Loop time complexity:

category - 1

① for (i = 1 ; i ≤ n ; i++)
pf ("xxx") ⟶ n times the printing taked place.

Time complexity O(n)

② for (i = n ; i ≥ 1 ; i--)

③ for (i = 1 ; i ≤ n ; i = i + 5)

i	1	6	11
	xx	xx	fail

pf ("xxx") ——— n/5 times the printing taked place.

TC = O(n/5) = O(n)

④ for (i = n ; i ≥ 1 ; i = i - 5) ——— n/5 times.

TC = O(n/5) = O(n).

⑤ for (i = 1 ; i ≤ n ; i = i + c)
or
for (i = n ; i ≥ 1 ; i = i - c) } n/c times.

TC = O(n/c) = O(n)

category 2:

① for (i = 1 ; i ≤ n ; i = i * 2)
pf ("---");

last i value.

i = 1, 2, 2^2, 2^3, ---------, 2^k, 2^{k+1}

$2^{k+1} > i$

So the loop runs for (k+1) times.

Assume $2^{k+1} > i \Rightarrow 2^k = n$

$\Rightarrow \log 2^k = \log_2 n$

$\Rightarrow k \log_2 2 = \log n$

$\Rightarrow k = \log n.$

estimated time = TC = O(log n)

② for (i = n ; i >= 1 ; i = i/2)
    Pf ("x x x x")

i = n, n/2, n/2/2, ... n/2^k

n/2^2

n/2^k is the last time where the condition is

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

$\Rightarrow n = 2^k$

$\Rightarrow k = \log_2 n$

So the loop runs for (k+1) times. So Printf executed for (k+1) times. or $(\log_2 n + 1)$ times.

$TC = O(\log_2 n)$

③ for (i = 1 ; i <= n ; i = i * 5)
    Pf (x x x x)

$i = 1, 5, 5^2, 5^3 \ldots\ldots 5^k$

$5^k \leq n \Rightarrow 5^k = n$

$\Rightarrow k = \log_5 n + 1 = O(\log n)$

for (i = 1 ; i <= n ; i = i/5)

or i = i * c or i = i/c then $O(\log_c n)$

Category 3:

① for (i = 2 ; i <= n ; i = i * i)

$i = 2, 2^2, 2^4, 2^8 \ldots\ldots 2^{2^k}$

$2^{2^0} + 2^{2^1} + 2^{2^2} + 2^{2^3} + 2^{2^4} + \ldots 2^{2^k}$

$2^{2^k} \leq n \Rightarrow 2^{2^k} = n$

$\Rightarrow 2^k \log_2 2 = \log n$

$\Rightarrow 2^k = \log n \Rightarrow \log 2^k = \log(\log n)$

$\Rightarrow k \log_2 2 = \log(\log n)$

$\Rightarrow k = \log(\log n)$

① for (i=n ; i>=2 ; i=√i)

printf ("xxxx")

i=	n	$n^{1/2}$	$n^{1/4}$	$n^{1/8}$	.....	$n^{1/2^k}$

$n^{1/2^k} \geq 2 \Rightarrow \frac{1}{2^k} = \log 2^2 = 1$

$\Rightarrow \log 2^n = 2^k \Rightarrow k = \log(\log n)$

② for (i=5 ; i<=n ; i=i^5)

$5\ (5^2)^1\ (5^5)^1\ (5^5)^3, \ldots (5^5)^k$

$k = \log(\log_5 n)$

③ for (i=n ; i>=5 ; i=$\sqrt[5]{i}$)

$n, n^{1/5}, n^{1/5^2}, n^{1/5^3}, \ldots n^{1/5^k}$

$k = (\log(\log_5 n))$

Different category:

① for (i=1 ; i<=2^n ; i=i*2)

Pf (^xxx)

i=	1	2	2^2	2^3	....	2^k

$2^k = 2^n$

$\Rightarrow k = n \Rightarrow O(n)$

② for (i=1 ; i<=n^2 ; i=i+10)

$i = 1, 1+10, (1+2\times10), \ldots 1+(k*10)$

$(1+10)+10$

$1 + k*10 \leq n^2$

$\Rightarrow k*10 = n^2 - 1$

$\Rightarrow k = \frac{n^2-1}{10} \simeq O(n^2)$

(10) for ($i = n/2$, $i \le n$; $i = i * 2$)

$n/2$, $n/2 \times 2$

($n/2$, n) It runs for 2 times.

$\Rightarrow$ TC = O(constant) = O (2).

1, 1×2, 1, 2, 2^2, ...

Solve the following.

