Four methods for solving Recurrence.

→ Substitution method.

→ Iteration method

→ Recursion tree

→ Master's Theorem.

(1)

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

$= O(\log n)$

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

n as n-1

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

$= T(n-2) + 2$

⋮

$T(n-k) + k.$

where $k = n-1$

$T(n-k) = T(n-n-1) + k.$

$= T(0) + 1$

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

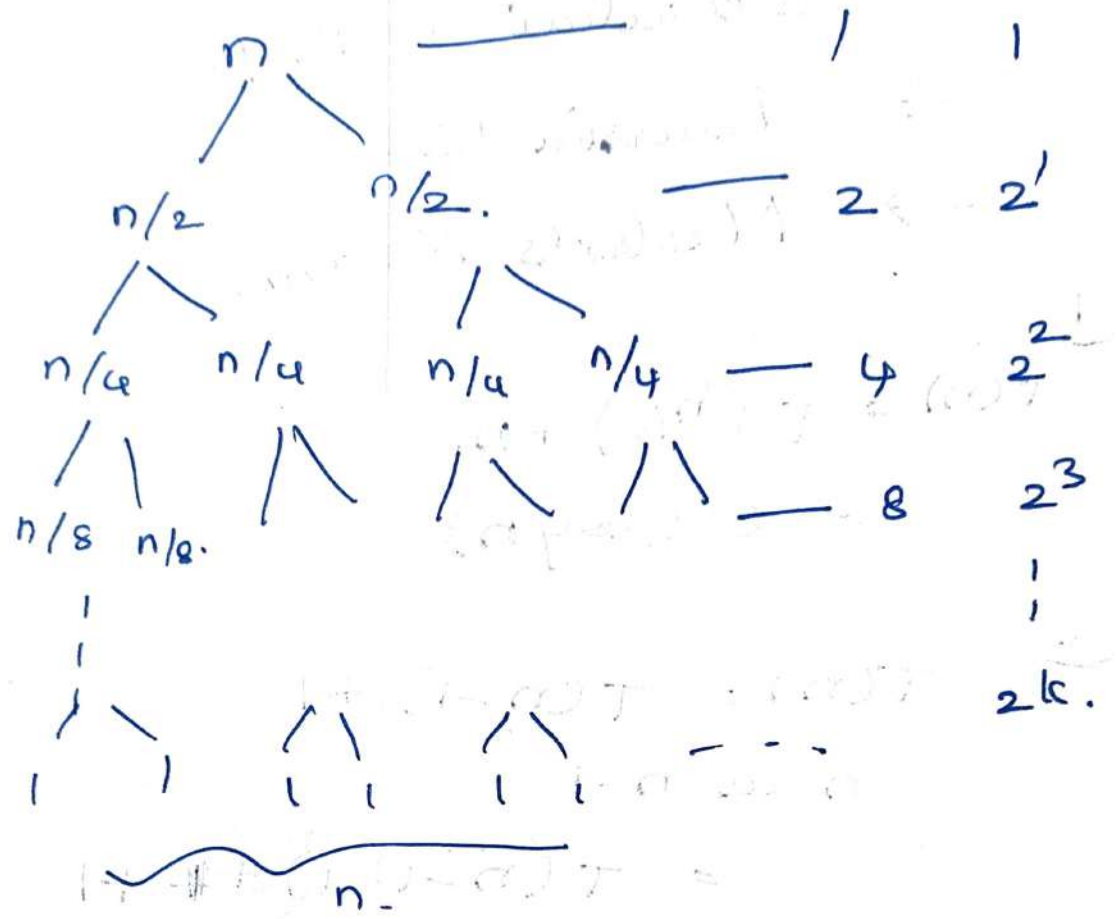
$= 1 + n - 1$

$= n.$

$T(n) = O(n)$

Recursion tree.

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



$2^0 + 2^1 + \cdots + 2^{\log_2^{n-1}} + n$

$= 2^{\log n} + n$

$= n - 1 + n$

$= 2n - 1)$

$= 2(n-1)$

$O(n)$

(n)

$n = 2^k \rightarrow$

$k = n/2$

$T(n) = O(n)$.

max element (A[0 -- n-1])
// input A[0 -- n-1]
// output max among A.

max ← A[0].
for i ← 1 to n-1 do.
    if A[i] > max.
        max ← A[i]
return max

$$T(n) = \sum_{i=1}^{n-1} 1 \qquad (u - l + 1) \times 1$$

$$= (n-1) \times 1$$

$$(n-1) \times 1 \quad \text{(remove constant)}$$

$$= n-1$$

$$T(n) = n$$

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

Unique element A[0 -- n-1]
// input A[0 -- n-1]
// output return 't' all elements are distinct otherwise false.

