???

Substitution method

- Guess a solution
  - $T(n) = O(g(n))$
  - Induction goal: apply the definition of the asymptotic notation
    - $T(n) \leq d\, g(n)$, for some $d > 0$ and $n \geq n_0$ (strong induction)
  - Induction hypothesis: $T(k) \leq d\, g(k)$ for all $k < n$
- Prove the induction goal
  - Use the **induction hypothesis** to find some values of the constants d and n_0 for which the **induction goal** holds

Example: Binary Search

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

- Guess: $T(n) = O(\lg n)$
  - Induction goal: $T(n) \le d \lg n$, for some d and $n \ge n_0$
  - Induction hypothesis: $T(n/2) \le d \lg(n/2)$
- Proof of induction goal:

$$T(n) = T(n/2) + c \le d \lg(n/2) + c$$

$$= d \lg n - d + c \le d \lg n$$

$$\text{if: } -d + c \le 0,\ d \ge c$$

- Base case?

Example 2

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

- Guess: $T(n) = O(n^2)$
  - Induction goal: $T(n) \le c\, n^2$, for some c and $n \ge n_0$
  - Induction hypothesis: $T(n-1) \le c(n-1)^2$ for all $k < n$
- Proof of induction goal:

$T(n) = T(n-1) + n \le c\,(n-1)^2 + n$

$= cn^2 - (2cn - c - n) \le cn^2$

if: $2cn - c - n \ge 0 \Leftrightarrow c \ge n/(2n-1) \Leftrightarrow c \ge 1/(2 - 1/n)$

  - For $n \ge 1 \Rightarrow 2 - 1/n \ge 1 \Rightarrow$ any $c \ge 1$ will work

Example 3

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

- Guess: $T(n) = O(n\lg n)$
  - Induction goal: $T(n) \le cn \lg n$, for some c and $n \ge n_0$
  - Induction hypothesis: $T(n/2) \le cn/2 \lg(n/2)$
- Proof of induction goal:

$$T(n) = 2T(n/2) + n \le 2c\,(n/2)\lg(n/2) + n$$

$$= cn \lg n - cn + n \le cn \lg n$$

$$\text{if: } -cn + n \le 0 \Rightarrow c \ge 1$$

- Base case?

Changing variables

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

– Rename: $m = \lg n \Rightarrow n = 2^m$

$T(2^m) = 2T(2^{m/2}) + m$

– Rename: $S(m) = T(2^m)$

$S(m) = 2S(m/2) + m \Rightarrow S(m) = O(m\lg m)$ (demonstrated before)

