Recurrences ch-4. (T. H. Cormen)

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

eg. Recurrence of merge sort

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

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

Methods to solve Recurrences

1) Substitution method.
2) Recursion Tree Method.
3) Master Theorem Method.

1. **Substitution method**:- consist of two steps :-

① Guess the form of the solution.
② use Mathematical Induction to find the constants & show that the solution works. (Merge Sort using Substitution Method)

eg. Determine the upperbound on the recurrence :-

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

Sol^n: we guess that the sol^n is $T(n) = O(n \log n)$

$$\therefore T(n) \leq c(n \log n). \quad (3)$$

start assuming, $T\left(\frac{n}{2}\right) \leq c\left(\frac{n}{2} \log \frac{n}{2}\right) - (2)$

Put (2) in (1), we get,

$$T(n) \leq 2c\left(\frac{n}{2} \log \frac{n}{2}\right) + n$$
$$\leq cn \log \frac{n}{2} + n$$
$$\leq cn \log n - cn \log 2 + n$$
$$\leq cn \log n - cn + n \leq cn \log n = O(n \log n)$$

(discarding constants)

2nd step. Mathematical Induction.

$T(n) \leq cn\log n$.

$n=1$,

$T(1) \leq c.1.\log 1. = 0$. ($1<0$, fails because $T(1)=1$).

$n=2$,

$T(2) \leq c.2.\log 2$ ($2=2$, True.)

$n=3$,

$T(3) \leq c.3\log 3$. (True).

Here, we take advantage of asymptotic Notation only requiring us to prove $T(n) \leq cn\log n$ for $n \geq n_0$.

$\therefore n_0 = 2$.

& consider $n=2$ & $n=3$ as the base cases.

$\therefore$ $T(n) \leq cn\log n$ $\forall$ $n \geq 2$.

---

using master Th. [$f(\frac{n}{4}) = (\frac{n}{4})^2$]

$T(n) = 3T(\frac{n}{4}) + \Theta(n^2)$, $f(n) = n^2$

$n^{\log_4 3} = n^{0.\ldots+\varepsilon} = n^2$ [Case III]

a $f(\frac{n}{b}) \leq c.f(n)$, $c<1$

$3(\frac{n}{4})^2 \leq c.n^2$

$(\frac{3}{4^2})n^2 \leq cn^2$ $\therefore c = \frac{3}{16} < 1$.

Thus ; $\boxed{f(n) = \Theta(n^2)}$

Recursion - Tree Method

In this method, each node represents the cost of a single subproblem somewhere 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 all the per-level costs to determine the total cost of all levels of the recursion. Recursion trees are useful when the recurrence describes the running time of a divide and conquer algorithm.

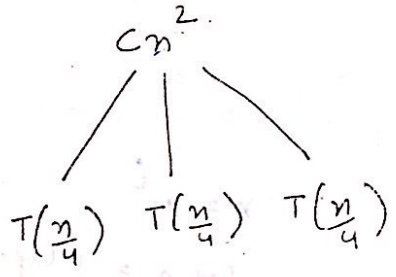
eg. $T(n) = 3T\left(\lfloor \frac{n}{4} \rfloor\right) + \Theta(n^2)$.

Sol^n. ⇐ we know that floors and ceilings are usually insubstantial in solving recurrences, we create a recursion tree for the recurrence $T(n) = 3T\left(\frac{n}{4}\right) + cn^2$ for $c > 0$.