Conti --

```
for i ← 0 to n-2 do.
    for j ← i+1 to n-1 do.
        if A(i) = A(j)
            return false.
return true.
```

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

$$\sum_{i=0}^{n-2} = ((n-1)-(i+1)+1) \times 1$$

$$= (n-1-i-1+1) \times 1$$

$$= \sum_{i=0}^{n-2} (n-1-i)$$

$$= (n-2-0+1) \times (n-1-i)$$

$$= (n-2+1) \times n-1-i$$

$$= (n-1) \times (n-1-i)$$

(remove i)

$$(n-1)(n-1)$$

remove constant

$n \cdot n$

n^2

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

(1)

Matrix multiplication

A[0 -- n-1, 0 -- n-1)
B[0 -- n-1, 0 -- n-1)

// input A, B.
// Output C = A x B

```
for i ← 1 to n-1 do.
    for j ← 1 to n-1 do.
        C(i,j) ← 0.
        for k ← 1 to n-1 do.
            C(i,j) = C(i,j) + A(i,k) * B(k,j)
return C.
```

$$T(n) = \sum_{i=1}^{n-1} \sum_{j=1}^{n-1} \sum_{k=1}^{n-1} 1$$

$$\sum_{i=1}^{n-1} \sum_{j=1}^{n-1} : (n-1) - 1 + 1) \times 1$$

$$\sum_{i=1}^{n-1} n. \quad n.$$

$n \cdot n \cdot n.$

$$T(n) = O(n^3)$$

Number of digits in binary.

7 — 111 $c \rightarrow 3$.

12 — 1100 (8 4 2 1) $c \leftarrow 4$

Digit

$c \leftarrow 1$

while $n > 1$ do.

$c \leftarrow c+1$

$n \leftarrow \lfloor n/2 \rfloor$

return c.

$T(n) = O(\log_2 n)$. ie $n/2$ - takes log. time.

Recurrence relation

Linear recursive relation.

$T(n) = T(n-1) + n$. — (1)

$T(0) = 1$

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

$T(n-2) = T(n-3) + n$.

$T(n) = n-1$

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

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

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

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

⋮

$T(n-10)$

Recurrence relation

Factorial.

```
If n=0 return 1
else
    return n x fact (n-1).
        i.e fact(n-1)xn.
```

Recurrence equation

$M(n) = M(n-1) + 1$; M – multiplication, 1 – const time

$M(0) = 0$ – initial condition

$M(n) = M(n-1) + 1$ —— (1)

Put $n = n-1$ in eqn (1).

$M(n-1) = M((n-1)-1) + 1$

$M(n-1) = M(n-2) + 1$ ⟶ (2)

Put 2 in (1)

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

$M(n) = M(n-2) + 2$ —— (3)

Put $n = n-2$ in eqn (1).

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

$M(n-2) = M(n-3) + 1$ —— (4)

Put 4 in 3

$M(n) = M(n-3) + 1 + 2$

$= M(n-3) + 3$ —— (5).

for k^{th} step

$M(n) = M(n-k) + k$ ——— *

Initial condition $M(0) = 0$.

$n - k = 0$.

$k = n$. put k in eqn *

$M(n) = M(0) + n$.

$M(n) = 0 + n$.

$T(n) = O(n)$

Fibonacci series.

0, 1, 1, 2, 3, 5, 8 ---

$fib(0) = 0$

$fib(1) = 1$

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

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

Assume $M(n-1) \simeq M(n-2)$

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

$M(n) = 2M(n-2) + 1$.

Initial condition

$M(0) = 0 \quad M(1) = 1$

$M(n) = 2M(n-2) + 1$ —— (1).

replace n by $n-2$

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

$M(n-2) = 2M(n-4) + 1$ → (2).

Put 2 in (1).

$M(n) = 2M.(2M(n-4)+1) + 1$

$= 4m(n-4) + 2 + 1$

$= 4m(n-4) + 3.$

$M(n) = 2^2 m(n-4) + 3$ → (3).

replace n by n-4 in eqn (1).

~~M(n)+4~~.

$M(n-4) = 2M(n-4-2) + 1$

$M(n-4) \quad 2M(n-6) + 1$ — (4).

Sub 4 in 3.

$M(n) = 2^2(m.(2m.(n-6)+1) + 3$

$= 2^3 m(n-6) + 4 + 3.$

$= 2^3 m(n-6) + 7$ → (5)

for k runs

$m(n) = 2^k m(n-2k) + 2^k - 1$ (*)

Put $m(0) = 0.$

$n - 2k = 0$

$n = 2k \Rightarrow k = n/2$

Sub - $k = n/2$ in * $\Rightarrow 2$

$$2^k m(n-2k) + 2^k - 1$$

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

$$= 2^{n/2} m(0) + 2^{n/2} - 1$$

$$= 2^{n/2}(0) + 2^{n/2} - 1$$

$$0 + 2^{n/2} - 1$$

$$= 2^{n/2} - 1$$ remove constant

2^n.

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

General plan for recursive algorithm.

- Decide on parameter indicating an 'input size'.
- Identify the algorithm basic operation.
- check whether number of times basic operation is executed.
- Set up recursive relation, with an appropriate initial condition.
- solve recurrence relation.