$T(n) = T(2^m) = S(m) = O(m\lg m)=O(\lg n\lg\lg n)$

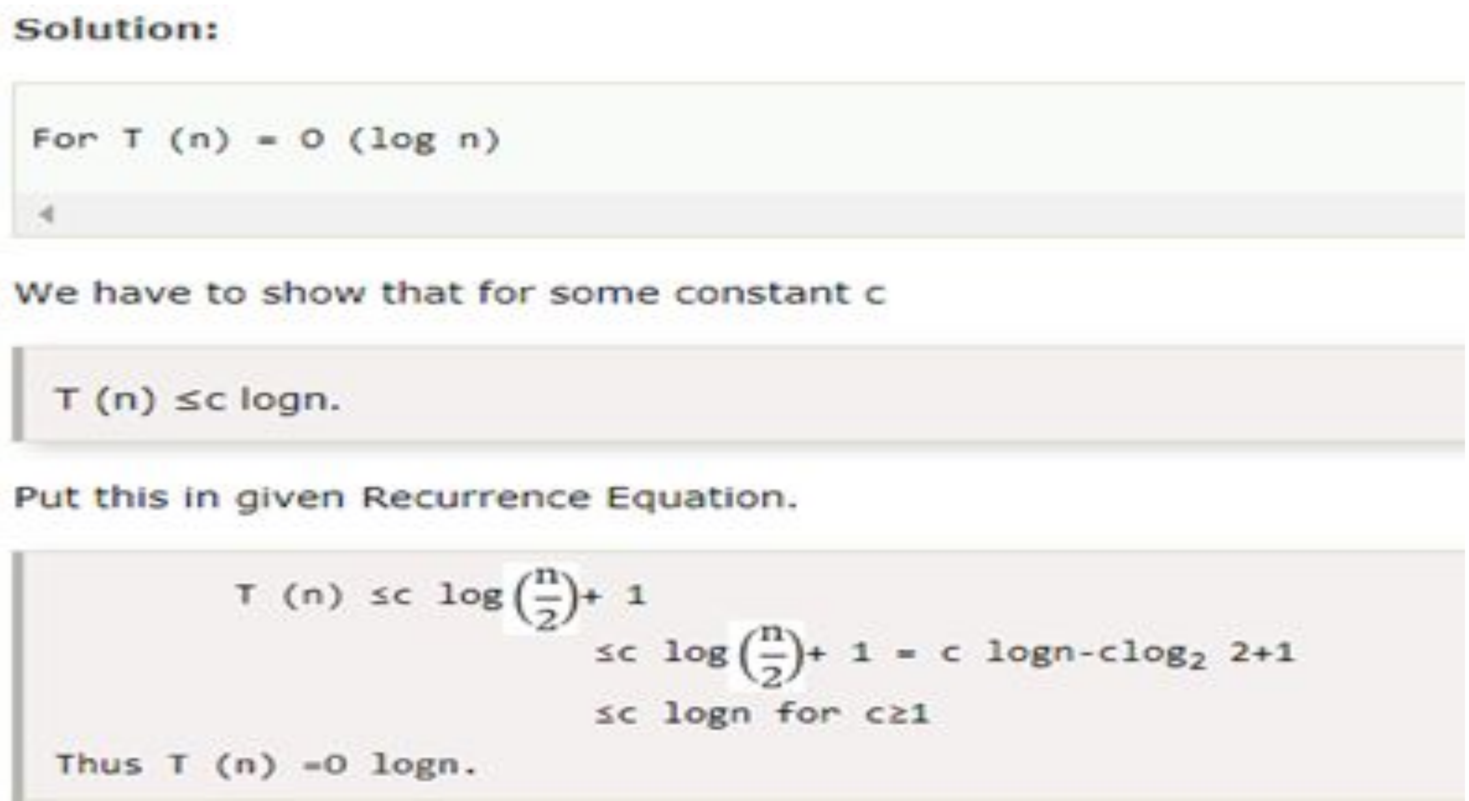
Idea: transform the recurrence to one that you have seen before

More Examples (Substitution method)

For Example1 Solve the equation by Substitution Method.

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

We have to show that it is asymptotically bound by O (log n).



Solution:

For T (n) = O (log n)

We have to show that for some constant c

T (n) $\leq$c logn.

Put this in given Recurrence Equation.

$$\begin{aligned} T\ (n) &\leq c\ \log\left(\frac{n}{2}\right) + 1 \\ &\leq c\ \log\left(\frac{n}{2}\right) + 1 = c\ \text{logn} - c\log_2 2 + 1 \\ &\leq c\ \text{logn for } c \geq 1 \end{aligned}$$

Thus T (n) =O logn.

(Example-Substitution method)

- T(n) = 2T(floor(n/2)) +n

  We guess that the solution is T(n)=0(n lg n).

  i.e. to show that T(n) ≤ cn lg n , for some constant c> 0 and n ≥ m.

  Assume that this bound holds for [n/2]. So , we get

  T(n) ≤ 2(c floor (n/2) lg(floor(n/2))) + n

  ≤ cn lg(n/2) + n

  = cn lg n – cn lg 2 + n

  = cn lg n – cn + n

  ≤ cn lg n

  where , the last step holds as long as c≥ 1.

- *Boundary conditions :*

Suppose , $T(1)=1$ is the sole boundary condition of the recurrence .

then , for $n=1$, the bound $T(n)\leq c\, n \lg n$ yields

$T(1)\leq c \lg 1=0$, which is at odds with $T(1)=1$.

Thus ,the base case of our inductive proof fails to hold.

To overcome this difficulty , we can take advantage of the asymptotic notation which only requires us to prove

$T(n)\leq c\, n \lg n$ for $n\geq m$.

The idea is to remove the difficult boundary condition $T(1)= 1$ from consideration.

Thus , we can replace $T(1)$ by $T(2)$ as the base cases in the inductive proof , letting $m=2$.