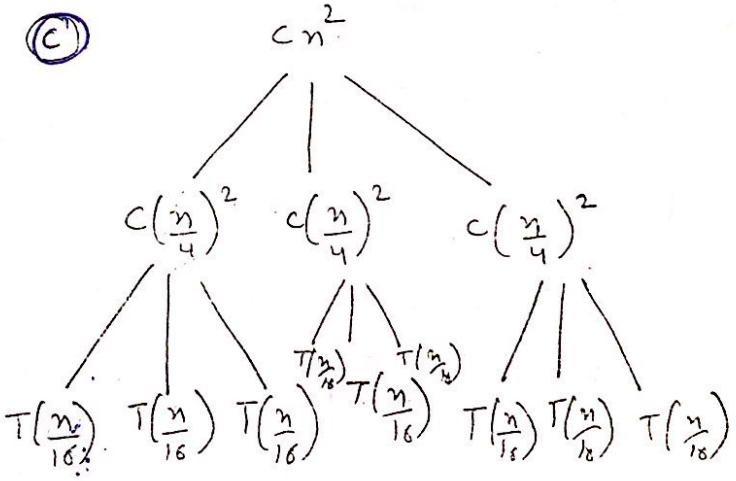
Also, we assume that 'n' is an exact power 4.

(a) $T(n)$

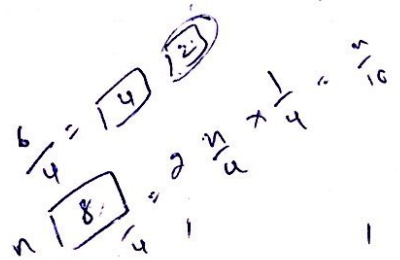
(b)



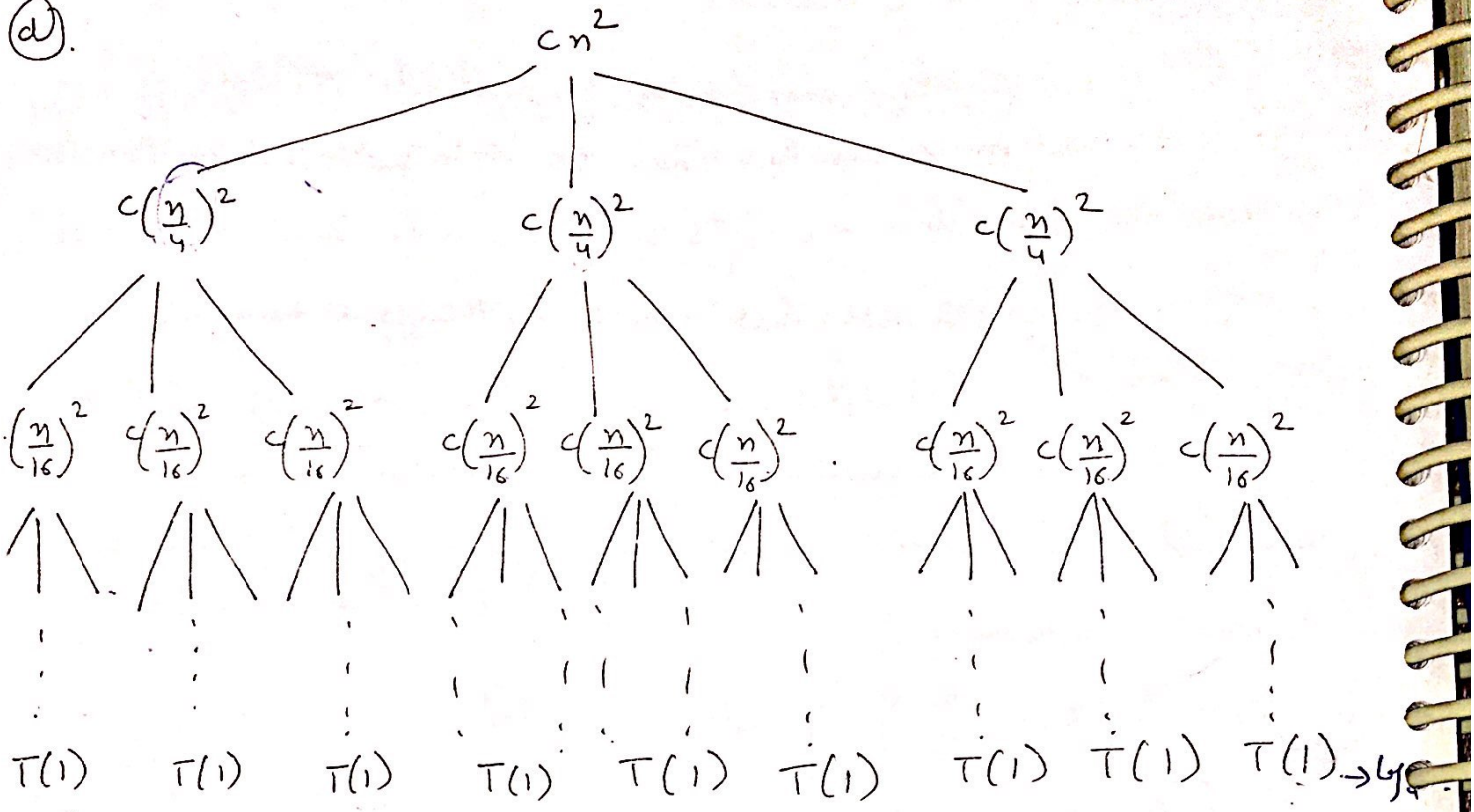
(c)



