Imp. Formula

If $A \subset B$, $A \cup B = B$, $A \cap B = A$

Properties of Union

(1) $A \cup B = B \cup A$ (Commutative)

(2) $(A \cup B) \cup C = A \cup (B \cup C)$

Problem - 5

Solve the following recurrence relation

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

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

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

$$T(n) = 2\left[2 \times T\left(\frac{n}{2^2}\right) + \frac{n}{2}\right] + n.$$

$$T(n) = 2^2 \times T\left(\frac{n}{2^2}\right) + 2 \times \frac{n}{2} + n. \quad \text{— (1).}$$

↓

Expanding this further.

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

$$T(n) = 2^3 T\left(\frac{n}{2^3}\right) + \underbrace{2^2 \times \frac{n}{2^2}}_{n} + \underbrace{2 \times \frac{n}{2}}_{n} + \underbrace{n}_{n}.$$

$$= 2^K T\left(\frac{n}{2^K}\right) + n + n + \cdots \text{ (K terms)}$$

$$T(n) = 2^K T\left(\frac{n}{2^K}\right) + K \cdot n.$$

but $\frac{n}{2^K} = 1 \Rightarrow K = \log_2 n.$

$$2^K = n.$$

$$T(n) = n\,T(1) + k \cdot n.$$

$$k = @\log_2 n.$$

$$= n + n \cdot \log_2 n. = n(1 + \log_2 n)$$

$$= n \log_2(2n).$$

$$T(n) = n \log_2 2n. = \underbrace{O(n \log n)}_{T.C}.$$

Problem - 6

Solve the following Recurrence relation

$$T(n) = \begin{cases} \sqrt{n}\,T(\sqrt{n}) + n & \text{if } n > 2 \\ 2 & \text{if } n = 2. \end{cases}$$

$$T(n) = n^{\frac{1}{2}}\,T(n^{\frac{1}{2}}) + n. \quad \text{——(1)}$$

$T(n^{\frac{1}{2}}) = n^{\frac{1}{4}}\,T(n^{\frac{1}{4}}) + n^{\frac{1}{2}}$. — put this into the main term.

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

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

$$T(n) = n^{\frac{3}{4}}\,T(n^{\frac{1}{4}}) + 2n. \quad \text{——(2)}$$

$T(n^{\frac{1}{4}}) = n^{\frac{1}{8}}\,T(n^{\frac{1}{8}}) + n^{\frac{1}{4}}$. — put this

$$T(n) = n^{\frac{3}{4}}\left[n^{\frac{1}{8}}\,T(n^{\frac{1}{8}}) + n^{\frac{1}{4}}\right] + 2n$$

$$= n^{\frac{7}{8}}\,T(n^{\frac{1}{8}}) + n + 2n$$

$$T(n) = n^{\frac{7}{8}} T\left(n^{\frac{1}{8}}\right) + 3n \cdot \quad - (3)$$

From (1), (2) and (3), we have some pattern.

$$T(n) = n^{\frac{2^K - 1}{2^K}} T\left(n^{\frac{1}{2^K}}\right) \cdot + Kn \cdot$$

Let say this reaches the base case.

$$T\left(n^{\frac{1}{2^K}}\right) = T(2) = 2.$$

$$\therefore n^{\frac{1}{2^K}} = 2$$

$$\frac{1}{2^K} = \log_n 2$$

$$2^K = \log_2 n \quad - (1).$$

$$K = \log_2 \log_2 n \quad - (2).$$

Now apply this.

$$T(n) = n^{\frac{\log_2 n - 1}{\log_2 n}} \times \frac{T(2)}{2} + n \times \log_2 \log_2 n$$

Solve this one.

$$n^{\left(1 - \frac{1}{\log_2 n}\right)} = \frac{n}{n^{\frac{1}{\log_2 n}}} = \frac{n}{n^{\frac{1}{2^K}}}$$

$\because 2^K = \log_2 n$

now, $n^{\frac{1}{2^K}} = 2$

$= \frac{n}{2}$

(134)

$T(n) = 2 \times \frac{n}{2} + n \times \log_2 \log_2 n$.

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

dominant term = $n \times \log_2 \log_2 n$.

$$\boxed{TC = O(n \log \log n).}$$

Problem - 7 ~~[GATE]~~

$$T(n) = \begin{cases} 2T(\sqrt{n}) + 1 & \text{if } n > 2 \\ 2 & \text{if } 0 < n \leq 2. \end{cases}$$

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

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

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

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

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

$\vdots$

$= 2^K \underbrace{T(n^{\frac{1}{2^K}})}_{\text{Base case reached}} + 2^{K-1} + \cdots + 2^2 + 2 + 1$

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

$n^{\frac{1}{2^K}} = 2 \Rightarrow \boxed{K = \log_2 \log_2 n}$. $\boxed{2^K = \log_2 n}$

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

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

$\frac{1(2^k - 1)}{2-1} = (2^k - 1)$.

$= 2^{k+1} + 2^k - 1 \quad = 2 \cdot 2^k + 2^k - 1$

$= 2^k(2+1) - 1 \quad = 3 \cdot 2^k - 1$.

$= 3 \cdot \log_2 n - 1$

$\therefore \boxed{T(n) = 3 \times \log_2 n - 1}$

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

Problem - 8 ~~(Gate)~~

Let $T(n)$ be defined by $T(1) = 10$ and $T(n+1) = 2n + T(n)$ for all integers $n \geq 1$. Which ~~one~~ of the following represents the order of growth of $T(n)$ as a function of n?

(A) $O(n)$ (B) $O(n \log n)$ (C) $\theta(n^2)$ (D) $\theta(n^3)$.

(ISRO 2021)

$T(1) = 10$

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

$$T(n+1) = \begin{cases} T(n) + 2n & \forall\ n \geq 1. \\ 10 & n = 0. \end{cases}$$

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

Substitute ~~n = n+1~~ $n \Rightarrow n-1$.

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

(136)

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

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

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

$$= T(n-K) + 2(n-K) + 2(n-(K-1)) + \cdots + 2(n-2) + 2(n-1)$$

$T(n-K)$ — base case

$n-K=1 \Rightarrow T(1)=10$

$K = n-1$

$$= \underbrace{T(1)}_{10} + 2\times1 + 2\times2 + \cdots + 2(n-1)$$

$$= 10 + 2\underbrace{[1+2+\cdots(n-1)]}_{(n-1)\text{ natural nos}}$$

$$= 10 + 2\,\frac{(n-1)\cdot n}{2}$$

$$\boxed{T(n) = 10 + n^2 - n}$$

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

$\Theta(n^2)$ if satisfied $\rightarrow O(n^2)$, $\omega(n^2)$ } will also be satisfied.

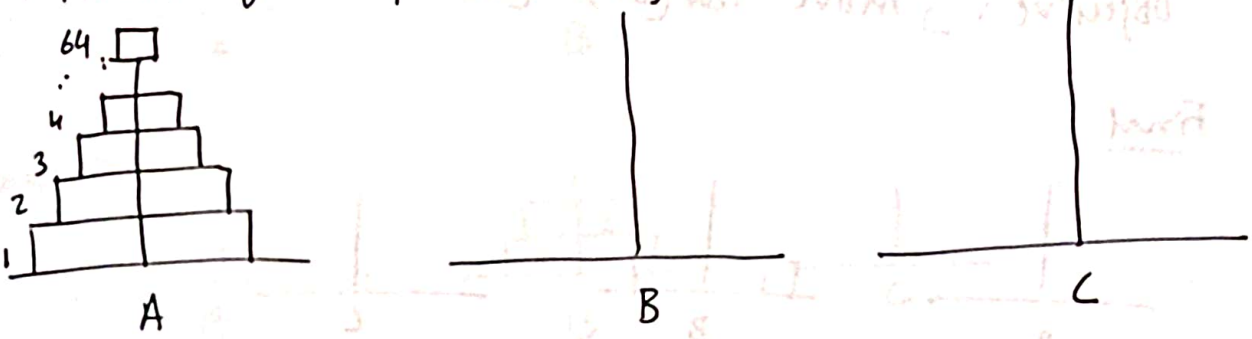
Towers of Hanoi

Introduction

. Classic puzzle of - maths + CS.