From the recurrence , with $T(1) = 1$, we get
$T(2)=4$
We require $T(2)\leq c\ 2 \lg 2$
It is clear that , any choice of $c\geq 2$ suffices for the base cases

- *Have we proved the case for $n = 3$.*
- *Have we proved that $T(3) \leq c\ 3 \lg 3$.*
- *No. Since $\text{floor}(3/2) = 1$ and for $n = 1$, it does not hold. So induction does not apply on $n = 3$.*
- *From the recurrence, with $T(1) = 1$, we get $T(3) = 5$.*

The inductive proof that $T(n) \leq c\ n \lg n$ for some constant $c\geq 1$
can now be completed by choosing c large enough that
$T(3)\leq c\ 3 \lg 3$ also holds.
It is clear that , any choice of $c \geq 2$ is sufficient for this to hold.
Thus we can conclude that $T(n) \leq c\ n \lg n$ for any $c \geq 2$ and $n \geq 2$.

Wrong Application of induction

Given recurrence: $T(n) = 2T(n/2) + 1$

Guess: $T(n) = O(n)$

Claim : $\exists$ some constant c and n_0, such that $T(n) <= cn$, $\forall n >= n_0$.

Proof: Suppose the claim is true for all values $<= n/2$

then

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

$<= 2.c.(n/2) +1$ (by induction hypothesis)

$= cn+1$

$<= (c+1)\,n$, $\forall\, n>=1$

Why is it Wrong?

Note that $T(n/2) <= cn/2 => T(n) <= (c+1)n$ (this statement is true but does not help us in establishing a solution to the problem)

$T(1) <= c$ (when $n=1$)

$T(2) <= (c+1).2$

$T(2^2) <= (c+2).2^2$.

.

.

.

$T(2^i) <= (c+i).2^I$

So, $T(n) = T(2^{\log n}) = (c + \log n)n$

$= \Theta(n \log n)$

What if we have extra lower order terms?

- So, does that mean that the claim we initially made that $T(n)=O(n)$ was wrong ?
- No.
- Recall:
- $T(n)=2T(n/2)+1$ (given)

  $\le 2.c.(n/2) + 1$ (by induction hypothesis)

  $= cn+1$
- Note that in the proof we have an extra lower order term in our inductive proof.

Given Recurrence: $T(n) = 2T(n/2) + 1$

Guess: $T(n) = O(n)$

Claim : ∃ some constant c and n_0 , such that $T(n) <= cn - b$, Ɐ $n >= n_0$ Proof:

Suppose the claim is true for all values <= n/2

then

$$\begin{aligned} T(n) &= 2T(n/2)+1 \\ &<= 2[c(n/2)-b]+1 \\ &<= cn-2b+1 \\ &<= cn-b+(1-b) \\ &<= cn-b , \text{Ɐ } b>=1 \end{aligned}$$

Thus, $T(n/2) <= cn/2 – b$

=> $T(n) <= cn – b$, Ɐ $b>=1$

Hence, by induction $T(n) = O(n)$, i.e. Our claim was true.

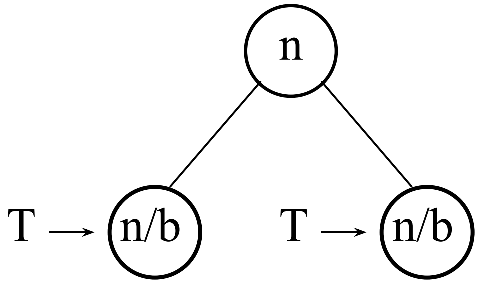
Hence proved.

Solving Recurrences using Recursion Tree Method

- Here while solving recurrences, we divide the problem into subproblems of equal size.