(1)

$\log n^2$ { for ($i = 1$; $i \le n^2$; $i = i \times 2$)
  2 times { for ($j = n$; $j \ge n/2$; $j = j/2$)
    $\log n$ { for ($k = 1$; $k \le 2^n$; $k = k^2$)
      pf (x x x x x).

$= \log n^2 \times 2 \times \log n$

$= O(\log n)^2$

(2)

$\log n^2$ { for ($i = n^2$, $i \ge 1$; $i = i/2$)
  $\log n$ { for ($j = 1$; $j \le n$; $j = j * 10$)
    { for ($k = 1$; $k \le 2^n$, $k = 2k$)
      n times.

$\log n^2 \times \log n \times n$.

(3) Prove that

$(64)^{\log_2 n} = n^{\log_2 64}$

$= n^{\log_2 2^6}$

$= n^{6 \log_2 2}$

$= n^6$

Nested Loops → Independent nested Loop.
→ Dependent nested Loops.

Independent

```
for (i=1; i<=n; i++)
    for (j=1; j<=n; j++)
        for (k=1; k<=n; k++)
            Printf (" ");
```

Independent of each other.

n n

TC = no. of times outer loop
× no. of times 2nd loop
× " " 3rd
= n × n × n
= n^3.

Dependent

```
for (i=1; i<=n; i++)
    for (j=i; j<=n; j++)
        for (k=1; k<=j; k++)
```

Expand the Loops

i =	1	2	3
j	1 to n	2 to n	3 to n
	1 2		
k	1 to n	2 to n	

i)
```
for (i=1; i<=n; i++)
    for (j=1; j<=n; j++)
        for (k=1; k<=n; k++)
```
1 1 n

$n \times 1 \times 1 = O(n)$.

i = 1
i = 1 to n
i = 1 to n

②
```
for (i=0; i<=n; i++)
    for (j=1; j<=n; j++)
        for (k=1; k<=n; k=k*3)
            Pf( ... )
```
1 1 $\log_2 n$

$TC = O(\log n)$

~~3~~ (4)

```
main()
{
  flag = 0
  i = n
  while (i >= 1).
    {
      if (flag == 0)
        {
          i = i - 1.
          flag = 1
        }
      else
          i = i/2
          flag = 0
    }
}
```

i = i - 1 ; i = i^2

i = i/2 ; i = i^2

n, n/2, n/2^2, n/2^3 ... n/2^k

n/2^k = 1

(3)

```
while ( )
{
  i = 1
  s = 1
  while (i <= n)
    {
      i = i + 1
      s = s + i
      Pf ("xxx")
    }
}
```

i	1	2	3	
s	1	1+2	1+2+3	...

1+2+3+....+k ≤ n

k(k+1)/2 ≤ n

k^2 ≤ n

k = √n

TC = O(√n)

i.m.p.
Whichever decrement / increment speedily the time complexity depends on that)

(6)

for (i=n; i>=1; i = i/2)

n

n-1

(n-1)/2 ≃ n/2

n/2 - 1

(n/2 - 1)/2 = n/2^2

more than.

√i — more decrement ; n/2^3.

O(log n)

O(log(log n))

Dependent Loop:

```
for (i=1 ; i ≤ n ; i++)
    for (j = 1 ; j ≤ i ; j++)
        for (k=1; k ≤ j ; k++)
            Pf ("x x x")
}}}
```

What is frequency count of Pf.
How many times it executes.

i = 1	2	3	n
j = 1	1, 2	1, 2, 3	1 to m
k = 1	1 to 1, 1 to 2	1-1, 1-2, 1-3	1-1, 1-2, ... 1..n
Pf = 1	1 + 2	1 + 2 + 3	1 + 2 + 3 + ... + n

$S_n = 1 + (1+2) + (1+2+3) + (1+2+3+4) + \cdots + (1+2+3 \cdots + n)$

$S_n = \sum t_n = \sum \frac{n(n+1)}{2} = \frac{1}{2}\left[\sum n^2 + \sum n\right]$ (★)

$\frac{n(n+1)}{2}$

$S_n = \sum t_m = \frac{1}{2}\left[\sum n^2 + \sum n\right]$

$= \frac{1}{2}\left[\frac{n(n+1)(2n+1)}{6} + \frac{n(n+1)}{2}\right]$

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

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

$S_n = \frac{n(n+1)(n+2)}{6} \simeq O(n^3)$

$= \frac{n(n+1)(n+2)}{6}$

2n+1+3
2n+4
2(n+2)