. ancient temple in Hanoi (Vietnam)

. Priests of temple working on 64 goldin disks



(pegs)

Task

. Shift all disks

A -> B or C

Rules

(1) Not allowed to move multiple disks at a time

(2) Can't place a larger disk on a smaller disk

~~(3)~~ Each move

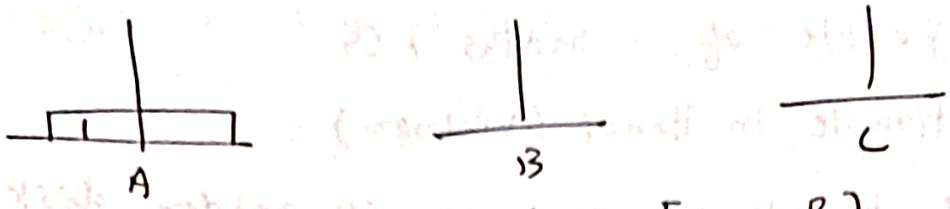
. Top disk from one stack ⟶ Place it on the top of another stack

Myth

This task will bring the world to its end. (millions of yrs)

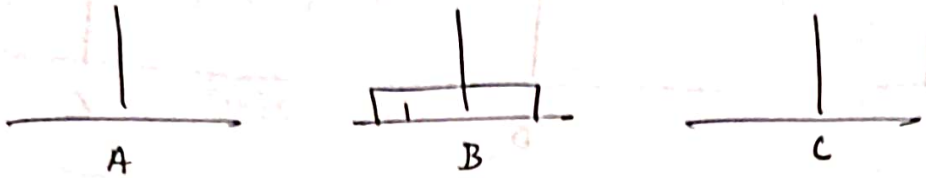
Examples

① $n = 1$



Objective: move ~~Disk~~ (D1) [A → B]

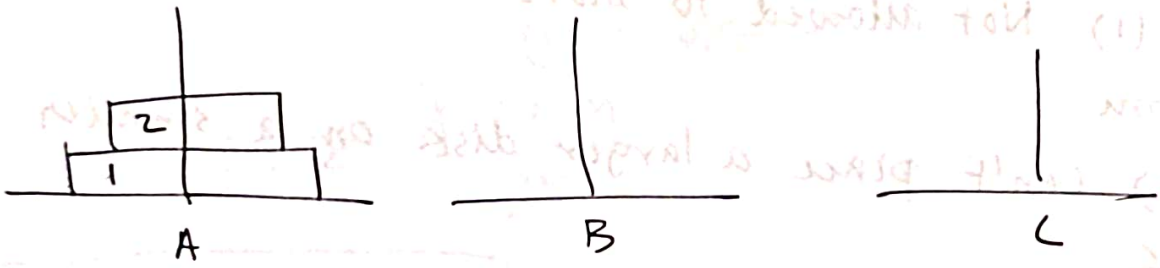
Final



Disk	Peg From		Peg To
~~D1~~ D1	A	→	B

Total = 1

② $n = 2$



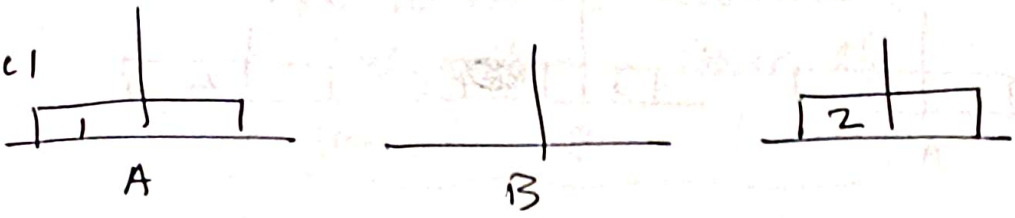
Objective: move ~~D1, D2~~ D1 + D2 from A → B.

Disk	Peg From		Peg To
D2	A	→	C
D1	A	→	B
D2	C	→	B

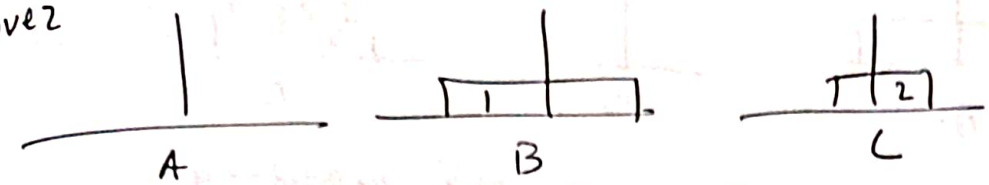
3

139

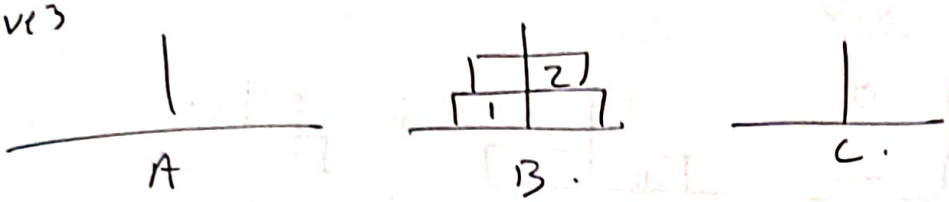
move 1



move 2



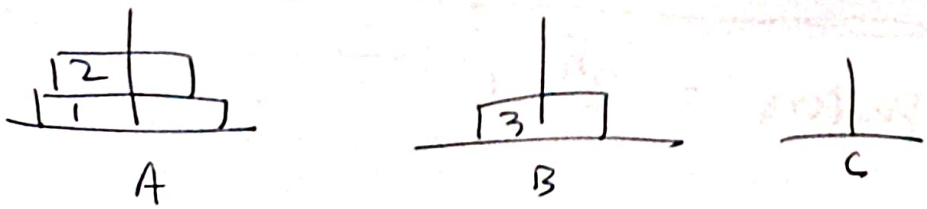
move 3



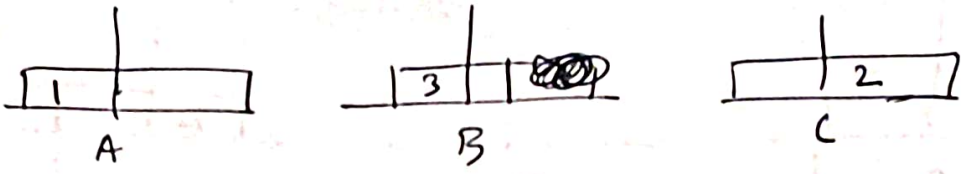
(3) $n = 3$

D1, D2, D3 : A → B : Goal.

	Disk	From		To	# moves
Print	(D2, D3)	A	→	C	3.
C1	D3	A	→	B.	
C2	D2	A	→	C	
C3	D3	B	→	C	



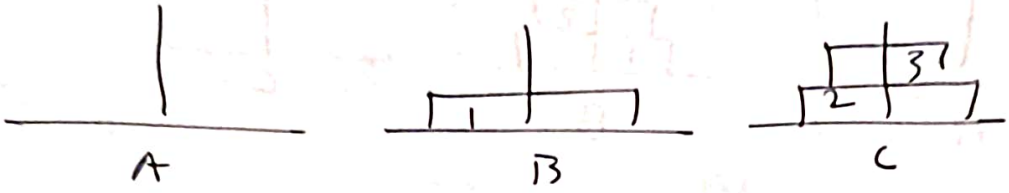
(2



(3

For 2 blocks → 3 moves are required.

(1)



• Again to move D2, D3 from C → B.
it takes 3 moves

Disk	From	To	via	moves
D2, D3	A	C	B	3.
D1	A	B	—	1
D2 D3	C	B	A	3
				7

3 examples : we have seen so far.

n	move
1	1
2	3
3	7

pattern = $2^n - 1$

How to write recursive algorithm - Towers of Hanoi

Thinking:

$n = 3$.

No. of moves = 7

Disk	From	To	Via	# moves
D2, D3	A	C	B	3.
D1	A	B	—	1
D2 D3	C	B	A	3.
				7

The above is what we did to transfer the disks from peg A → peg B for 3 disks.

Now, what if we have $n = K$.

1. move Top (K-1) disks from A → C via B.
2. move the 1st one from A → B or remaining only
3. move Top (K-1) disks from C → B via A.

Recursive Algorithm.

Algo TOH (A, B, C, n)
(from A, to B, via C)

{
// Base case
if n == 1
move A to B.
// Recursive case
else
TOH (A, C, B, n-1).
Move A to B.
TOH (C, B, A, n-1).
}.

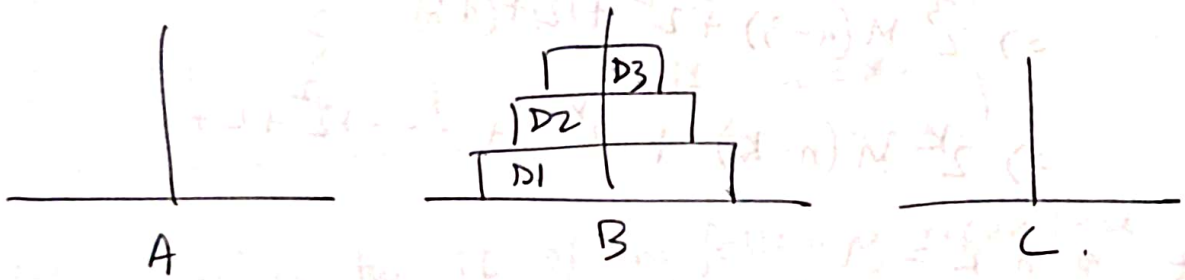
This will work for any 'n'.