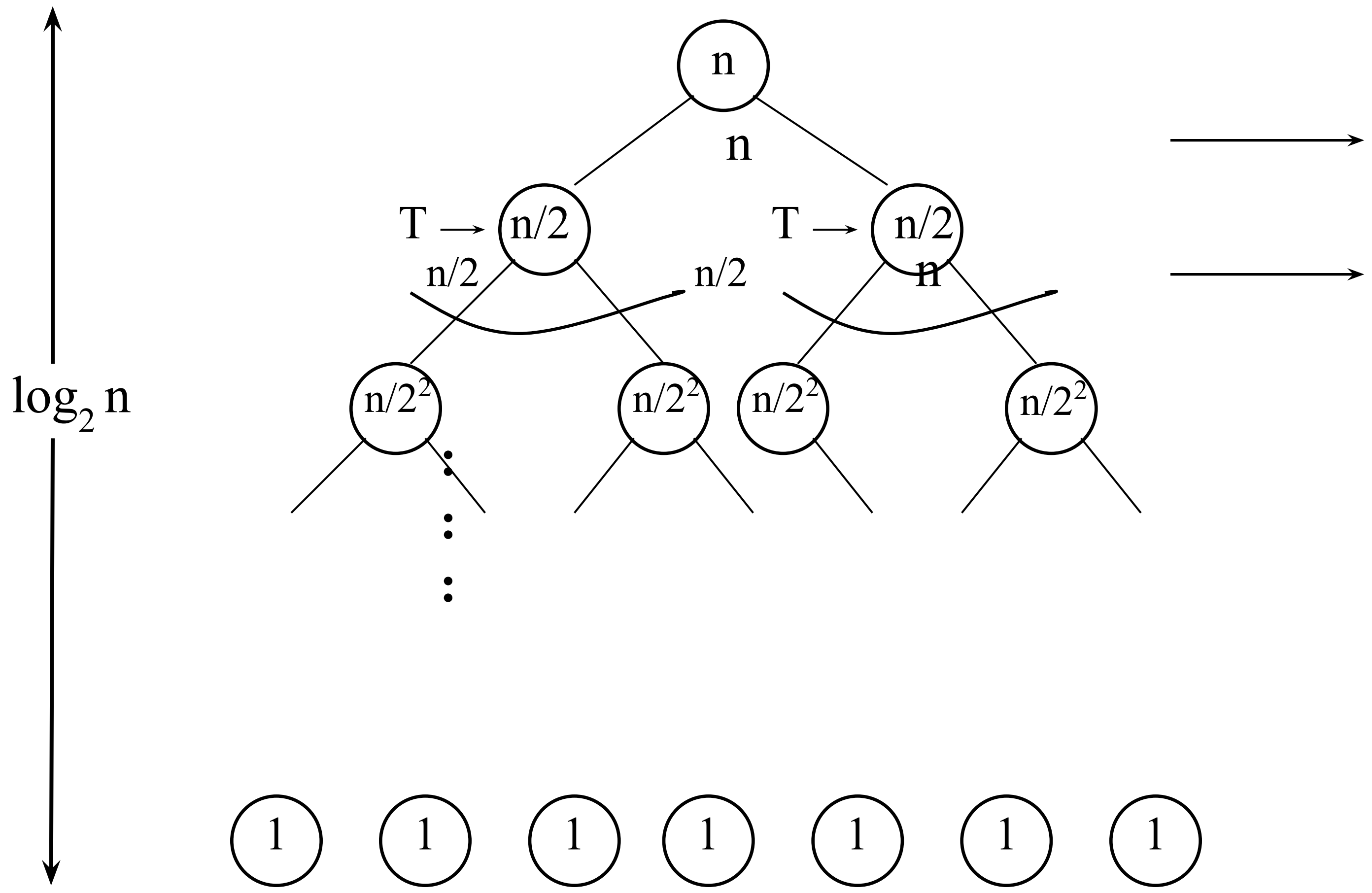
  For e.g., $T(n) = a\,T(n/b) + f(n)$ where $a \geq 1$,$b > 1$ and $f(n)$ is a given function .

  F(n) is the cost of splitting or combining the sub problems.



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

The recursion tree for this recurrence is :



When we add the values across the levels of the recursion tree, we get a value of n for every level.

We have $n + n + n + \ldots\ldots$ log n times

$= n\,(1 + 1 + 1 + \ldots\ldots$ log n times)