① $\log_c ab = \log_c a + \log_c b$

② $\log \frac{a}{b} = \log_c a - \log_c b$

$\log a^b = b \log_c a$

$\log_b a = \frac{\log_c a}{\log_c b}$

$a^{\log_c b} = b^{\log_c a}$

$\log_b a = \frac{1}{\log_a b}$

$64^{\log_2 n} = n^{\log_2 64}$
$= n^{\log_2 2^6}$
$= n^{6 \log_2 2}$
$= n^6$

$1 + \frac{1}{2} + \frac{1}{3} + \frac{1}{4} + \cdots + \frac{1}{n} = \log_e n$

proof $\int_{x=1}^{n} \frac{1}{x} dx = \log n$

$\frac{1}{2} + \frac{1}{3} + \frac{1}{5} + \frac{1}{7} + \frac{1}{11} + \frac{1}{13} + \cdots + \frac{1}{p}$ ($p <= n$ prime)

$= \log \log_e n$

⑤
```
for(k=1; k<=n; k++)
    for(i=k; i<=n; i++)
        for(j=i; j<=n; j++)
            Pf(xxx)
```

k	1	2	...	n
i	1, 2 ... n	2, 3, ... n	...	n
j	1 to n, 2 to n, n to n	2 to n, 3 to n, n to n	...	n to n

$[n + (n-1) + \cdots 1] + [(n-1) + (n+2) \cdots + 1] + \cdots$

$= \frac{n(n+1)(n+2)}{6}$ $O(n^3)$

Recursive algo :

The running time of the recursive algo. can be obtained by a recurrence

or.

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

For solving the recurrence we will discuss follow methods

1. Substitution method
2. Recursion Tree method
3. Master method.

(Iteration, changing variable, chain eqn).

ex. 1

```
rec(n)
{
    if (n <= 1)
        return 1;
    else
        return (rec(n-1) + n)
}
```

Time complexity

$T(n)$ = TC of rec(n)

$$T(n) = \begin{cases} 1 & n < 1 \\ T(n-1) + 1 & n \geq 1 \end{cases}$$

Then solve the recurrence relations

$$T(n) = T(n-1) + 1$$
$$= T(n-2) + 1 + 1$$
$$= T(n-3) + 3$$
$$\vdots$$
$$= T(n-k) + k$$

$\begin{bmatrix} n-k = 1 \\ k = n-1 \end{bmatrix}$ substituting this value in the above

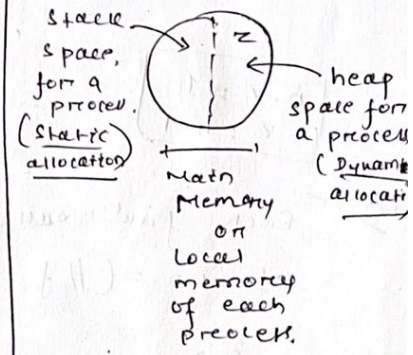
program $T(n) = T(1) + n - 1$

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

(1) Recursive algo. are like sub routine calls.

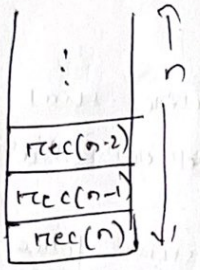
(2) Stack is used to store return address

(3) Program under execution → process.



Space Complexity:

Though it is Recursive algo. we have to see stack space.



stack entry is required

Space comm of it is $O(n)$.

No. of function calls = $\theta(n)$

rec(n)
/
rec(n-1)
/
.
.
rec(1)

(for space find depth of rec tree

Value return by algorithm

$$\text{return} \begin{cases} 1 & n = 1 \\ \text{rec}(n-1) + n & n > 1 \end{cases}$$

$\text{rec}(n) = \text{rec}(n-1) + n$

$\text{rec}(n) = [\text{rec}(n-2) + (n-1)] + n$ — (k times) $= \text{rec}(n-3) + (n-2) + (n-1) + n$

$= \text{rec}(n-k) + (n-2) + (n-1) + n$

$= \text{rec}(n-k) + (n-(k-1)) \cdots + (n-1) + n$

$n - k = 1$
$k = n-1$

$= \text{rec}(1) + 2 + 3 \cdots$

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

Note: Each func call $O(1)$ if exclude subfunc call then TC of algo. = (# of function call).

(2)

```
Rec(n)
{
  if (n <= 1)
    return (1)          ------ 1
  else
  { rec(n-1);           ------ T(n-1)
    for(i=0; i<=n; i++) ------ n
      P+(  );
  }
```

Recurrence Rel^n

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

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

No. of function call

Step of func call = $O(n)$

```
if (n <= 1)
    return (1)                    ——— (1)
else
{
    return (n-1) + rec(n-1) + 1
}
```

$T(n-1)$ $T(n-1)$ c (for 2 Add^n + return)

Time complexity Recurrence rel^n

$$T(n) = \begin{cases} 1 & n \le 1 \\ 2T(n-1) + c & n > 1 \end{cases}$$

$$T(n) = 2T(n-1) + c$$

$[\because T(n-1) = 2T(n-2) + c]$

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

$= 2^2 T(n-2) + c(2+1)]$

$T(n-2) = 2^3 T(n-3) + c$

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

Substitute until k terms.

