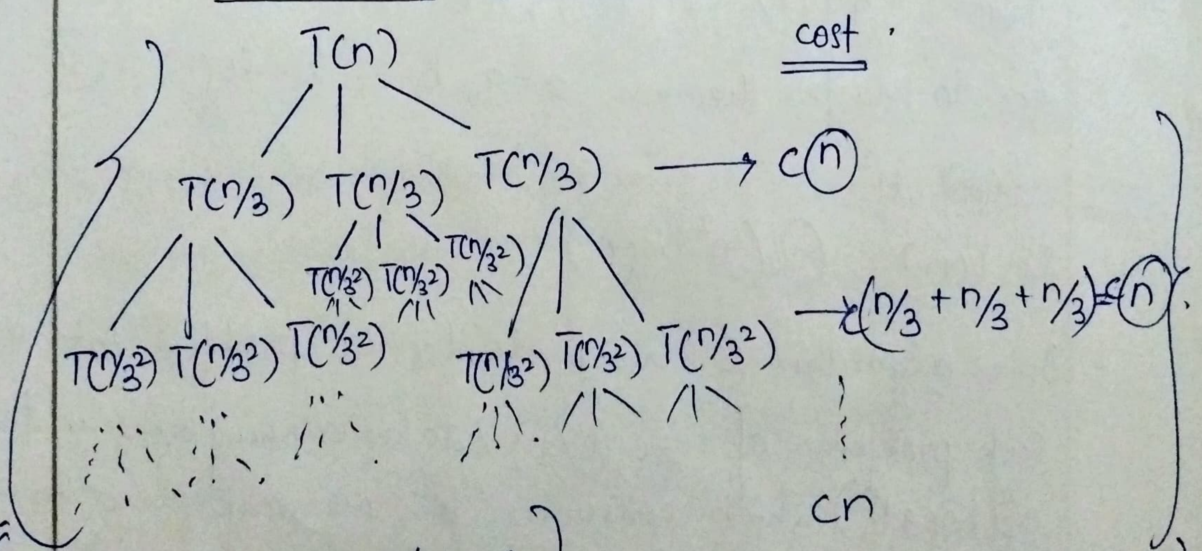
(35) function (int n)
{ if (n==1) → (1)
    return 1; → (1)
  else
    function (n/3); function (n/3); function (n/3);
                                              ↳ $3T(n/3)$
    for (i=1; i<=n; i++) → (n+1)
        x = x+1; → (n)
}

Find time & space complexity of the given algorithm.

ans:- <u>Time complexity</u> :- $T(n) = 1+1+3T(n/3)+n+1+n$

$$= 3T(n/3) + O(n)$$

Now, <u>recursive trace</u> <u>for the function</u> :-



will run, upto, $n/3^k = 1$

$\Rightarrow 3^k = n$

$\Rightarrow k = \log_3 n$ (log₃n times)

↓

height of the tree.

So, Time complexity

$= cn * \log_3 n$

$= \Theta(n \log_3 n)$.

Space complexity :- As the $T(n)$ is in recursive form

→ So, the space complexity is the maximum height of the tree (as that many (max) times the fun^c is stored in the stack)

→ As the tree is in the same level, so, the height of the tree $= \log_3 n$

Also, there some constant variables in the algorithm, not any array of variables space required.

So, $S(n) = \log_3 n + O(1)$

Space complexity $= O(\log_3 n)$

(36) Void function(int n) {

```
    Temp = 1;
    Repeat
        For (i = 1 to n)
            temp = temp + 1
            n = n/2;
    until n <= 1
}
```

Find the time complexity & space complexity of the given algorithm.

ans:- We can write the algo properly as:-

void function (int n) { → 1

temp = 1;

for (int i = 1 ; i ≤ n ~~n≤1~~ ; n = n/2) → $\log_2 n$

temp = temp+1; → $\log_2 n - 1$

} So, $T(n) = \cancel{1} + \log_2 n + \log_2 n - \cancel{1} = 2\log_2 n = O(\log_2 n)$

Time complexity : $O(\log_2 n)$

Space complexity : $O(1)$ → As only constant no. of variables required, & no variables requires an array amt. of space.

(37) int function (int n) {

```
if (n<=2)                    → 1
        return 1;            → 1
else                         → 1
    return (function (floor (sqrt(n))) +1);  → T(√n) +1
}
```

Find the time & space complexity of the given algorithm

ans:- $T(n) = \begin{cases} 1, & n \leq 2 \\ T(\sqrt{n})+1, & n > 2 \end{cases}$

Recursive trace for Time complexity :-

T(n)   cost :

$T(n)$
↓
$T(\sqrt{n}) \longrightarrow c$
↓
$T(n^{1/2^2}) \longrightarrow c$
↓
$T(n^{1/2^3}) \longrightarrow c$
⋮
$T(n^{1/2^k}) \longrightarrow c$

k times

$n^{1/2^k} = 2$

$\Rightarrow \frac{1}{2^k} \log_2 n = 1$

$\Rightarrow 2^k = \log_2 n$

$\Rightarrow \boxed{k = \log_2 \log_2 n}$

So, Time complexity $= c * k$

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

Now, Space complexity $= S(n) =$ height of the tree
$= O(\log \log n)$.

