Recurrence Relation

Introduction

- Many algorithms (divide and conquer) are recursive in nature.
- When we analyze them, we get a recurrence relation for time complexity.
- We get running time as a function of n (input size) and we get the running time on inputs of smaller sizes.
- A recurrence is a recursive description of a function, or a description of a function in terms of itself.
- A recurrence relation recursively defines a sequence where the next term is a function of the previous terms.

Methods to Solve Recurrence

- Substitution
- Recurrence tree
- Master method

Substitution Method – Example 1

- We make a guess for the solution and then we use mathematical induction to prove the guess is correct or incorrect.

Example 1:

Time to solve the instance of size $n-1$

Time to solve the instance of size n

$$\boldsymbol{T(n) = T(n-1) + n} \quad (1)$$

- Replacing n by $n-1$ and $n-2$, we can write following equations.

$$\boldsymbol{T(n-1) = T(n-2) + n - 1} \quad (2)$$

$$\boldsymbol{T(n-2) = T(n-3) + n - 2} \quad (3)$$

- Substituting equation 3 in 2 and equation 2 in 1 we have now,

$$\boldsymbol{T(n) = T(n-3) + n - 2 + n - 1 + n} \quad (4)$$

Substitution Method – Example 1

$$T(n) = T(n-3) + n - 2 + n - 1 + n$$ 4

- From above, we can write the general form as,

$$T(n) = T(n-k) + (n-k+1) + (n-k+2) + \ \dots + n$$

- Suppose, if we take $k \ = \ n$ then,

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

$$T(n) = 0 + 1 + 2 + \ \dots + n$$

$$\boxed{T(n) = \frac{n(n+1)}{2} = O(n^2)}$$

Substitution Method – Example 2

$$t(n) = \begin{cases} c1 & if\ n = 0 \\ c2 + t(n-1) & o/w \end{cases}$$

- Rewrite the equation, $t(n) = c2 + \underline{t(n-1)}$
- Now, replace **n** by **n – 1** and **n − 2**

$$t(n-1) = c2 + \underline{t(n-2)}$$
$$\underline{t(n-2)} = c2 + t(n-3)$$

$$\therefore \underline{t(n-1)} = c2 + c2 + t(n-3)$$

- Substitute the values of **n – 1** and **n − 2**

$$t(n) = c2 + c2 + c2 + t(n-3)$$

- In general,

$$t(n) = kc2 + t(n-k)$$

- Suppose if we take $k = n$ then,

$$t(n) = nc2 + t(n-n) = nc2 + t(0)$$
$$\boxed{t(n) = nc2 + c1 = \boldsymbol{O(n)}}$$

Substitution Method Exercises

- Solve the following recurrences using substitution method.

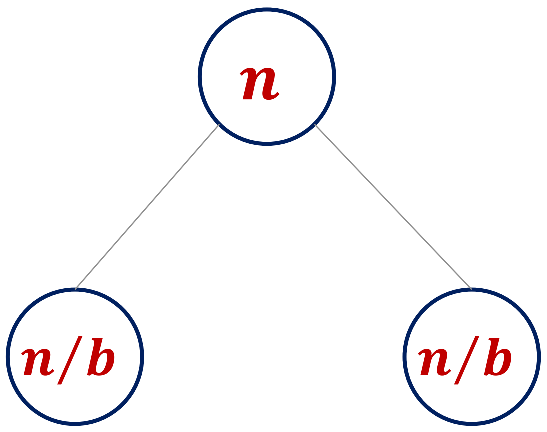
1. $T(n) = \begin{cases} 1 & \text{if } n = 0 \text{ or } 1 \\ T(n-1) + n - 1 & \text{o/w} \end{cases}$

2. $T(n) = T(n-1) + 1$ and $T(1) = \theta(1)$.

$$\begin{aligned} T(n) &= T(n-1) + n \\ &= T(n-2) + (n-1) + n \\ &= T(n-3) + (n-2) + (n-1) + n \\ &\vdots \\ &= T(0) + 1 + 2 + \ldots + (n-2) + (n-1) + n \\ &= T(0) + \frac{n(n+1)}{2} = O(n^2) \end{aligned}$$

