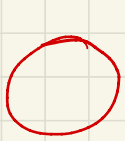
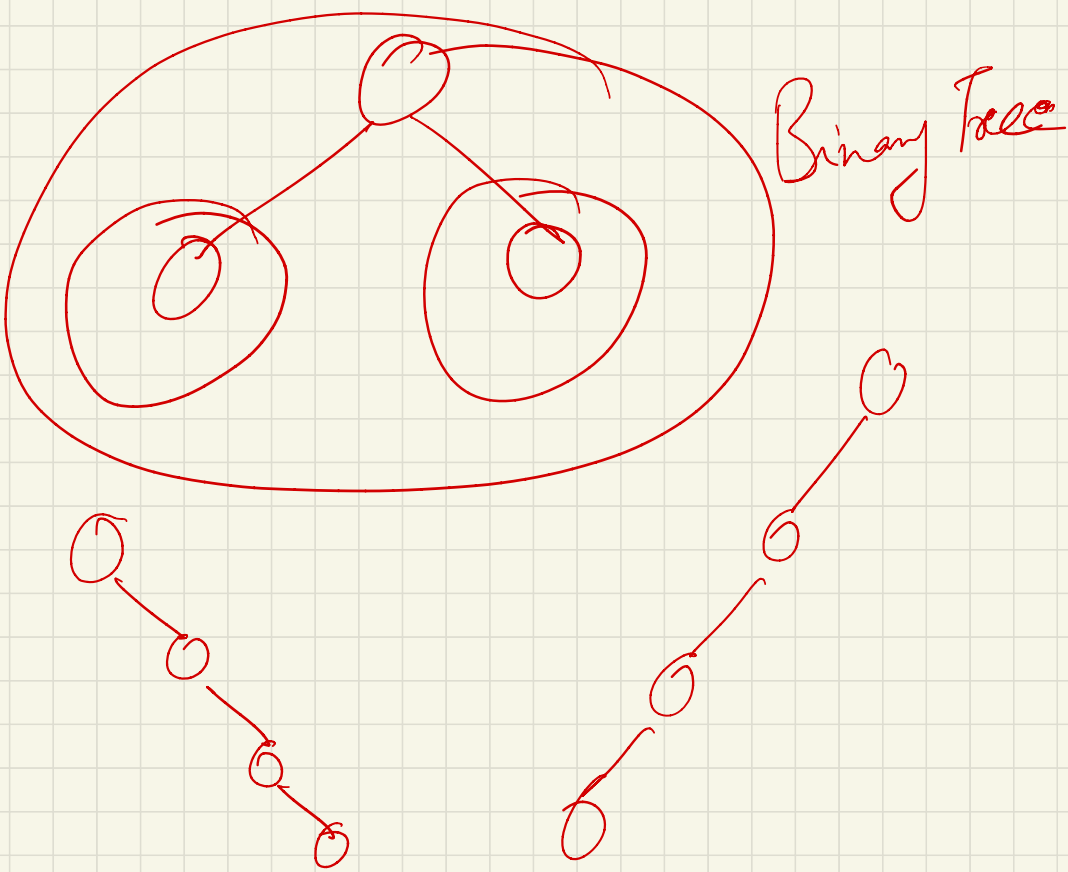
DS & ALGO - 12 Jul 20.

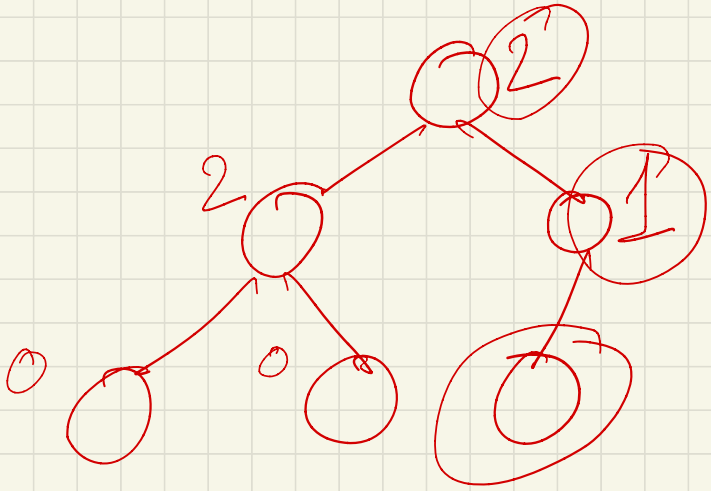
Trees.