$= n\,(\log_2 n)$

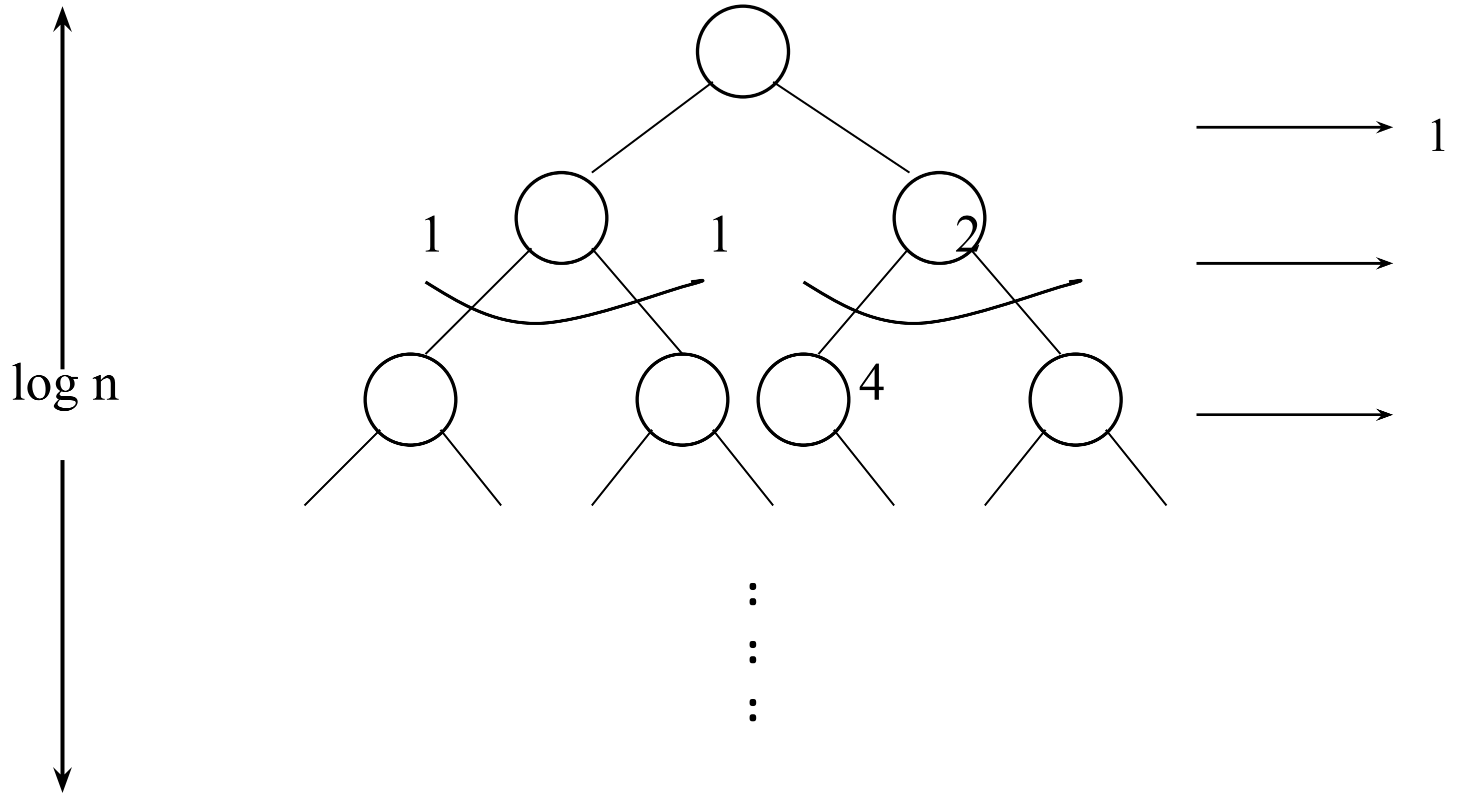
$= \Theta\,(n \log n)$

$T(n) = \Theta\,(n \log n)$

II.

Given : **T(n) = 2T(n/2) + 1**

Solution : The recursion tree for the above recurrence is



Now we add up the costs over all levels of the recursion tree, to determine the cost for the entire tree :

We get series like

$1 + 2 + 2^2 + 2^3 + \ldots\ldots$ log n times which is a G.P.

[So, using the formula for sum of terms in a G.P. :

$$a + ar + ar^2 + ar^3 + \ldots\ldots + ar^{n-1} = \frac{a(r^n - 1)}{r - 1} \quad]$$