~~In short, what~~
Re-write the algorithm.

Algo TOH (src, dest, via, n).
{
// Base case
if n == 1
move src -> dest -> mov1
else
TOH (src, ~~dest~~ via, dest, n-1) -> TOH1
move src - dest -> mov2
TOH (via, dest, src, n-1). -> TOH2

(143)

Call	Src	Dest	Via	n	A	B	C
Main	A	B	C	3	D1, D2, D3.		
TOH1	A	C	B	2			
TOH1	A	B	C	1	D1, D2	D3.	
MOV2	A	C			D1	D3	D2
TOH2	B	C	A	1	D1		D2, D3. ~~D2, D3~~
MOV2	A	B				D1	D2, D3
TOH2	C	B	A	2		D1	D2, D3
TOH1	C	A	B	1	D3	D1	D2
MOV2	C	B			D3	D1, D2	
TOH2	A	B	C	1		D1, D2 D3	



Objective: Achieved.

How to write the recurrence relations - Tower of Hanoi

$M(n)$ = no. of moves in TOH

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

$M(n)$ ↓ no. of disks

$M(n) = 2M(n-1) + 1$ — (1)

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

$\Rightarrow 2^2 M(n-2) + 2 + 1$ — (2)

$\Rightarrow 2^2 [2M(n-3) + 1] + 2 + 1$

$\Rightarrow 2^3 M(n-3) + 2^2 + 2 + 1$

$\Rightarrow 2^K M(n-K) + 2^{K-1} + \cdots + 2^2 + 2 + 1$

Let $n - K = 1$

$K = n - 1$

$\Rightarrow 2^{n-1} M(1) + 2^{n-2} + \cdots + 2^2 + 2 + 1$

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

$$S_n = \frac{1(2^n - 1)}{2-1} = 2^n - 1$$

$M(n) = 2^n - 1 = \underline{O(2^n)}$.

exponential time.

2^{64} seconds. ≅ 584 billion yrs.

Some Rapid fire Quiz

(1) Recurrence relation

=> math expression.

Solve ——> Big Prob. ——> smaller sub problems (n, n, n).

(2) Algo Sum (n) {

if n == 1

return 1

else

return n + Sum(n-1). ——> constant amount of time.

For Sum(n) —> T(n) time.

Sum(n-1) —> T(n-1).

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

(3) What is the TC of the following recurrence relation ?

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

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

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

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

$$= T(n-k) + (n-(k-1)) + (n-(k-2)) + \dots + (n-1) + n.$$

$n-k=1$.

$k = n-1$.

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

$= 1+2+3+\cdots+n = \frac{n(n+1)}{2}$

$\therefore \boxed{T(n) = O(n^2)}$.

Q.4. What is the recurrence relation for factorial

```
Algo fact(n) {
    if n == 1
        return 1.
    else
        return n x fact(n-1).
}.
```

$$M(n) = \begin{cases} M(n-1)+1 & \text{if } n>1 \\ 0 & \text{if } n=1 \end{cases}$$

↑ no multiply operation. — Option (only return).

Time complexity.

Q.5. What is the recurrence relations for the number of additions performed in the following algorithm.

```
Algo SumArray (arr, n) {
    if n == 1
        return arr[0]
    else
        return arr[n-1] + SumArray (arr, n-1)
}.
```

(147) Question:

$$A(n) = \begin{cases} A(n-1) + 1 \\ 0 \quad \text{if } n = 1. \end{cases}$$

Option A.

↑ no addition or any other math op.

→ but how arr[n-1] will be computed? Should it be part of the problem?

---

6. What is the recurrence relation for the number of comparisons performed in the following algorithm.

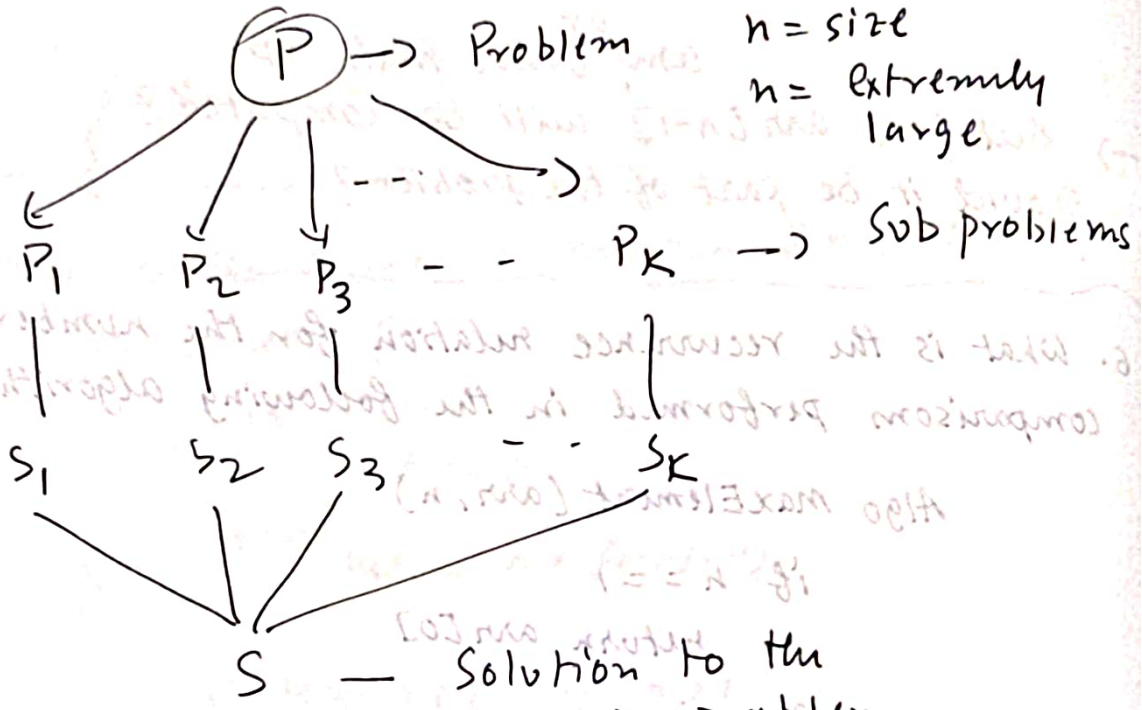
```
Algo maxElement (arr, n)
    if n == 1
        return arr[0]
    else
        max = MaxElement (arr, n-1).
        if arr[n-1] > max
            return arr[n-1]
        else
            return max.
```

~~C(n) = {~~

~~C(n) = {~~ $C(n) = \begin{cases} C(n-1) + 1 & \text{if } n > 1. \\ 1 & \text{if } n = 1. \end{cases}$

Option (c).

The strategy that we adopted in all the problems we dealt with so far - Divide & conquer



Assumption

P = Sort algo.

P1 P2 -- Pk = are also sort.

In this aspect, they are recursive in nature.

Divide & Conquer

Topics :

1. Binary Search
2. Finding max & min
3. Merge sort
4. Quick sort
5. Strassen's Matrix Multiplication.

Few examples on recurrence relations

Decreasing function.

```
void Test(int n)          ---> T(n).
{
  if(n>0)                 - 1
  {
    for(i=0; i<n; i++).   - 2n+2
    {                       (1)  (n+1)  (n)
      printf("%d", n)     - n.
    }
    Test(n-1)             - T(n-1) times
  }
}
```

$$T(n) = \begin{cases} T(n-1) + 1 + (2n+2) + n, & n > 0 \\ 1 & n = 0 \end{cases}$$

$$\boxed{T(n) = T(n-1) + 3n + 3.}$$

We can approximate this to $O(n)$.