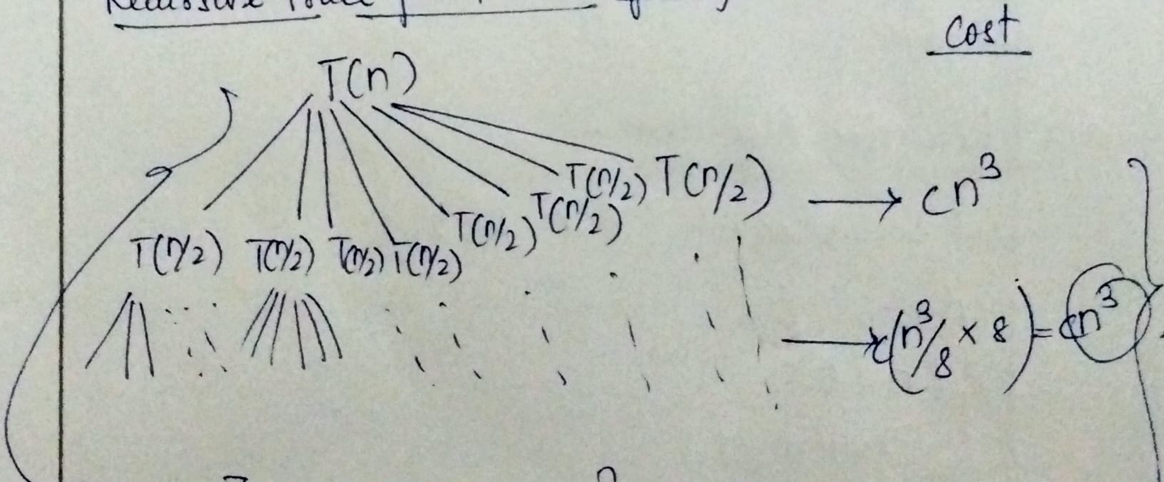
(38) void function (int n) {

```
    if (n==1)
        return 1;
    else
        for (i=1; i<=8; i++)
            function(n/2);
    for (i=1; i<=n^3; i++)
        count = count + 1;
}
```

Find the time & space complexity of the given algorithm.

ans:- $T(n) = 1 + 1 + 8T(n/2) + O(n^3)$

Recursive trace for time complexity :-



$T(n/2^k) = 1$ (Termination point)
$\Rightarrow k = \log_2 n$.

So, Time Complexity
$= c(n^3) \times \log_2 n$
$= O(n^3 \log_2 n)$.

Space Complexity : $S(n) \rightarrow$ max height of the recursive tree

So, $S(n) = \log_2 n + O(1)$

$\hookrightarrow$ (Constant no. of variables)

So, Space complexity $(S(n)) = \underline{O(\log n)}$.

(39) The following pseudocode performs linear search on an array of size n to find the presence of an element el.

```
Linear Search (A, n, el)
for i = 1 to n do
    If A[i] = el then
  return i.
  return NIL
```

Write the recursive version of the Linear Search Algorithm & compare the time & space complexity with the iterative version.

ans:- Recursive Algorithm :-

```
int LinearSearch (A, n, el) {
  n--;
  if (n < 0)
     return -1;
  if (A[n] == el)
     return n;
  return LinearSearch (A, n, el); }
```

Time complexity :- $T(n) \to n+1+n+1+1$

$= \underline{O(n)}$

As the in linear search iterative algorithm in the worst case time it will traverse through the whole array

Space complexity :- $S(n) = O(1)$

→ fixed/constant variables

→ No extra array space is required, hence $O(1)$.

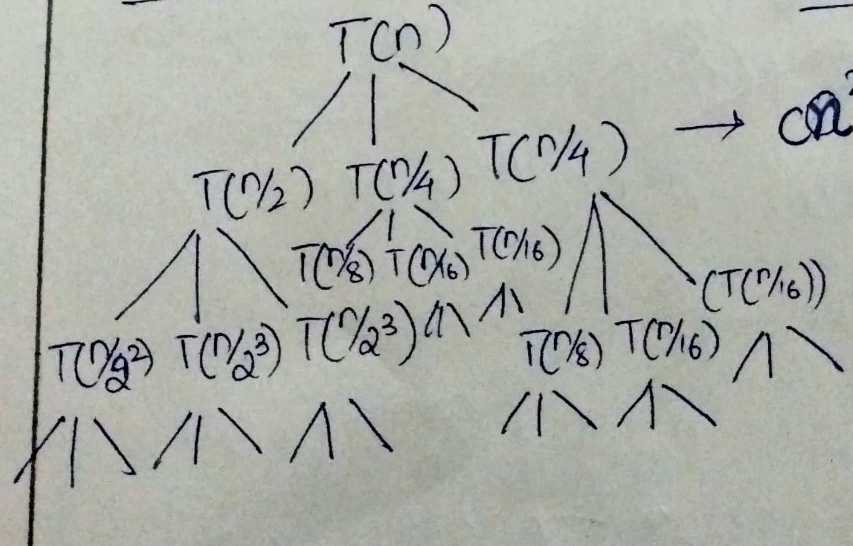
(40) Solve the following recurrence using any of the suitable methods. If no solution is possible, justify using proper reasoning.