$$= \frac{1\,(2^{\log n} - 1)}{2 - 1}$$

$= n - 1$

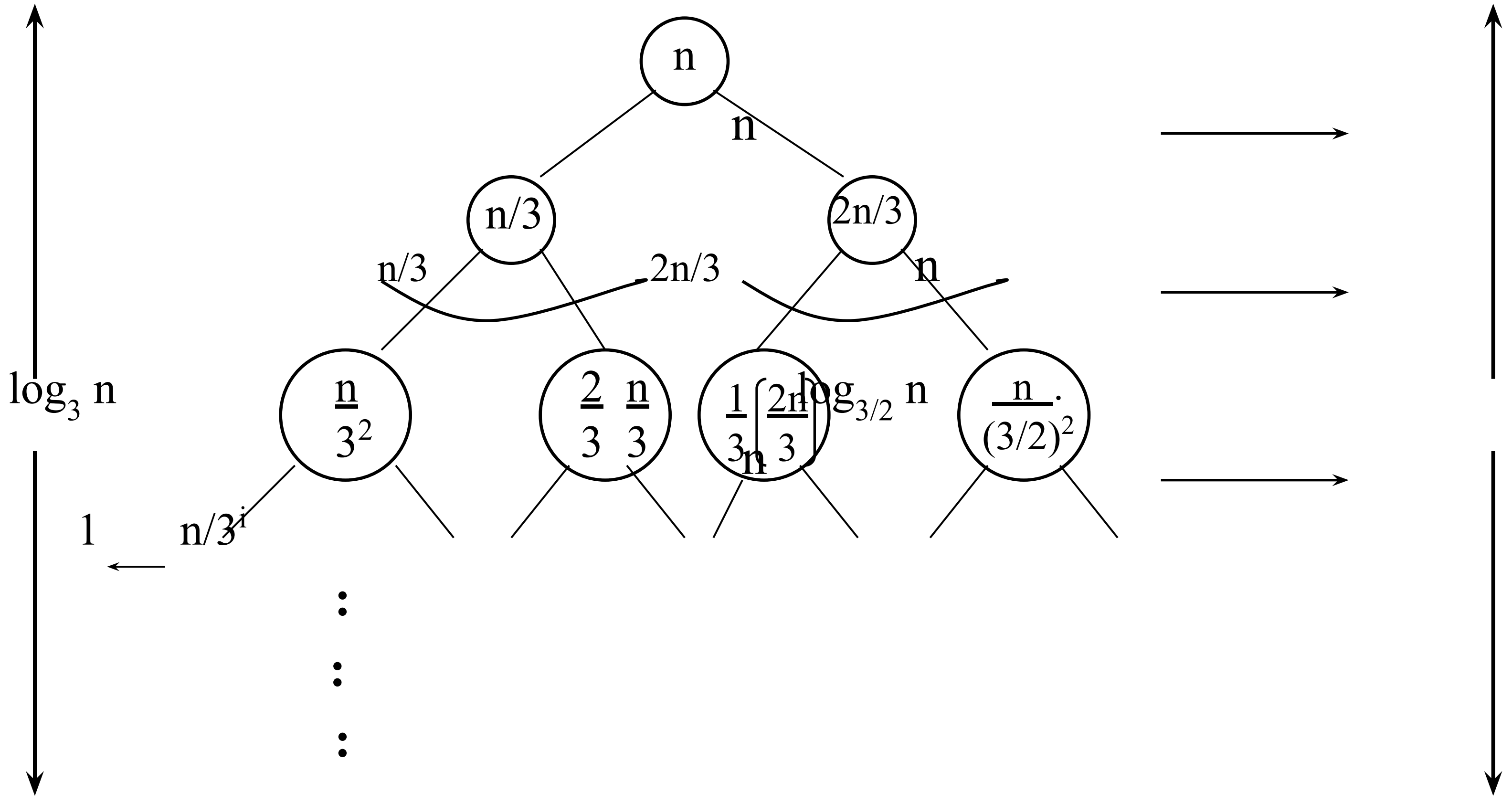
$= \Theta\,(n - 1)$ (neglecting the lower order terms)

$= \Theta\,(n)$

III.

Given : $\mathbf{T(n) = T(n/3) + T(2n/3) + n}$

Solution : The recursion tree for the above recurrence is



When we add the values across the levels of the recursion tree , we get a value of n for every level.

Since the shortest path from the root to the leaf is

$$n \rightarrow \frac{n}{3} \rightarrow \frac{n}{3^2} \rightarrow \frac{n}{3^3} \rightarrow \ldots.\ 1$$

we have 1 when $\frac{n}{3^i} = 1$

$\Rightarrow \quad n = 3^i$

Taking $\log_3$ on both the sides

$\Rightarrow \quad \log_3 n = i$

Thus the height of the shorter tree is $\log_3 n$

$T(n) \geq n \log_3 n \quad \ldots \quad A$

Similarly, the longest path from root to the leaf is

$$n \rightarrow \left(\frac{2}{3}\right) n \rightarrow \left(\frac{2}{3}\right)^2 n \rightarrow \dots 1$$