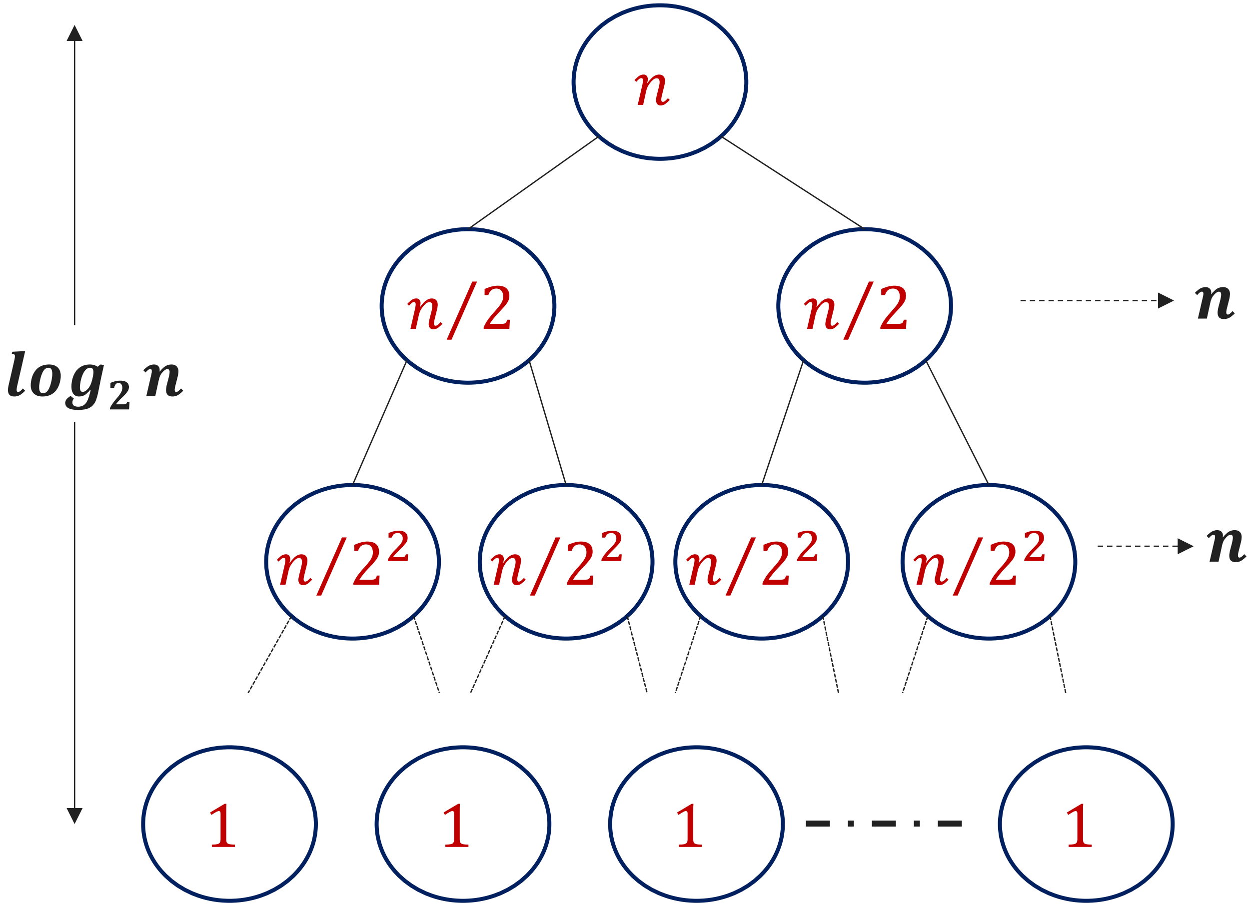
Recurrence Tree Method

- In recurrence tree, each node represents the **cost of a single sub-problem** in the set of recursive function invocations.
- We sum the **costs within each level** of the tree to obtain a set of per level costs.
- Then we sum the all the **per level costs** to determine the total cost of all levels of the recursion.
- Here while solving recurrences, we **divide the problem** into sub-problems of equal size.
- E.g., $T(n) = a\,T(n/b) + f(n)$ where $a > 1$, $b > 1$ and $f(n)$ is a given function.
- $F(n)$ is the cost of **splitting or combining** the sub problems.