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

$$\therefore T(n) = \begin{cases} T(n-1) + n, & n > 0 \\ 1, & n = 0. \end{cases}$$

Now solve this:

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

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

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

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

$$\vdots$$

$$= T(n-K) + (n-(K-1)) + \cdots + (n-2) + (n-1) + n.$$

Let $n-K = 0$

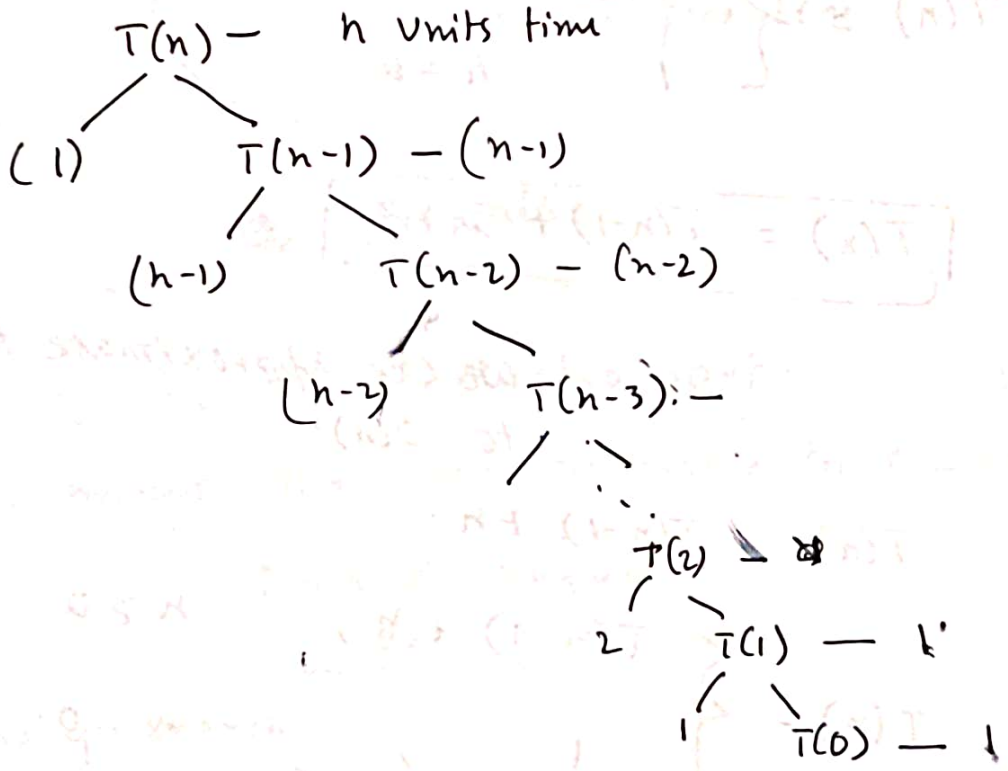
$n = K.$

$T(1) = 1$

$$\therefore T(n) = T(0) + 1 + 2 + \cdots + n.$$

$$= 1 + \frac{n(n+1)}{2} = O(n^2) \equiv \Theta(n^2).$$

Tree method



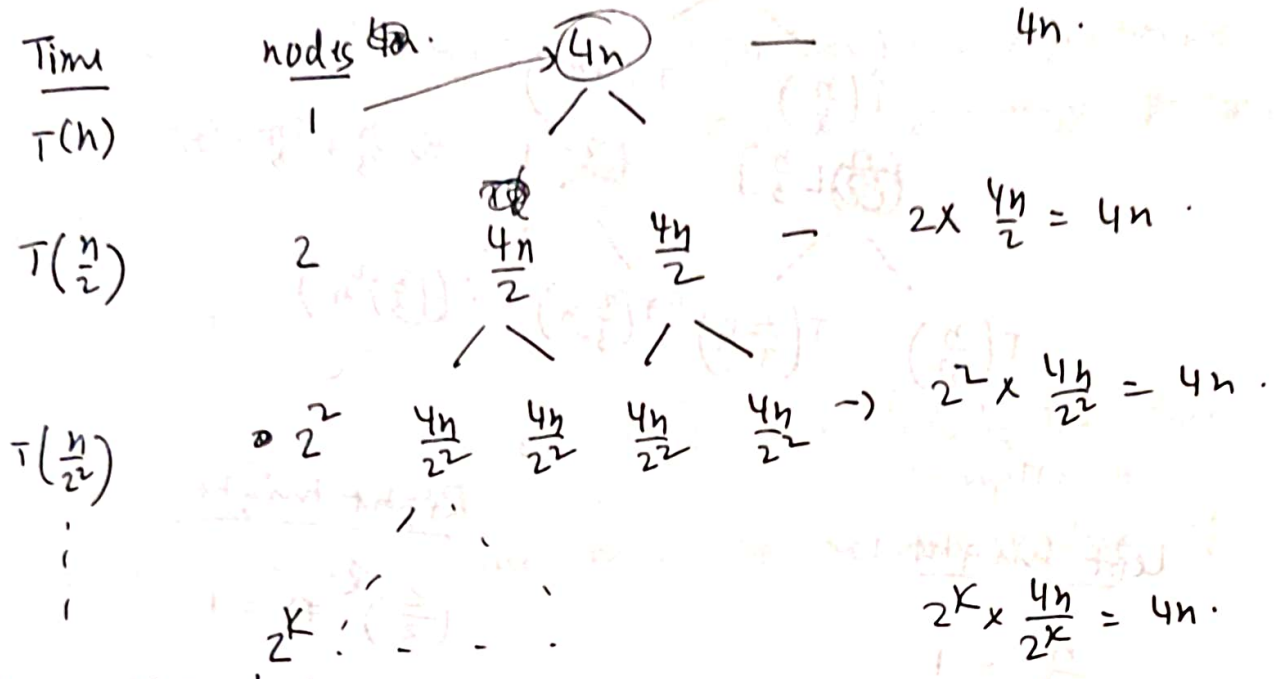
(151)

Tree method explanation

Equal split.

ex: 1) $T(n) = 2T\left(\frac{n}{2}\right) + 4n$.

$T(1) = 4 \cdot 1$



Generalize this's

$T\left(\frac{n}{2^k}\right)$ θ is when exit happens. $= T(1)$.

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

$$\Rightarrow k = \log_2 n.$$

↑ no. of levels = height.

$$\therefore \sum_{i=0}^{\log_2 n} 4n = 4n \sum_{i=0}^{\log_2 n} 1$$

$$= 4n \underbrace{(1+1+\cdots+1)}_{(1+\log_2 n) \text{ times}}$$

$$= 4n \times (1+\log_2 n)$$

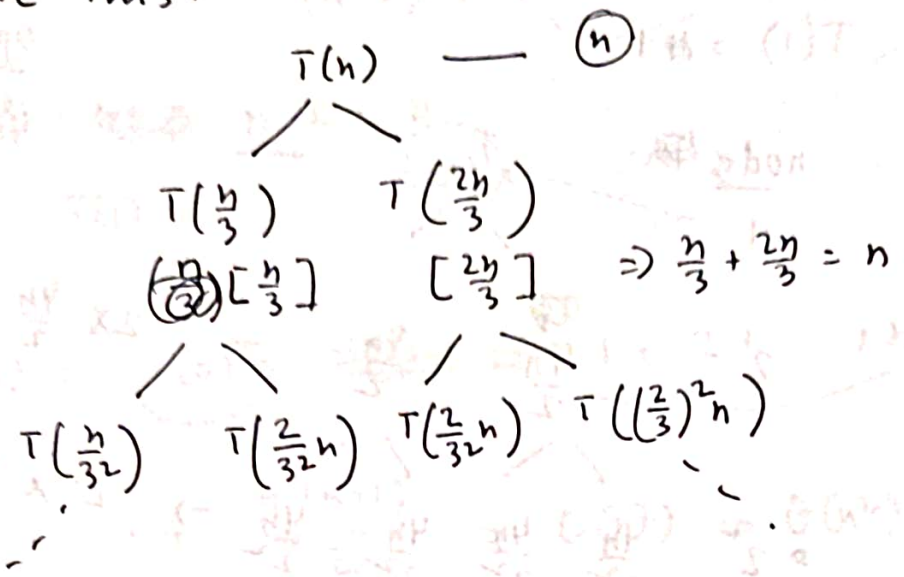
$$TC = O(n \log n)$$

1 (152)

Ex. 2 $T(n) = T(\frac{n}{3}) + T(\frac{2n}{3}) + n$.

** $T(1) = 1$

Solve this.



Left height

$\frac{n}{3^k} = 1$

$k = \log_3 n$.

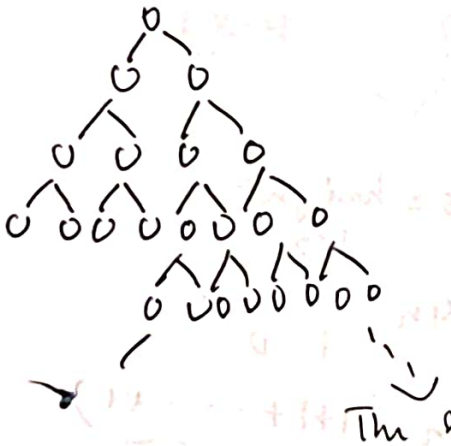
Right height.

$(\frac{2}{3})^l n = 1$

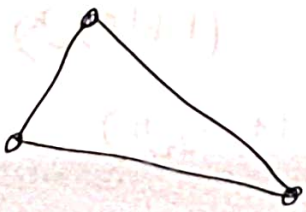
$l = \log_{(\frac{3}{2})} n$.

$l > k$. -- approximately

$\boxed{l \simeq 2k}$



The right side will go on much further.



$T(n) \quad - \left(\frac{2}{3}\right)^0 \cdot n \cdot$ ~~Note $T(n) = \Theta(n \log n)$.~~

$\vdots$

$\left(\frac{2}{3}\right)^k \cdot n \cdot$ Left Hand height

$\therefore$ Total height $= (1 + \log_2 n) \quad = \Omega(n \log n)$

$T(n) = n \times (1 + \log_2 n) \cdot = O(n \cdot \log n) \rightarrow$ Upper bound.