So rightmost will be the longest

when $\left(\frac{2}{3}\right)^k n = 1$

or $\frac{n}{(3/2)^k} = 1$

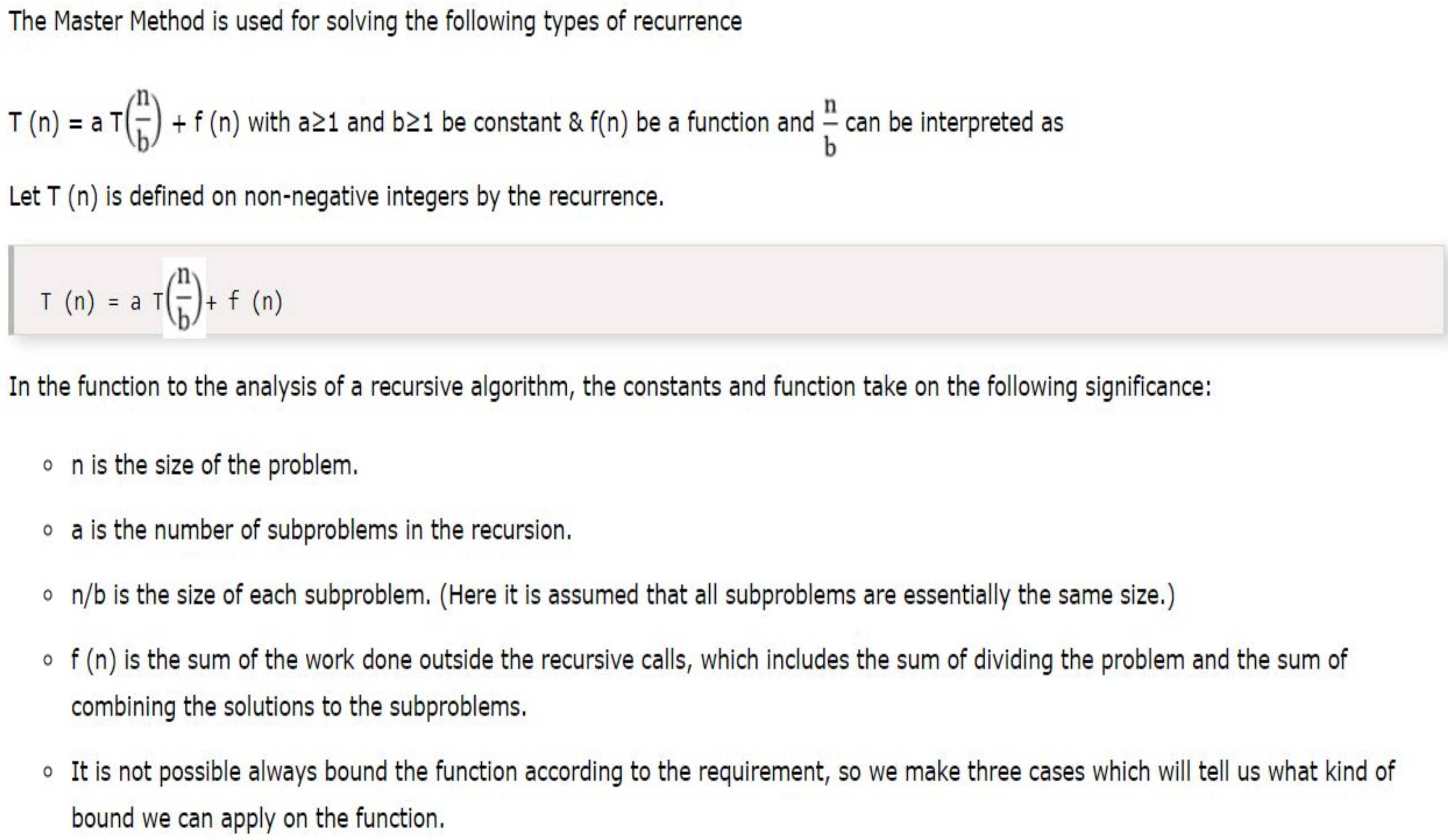
$\Rightarrow \quad k = \log_{3/2} n$

$T(n) \le n \log_{3/2} n \quad \dots \textcircled{B}$

Since base does not matter in asymptotic notation , we guess from $\textcircled{A}$ and $\textcircled{B}$

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

Master Method



The Master Method is used for solving the following types of recurrence

$T(n) = a\,T\left(\frac{n}{b}\right) + f(n)$ with a≥1 and b≥1 be constant & f(n) be a function and $\frac{n}{b}$ can be interpreted as

Let T (n) is defined on non-negative integers by the recurrence.

$$T(n) = a\,T\left(\frac{n}{b}\right) + f(n)$$

In the function to the analysis of a recursive algorithm, the constants and function take on the following significance:

- n is the size of the problem.
- a is the number of subproblems in the recursion.
- n/b is the size of each subproblem. (Here it is assumed that all subproblems are essentially the same size.)
- f (n) is the sum of the work done outside the recursive calls, which includes the sum of dividing the problem and the sum of combining the solutions to the subproblems.
- It is not possible always bound the function according to the requirement, so we make three cases which will tell us what kind of bound we can apply on the function.

