Unit 2 Recurrence Solving

⇒ A recurrence is an equation or inequality that describes a function in terms of its value on smaller inputs.

For example

Worst case running time $T(n)$ of Mergesort -

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

whose solution is -

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

In merge sort - we have two recursive call each for size $n/2$ and one call to function merge which merges two sorted arrays.

⇒ In this chapter, we study 3 methods of solving recurrences -

1) Substitution Method -
   ↳ we guess bound and then mathematical induction to prove our guess correct.

2) Recursion Tree -
   ↳ convert recurrence into a tree, whose nodes represent cost incurred at various levels of recursion. Then we bound summation to solve recurrence.

3) Masters Theorem -
   ↳ it provides bound for recurrences of form $T(n) = a\,T(n/b) + f(n)$. $a \geq 1$ $b > 1$ $f(n)$ is function

Technicalities in Recurrences

In practice, we neglect certain things when we state and solve recurrences.

Ex. Merge sort on 'n' elements when 'n' is odd gives subproblems of size $\lfloor n/2 \rfloor$ and $\lceil n/2 \rceil$.

Thus -

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

Boundary conditions is another class of details that we typically ignore.

Thus:

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

Ignoring floors, ceilings and boundary conditions.

1) Substitution Method -

This involves two steps-

1) Guess the form of solution.
2) Use mathematical induction to find the constant and show that solution works.

We substitute the guessed solution for the function when applying hypothesis to smaller values. This is a powerful method but it requires to guess the form of answer in order to apply it.

(1) $T(n) = 2T(\lfloor n/2 \rfloor) + n$.

Guess: $T(n) = O(n \log n)$.

Thus substitution method requires us to prove that $T(n) \leq cn\log n$ for appropriate choice of $c > 0$

we start by assuming that this bound is true for all positive $m < n$ i.e. $T(m) \leq cm\log m$

Let $m = \lfloor n/2 \rfloor$

$$T(\lfloor n/2 \rfloor) \leq c \cdot \lfloor n/2 \rfloor \log(\lfloor n/2 \rfloor)$$

Substituting this in the given recurrence eq -

$$\begin{aligned} T(n) &\leq 2 \cdot (c \cdot \lfloor n/2 \rfloor \log(\lfloor n/2 \rfloor)) + n \\ &\leq cn \log(n/2) + n \\ &= cn[\log n - \log 2] + n \\ &= cn\log n - cn + n \\ &\leq cn\log n - n(c-1) \end{aligned}$$

Thus $T(n) \leq cn\log n$ for $c \geq 1$

Now, this solution should hold for the boundary condition.

Let $T(1) = 1$ is the only boundary condition.

Then for $n = 1$, $T(n) \leq c \cdot n\log n$ gives.

$T(1) \leq c \cdot 1 \log 1$

$\leq 0 \longrightarrow$ fail for boundary condition

Here, we use the definition of asymptotic notation to overcome this obstacle in inductive hypothesis.

We notice that the given recurrence does not depend upon $T(1)$ directly when $n \geq 3$

As $n=2 \rightarrow T(2) = 2T(1) + 2$
$n=3 \rightarrow T(3) = 2T(1) + 3$.
} Directly depend on $T(1)$.

Thus we remove this boundary condition from our inductive hypothesis proof.

Asymptotic notation requires us to prove that $T(n) \leq c \cdot n \log n$ for all $n \geq n_0$ where n_0 is constant that we can choose.

Thus, we replace boundary condition $T(1)$ by $T(2)$ and $T(3)$ in inductive hypothesis proof by letting $n_0 = 2$.

We derive from $T(1) = 1$

$T(2) = 2T(1) + 2 = 2 + 2 = 4$.
$T(3) = 2T(1) + 3 = 2 + 3 = 5$

Now, we complete the inductive proof.

$T(2) = c \cdot 2 \log 2 = 2c$
$T(3) = c \cdot 3 \log 3 = 3c \cdot \log 3$
} $c \geq 2$

* Making a good guess

- no general way to guess the correct solution.
- Creativity and experience.
- Loose upper bound and lower bound (heuristic).
- recursion tree.

$T(n) = 2T(\lfloor n/2 \rfloor + 17) + n$.

This becomes difficult to guess because of added "17" in T.
However this term does not substantially affect the solution to recurrence equation.