$[2^k T(n-k) + c(2^{k-1} + 2^2 + 2 + 1)]$ — $\frac{1(2^k - 1)}{2}$

$n - k = 1$
$k = n - 1$

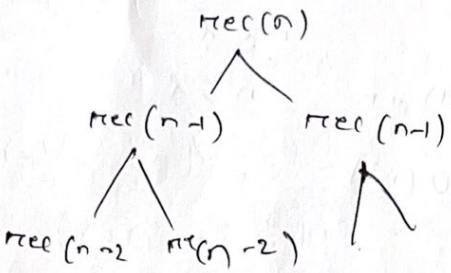
$2^k - 1$

$\Rightarrow 2^{n-1} T(1) + c\{2^{k+1} - 1\}$

$\vdots$

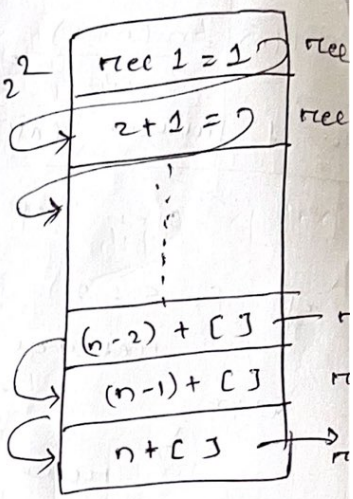
$T(n) = O(2^n)$

fun' call



Total No = $1 + 2 + 2^2 + \dots + 2^{n-1}$

$= 2^n - 1 = \theta(2^n)$



④
```
Rec (n)
{
  if (n ≤ 1)
    return
  else
    return (rec (n-1) + n);
}
```
fun' call

① Stack space used

= No. of stack entries = n

n = depth of recursion.

Space complexity = O(n)

② Value returned by algo :

$\frac{n(n+1)}{2}$ is $O(n^2)$.

Let n = 10 it returns $= \frac{10 \times 11}{2} = 55$

③ How many additions = $1 + 2 + 3 + \dots + n$ is $O(n)$.

```
Rec (n)
{
  if (n ≤ 1)
    return (1)
  else
    return ( 2 * Rec (n-1) + n )
}
```

TC

$$TC(n) = \begin{cases} 1 & n \leq 1 \\ T(n-1) + c & n \geq 1 \end{cases} \Rightarrow TC(n) = T(n-1) + c$$

$\vdots$

$O(n^2)$

(5) Recursive Algo:

```
Algo rec(n)
{
  if ( )
    rec (1)
  else
    rec ( rec (n-1) + rec (n-1) + rec (n-1) + n )
}
```

T(n) → T(n-1), T(n-1), T(n-1)

rec(n-1) ; → T(n-1)
rec(n-1) ; → T(n-1)
rec(n-1) ; → T(n-1)
rec(- - - ;) → n

TC = Time complexity

$$T(n) = \begin{cases} c & n \le 1 \\ 3T(n-1) + a & n \ge 1 \end{cases}$$

$T(n) = 3T(n-1) + a$ [$\because T(n-1) = 3T(n-2) + a$]

$= 3[3T(n-2) + a] + a$

$= 3^2 T(n-2) + 3a + a$

$= 3^2 [3T(n-3) + a] + 4a$

$= 3^3 T(n-3) + a[3^2 + 3 + 1]$

$n - k = 1$

$= 3^k T(n-k) + a[3^{k-1} + \dots + 3^2 + 3]$

$T(n) = 3^k \cdot T(n-k) + \frac{a}{2}[3^k - 1]$

$T(n) = 3^{n-1} \cdot c + \frac{a}{2}[3^n - 1]$

$\simeq O(3^n)$.

value returned by algo:

$$\pi(n) = \begin{cases} 1 \\ 3\pi(n-1) + n \end{cases}$$

$= 3\pi(n-1) + n$

$= 3[3\pi(n-2) + n - 1] + n$

$= 3^2 \cdot \pi(n-2) + 3(n-1) + n$

$= 3^2 [3\pi(n-3) + (n-2)] + 3(n-1) + n$

$= 3^3 \pi(n-3) + 3^2(n-2) + 3(n-1) + n$

⋮ k times.