Is it a binary tree ?

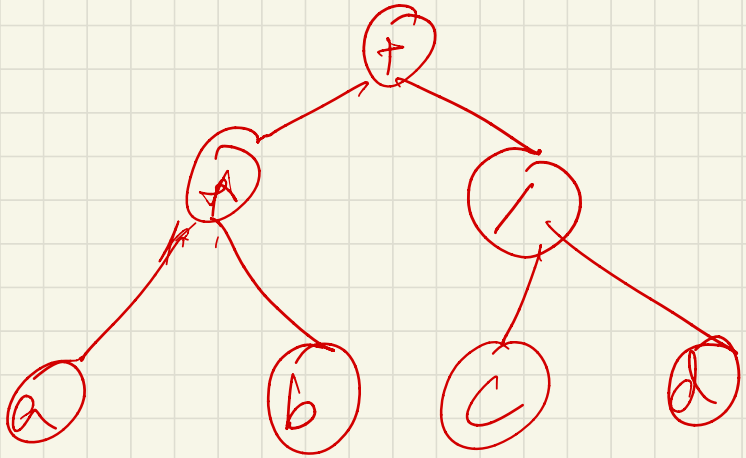
— Yes.





0 → 1 → 2

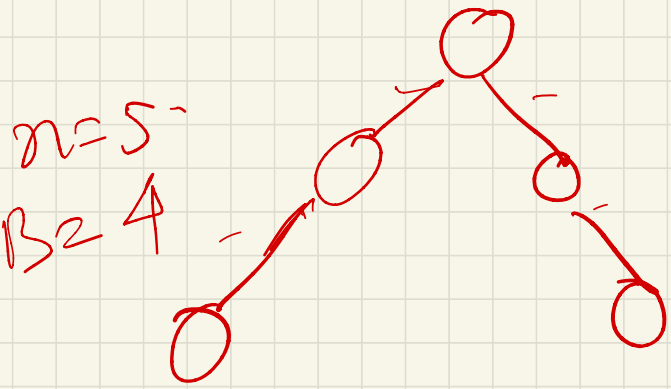
0, 1, 2 → children ..

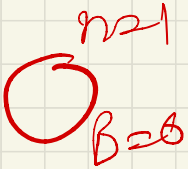


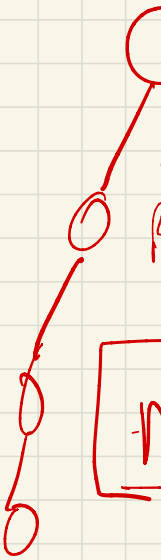
(a*b) + (c/d)

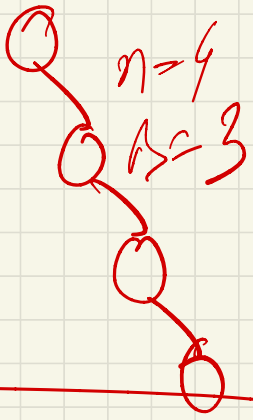
Properties of Binary Tree.

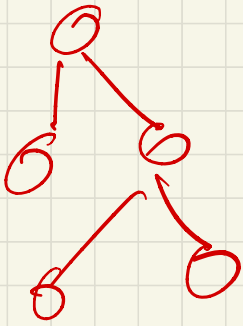
1.