(a) $$T(n) = \begin{cases} 1 & \text{if } n=2 \\ 1 & \text{if } n=4 \\ T(n/2) + 2T(n/4) + \Theta(n^2) & \text{if } n>4 \end{cases}$$

where n is assumed to be a power of 2.

ans:- Recursive Trace for the time complexity :-

cost



$\to cn^2$

$\to (n/2)^2 + (n/4)^2 + (n^2/16)$

$= \frac{4n^2+n^2+n^2}{16}$

$= \frac{3}{8}n^2$

$\to n^2/16 + n^2/64 + n^2/64 + n^2/64 + n^2/256 +$

$+ n^2/256 + n^2/64 + n^2/256 + n^2/256$

$$= \frac{16+4+4+4+1+1+4+1+1}{256} n^2 = \frac{\cancel{36}^{\,9}}{\cancel{256}_{\,64}} n^2 = \frac{9}{64} n^2$$

$$= \underline{\left(\tfrac{3}{8}\right)^2 n^2}.$$

So, $T(n) = n^2 + \frac{3}{8}n^2 + \left(\frac{3}{8}\right)^2 n^2 + \cdots$ h times (where h is the height of the tree).

$\Rightarrow T(n) \leq n^2\left(1 + \frac{3}{8} + \left(\frac{3}{8}\right)^2 + \cdots \infty \text{ times}\right)$

$\Rightarrow T(n) \leq n^2 \left(\frac{1}{1-3/8}\right)$

$\Rightarrow \boxed{T(n) = O(n^2)}$ → answer.

(b) $T(n) = n^{1/3} T\left(n^{2/3}\right) + \theta(n)$.

ans:- Recursive trace ~~for Time complexity~~ :- cost

$T(n)$

$\downarrow$

$n^{1/3}\left(T\left(n^{2/3}\right)\right) \rightarrow n$

$\downarrow$

$n^{2/9}\, T\left(n^{(2/3)^2}\right)) \rightarrow n^{2/3}$

We have:- $T(n) = n^{1/3} T\left(n^{2/3}\right) + \theta(n)$

$= n^{1/3}\, T\left(1/n^{-2/3}\right) + \theta(n)$

$= n^{1/3}\, T\left(n/n^{-2/3+1}\right) + \theta(n)$

Using Master's theorem:

$= n^{1/3}\, T\left(n/n^{1/3}\right) + \theta(n)$.

$a = n^{1/3}, \quad b = n^{1/3}$ $\quad f(n) = \theta(n)$.

So, acc. to Master's theorem,

$$n^{\log_b a} = n^{\log_{n^{1/3}} n^{1/3}} = n = \Theta f(n)$$

So, acc to <u>case-II</u> :- $T(n) = \Theta(n^{\log_{n^{1/3}} n^{1/3}} \cdot \log n)$

$$= \Theta(n \cdot \log n) \rightarrow \text{(ans)}$$

(c) $T(n) = \sqrt{n}\, T(\sqrt{n}) + \log n$

ans:- Using recursive trace, for time complexity :-

$$\begin{array}{ccc} T(n) & & \underline{\text{cost}} \\ \downarrow & & \\ (n)^{1/2}\, T(n^{1/2}) & \longrightarrow & \log n \\ \downarrow & & \\ (n)^{(1/2)^2}\, T(n^{1/2})^{1/2} & \longrightarrow & \frac{1}{2}\log n \\ \downarrow & & \\ (n)^{(1/2)^3}\, T(n^{(1/2)^3}) & \longrightarrow & (\frac{1}{2})^2 \log n \\ \vdots & & \\ (n)^{(1/2)^k}\, T(n^{(1/2)^k}) & \longrightarrow & (\frac{1}{2})^{k-1} \log n \end{array}$$

So, $T(n) = \log n \left(1 + \frac{1}{2} + (\frac{1}{2})^2 + (\frac{1}{2})^3 \cdots (\frac{1}{2})^{k-1}_{\infty}\right)$

$$\leq \log n \left(1 + \frac{1}{2} + \cdots \infty\right)$$

$$\Rightarrow T(n) = O(\log n) \rightarrow \text{(ans)}$$