$= 3^k \cdot \pi(n-k) + 3^{k-1}(n-(k-1))$

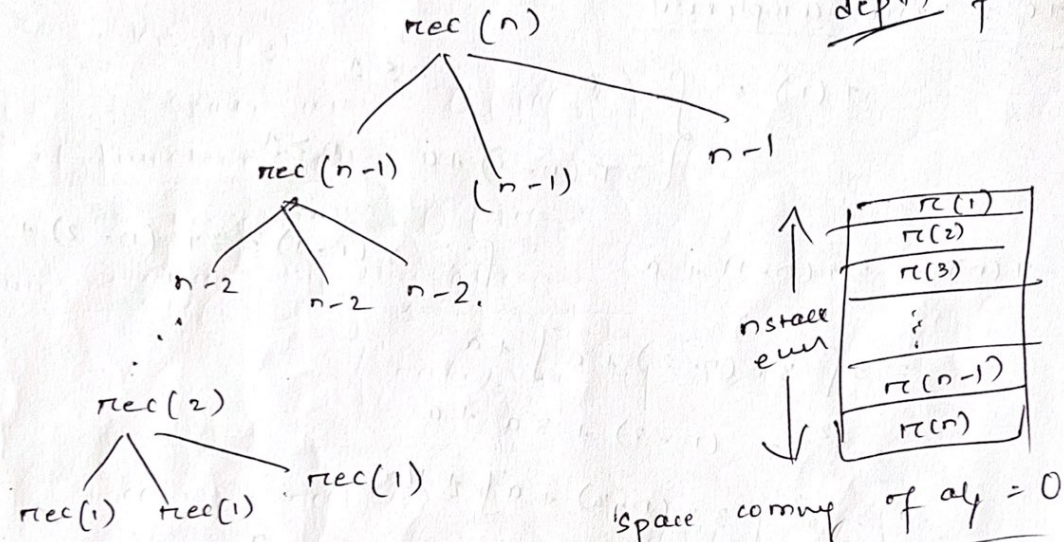
$+ \cdots\cdots + 3^2(n-2) + 3(n-1) + n$

$\pi(n) = 3^{(n-1)} \cdot 1 + 3^{(n-2)} \cdot 2 + 3^{n-3} \cdot 3 + \cdots\cdots +$

G.P. $3^2(n-1) + 3(n-1) + n$

Space complexity

depth of tree



Space comp of alg = O(

$T(n) = 3^{n-1} T(1) + 3^{n-2}(n-n+2) + 3^{n-3}(3) + \cdots$

$+ 3^2(n-2) + 3(n-1) + 3^0(n)$

$n-2=1$

$k = (n-1)$

$T(n) = 3^{n-1} T(1) + 3^{n-2}(n-n-2) + 3^{n-3}(3) + \cdots$

$+ 3^2(n-2) + 3(n-1) + 3^0(n)$ —— (1)

$n-(n-1-1)$

$n-n$

$\Rightarrow 3T(n) = 3^n \times 1 + 3^{n-1} \cdot 2 + 3^{n-2} \cdot 3 + \cdots$

(multiplying 3 both sides) $+ 3^3(n-2) + 3^2(n-1) + 3 \cdot n$ —— (2)

Now substract 2nd eqn from 1st eqn.

$\Rightarrow 3(T(n) - T(n)) = 3^n \cdot 1 + (3^{n-1} \cdot 2 - 3^{n-1} \times 1) + (3^{n-2} \cdot 3 - 3^{n-2} \cdot 2)$

$= 3^n \cdot 1 + 3^{n-1}(2-1) + 3^{n-2}(3-2)$

$+ 3^{n-3}(4-3) - \cdots + 3^3(n-2-n+3)$

$+ 3^2(n-1-n+2)$

$$2T(n) = 3^n \cdot 1 + 3^{n-1} + 3^{n-2} + 3^{n-3} - \cdots + 3^3 + 3^2 + 3$$

A.P

$$= \frac{3(3^n - 1)}{3 - 1} - n$$

$$T(n) = \frac{3^{n+1} - 3 - 2n}{4}$$

$$T(n) = \frac{1}{4}\left[3^{n+1} - 3 - 2n\right]$$

$$T(n) = \theta(3^n)$$

) Time complexity find out

```
rec(n)
{
  if (n ≤ 1)
    return
  else
    return (rec(n/2) + n)
}
```

$T(n/2)$

Recurrence rel^n

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

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

$$\left[\because T(n/2) = T(n/2^2) + 1\right]$$

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

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

⋮ k times

$$= T\left(n/2^k\right) + k$$

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

$$= T(1) + \log 2^n$$

$$= O(\log 2^n).$$

Time complex.

rec(n) or rec(n/2^0) — 1st

rec(n/2) — 2nd

rec(n/2^2) — 3rd.