Recurrence Tree Method

The recursion tree for this recurrence is



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

- When we add the values across the levels of the recursion tree, we get a value of n for every level.
- The bottom level has $2^{\log n}$ nodes, each contributing the cost $T(1)$.
- We have $n + n + n + \dots\dots$ $\log n$ $times$

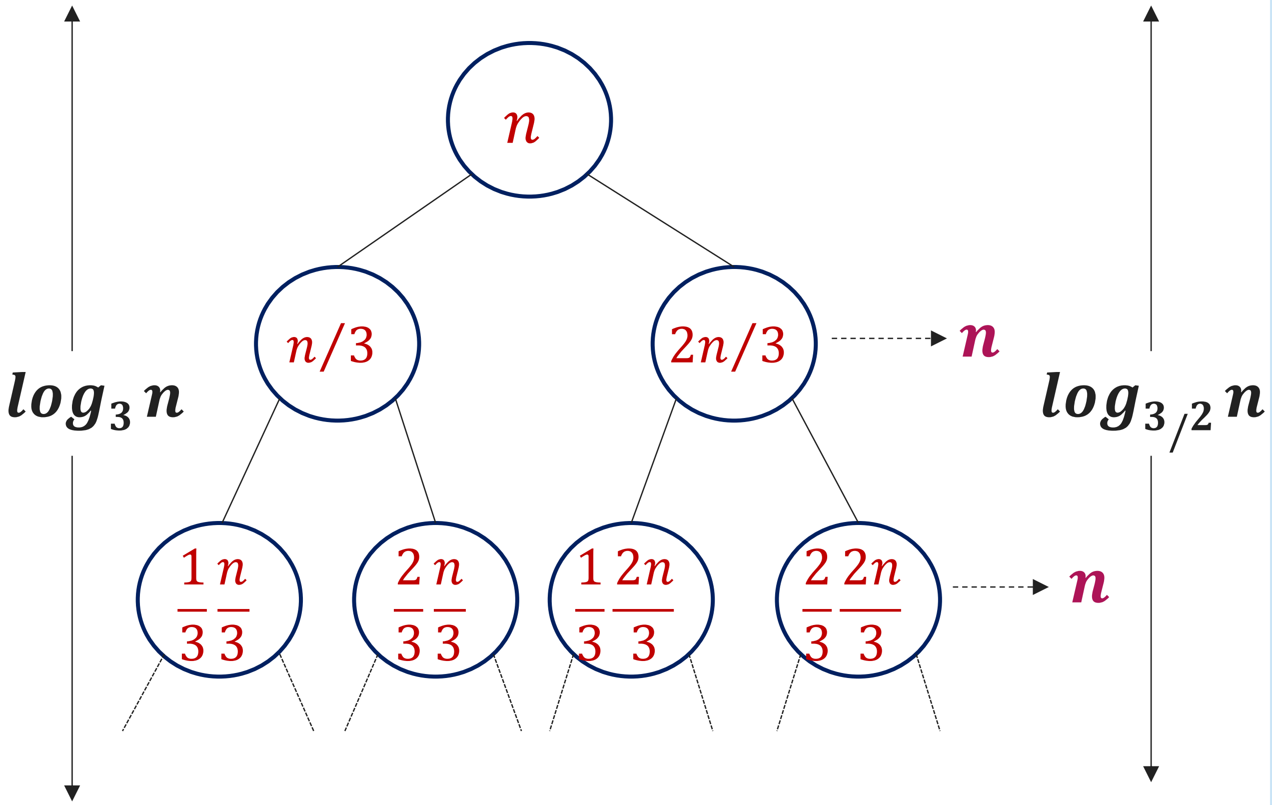
$$T(n) = \sum_{i=0}^{log_2 n - 1} n + 2^{log\,n} T(1)$$