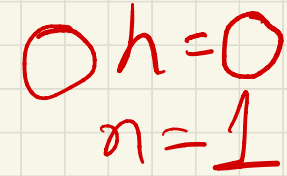


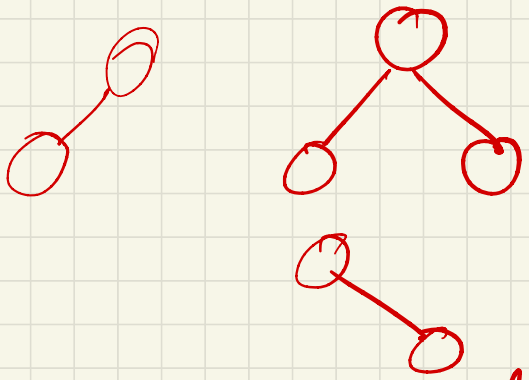
No of Branches = n - 1.

2. height = 2., h = 0 ✓

a) Minimum no. of elements.

$n = h + 1$.





h = 1.

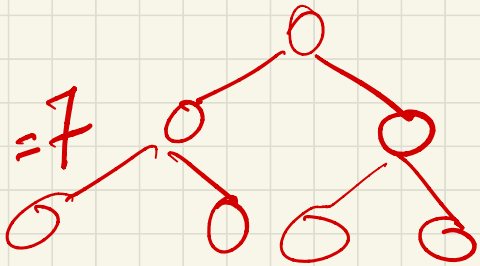
$n = h + 1 = 2$.

b) Maximum no. of elements.

For height h, max = $2^{h+1} - 1$.

h = 2

$n = 2^3 - 1 = 7$



h = 1.

$n = 2^{1+1} - 1$

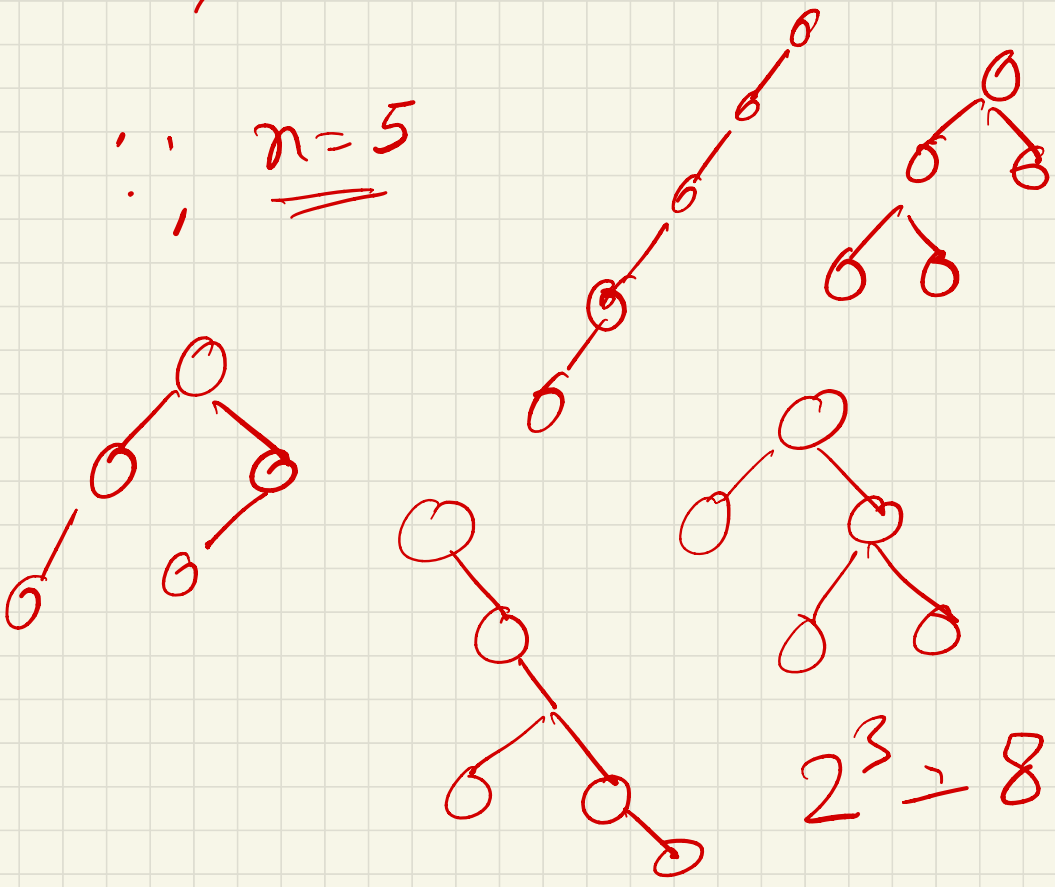
$= 3$

3) no. of element, n.