When n is large - difference between $\lfloor n/2 \rfloor$ and $\lfloor n/2 \rfloor + 17$ is not that large.
We can make the guess as $T(n) = O(n \log n)$.

Another way to make a good guess is to prove loose upper and lower bound. and then reduce the range of uncertainty.

Ex: We may start with a lower bound $T(n) = \Omega(n)$ since we have the term 'n' in recurrence.
Also, we can prove initial upperbound of.
$T(n) = O(n^2)$.

Then, we gradually lower the upperbound and raise the lower bound until we converge on correct asymptotically tight solution of $T(n) = \Theta(n \log n)$.

Subtleties -

Sometimes, it may happen that we made a correct asymptotic guess, but the math fails to work out in the induction.

- [Inductive assumption is not strong enough to prove desired bound].
- Then, we revise the guess by subtracting a lower order term when we hit such snag.

Ex:- $T(n) = T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + 1$

Guess $T(n) = O(n)$.
$\leq c \cdot n$. for appropriate constant c.

$T(n) \leq c \cdot \lfloor n/2 \rfloor + c \cdot \lceil n/2 \rceil + 1$
$= cn + 1 \not\leq cn$ for any value of 'c'.

We may try a larger guess $O(n^2)$, which may work.
But our initial guess of $O(n)$ is correct.
New guess $T(n) \leq cn - d$ where $d \geq 0$.

$T(n) \leq (c \cdot \lfloor n/2 \rfloor - d) + c(\lceil n/2 \rceil - d) + 1$
$= cn - 2d + 1$
$\leq$ ~~$cn - d(d+1)$~~
$\leq cn - d - (d-1)$
$d - 1 \geq 0$ $(d \geq 1)$ and large enough c.

Avoiding pitfalls

It is easy to err in use of asymptotic notation

Ex :- we can falsely prove $T(n) = O(n)$ by guessing $T(n) \leq c \cdot n$ and then arguing -

$$T(n) \leq 2(c\lfloor n/2 \rfloor) + n$$
$$\leq cn + n$$
$$\leq (c+1)n$$
$$= O(n) \quad \text{as 'c' is constant.}$$

THIS IS WRONG

Because we have not proved the exact form of inductive hypothesis that is $T(n) \leq cn$.

Changing variables

$$T(n) = 2T(\lfloor \sqrt{n} \rfloor) + \log n.$$

Now this equation looks difficult.

we do change of variables.

Let $m = \log n \Rightarrow n = 2^m$.

$$T(2^m) = 2T(2^{m/2}) + m.$$ — don't worry about $\sqrt{n}$ to be integer.

Let $S(m) = T(2^m)$

$$S(m) = 2S(m/2) + m.$$

$\rightarrow$ we have already solved this recurrence equation.

$S(m) = O(m \log m)$.

Changing back from $S(m)$ to $T(n)$,

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

2) Recursion Tree Method -

In recursion tree, each node represent a cost of a single subproblem.

A recursion tree is used to make a good guess, which can be verified by substitution method.

When using a recursion tree to generate a good guess, we can often tolerate a small amount of "sloppiness", since we will be verifying our guess later on using substitution.

Also, if we are careful while drawing out a recursion tree, we can use it as a direct proof of a solution to a recurrence.

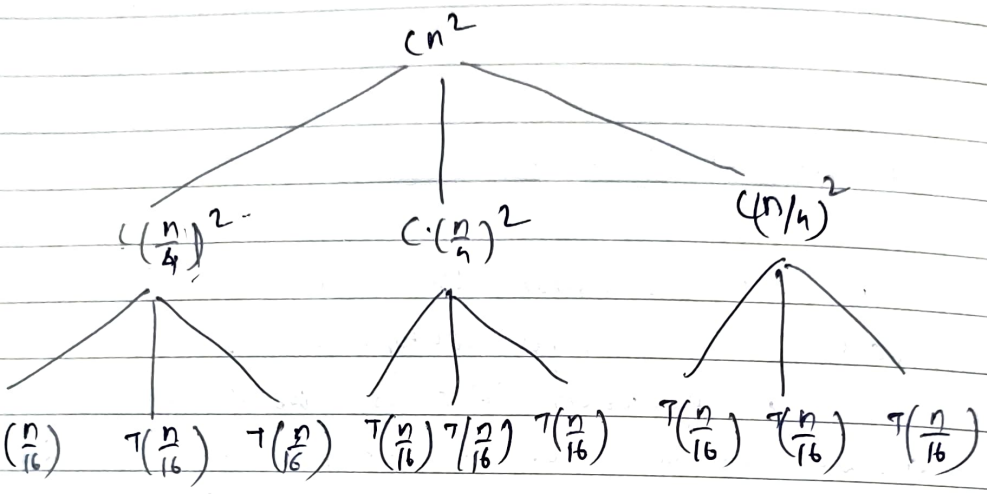
Ex: $T(n) = 3T(\lfloor n/4 \rfloor) + \Theta(n^2)$.