⋮

rec(n/2^k) = rec(1) — k level

k = log_2 n

Space comm = O(log_2 n)

No. of function call

k times call O(log_2 n)

Value return by this algorithm.

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

rec(n) = rec(n/2) + n

[rec(n/2) = rec(n/2^2) + n/2]

= rec(n/2^2) + n/2 + n

= rec(n/2^3) + n/2^2 + n/2 + n

= rec(n/2^3) + n[1/2^2 + 1/2 + 1]

⋮ k sum

= rec(n/2^k) + n[1/2^{k-1} + ... 1/2 + 1]

$$\frac{1\left(1-\frac{1}{2}\right)^k}{1-1/2}$$

= rec(n/2^k) + 2n[1 − 1/2^k]

[∵ n/2^k = 1 ⇒ n = 2^k]

= rec(n) = T(1) + 2n − 2n/2^k

= 2n − 1

Rec(n) = O(n).

(7)

```
rec(n)
{ if (n<=1)
    return(1)
  else
    rec(n/2)
    for(i=1; i<=n; i++)
    Pf(----)
```

Time complexity:

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

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

$\vdots$

$T(n) = 2n - 1 = \Theta(n)$

Space complexity

$\theta(\log n)$

No. of function call

$O(\log_2 n)$

Here time complexity is bigger than no. of function call because of the for loop which taken n times.

(8)

```
rec(n)
{ if(n<=1)
    return(1)
  else {
    rec(n/2)
    rec(n/2)
    for(i=1; i<=n; i++)
    Pf(----)
  }}
```

Time complexity:

$$T(n) = \begin{cases} 1 & n = 1 \\ 2T(n/2) + n & n \geq 2. \end{cases}$$

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

$$= 2\left[2T(n/2^2) + n/2\right] + n$$

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

$$= 2^2\left[2T(n/2^3) + \frac{n}{4}\right] + 2n$$

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

$$\vdots$$

$$= 2^k T(n/2^k) + kn$$

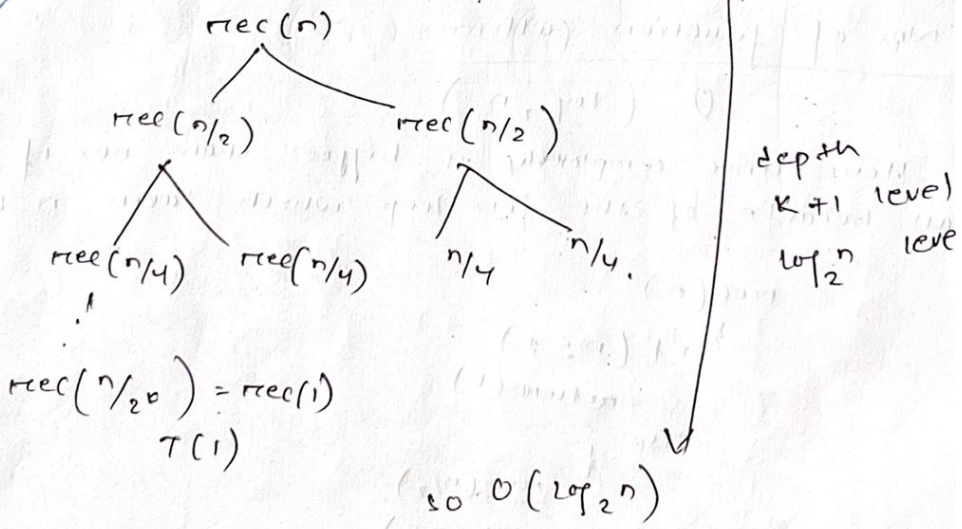
$\frac{n}{2^k} = 1 \Rightarrow n = 2^k$ and $k = \log_2 n$

$$= n\,T(1) + (\log_2 n)\,n$$

$$= n + n\log_2 n$$

$$= O(\log_2 n)$$

Space complexity



so $O(\log_2 n)$

(9)

```
rec(n)
{
  if (n ≤ 1)
    return 1
  else
    return (rec (sqrt(n) + 1))
}
```

$T(\sqrt{n})$

Time complexity

$$T(n) = \begin{cases} 1 & ; \ n < 1 \\ T(\sqrt{n}) + 1 & ; \ n > 2 \end{cases}$$