Master Theorem:

It is possible to complete an asymptotic tight bound in these three cases:

$$T(n)=\begin{cases}\Theta\left(n^{\log_b a}\right) & f(n)=O\left(n^{\log_b a-\varepsilon}\right)\\ \Theta\left(n^{\log_b a}\log n\right) & f(n)=\Theta\left(n^{\log_b a}\right)\\ \Theta(f(n)) & f(n)=\Omega\left(n^{\log_b a+\varepsilon}\right)\text{AND}\\ & af(n/b)<cf(n)\text{ for large } n\end{cases}\quad \begin{matrix}\varepsilon>0\\ c<1\end{matrix}$$

Case1: If f (n) = $O\left(n^{\log_b a-\varepsilon}\right)$ for some constant ε >0, then it follows that:

> T (n) = Θ $\left(n^{\log_b a}\right)$

Example:

T (n) = 8 T $\left(\frac{n}{2}\right)+1000n^2$ apply master theorem on it.

Solution:

Compare T (n) = 8 T $\left(\frac{n}{2}\right)+1000n^2$ with

T (n) = a T $\left(\frac{n}{b}\right)+f(n)$ *with* $a \geq 1$ *and* $b > 1$

a = 8, b=2, f (n) = 1000 n^2, $\log_b a = \log_2 8 = 3$

Put all the values in: f (n) = $O\left(n^{\log_b a-\varepsilon}\right)$

1000 n^2 = O ($n^{3-\varepsilon}$)

If we choose ε=1, we get: 1000 n^2 = O (n^{3-1}) = O (n^2)

Since this equation holds, the first case of the master theorem applies to the given recurrence relation, thus resulting in the conclusion:

T (n) = Θ $\left(n^{\log_b a}\right)$

Therefore: T (n) = Θ (n^3)

Case 2: If it is true, for some constant k ≥ 0 that:

Case 2: If it is true, for some constant k ≥ 0 that:

F (n) = θ $\left(n^{\log_b a} \log^k n\right)$ then it follows that: T (n) = θ $\left(n^{\log_b a} \log^{k+1} n\right)$

Example:

T (n) = 2 $T\left(\frac{n}{2}\right)+10n$, solve the recurrence by using the master method.

As compare the given problem with T (n) = a T$\left(\frac{n}{b}\right)+f(n)$ *with* $a \geq 1$ *and* $b > 1$ a = 2, b=2, k=0, f (n) = 10n, $\log_b a = \log_2 2 = 1$

Put all the values in f (n) =θ $\left(n^{\log_b a} \log^k n\right)$, we will get

10n = θ (n^1) = θ (n) which is true.

Therefore: T (n) = θ $\left(n^{\log_b a} \log^{k+1} n\right)$

= θ (n log n)

Case 3: If it is true f(n) = $\Omega\left(n^{\log_b a+\varepsilon}\right)$ for some constant ε >0 and it also true that: a f $\left(\frac{n}{b}\right) \leq cf(n)$ for some constant c<1 for large value of n ,then :

```
T (n) = Θ((f (n))
```

Example: Solve the recurrence relation:

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

Solution:

Compare the given problem with T (n) = a T $\left(\frac{n}{b}\right)+ f(n)$ *with* $a \geq 1$ *and* $b > 1$

a= 2, b =2, f (n) = n^2, $\log_b a$ = $\log_2 2$ =1

Put all the values in f (n) = Ω $\left(n^{\log_b a+\varepsilon}\right)$ ..... (Eq. 1)

If we insert all the value in (Eq.1), we will get

  $n^2 = \Omega(n^{1+\varepsilon})$ put ε =1, then the equality will hold.

  $n^2 = \Omega(n^{1+1}) = \Omega(n^2)$

Now we will also check the second condition:

  $2\left(\frac{n}{2}\right)^2 \leq cn^2 \Rightarrow \frac{1}{2}n^2 \leq cn^2$

If we will choose c =1/2, it is true:

  $\frac{1}{2}n^2 \leq \frac{1}{2}n^2$ ∀ n ≥1

So it follows: T (n) = Θ ((f (n))

    T (n) = $\Theta(n^2)$