↳ $T(n) = 3T(n/4) + cn^2$. — Ignoring ceiling and $\epsilon \to 0$.

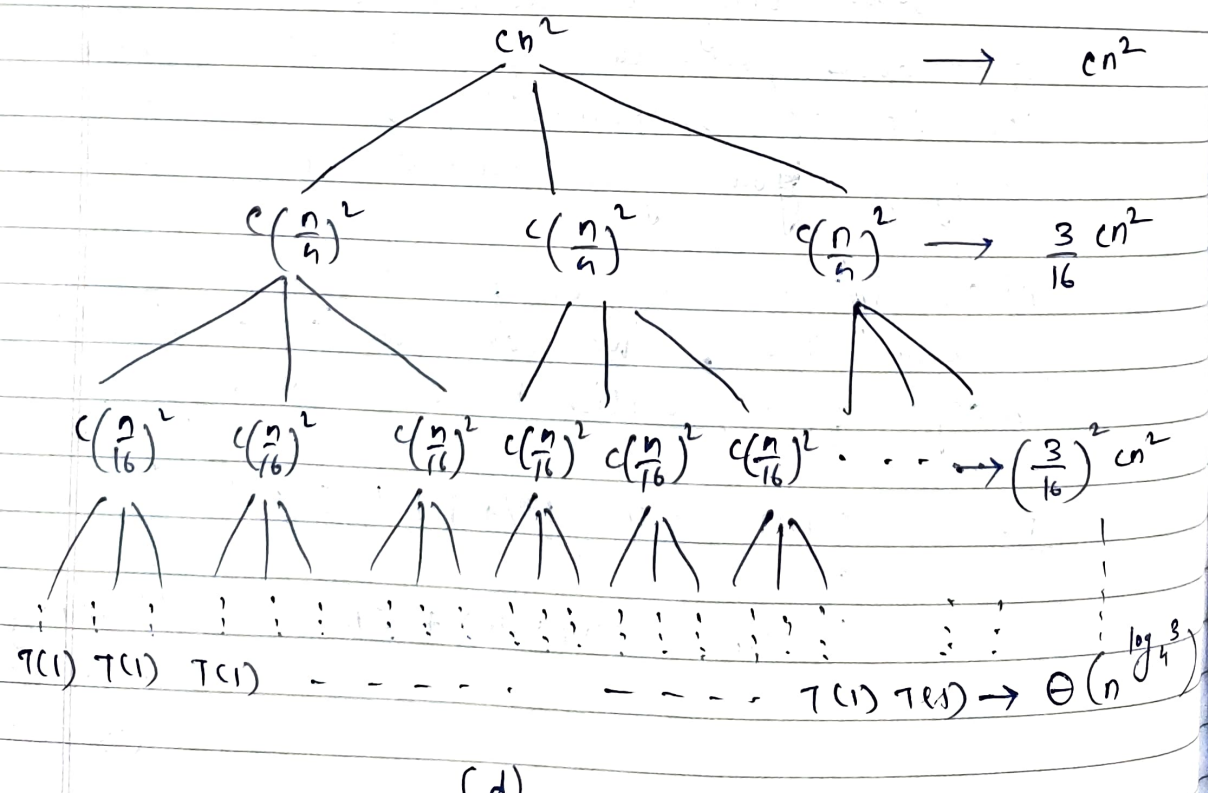
$T(n)$ cn^2

$T(n/4)$ $T(n/4)$ $T(n/4)$

(a) (b)



(c)



(d).

Problem size for a node at depth 'i' is $(n/4^i)$

Thus, subproblem hit $n=1$ when $\frac{n}{4^i} = 1$

$$\boxed{i = \log_4 n}$$

Thus tree has $(\log n + 1)$ levels – at depths. $0, 1, 2, \ldots \log_4 n$.

Next, we find cost at each level.