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

$$\left[\because T(n^{1/2}) = T(n^{1/2^2}) + 1 \right]$$

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

$$= T(n^{1/2^3}) + 3$$

$$\vdots$$

$$k$$

$$= T(n^{1/2^k}) + k$$

Now $\Rightarrow n^{1/2^k} = 2$ $\quad k = \log(\log n)$

$$T(n) = T(2) + \log(\log n)$$

$$= 1 + \log(\log n)$$

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

Space complexity

rec(1)

rec($n^{1/2^k}$) = rec(2) $\quad k = \log(\log n)$

$O(\log(\log n))$

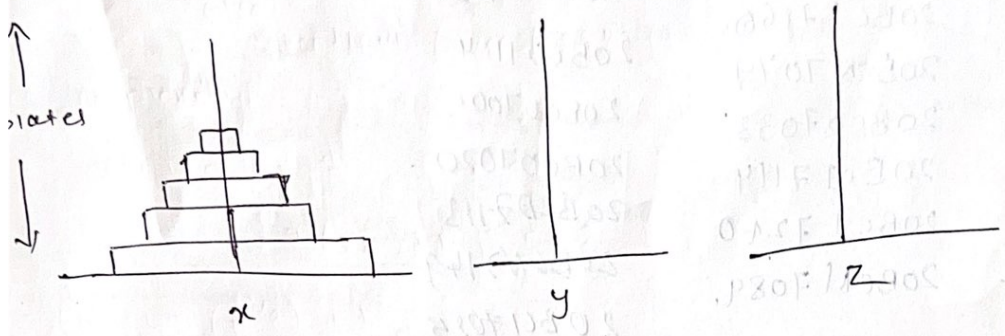
1. M.P

$rec(n) = rec(n-a) + rec(n-b)$	$O(2^n)$
$rec(n) = rec(n-1) + rec(n-1)$	2^n
$rec(n) = rec(n-2) + rec(n-2)$	$2^{n/2}$
$rec(n) = rec(n-1) + rec(n-1) + rec(n-1)$	3^n
$rec(n) = rec(n-1) + rec(n-2)$	2^n
$rec(n) = rec(n-1)$	n

	TC	SC
$rec(n) = \begin{cases} rec(n-a) \\ rec(n-a) + rec(n-b) \end{cases}$	n 2^n	n/a $\max(n/a$
$rec(n) = \begin{cases} rec(n/a) \\ rec(n/a) + rec(n/a) \end{cases}$	$\log n$ n	$\log n$ $\log n$
$rec(n) = \begin{cases} rec(\sqrt{n}) \\ rec(\sqrt[a]{n}) + rec(\sqrt[a]{n}) \end{cases}$	$\log(\log n)$ "	$\log(\log$ "

Tower's of Hanoi

Let's assume 3 towers and n plates in tower where diameters of the plate are in decreasing order.



Find no. of plates moves to transfer n plates from tower x to tower y.

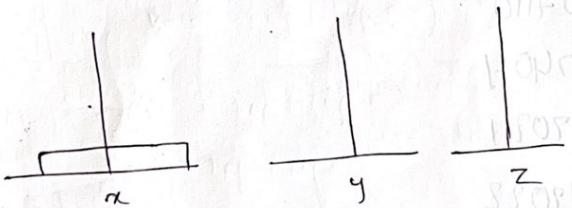
Condition:

1. Only top plate can move at a time.
2. At any time low diameter plate should not below the upper diameter.

Solution:

No. of moves

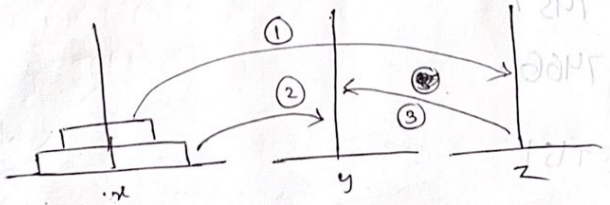
$n = 1$



1 move.

$x \to y$

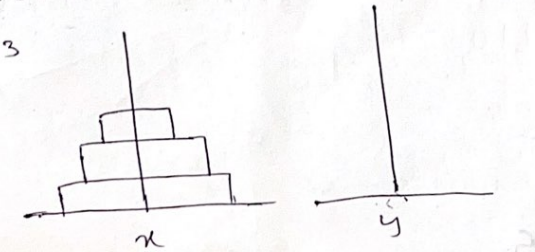
$n = 2$



3 moves

$x \to z$, $x \to y$, $z \to y$

$n = 3$



7 moves.

1. $x \to y$, $x \to z$, $y \to z$
2. $x \to y$
3. $z \to x$, $z \to y$, $x \to y$.