(d)



(d).



∓ Because subproblem sizes decrease as we get further from the root, we must reach a boundary condition.

∓ The subproblem size of a node at depth i is $n/4^i$.

depth 1 $\longrightarrow \frac{n}{4}$,

depth 2 $\longrightarrow \frac{n}{4^2} = \frac{n}{16}$ and so on.

depth $\to 0$ $\quad \frac{n}{4^0} = n$

∓ Thus, at the last level when size of n is 1 ie $n = 1$, we get, $\frac{n}{4^i} = 1$ or $i = \log_4 n$. $\quad n = 4^i$

$\log n = i \log 4$

∓ Thus the tree has $\log_4 n + 1$ levels $(0, 1, 2, \ldots \log_4 n)$.

Next, we determine the cost at each level of the tree. Each level has three times more nodes than the level above & so the no. of nodes at depth i is 3^i.

depth 0 $\longrightarrow$ 3^0 = 1 node

depth 1 $\longrightarrow$ 3^1 = 3 nodes

depth 2 $\longrightarrow$ 3^2 = 9 nodes & so on.

Because subproblem sizes reduce by a factor of 4 for each level we go down from the root, each node at depth i, for $i = 0, 1, 2, \ldots \log_4 n - 1$ (second last level) has a cost of $c\left(\frac{n}{4^i}\right)^2$.

depth 1 $\longrightarrow$ $c\left(\frac{n}{4^1}\right)^2$

depth 2 $\longrightarrow$ $c\left(\frac{n}{4^2}\right)^2$ & so on.

Multiplying, we see that the total cost over all nodes at depth i, for $i = 0, 1, 2, \ldots \log_4 n - 1$ is,

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

At the last level, the depth $\log_4 n$ has $3^{\log_4 n}$ nodes or $n^{\log_4 3}$ nodes, each contributing cost $T(1)$, for total cost of $n^{\log_4 3} \cdot T(1) = \Theta\left(n^{\log_4 3}\right)$.

$[T(1) = 1]$

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

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

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

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

$$\leq \frac{1}{1-\left(\frac{3}{16}\right)} cn^2 + \Theta\left(n^{\log_4 3}\right)$$

$$\left[\sum_{k=0}^{\infty} x^k = \frac{1}{1-x} \quad [(A.6)] \right]$$

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

$$= O(n^2)$$

(III). Master Theorem Method

Master Theorem solves recurrences of the form,

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

where $a \geq 1$ & $b > 1$ and $f(n)$ is an asymtotically Positive function.

$T(n)$ can be bounded as follows :-

1) If $f(n) = O\left(n^{\log_b a - \varepsilon}\right)$ for some constant $\varepsilon > 0$, then $T(n) = \Theta\left(n^{\log_b a}\right)$.