Note:

$n \log_3 n \cdot < T(n) < n \log_{\frac{3}{2}} n \cdot$

$\therefore \boxed{T(n) = \Theta(n \log n)} \rightarrow$ This is the time complexity

, We don't know what will be actual value of $T(n)$.

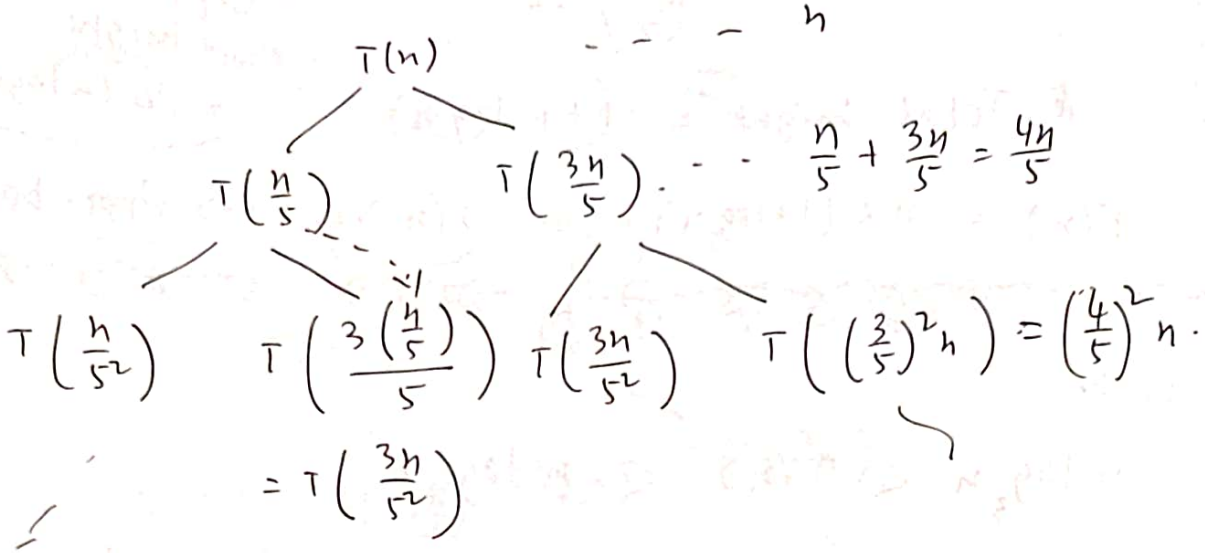
Solve recurrence relations

- Back Substitution method ✓
  - usually applied where unary tree is present.
  - ~~values are functions are pass present.~~
- Tree method ✓
- Master Theorem ✓

(154)

ex: 3 $T(n) = T\left(\frac{n}{5}\right) + T\left(\frac{3n}{5}\right) + n$.

$T(1) = 1$



Left Height < Right Height.

Let us compute the left height.

$T\left(\frac{n}{5^k}\right) = T(1)$

$\frac{n}{5^k} = 1$

$\Rightarrow \boxed{k = \log_5 n}$

Left Height = $1 + \log_5 n$.

Right height

~~$T\left(\left(\frac{3}{5}\right)^{\ell} n\right)$~~ → Right height

$T\left(\left(\frac{3}{5}\right)^{\ell} n\right) = T(1)$

$n = \left(\frac{5}{3}\right)^{\ell} \Rightarrow \ell = \log_{\left(\frac{5}{3}\right)} n$.

RH = $1 + \ell = 1 + \log_{\left(\frac{5}{3}\right)} n$.

(155) Left Hand T.C (LTC)

$$\left(\frac{4}{5}\right)^0 n + \left(\frac{4}{5}\right)^1 n + \left(\frac{4}{5}\right)^2 n + \cdots + \left(\frac{4}{5}\right)^{\log_5 n} \times n.$$

$$= n\left[\left(\frac{4}{5}\right)^0 + \left(\frac{4}{5}\right)^1 + \cdots + \left(\frac{4}{5}\right)^{\log_5 n}\right]$$

$$a = 1 \qquad r = \left(\frac{4}{5}\right).$$

$$S_n = \frac{1(1-r^n)}{1-r} = \frac{1-\left(\frac{4}{5}\right)^{\log_5 n}}{1-\frac{4}{5}}.$$

$$= 5\left[1-\left(\frac{4}{5}\right)^{\log_5 n}\right]$$

b $LTC = 5n\left[1-\left(\frac{4}{5}\right)^{\log_5 n}\right]$

Now, $\left(\frac{4}{5}\right)^{\log_5 n} = n^{\log_5\left(\frac{4}{5}\right)}.$

Now $\log_5\left(\frac{4}{5}\right) = -0.1386$

For large values of n.

$$1 - n^{\log_5\left(\frac{4}{5}\right)} \approx 1$$

$LTC \simeq 5n.$

Right Hand TC (RTC)

$$= 5n\left[1 - n^{\log_{\frac{5}{3}}\left(\frac{4}{5}\right)}\right]$$

$$\log_{\frac{5}{3}}\left(\frac{4}{5}\right) = -0.4369$$

$\simeq 5n.$

$5n < T(n) < 5n$.

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

---

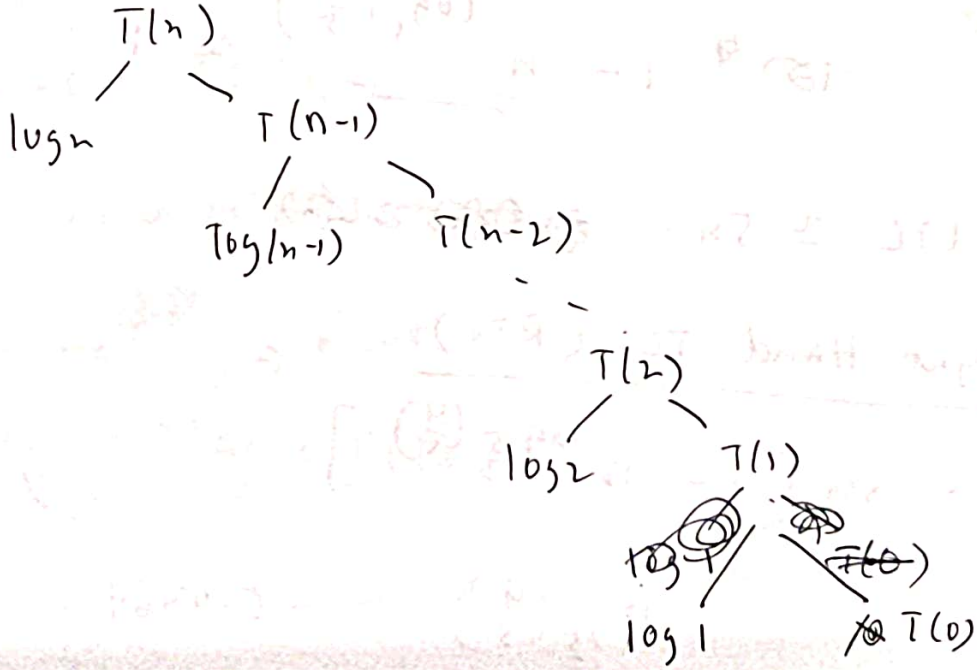
ex 4)

```
void Test(int n)
{
    if (n>0)
    {
        for(i=1; i<n; i=i*2)
        {
            print(i)
        }
        Test(n-1)
    }
}
```

$\log_2 n$

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

$$T(n) = \begin{cases} 1, & n = 0 \\ T(n-1) + \log n, & n > 0 \end{cases}$$



157

$\log 1 + \log 2 + \cdots + \log n$.

$\log(2 \times 3 \times \cdots n) = \log n!$

$T(n) = \log n!$

$1 < \log n < \sqrt{n} < n < n \log n < n^2 < n^2 \log n < n^3$
$\cdots < 2^n < 3^n < n^n$.

$\log 1 \leq \log(1 \times 2 \times 3 \cdots \times n) \leq \log(n \times n \times \cdots n \text{ times})$

$\Rightarrow \log n^n$.

$\Omega(1) \rightarrow$ lower bound $\Rightarrow n \log n$.

$O(n \log n) \rightarrow$ upper bound.

There is no Θ for this.

$\therefore \boxed{T(n) = O(n \log n)}$.

Masters theorem for Decreasing functions

Type-1

Set 1

$T(n) = T(n-1) + 1$ — $O(n)$

$T(n) = T(n-1) + n$ — $O(n^2)$

$T(n) = T(n-1) + \log n$ — $O(n \log n)$

Set 2

$T(n) = 2T(n-1) + 1$ — $O(2^n)$

$T(n) = 3T(n-1) + 1$ — $O(3^n)$

$T(n) = 2T(n-1) + n$ — $O(n \cdot 2^n)$.