Procedure :

To transfer n plates from x ⇒ y

① Transfer (n-1) plates from x → z recursively

② Transfer 1 plate from x → y

③ Transfer (n-1) " " z → y recursively.

gorithm TOH (x, y, z, n) (x: source, y: destin, z: Temporary.)

{

Transfer n plates from x to y using z as intermediate temporary.

if (n == 1) // one plate in the tower x

move (x → y)

else // more than one plate.

{

TOH (x, z, y, n-1) - - - - - - - - - T(n-1)

move (x → y) ——————— (1)

TOH (z, y, x, n-1) ——————— T(n-1)

}

T(n) = No. of plate moves to transfer n plate

$$T(n) = \begin{cases} 1 & n == 1 \\ 2T(n-1)+1 & n > 1 \end{cases}$$

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

$= 2[2T(n-2) + 1] + 1$

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

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

!
! k times

$= 2^k T(n-k) + 2^{k-1} + \cdots\cdots\cdots + 2 + 1$

$$= 2^k T(n-k) + 2^{k-1} + \cdots\cdots + 2 + 1$$

$n - k = 1$

$k = n - 1$

$$= 2^{n-1} T(1) + 2^{n-2} + \cdots\cdots + 2 + 1.$$

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

$$T(n) = O(2^n)$$

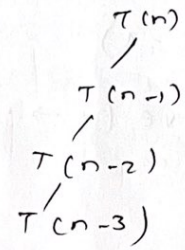
exponential problem

Un-decidable pr

Space complexity

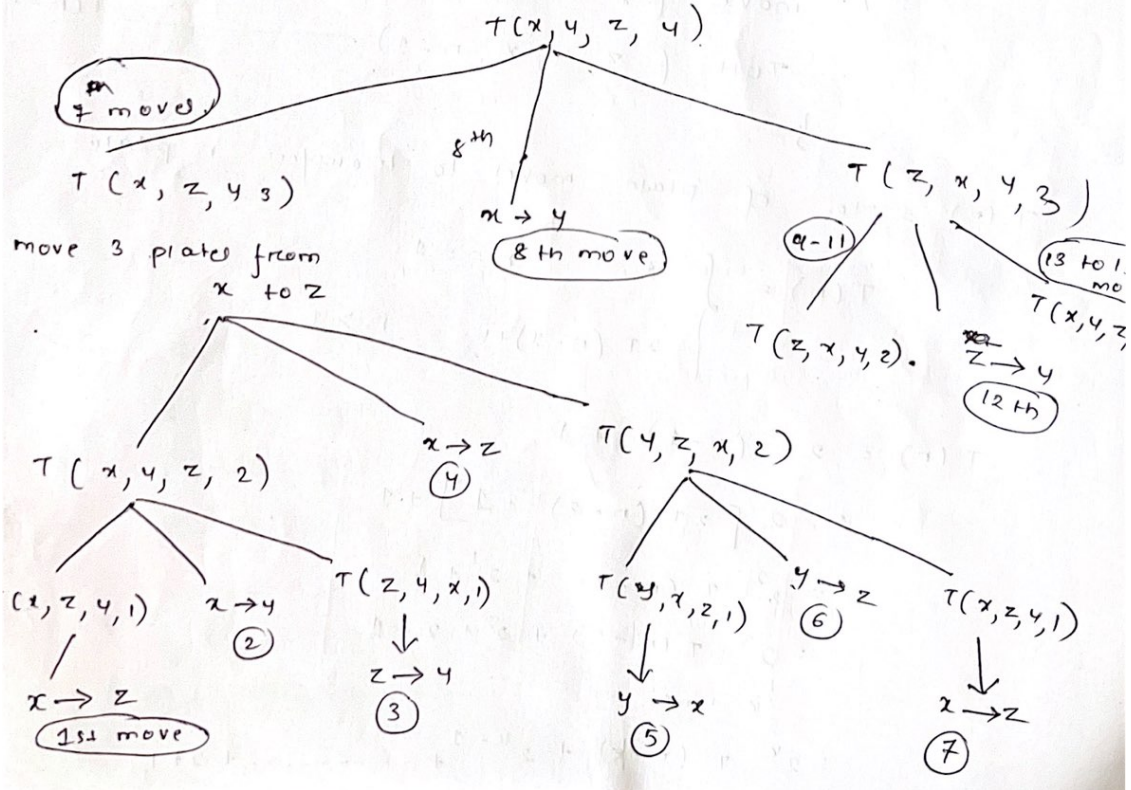
It is order of n. $O(n)$ because n is h

depth of recursion.



Question :-

(1) {x, y, z} towers n = 4 plates where what is 12th move.



(2) (x, y, z) towers n=5 plate in x. what is 28th move.