2) If $f(n) = \Theta\left(n^{\log_b a}\right)$, then,

$$T(n) = \Theta\left(n^{\log_b a} \cdot \log n\right) \text{ where } \varepsilon = 0.$$

3) If $f(n) = \Omega\left(n^{\log_b a + \varepsilon}\right)$ for some constant $\varepsilon > 0$ & if $a f\left(\frac{n}{b}\right) \leq c \cdot f(n)$ for some constant $c < 1$ & sufficiently large n, then

$$T(n) = \Theta\left(f(n)\right).$$

eg.

① $T(n) = 9T\left(\frac{n}{3}\right) + n.$

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

Find $n^{\log_b a} = n^{\log_3 9} = n^2$

but $f(n) = n.$

$\therefore f(n) = O(n^{\log_b a - \varepsilon})$ where $\varepsilon = 1.$

$= O(n^{2-1}) \Rightarrow T(n) = \Theta(n^2)$ [case I].

② $T(n) = T\left(\frac{2n}{3}\right) + 1$

Here $a=1$, $b=\frac{3}{2}$, $f(n) = 1.$

Find $n^{\log_b a} = n^{\log_{3/2} 1} = n^0 = 1 = f(n).$ [case II].

$\therefore \varepsilon = 0,$

$\therefore T(n) = \Theta(1 . \log n) = \Theta(\log n).$

3) $T(n) = 3T\left(\frac{n}{4}\right) + n \log n$

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

Find $n^{\log_b a} = n^{\log_4 3} = n^{0.793 \text{ (appx)}}$.

$f(n) = n \log n$

since $f(n) = \Omega\left(n^{\log_4 3 + \varepsilon}\right)$ where $\varepsilon = 0.2$ appx.

And

To check if $a f\left(\frac{n}{b}\right) \leq c f(n)$ for some constant $c < 1$ & sufficiently large n,

$\Rightarrow 3\left(\frac{n}{4}\right) \log\left(\frac{n}{4}\right) \leq \left(\frac{3}{4}\right) n \log n$ (Take $c = \frac{3}{4} < 1$)

$\log\left(\frac{n}{4}\right) \leq \log n \quad \therefore \Theta(n \log n)$

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

$a = 2,\ b = 2,\ f(n) = n \log n$

Find $n^{\log_b a} = n^{\log_2 2} = n^1 \neq n \log n$

$f(n)$ ie $n \log n$ is asymptotically larger than n but it is not Polynomially larger than n.

Hence master theorem cannot be applied to the given problem.

c) $T(n) = 3T\left(\frac{n}{5}\right) + n$

$a = 3,\ b = 5,\ f(n) = n$

Find $n^{\log_b a} = n^{\log_5 3} = n^{0.6 \text{ (approx)}} \neq n.$

$n^{\log_b a + \varepsilon}$ where $\varepsilon = 0.4$ (approx).

AND.

$a f\left(\frac{n}{b}\right) \leq c f(n)$

$3\left(\frac{n}{5}\right) \leq c.n$, Take $c = \frac{3}{5} < 1$, we get,

$\frac{3n}{5} \leq \frac{3}{5}n$

$1 \leq 1$ (True). $\therefore T(n) = \Theta(n).$

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

$a = 2,\ b = 2,\ f(n) = n$

find $n^{\log_b a} = n^{\log_2 2} = n^1 = f(n).$

$\therefore \varepsilon = 0.$

$\therefore T(n) = \Theta(n \log n)$ [case II].

(7) $T(n) = 2T\left(\frac{n}{2}\right) + 1$, $a=2$, $b=2$, $f(n)=1$.