$$h_{min} = \lceil \log_2(n+1) \rceil - 1.$$

$$h_{max} = n - 1.$$

$\therefore n = 5$



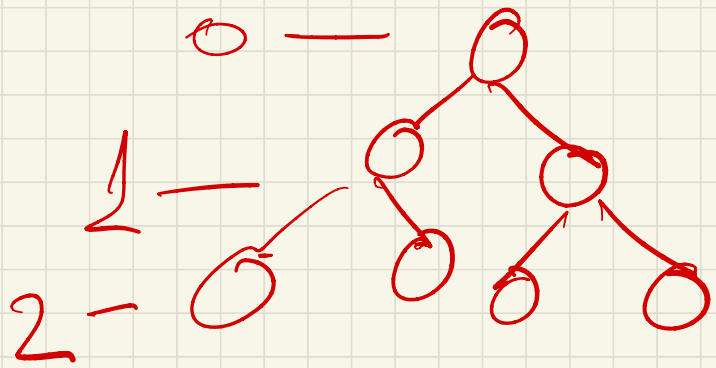
$2^3 = 8$.

$\log_2 8 = 3$

$n = 7$.

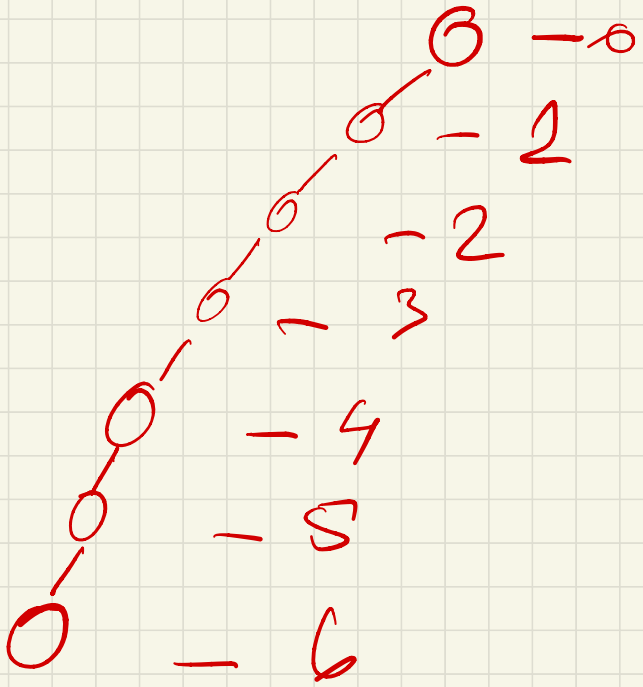
$h_{min} = \log 8 - 1$.

$= 2$.

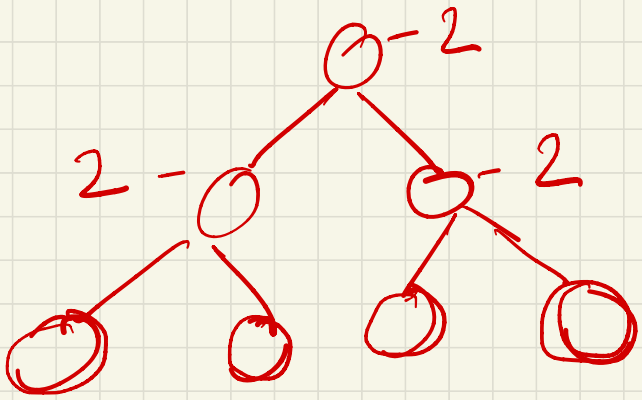


$h = 2$.

$h_{max} = 7 - 1 = 6$.