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

$$\boxed{T(n) = O(n \log n)}$$

Recurrence Tree Method

The recursion tree for this recurrence is



Example 2: $T(n) = T(n/3) + T(2n/3) + n$

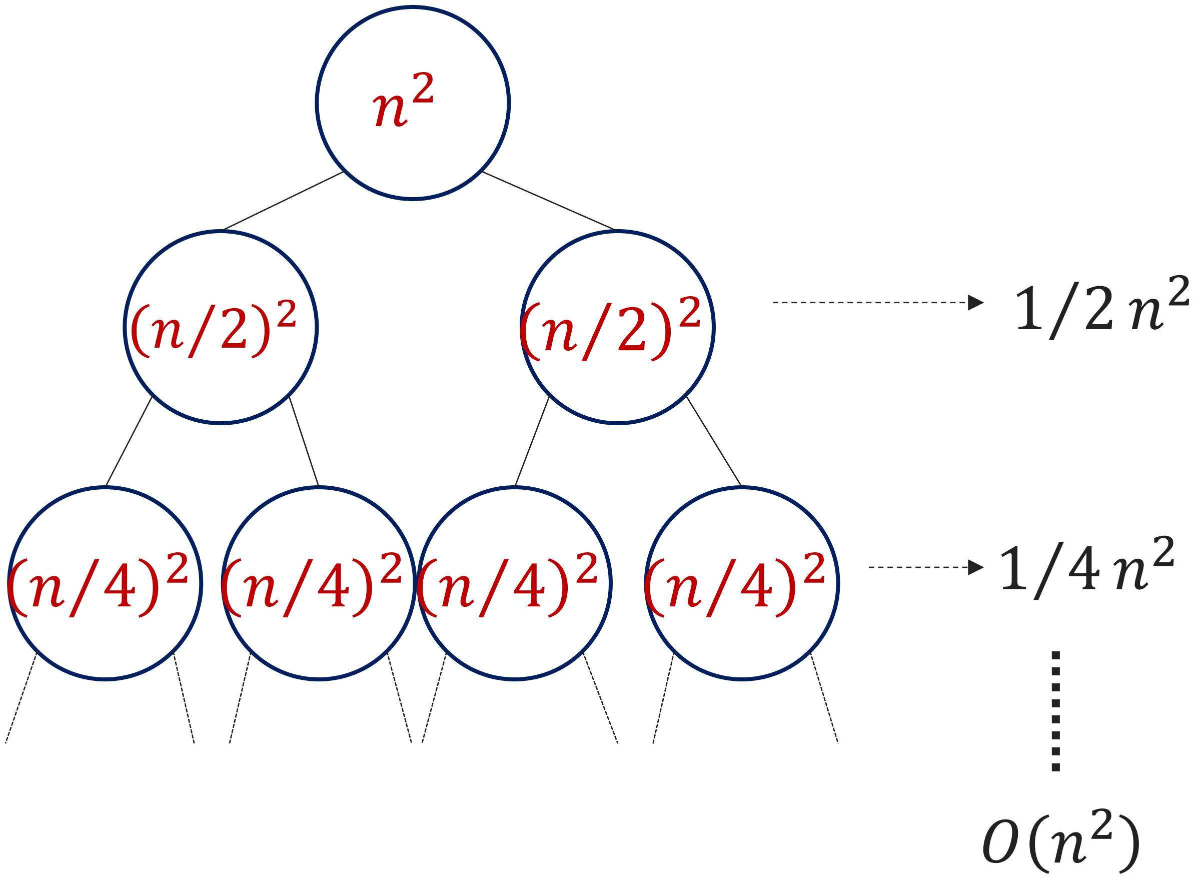
- When we add the values across the levels of the recursion tree, we get a value of n for every level.