General form

$T(n) = aT(n-b) + f(n)$ → Look for definition (next pg)

$a > 0$, $b > 0$, $f(n) = O(n^k)$ where $k \geq 0$

Let us see how set 1 complies with it

For set 1, $a = 1$ $b = 1$, $f(n)$ → 1, n, log n

In general way, the time complexity = $O(n \times f(n))$ as a result.

(coming from $T(n-1)$.)

Question is how...

$O(n^k) = \log n$ (set 1, 3rd example)

well $\log n = O(n^k)$ indeed for $k > 0$

Def

a = no. of subproblems

b = Amt by which the problem decreases

$f(n)$ = cost of the work done outside of recursive calls

There are 3 cases here.

If $a = 1 \Rightarrow O(n \times f(n)) = O(n \times n^k) = \underline{O(n^{k+1})}$.

$a > 1 \Rightarrow O(a^n \times f(n))$.

$= O(a^n \times n^k)$.

Special case : if $b > 1$

$= O(a^{\frac{n}{b}} \times n^k)$.

$0 < a < 1 \Rightarrow O(f(n)) = O(n^k)$.

This is masters theorem.

- Type 1.

Decrement Types

Ex:

- $(n-b)$ / $(n-1)$
- $\frac{n}{b}$ / $\frac{n}{2}$
- $\sqrt[b]{n}$ / $\sqrt{n}$.

Divide & Conquer

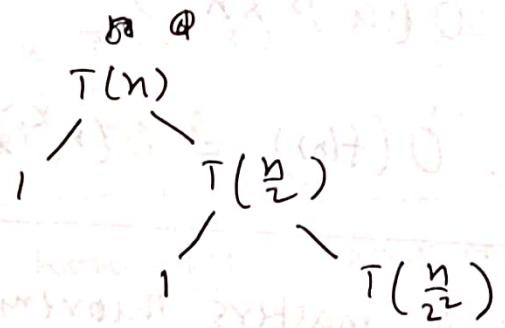
(1) Algorithm (Test n).

```
{
    if (n>1)
    {
        printf ("%d", n)
        Test(n/2)
    }
}
```

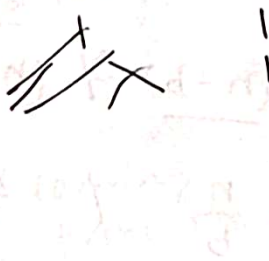
Recurrence relation

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

(a) Solve it using Tree method.



or else we can write the tree in "Time taken" format



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

$k = \log_2 n$

$$\sum_{i=0}^{\log_2 n} 1 = \log_2 n + 1$$

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

160

(1b) Back Substitution method

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

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

$$\vdots$$

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

let $T\left(\frac{n}{2^k}\right) = T(1)$. $\Rightarrow \frac{n}{2^k} = 1$

$\Rightarrow k = \log_2 n$.

$$= 1 + \log_2 n.$$

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

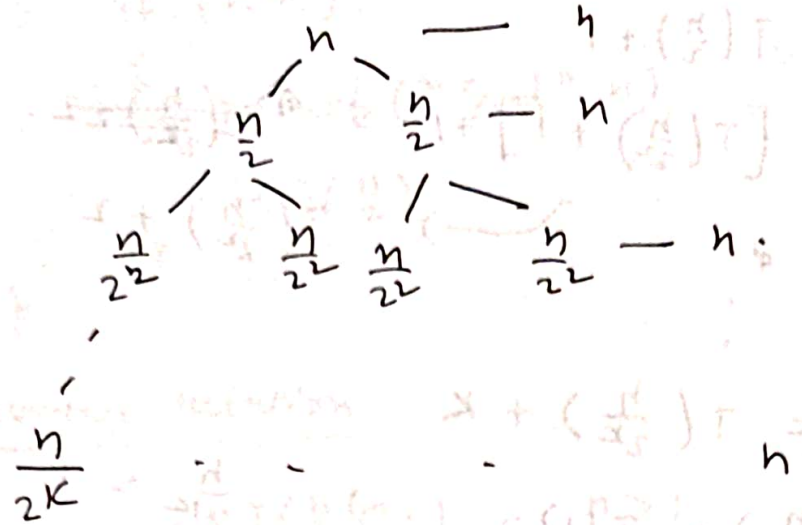
---

(2)

```
Void Test(int n).
{
  if (n>1).
  {
    for(i=0; i<n; i++)
    {
      stmt;
    }
    Test(n/2)
    Test(n/2)
  }
}
```

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

(6) (a) Solve the recurrence relation using Tree method.



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

$n + n + \cdots$ K times.

$= n \cdot K = n \cdot \log_2 n.$

$\therefore \boxed{T(n) = O(n \log n).}$

(b) Solve the relation using back sub. ~~recurrence~~ ~~root~~ method.

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

$T(n) = 2T(\frac{n}{2}) + n$ — (1)

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

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

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

$= 2^3 T(\frac{n}{2^3}) + n + 2n = 2^3 T_1(\frac{n}{2^3}) + 3n.$

(162)

$$T(n) = 2^K T\left(\frac{n}{2^K}\right) + Kn.$$

Now, let $T\left(\frac{n}{2^K}\right) = T(1)$.

$\Rightarrow K = \log_2 n \qquad 2^K = n.$

$$T(n) = n + n \cdot \log_2 n.$$

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

---

(3)

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

Function $T(n)$:

if $n <= 2$

return 1

else:

print("%d", n)

$T(\sqrt{n})$.

$$T(n) = T(n^{\frac{1}{2}}) + 1, \quad n > 2 \qquad -(1)$$

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

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

$$= T(n^{\frac{1}{2^K}}) + K \quad \cdots (4)$$

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

$$\frac{1}{2^k} = \log_n 2$$

$$2^k = \log_2 n$$

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

$$T(n) = k = \log_2 \log_2 n.$$

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

Summary of all results

		Time
①	$T(n) = \begin{cases} T(\frac{n}{2}) + 1, & n > 1 \\ 1, & n = 1 \end{cases}$	$O(\log n)$
②	$T(n) = \begin{cases} 2T(\frac{n}{2}) + n, & n > 1 \\ 1, & n = 1 \end{cases}$	$O(n \log n)$
③	$T(n) = \begin{cases} T(\sqrt{n}) + 1, & n > 2 \\ 1, & n \leq 2 \end{cases}$	$O(\log \log n)$

Masters Theorem - Divide & Conquer

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

$a \geq 1 \qquad b > 1 \qquad f(n) = \Theta(n^k \log^p n)$.

a = Subproblems $\qquad$ b = Amt by which you are decrementing.

Steps

1. $\log_b a$
2. k, p.

Rules

case 1:

If $\log_b a > k \longrightarrow \Theta(n^{\log_b a})$.

case 2:

If $\log_b a = k$

(a) If $p > -1 \longrightarrow \Theta(n^k \log^{p+1} n)$.

(b) If $p = -1 \longrightarrow \Theta(n^k \log \log n)$.

(c) If $p < -1 \longrightarrow \Theta(n^k)$.

case 3:

If $\log_b a < k$

(a) If $p \geq 0 \longrightarrow \Theta(n^k \log^p n)$.

(b) If $p < 0 \longrightarrow \Theta(n^k)$.

(165) Let us take some recurrence relations that satisfies case-1

① $T(n) = 2T(\frac{n}{2}) + 1$

$a = 2 \quad b = 2 \quad \& \quad f(n) = \theta(n^k \log^p n)$

$k = 0 \quad p = 0$

$\log_b a = \log_2 2 = 1$

$\therefore \log_b a > k. \dashrightarrow$ falls into case-1

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

Try solving that using back substitution

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

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

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

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

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

$\vdots$

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

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

$2^k = n$

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

$= n + n - 1 = 2n - 1$

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

ex. 3 $T(n) = 4T(\frac{n}{2}) + n$.

$a = 4 \quad b = 2 \quad k = 1 \quad p = 0$.

$\log_b a = \log_2 4 = 2$

$\log_b a > k$

$\therefore \quad T(n) = \theta(n^{\log_b a})$.

$= \theta(n^2)$.

---

ex. 4) $T(n) = 8T(\frac{n}{2}) + n$.

$a = 8 \quad b = 2 \quad k = 1 \quad p = 0$.

$\log_b a = \log_2 8 = 3$.

$\therefore \log_b a > k$. $\rightarrow \theta(n^{\log_b a})$

$\theta(n^3)$.

---

ex. 5) $T(n) = 9T(\frac{n}{3}) + 1$

$a = 9 \quad b = 3 \quad k = 0 \quad p = 0$

$\log_b a = 2 \rightarrow \theta(n^2)$.

Examples for case - 2

ex. 6 $T(n) = 2T(\frac{n}{2}) + n$.