4)

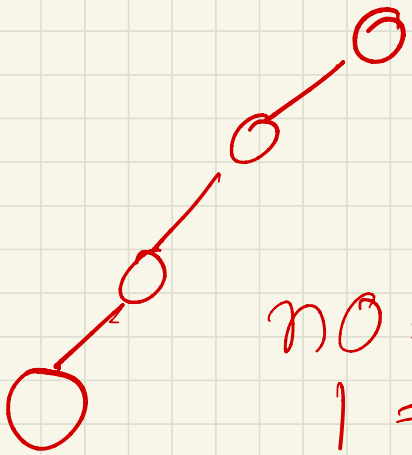


$n_0 = 4$. no node degree (2)

$= 3$.

$n_0 = n_2 + 1$

$4 = 3 + 1$.



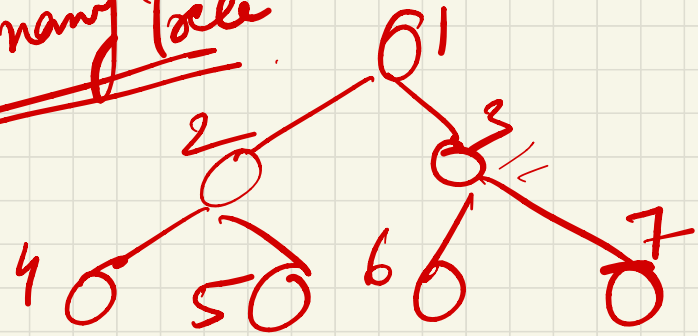
$n_0 = 1$.

$n_2 = 0$.

$n_0 = n_2 + 1$.

$1 = 0 + 1$

Full Binary Tree



node (i) ⇒ node 4 → Parent ?

$\lfloor i/2 \rfloor = 2.$

node (5) = $\lfloor i/2 \rfloor = 2.$

node (1) = $\lfloor 1/2 \rfloor = 0.$

Root.

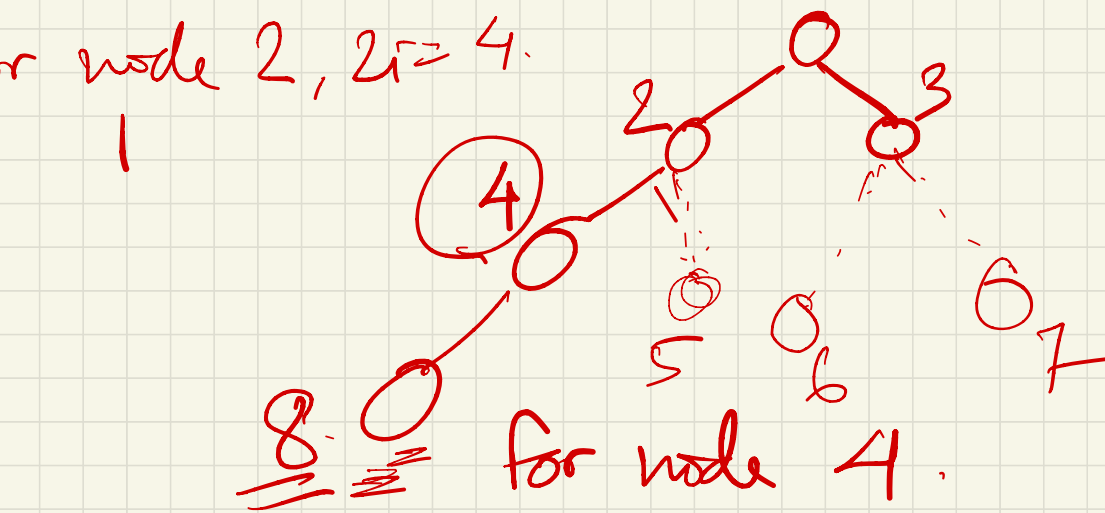
Left child.