At each node at depth 'i' – $c \cdot \left(\frac{n}{4^i}\right)^2$ for i=0 to $\log n$

Total cost = $3^i \cdot c \cdot \left(\frac{n}{4^i}\right)^2 = \left(\frac{3}{16}\right)^i cn^2$

At bottom level – depth $\log_4 n$ – has $3^{\log_4 n}$ nodes.

$= n^{\log_4 3}$.

$$T(n) = cn^2 + \left(\frac{3}{16}\right) cn^2 + \left(\frac{3}{16}\right)^2 cn^2 + \ldots + \left(\frac{3}{16}\right)^{\log_4 n - 1} cn^2 + \theta\left(n^{\log_4 3}\right)$$

$$= \sum_{i=0}^{\log_4 n - 1} \left(\frac{3}{16}\right)^i cn^2 + \theta\left(n^{\log_4 3}\right)$$

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

Consider summation upto ∞. as 3/16 is decreasing with increase in value of 'i'.

$$T(n) = \sum_{i=0}^{\infty} \left(\frac{3}{16}\right)^i cn^2 + \theta\left(n^{\log_4 3}\right)$$

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

$$= \frac{16}{13} cn^2 + \theta\left(n^{\log_4 3}\right)$$

$$= \underline{\underline{O(n^2)}}$$

Now, we prove our guess by substitution method.

$T(n) \leq d \cdot n^2$ for some constant $d > 0$.

$$T(n) \leq 3T(n/4) + cn^2$$

$$\leq 3d\left(\frac{n}{4}\right)^2 + cn^2$$

$$\leq \frac{3}{16} dn^2 + cn^2$$

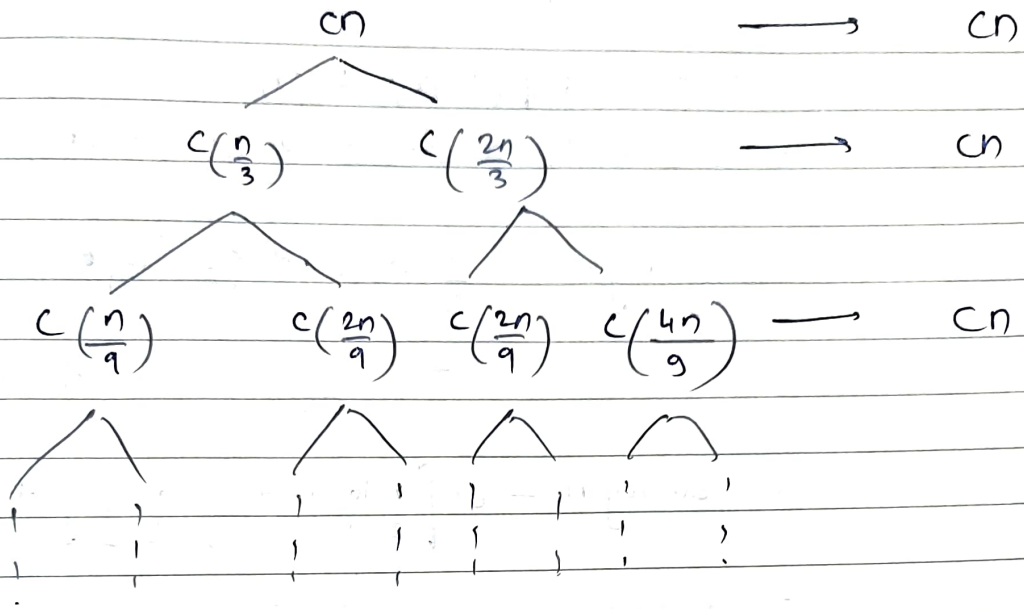
$$= dn^2 - \frac{13}{16} dn^2 + cn^2$$

$$= dn^2 - \left[\frac{13}{16} dn^2 - cn^2\right]$$

$T(n) \leq dn^2$ if $\frac{13}{16} dn^2 - cn^2 \geq 0$

$$\boxed{d \geq \frac{16}{13} c}$$

Ex :- $T(n) = T(n/3) + T(2n/3) + O(n)$.