Find $n^{\log_b a} = n^{\log_2 2} = n^1 \neq f(n)$

$\therefore n^{\log_b a - 1}$ applies where $\varepsilon = 1$ [case I]

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

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

$a=1$, $b=2$, $f(n)=n$

Find $n^{\log_b a} = n^{\log_2 1} = n^0 = 1 \neq f(n)$

$\therefore n^{\log_b a + \varepsilon} = f(n)$ where $\varepsilon = 1$ [case III]

And $a.f\left(\frac{n}{b}\right) \leq c\, f(n)$

$1.\frac{n}{2} \leq c.n$

Take $c = \frac{1}{2}$ $(c < 1)$, we get,

$\frac{n}{2} \leq \frac{n}{2}$

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

(9) $T(n) = T\left(\frac{n}{2}\right) + 1$

$a = 1,\ b = 2,\ f(n) = 1.$

find $n^{\log_b a} = n^{\log_2 1} = n^0 = 1 = f(n)$ [case 2]

$\therefore T(n) = \Theta(1 . \log n) = \Theta(\log n).$

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

$a = 2,\ b = 2,\ f(n) = n / \log n$

Find $n^{\log_b a} = n^{\log_2 2} = n^1 \neq n / \log n$

$f(n)$ is not Polynomially larger than $n^{\log_b a}$ ie n.

Thus master Theorem cannot be applied.

NOV, 12

(11) $T(n) = 4T\left(\frac{n}{2}\right) + n^2 \log_2 n$

$a = 4,\ b = 2,\ f(n) = n^2 \log n$

$n^{\log_b a} = n^{\log_2 4} = n^2 \neq n^2 \log n$

not applicable.

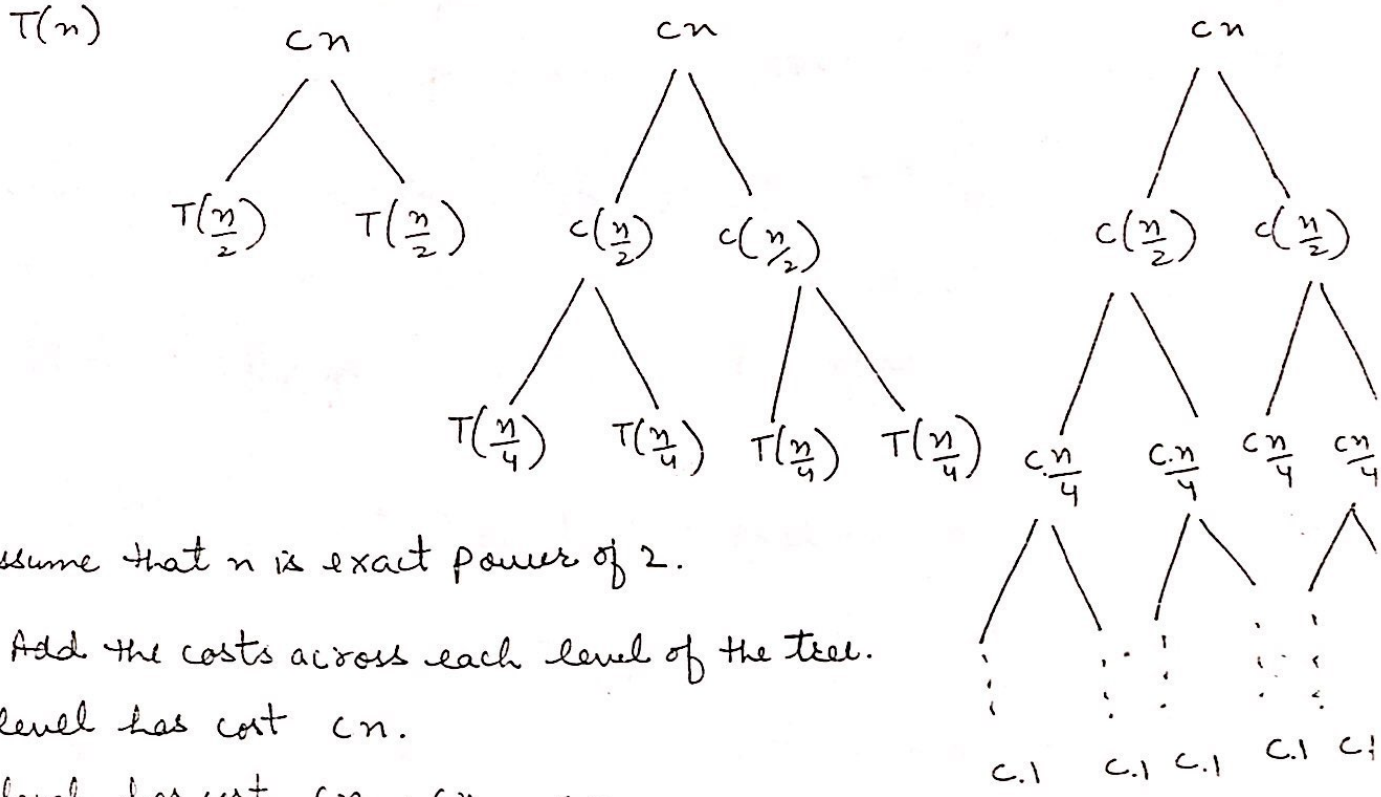
(12) $T(n) = 4T\left(\frac{n}{2}\right) + n$