$(2i)$

left child = 2i

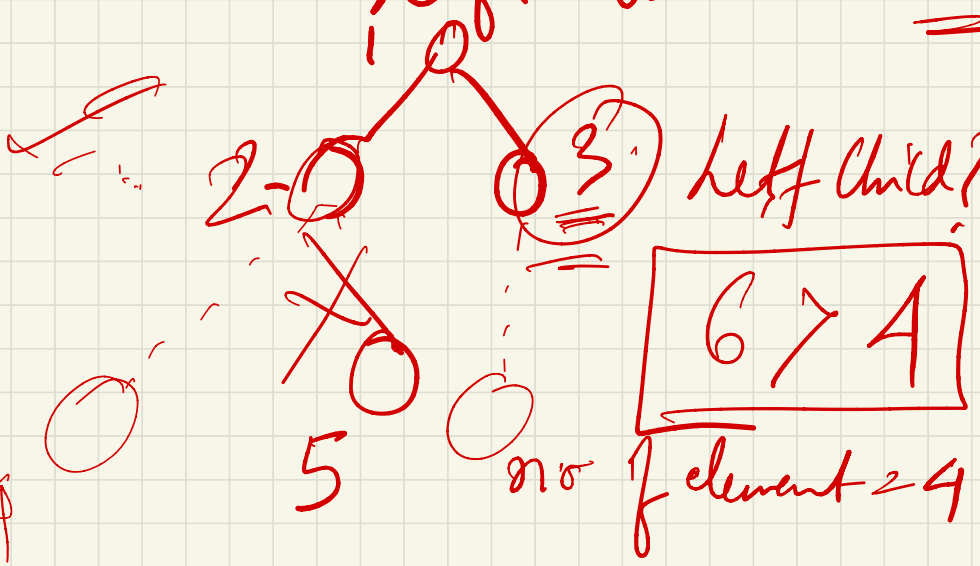
node 1, 2i = 2.

for node 2, 2i = 4.



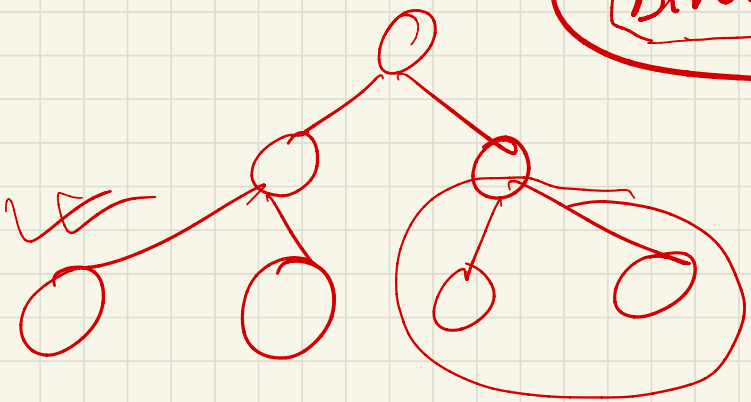
for node 4.

left child = 8

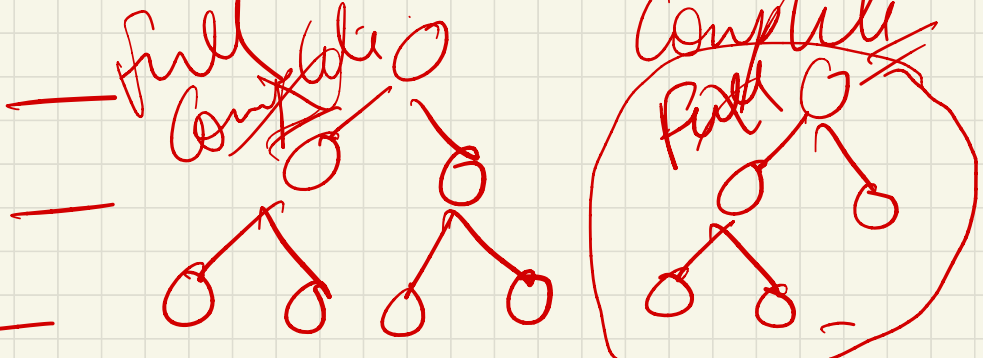


if $2i > n$, node i has no left child.

Full Binary Tree, Complete Binary Tree



Right child $\rightarrow 2i+1$.