In worst case -

$$n \to \frac{2}{3}n \to \left(\frac{2}{3}\right)^2 n \to \left(\frac{2}{3}\right)^3 n \to \cdots \to 1.$$

$$\left(\frac{2}{3}\right)^k n = 1 \qquad n = \left(\frac{3}{2}\right)^k. \qquad \boxed{k = \log_{3/2} n}$$

Thus - $T(n) = O(cn \log_{3/2} n)$

$= O(n \log n)$

If tree is complete tree of height $\log_{3/2} n$, we would have $2^{\log_{3/2} n} = n^{\log_{3/2} 2}$ leaves. But it is not the case.

we would have less than cn^2 cost for lower levels.

But we tolerate this sloppiness and attempt to prove that our guess is correct.

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

$\leq d n \log n$. where $d > 0$

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

$= d\frac{n}{3}\log\frac{n}{3} + d\cdot\frac{2n}{3}\log\frac{2n}{3} + cn$

$= \frac{dn}{3}\log n - \frac{dn}{3}\log 3 + \frac{2dn}{3}\log 2n$

$- \frac{2dn}{3}\log 3/2 + cn.$

$= dn\log n - d\left[\frac{n}{3}\log 3 + \frac{2n}{3}\log\frac{3}{2}\right] + cn.$

$= dn\log n - d\left[\frac{n}{3}\log 3 + \frac{2n}{3}\log 3 - \frac{2n}{3}\log 2\right] + cn$

$= dn\log n - dn\left[\log 3 - \frac{2}{3}\right] + cn.$

$T(n) \leq dn\log n$ as long as -

$$\boxed{d \geq \frac{c}{\log 3 - 2/3}}$$

3) Master's Theorem

It solves recurrences of form -

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

where $a \geq 1$ and $b > 1$ are constants and $f(n)$ is asymptotically positive function.

$T(n)$ is asymptotically bounded as below -

1. If $f(n) = O(n^{\log_b a - \epsilon})$ for some constant $\epsilon > 0$ then -

$$T(n) = \Theta(n^{\log_b a})$$

2. If $f(n) = \Theta(n^{\log_b a})$ then -

$$T(n) = \Theta(n^{\log_b a} \log_2 n)$$

3. If $f(n) = \Omega(n^{\log_b a + \epsilon})$ for some $\epsilon > 0$ and if $a \cdot f(n/b) \leq c \cdot f(n)$ for some constant $c < 1$ and sufficiently large n. then -

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

1st case - $f(n)$ must be polynomially smaller than n^{ϵ}

3rd case - $f(n)$ must be ——"—— larger than n^{ϵ} and must satisfy regularity condition.

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

$a = 9$, $b = 3$ $f(n) = n$.

$n^{\log_b a} = n^{\log_3 9} = n^2$ $f(n) = n$.

$f(n) = O(n^{\log_3 9 - \epsilon})$

For $\epsilon = 1$, we can apply case 1 of M.T.

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

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

$a = 1 \quad b = 3/2 \quad f(n) = 1$

$n^{\log_b a} = n^{\log_{3/2} 1} = n^0 = 1$

Case 2 applies as -

$f(n) = \theta(n^{\log_b a}) = \theta(1)$

$\therefore \boxed{T(n) = \theta(\log_2 n)}$

(3) $T(n) = 3T(n/4) + n\log n$.

$a = 3, \quad b = 4 \quad f(n) = n\log n$

$n^{\log_b a} = n^{\log_4 3} = O(n^{0.793})$

$f(n) = \Omega(n^{\log_4 3 + \epsilon})$ where $\epsilon \approx 0.2$

Regularity condition -

$a f(n/b) = 3 f(n/4) \leq \frac{3n}{4} \log \frac{n}{4}$

$= \frac{3}{4} n [\log n - \log 4] = \frac{3}{4} n\log n - \frac{3 \cdot 2}{4}$

$\leq c \cdot f(n)$.

for $c = 3/4 < 1$

Case 3 applies $\boxed{T(n) = \theta(n\log n)}$

(4) $T(n) = 2T(n/2) + n\log n$.

Here $a=2$ $b=2$ $f(n) = n\log n$.

$n^{\log_b a} = n^1$.

We may mistakenly think that case 3 should apply since $f(n) = n\log n$ is asymptotically larger than $n^{\log_b a} = n$.

But it is not polynomially larger.

Ratio $\frac{f(n)}{n^{\log_b a}} = \frac{n\log n}{n} = \log n$ is

asymptotically less than n^{ϵ} for any +ve constant ϵ.

Thus it falls in the gap between case 2 & case 3