$a = 2 \quad b = 2 \quad k = 1 \quad p = 0$

$\log_b a = 1$

Here $\boxed{\log_b a = k}$ $\rightarrow \theta(n \cdot \log n)$

If $p > -1 \rightarrow \theta(n^k \log^{p+1} n)$.

If $p = -1 \rightarrow \theta(n^k \log \log n)$

If $p < -1 \rightarrow \theta(n^k)$

(Ex-7) $T(n) = 8\,T(\frac{n}{2}) + n^3$.

$a = 8 \quad b = 2 \quad k = 3 \quad p = 0$.

$\log_b a = 3$.

$\log_b a = k$

$\theta(n^k \log^{p+1} n)$.

$= \theta(n^3 \log n)$.

Ex-8 $T(n) = 2\,T(\frac{n}{2}) + \frac{n}{\log n}$.

$a = 2 \quad b = 2 \quad k = 1 \quad p = -1$

$\log_b a = 1$

$T(n) = \theta(n^k \log \log n)$.

$= \theta(n \cdot \log \log n)$

Case-3.

Ex-9 $T(n) = T(\frac{n}{2}) + n^2$

$a = 1 \quad b = 2 \quad k = 2 \quad p = 0$.

$\log_b a = 0$

If $p \geq 0$ $\theta(n^k \log^p n)$

$p < 0$ $\theta(n^k)$

$T(n) = \theta(n^2)$

Next Topics

1. Root function
2. Binary search
3. Heap sort
4. Merge sort
5. Quick sort
6. Insertion sort — study separately.
7. Strassen matrix multiplication
8. Greedy method
9. Knapsack Problem
10. Job sequencing - Greedy method
11. ~~Optical merge~~ Optimal merge
12. Huffman coding
13. Prims
14. Kruskals
15. Dijkstra

(DS plan)

- Arrays
- Linked List - Single, Doubles
- ~~Graphs~~
- O

Root function

Algorithm:

```
void Test (int n)
{
    if (n > 2)
    {
        stmt;
        Test(√n)
    }
}
```

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

Solve it using recurrence relation

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

$\rightarrow = [T(n^{\frac{1}{2^2}}) + (1+1)]$ — (2)

$\rightarrow = T(n^{\frac{1}{2^3}}) + (1+1+1)$ — (3)

$\vdots$

$= T(n^{\frac{1}{2^k}}) + \underbrace{(1+1+1+\cdots \text{K times})}_{K}$

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

$\frac{1}{2^k} = \log_n 2$

$2^k = \log_2 n$

$K = \log_2 \log_2 n$

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

~~$T(n) = \Theta(\log_2 \log$~~

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

170

Now how root function works?

This wasn't required unless there is a theorem

Assume $n = 2^m$ -> because $n \to n^{\frac{1}{2}} \to n^{\frac{1}{2^2}} \cdots$

~~$T(2^m)$~~ Then how to write

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

$T(2^m) = T(2^{\frac{m}{2^k}}) + k.$

Assume, $2^{\frac{m}{2^k}} = 2$

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

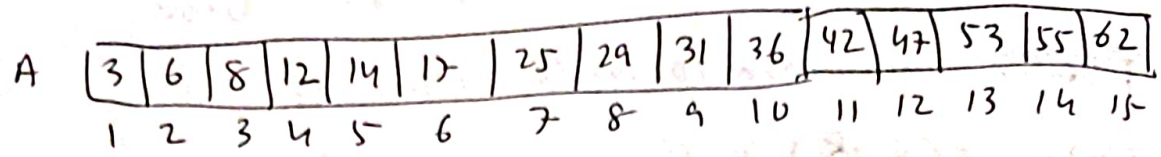
$k = \log_2 \log_2 n.$

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

$\Theta(\log \log n)$

Binary Search - Iterative

- Binary search follows → Divide & conquer strategy.



Rules

① List must be in the sorted order.

② low = 1 (static)
high = 15 (dynamic)

Points

let say key = 42

l h $mid = \left[\frac{l+h}{2}\right]$ ← floor value

$\lfloor x \rfloor$ = largest integer $\leq x$.

Ex	$\lfloor x \rfloor$
7.9	7
-3.2	-4

Step #	l	h	$mid = \left\lfloor \frac{l+h}{2} \right\rfloor$
1	1	15	$\frac{1+15}{2} = 8$

check A[8] == 42 ? No

(29) ← A[8] < 42

∴ ~~value 42~~ Key = 42 is on the right hand side.

Why? ← The list is sorted.

(172)

Step #	l	h	mid = $\lfloor \frac{l+h}{2} \rfloor$
2>	9	15	$\lfloor \frac{9+15}{2} \rfloor = 12$

$l = (mid + 1)$

Note:- If we had to search on the lower side.

l = remained intact $h = mid - 1$.

A[12] == 42 ? NO.

42 < A[12] (47)

3> 9 11 $\lfloor \frac{9+11}{2} \rfloor = 10$

$h = mid - 1$

~~(intact)~~

~~A[10]~~

A[10] = 36 == 42 ? NO.

42 > A[10]

4> $l = mid + 1$

11 11 $\lfloor \frac{11+11}{2} \rfloor = 11$

A[11] == 42

~~stop!~~ found.

Step #	gap = h - l + 1
1	h - l + 1 = 15
2	15 - 9 + 1 = 7
3	11 - 9 + 1 = 3
4	11 - 11 + 1 = 1

narrowing down.

How many comparisons = 4

$2^K = 15$

$K \approx \log_2 2^4 = 4$.

For Linear search → 15 comparisons

Binary search → 4 comparisons

---

ex:-2) With the same array

Key = 12

Step#	l	h	mid = $\lfloor \frac{l+h}{2} \rfloor$
1>	1	15	$\frac{1+15}{2} = 8$
			12 < A[8]
2>	1	7 (8-1)	$\frac{1+7}{2} = 4$
			12 = = A[4]
			Key found.

---

ex:-3 Let us search for any element that is not present in the list

Key = 30

Step#	l	h	mid = $\lfloor \frac{l+h}{2} \rfloor$	Comparison
1	1	15	$\frac{1+15}{2} = 8$	A[8] = 29 29 < 30
2	9 (8+1)	15	$\frac{9+15}{2} = 12$	A[12] = 47 30 < 47
3)	9	11	$\frac{9+11}{2} = 10$	A[10] = 36 30 < 36
4)	9	9	~~$\frac{9+9}{2} = 9$~~	~~A[9] = 42~~ ~~42 ≠ 30~~ A[9] = 31

Note:-

- Algorithm must run as long as ~~low <= high~~ low <= high.

∴ ~~at low = = high~~

~~high~~.

- At low = high, if the ~~A[mid] ≠ Key~~ exit.

5) 9 8 ✗ Element is not found.

Another observation

. By the time the algorithm exits / ends. A[mid] comes closest to Key value among all other values.

Algorithm : BINARY SEARCH. (Iterative Process).

① First teach algorithm → code.

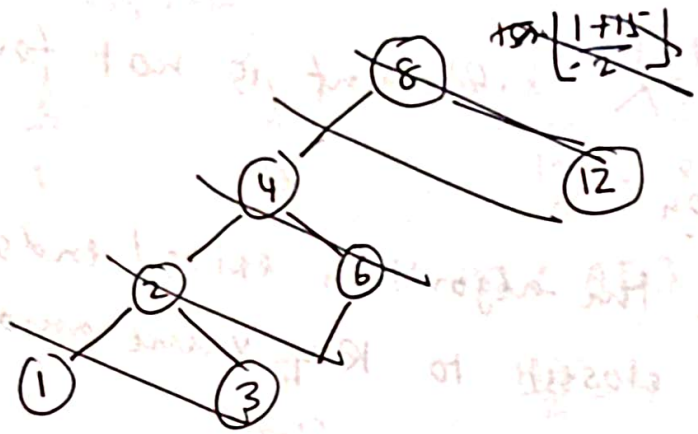
```
def binary_search (arr, key):
    low = 0
    high = len(arr) - 1
    while (low <= high)
    { ① Find mid = (low + high) // 2
      ② if (key == arr[mid])
            return mid  // Element is found

      ② elif key > A[mid]
            # Key is on the right
            low = mid + 1

      ③ else:
            # Key is on the left
            high = mid - 1
    }
    return (-1)
```

Now let's trace the algorithm - with a help of a tree

Plot the mid points.



generate the tree with the following rule

(1) left < node ; right > node.

node

left right

(2) Each node = $\left\lfloor \frac{low + high}{2} \right\rfloor$

~~(low + mid) // 2~~ (low + high) // 2

(3) Once a node is listed, we can't use it for - low or - high.

~~1 2 3 4 5 6~~

~~1 2 3 4 5 6 7 8 9~~ 10 11 12 13 14

root node = $\left\lfloor \frac{1+15}{2} \right\rfloor = 8$

176

(1) (8)

Left side of 8.