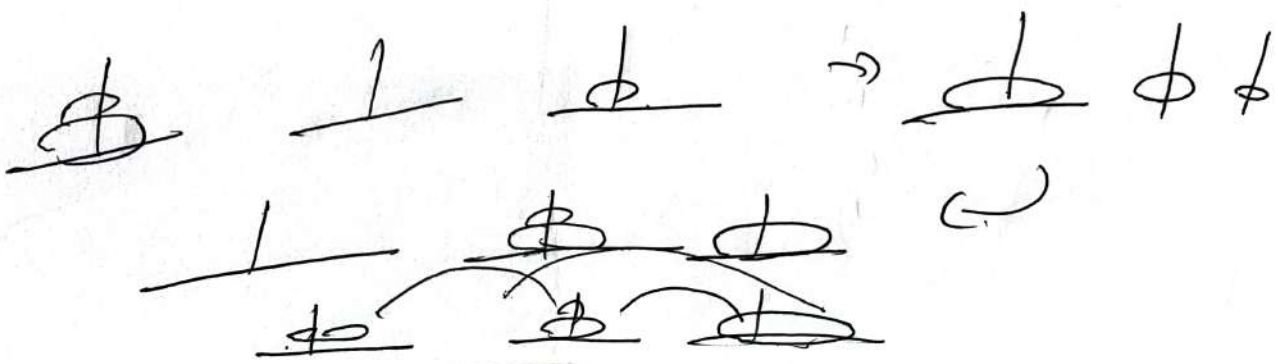
Tower of Hanoi.

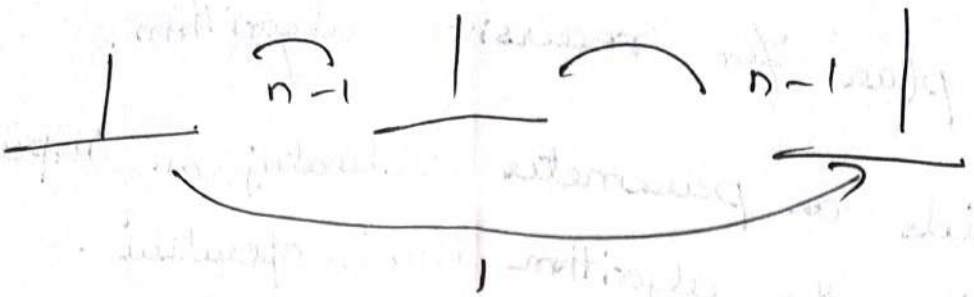
- Three plates (A, B, C)
  - we have to transfer 3 plates to right with 3 conditions.

→ Only one plate can be transferred at a time.

→ Only top plate can be transferred

→ Large size plate should be bottom





$M(n) = 2M(n-1) + 1 \longrightarrow (1)$

$M(1) = 1$

replace n with n-1 in eqn (1).

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

$= 2M(n-2) + 1 \longrightarrow (2)$

Subs (2) in (1).

$M(n) = 2(2m(n-2)+1) + 1$

$M(n) = 2^2 m(n-2) + 3 \longrightarrow (3)$

replace n with n-2 in eqn (1).

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

$M(n-2) = 2m(n-3) + 1 \longrightarrow (4)$

Sub (4) in (3)

$M(n) = 2^2 (2m(n-3)+1) + 3$

$2^3 m(n-3) 4 + 3$

$2 = 2^3 m(n-3) + 7 \longrightarrow (5)$

k runs.

$$M(n) = 2^k * M(n-k) + 2^k - 1 \rightarrow (*)$$

$M(1) = 1$ — initial condn.

$n - k = 1$

$k = n - 1$ s.

Sub k in eqn (*)

$M(n) = 2^{n-1} + 2^{n-1} - 1 \quad (M(1) = 1)$

$= \frac{2^n}{2} + \frac{2^n}{2} - 1$ (remove const

$= 2^n + 2^n$

$= 2 \cdot 2^n$. (remove consts).

$= 2^n$.

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

No. of digits in binary.

if $n=1$
return 1
else return bit$\lfloor n/2 \rfloor + 1$

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

$$T(1) = 1$$

replace n with $n/2$

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

$$= T(n/4) + 1$$

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

Sub $T(n/2)$ in eqn (1).

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

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

replace n by $n/2^2$ in eqn (1)

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

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

Sub $T(n/2^2)$ in eqn (3)

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

$$= T(n/2^3) + 3 \longrightarrow 5$$

for k runs.

$$T(n) = T(n/2^k) + k. \quad \text{——} \quad (*)$$

$$T(1) = 1$$

$$n/2^k = 1$$

$$n = 2^k$$

take log on both sides

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

$$k = \log_2 n \rightarrow (s)$$

Sub. (s) in (*)

$$T(n) = T(n/2^{\log_2 n}) + k \log_2 n$$

$$T(1) + \log_2 n.$$

$$= 1 + \log_2 n.$$

$$\boxed{T(n) = O(\log_2 n)}$$