$n^{\log_b a} = n^{2-\varepsilon} = f(n)$

$T(n) = \Theta\left(n^{\log_b a}\right) = \Theta\left(n^2\right)$

Recursion Tree Method. [Merge sort using Recursion Tree]

$T(n) = 2T(\frac{n}{2}) + cn.$



+ Assume that n is exact power of 2.

ow, Add the costs across each level of the tree.

p level has cost cn.

ext level has cost $\frac{cn}{2} + \frac{cn}{2} = cn$.

ext level has cost $\frac{cn}{4} + \frac{cn}{4} + \frac{cn}{4} + \frac{cn}{4} = cn$. & so on.

OR

ow, each level has 2^i Nodes. ie level 0 $= 2^0 = 1$

level 1 $= 2^1 = 2$

level 2 $= 2^2 = 4$. & so on.

ost of each node is $c(\frac{n}{2^i})$ ie level 0 $= cn$

level 1 $= c\frac{n}{2}$

level 2 $= \frac{cn}{4}$ & so on.

$\therefore$ Total cost at each level $= 2^i . c(\frac{n}{2^i}) = cn.$

At the last level there are n nodes each contributing a cost of c $\therefore$ Total cost $= cn$.

last level is $\log n$.

size of each node at each level is $\frac{n}{2^i}$.

At the last level, when size of n is 1

$$\frac{n}{2^i} = 1 \quad \text{or} \quad i = \log_2 n.$$

∴ Total no. of levels are :-

$$\log_2 n + 1 \quad [0, 1, 2, \ldots \log_2 n]$$

∴ Total cost by adding cost at each level is,

$$cn(\log_2 n + 1) = cn\log n + cn.$$

Ignoring low-order term & the constant c gives the desired result of $O(n\log n)$

Iterative method (Merge sort)