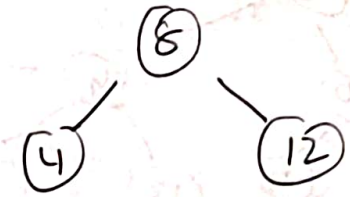
low = 1 high = 7

$\left\lfloor \frac{low+high}{2} \right\rfloor = \left\lfloor \frac{1+7}{2} \right\rfloor = 4$

right side of 8.

low = 9 high = 15

$\left\lfloor \frac{low+high}{2} \right\rfloor = \left\lfloor \frac{9+15}{2} \right\rfloor = \left\lfloor \frac{24}{2} \right\rfloor = 12$



(2) For node = 4

Left side of 4

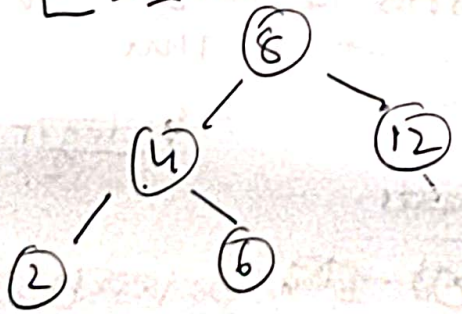
low = 1 high = 3

$\left\lfloor \frac{1+3}{2} \right\rfloor = 2$

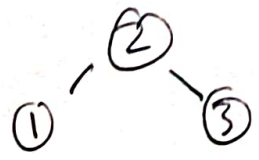
right side of 4.
up to 7

low = 5 high = 7

$\left\lfloor \frac{5+7}{2} \right\rfloor = 6.$

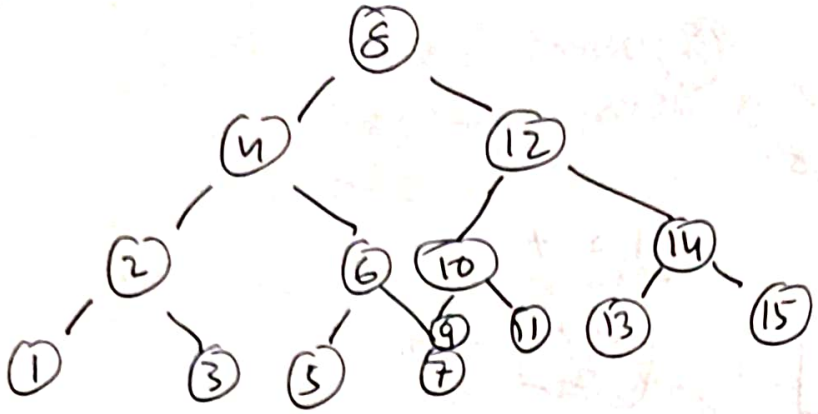


(3) For node = 2



Left	Right
low = 1	low = 3
high = 1	high = 3

177



Left of 12

low = 9 high = 11

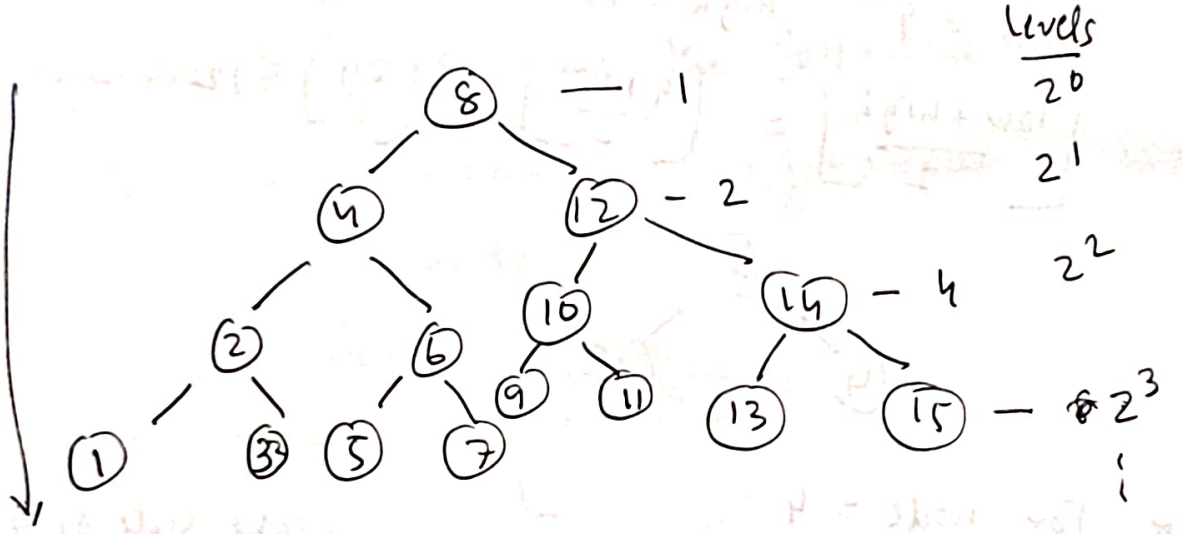
$\left\lfloor \frac{9+11}{2} \right\rfloor = 10$

right of 12

low = 13

high = 15

$\frac{13+15}{2} > 14$



Binary Search Tree

- maximum comparisons = height of the tree

(178)

Level	elements	representation
1	1	2^0
2	4	2^2
3	8	2^3
.	.	.
.	.	.
K		2^{K-1}

$$S_n = 2^0 + 2^2 + 2^3 + \cdots + 2^{K-1} = N.$$

$$\therefore \frac{1(2^K - 1)}{2-1} = N$$

$$2^K - 1 = N$$

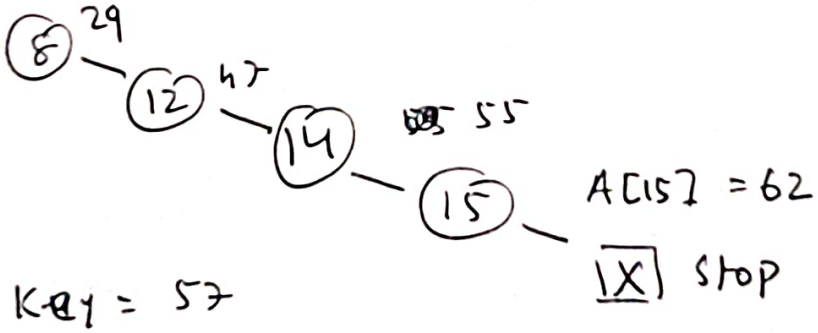
$$2^K = N+1$$

$$\boxed{K = \log_2 (N+1).}$$

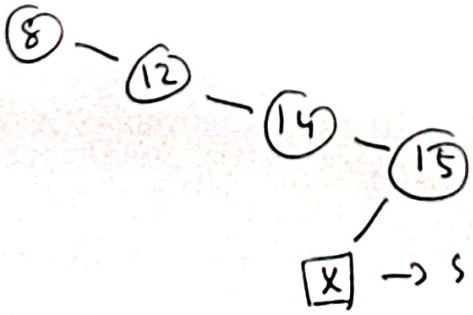
$$\boxed{K = O(\log n).}$$

n = no. of elements.
K = height of the tree.

Let say Key = 65 what is the traversal route? where it will stop?



Let say Key = 57



- For unsuccessful search = any. 4 comparison
  $\log N$.

- min time — 1 — $\Omega(1)$ — best case
  max time — $O(\log N)$. — worst case

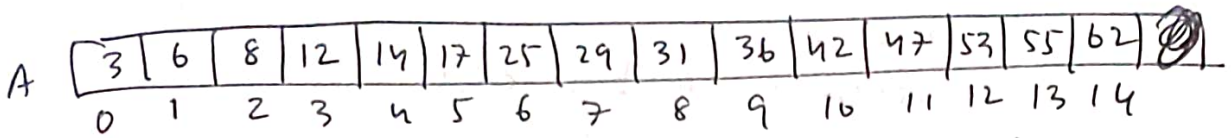
max time

element present in the leaf

unsuccessful attempt.

Avg time = $\frac{\text{Add time for each \& every elem}}{N}$.

Binary Search - Recursive




```
def r_binary_search (arr, target, low, high):
    if low > high:
        return (-1)    # Base case: target not found
    mid = (low + high) // 2
    if arr[mid] == target:
        return mid     # Target found.
    elif arr[mid] > target
        return r_binary_search (arr, target, low, mid-1).
    else:
        return r_binary_search (arr, target, mid+1, high).
```

Code Tracing available in algorithm.ipynb file > GitHub

Recursive algorithm:

```
def fact(n):
    if n == 0 or n == 1:
        return 1    # base case - exit clause
    else:
        return n * fact (n-1)    # recursive call.
```

181)

```
def fib(n):
    if n <= 0
        return 0    # base case

    elif n == 1
        return 1    # base case

    else:
        return fib(n-1) + fib(n-2)
```