$$T(n) = \sum_{i=0}^{\log_{3/2} n - 1} n + n^{\log_{3/2} 2} T(1)$$

$$\boxed{T(n) \in n \log_{3/2} n}$$

Recurrence Tree Method

The recursion tree for this recurrence is



Example 3: $T(n) = 2T(n/2) + c.n^2$

- Sub-problem size at level i is $n/_{2^i}$
- Cost of problem at level i Is $\left(n/_{2^i}\right)^2$
- Total cost,

$$T(n) \le n^2 \sum_{i=0}^{\log_2 n-1} \left(\frac{1}{2}\right)^i$$

$$T(n) \le n^2 \sum_{i=0}^{\infty} \left(\frac{1}{2}\right)^i$$

$$T(n) \le 2n^2$$

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

Recurrence Tree Method - Exercises

- Example 1: $T(n) = T(n/4) + T(3n/4) + c.n$
- Example 2: $T(n) = 3T(n/4) + c.n^2$
- Example 3: $T(n) = T(n/4) + T(n/2) + n^2$
- Example 4: $T(n) = T(n/3) + T(2n/3) + n$

Master Theorem

- The master theorem is a **cookbook method** for solving recurrences.
- Suppose you have a recurrence of the form

$$T(n) = aT(n/b) + f(n)$$

Time to divide & recombine

Number of sub-problems

Time required to solve a sub-problem

- **This recurrence would arise in the analysis of a recursive algorithm.**
- When input size $\boldsymbol{n}$ is large, the problem is divided up into $\boldsymbol{a}$ sub-problems each of size $\boldsymbol{n/b}$. Sub-problems are solved recursively and results are recombined.
- The work to split the problem into sub-problems and recombine the results is $\boldsymbol{f(n)}$.

Masters Theorem for Dividing Functions

$$\boldsymbol{T(n) = aT(n/b) + f(n)}$$

where $a >= 1$ and $b > 1$, $f(n) = \theta(n^k log^p n)$

First find out the value of : 1. log_b^a

2. k

Based on the values of log_b^a and k, we define 3 cases:

Case 1: if $log_b^a > k$, then $\theta(n^{log_b^a})$

Case 2: if $log_b^a = k$,

if p > -1 then $\theta\left(n^k log^{p+1} n\right)$

if p = -1 then $\theta\left(n^k loglogn\right)$

if p < -1 then $\theta\left(n^k\right)$

Case 3: if $log_b^a < k$,

if p>=0, then $\theta\left(n^k log^p n\right)$

if p<0, then O(n^k)

Example 1 : $\boldsymbol{T(n) = 2T(n/2) + 1}$

Given, a = 2, b = 2,

f(n) = $\theta(1)$

= $\theta(n^0(\log n)^0)$

$k = 0, p = 0$

$log_2^2 = 1 > k = 0$

Case1 :$\theta(n^{log_2^2})$ = $\theta(n^1)$

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

$log_2^4 = 2 > k = 1$, p = 0

therefore case 1: $\theta(n^2)$

Solve Examples:

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

Master Theorem for dividing function

Case 1:

$T(n) = 2T(n/2) + 1$	$O(n)$
$T(n) = 4T(n/2) + 1$	$O(n^2)$
$T(n) = 4T(n/2) + n$	$O(n^2)$
$T(n) = 8T(n/2) + n^2$	$O(n^3)$
$T(n) = 16T(n/2) + n^2$	$O(n^4)$

Case 3:

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

$T(n) = 2T(n/2) + n^2$ $\quad O(n^2)$

$T(n) = 2T(n/2) + n^2\, logn$ $\quad O(n^2\, logn)$

Case 2:

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

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

Solve for:

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

Master Theorem For Subtract and Conquer/Decreasing Recurrences

Let T(n) be a function defined on positive n as shown below:

$$T(n) \leq \begin{cases} c, & if\, n \leq 1, \\ aT(n-b) + f(n), & n > 1, \end{cases} ,$$

For some constants c, a>0, b>0, k>=0 and function f(n). If f(n) is $O(n^k)$, then

1. If a<1 then $T(n) = O(n^k)$
2. If a=1 then $T(n) = O(n^{k+1})$
3. If a>1 then $T(n) = O(n^k a^{n/b})$

Examples :

Master Theorem for decreasing function

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

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

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

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

$$T(n) = T(n-2) + 1 \rightarrow \frac{n}{2} \rightarrow O(n)$$

$$T(n) = T(n-50) + n --------- O(n^2)$$