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

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

Div. both sides by n,

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

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

$$\frac{T(n/4)}{n/4} = \frac{T(n/8)}{n/8} + c$$

$$\frac{T(2)}{2} = \frac{T(1)}{1} + c$$

Backtrack,

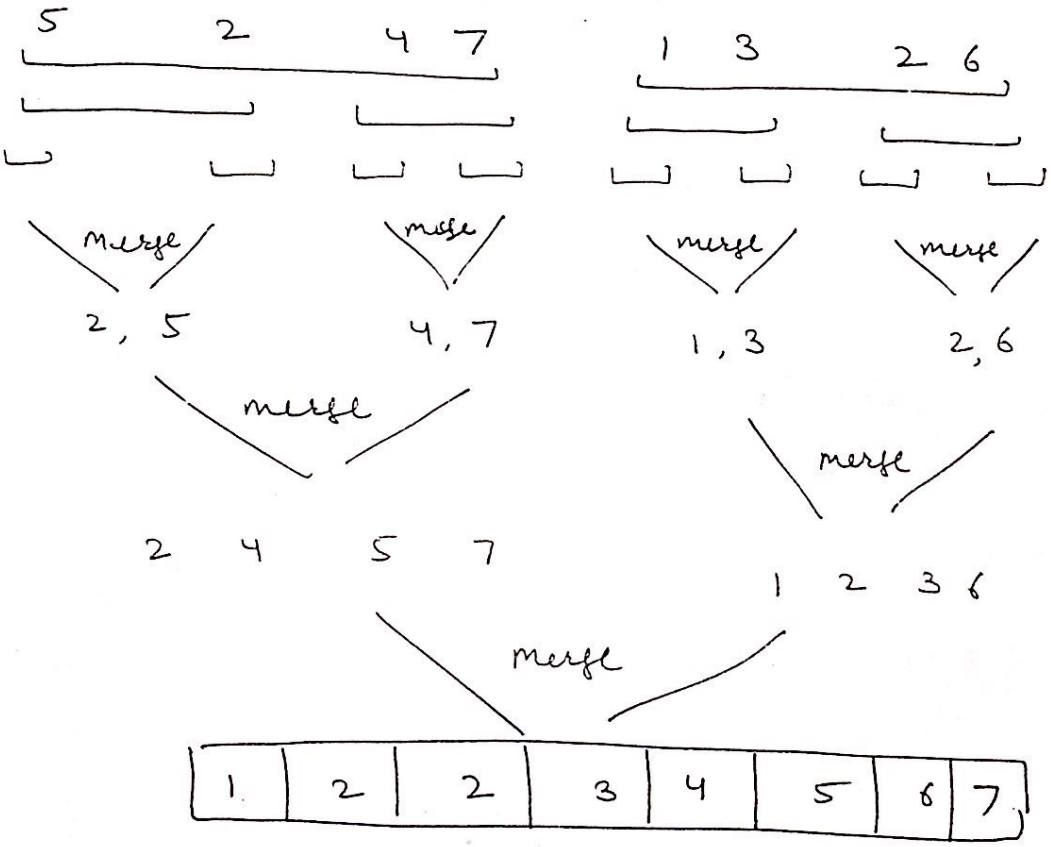
$$\frac{T(n)}{n} = 1 + c.1 \quad [T(1) = 1]$$

$$T(n)/n = 1 + c\log n \quad [\log 2 = 1]$$

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

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

Merge - sort Eg.



Loop Invariants

We must show three things about a loop invariant :-

1) Initialization :- It is true prior to the first iteration of the loop.
2) Maintenance :- If it is true before an iteration of the loop, it remains true before the next iteration.
3) Termination :- When the loop terminates, the invariant gives us a useful property that helps show that the algorithm is correct.

Eg. Loop invariant holds for Insertion Sort.

```
for(i=2; i<=n; i++)
{
    v = a[i];
    j = i-1;
    while ((j>=1) && v <= a[j])
    {
        a[j+1] = a[j];
        j = j-1;
    }
    a[j+1] = v;
}
```

Initialization :- show that loop invariant holds before the first loop iteration when $i = 2$. The subarray $A[1..i-1]$ consist of just a single element $A[1]$ & this subarray is sorted. Therefore, loop invariant holds for prior to the first iteration of the loop.

Maintenance :- second property is, each iteration maintains loop invariant. ie the array is partially sorted after each iteration.

3) Termination :- The outer for loop ends when i exceeds n, ie when i = n+1. array A[1..n] is sorted & terminated because i > n, ∴ the algorithm is correct.

Divide And Conquer Approach.

Break the problem into several subproblems that are similar to the original problem but smaller in size, then solve the subproblems recursively, and then combine these solutions to create a solution to the original problem.

Divide the problem into a no. of subproblems.

Conquer the subproblems by solving them recursively. If the subproblem size is small enough, then solve it in straightforward manner.

Combine the solutions to the subproblems into the solution for the original problem.

Eg. Merge Sort algorithm follows divide & conquer paradigm.

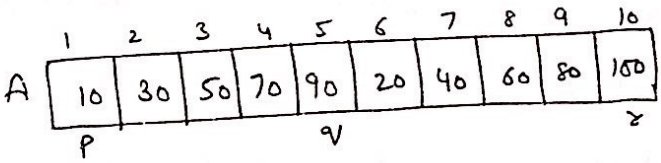
Divide :- Divide n-element sequence to be sorted into two subsequences of $n/2$ elements each.

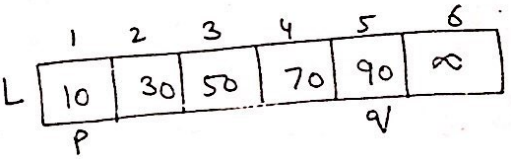
Conquer :- Sort the two subsequences recursively using merge sort.

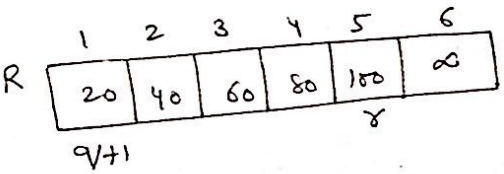
Combine :- Merge the two sorted subsequences to produce the sorted answer.

Merge (A, p, q, r)

1. $n_1 \leftarrow q - p + 1$
2. $n_2 \leftarrow r - q$
3. create arrays $L[1..n_1+1]$ and $R[1..n_2+1]$
4. for $i \leftarrow 1$ to n_1
5. do $L[i] \leftarrow A[p+i-1]$
6. for $j \leftarrow 1$ to n_2
7. do $R[j] \leftarrow A[q+j]$
8. $L[n_1+1] \leftarrow \infty$
9. $R[n_2+1] \leftarrow \infty$
10. $i \leftarrow 1$
11. $j \leftarrow 1$
12. for $k \leftarrow p$ to r
13. do if $L[i] \leq R[j]$
14. then $A[k] \leftarrow L[i]$
15. $i \leftarrow i+1$
16. else $A[k] \leftarrow R[j]$
17. $j \leftarrow j+1$







$n_1 = 5 - 1 + 1 = 5$

$n_2 = 10 - 5 = 5$

± The merge-procedure takes time $\Theta(n)$, where $n = r - p + 1$ is the no. of elements being merged. (for loop runs for n no. of times)

Loop Invariant

> Initialization :- Before the first iteration, we have $k = p$.

∴ The subarray becomes $A[p \ldots k-1]$ which is empty. This empty subarray contains 0 smallest elements of L & R & since $L[i]$ & $R[j]$ are smallest elements of

their respective arrays that have not been copied back into A.

Maintenance :- Suppose $L[i] \leq R[j]$, Then $L[i]$ is the smallest element not copied back to A. Because $A[p..k-1]$ contains the $k-p$ smallest elements, the subarray $A[p..k]$ will contain $k-p+1$ smallest elements. Incrementing k and i reestablishes the loop invariant for the next instruction.

Termination :- At termination, $k = r+1$. By the loop invariant, the subarray $A[p..k-1]$, which is $A[p..r]$, contains the $k-p = r-p+1$ smallest elements of $L[1..n_1+1]$ and $R[1..n_2+1]$, in sorted order. The arrays L & R together contain $n_1+n_2+2 = r-p+3$ elements. All but the two largest have been copied back into A, & these two largest are the sentinels(∞).

Merge Sort (A, p, r)

. If $p < r$

:. then $q \leftarrow L(p+r)/2 \rfloor$.

3. Merge-sort (A, p, q)

4. merge-sort (A, q+1, r)

5. merge (A, P, q, r).