Data Structure with C - Lipschutz

Representation — Array (X)
— LL. - LL.

TREE TRAVERSAL

BFS, DFS.

- Graph
- Tree
  - Preorder
  - Inorder
  - Postorder
  - level order

19 Jul 20 . Trees .

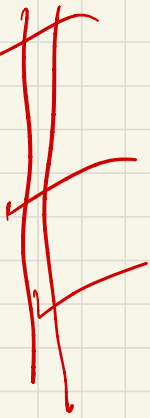
// Traversal — Pre order
— Inorder
— Post order .

Order is always in respect of root .

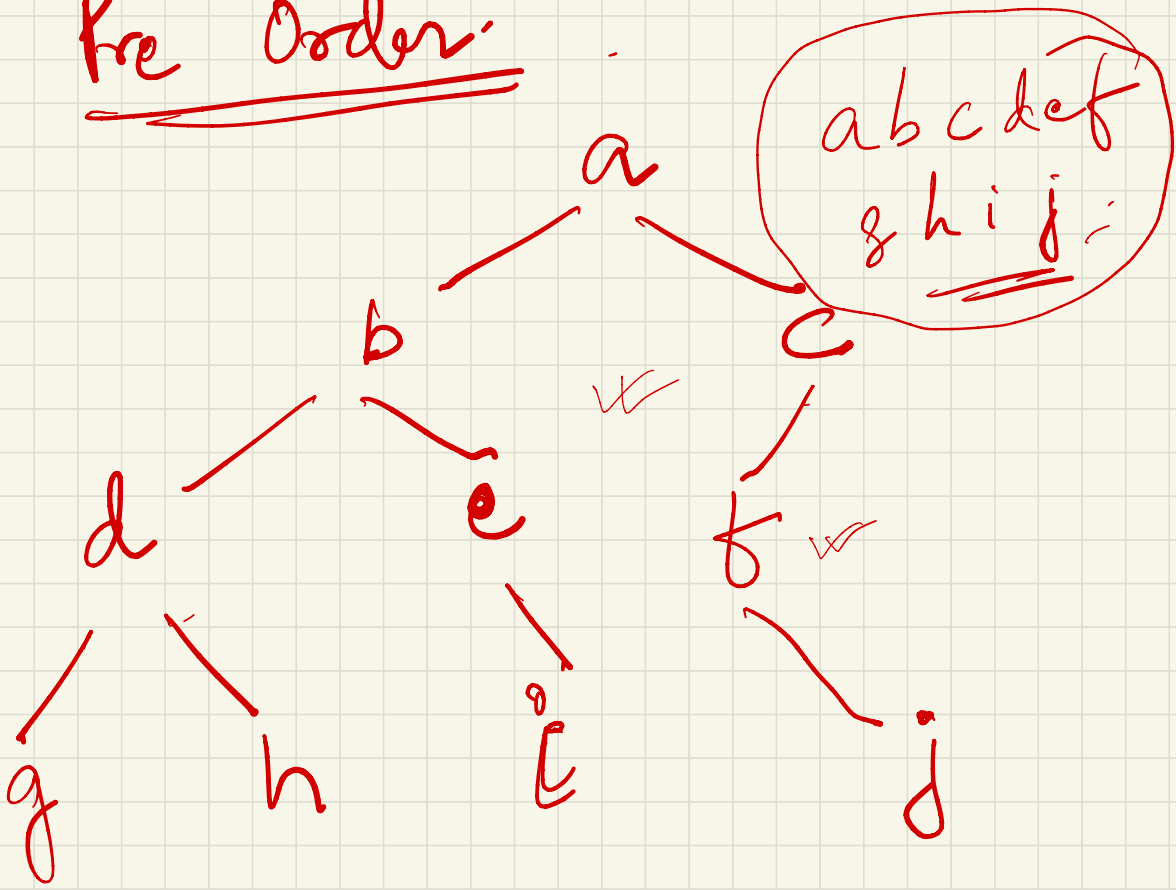
Pre - root , left, right .

In - left, root, right .

Post - left, right, root .



Pre Order:



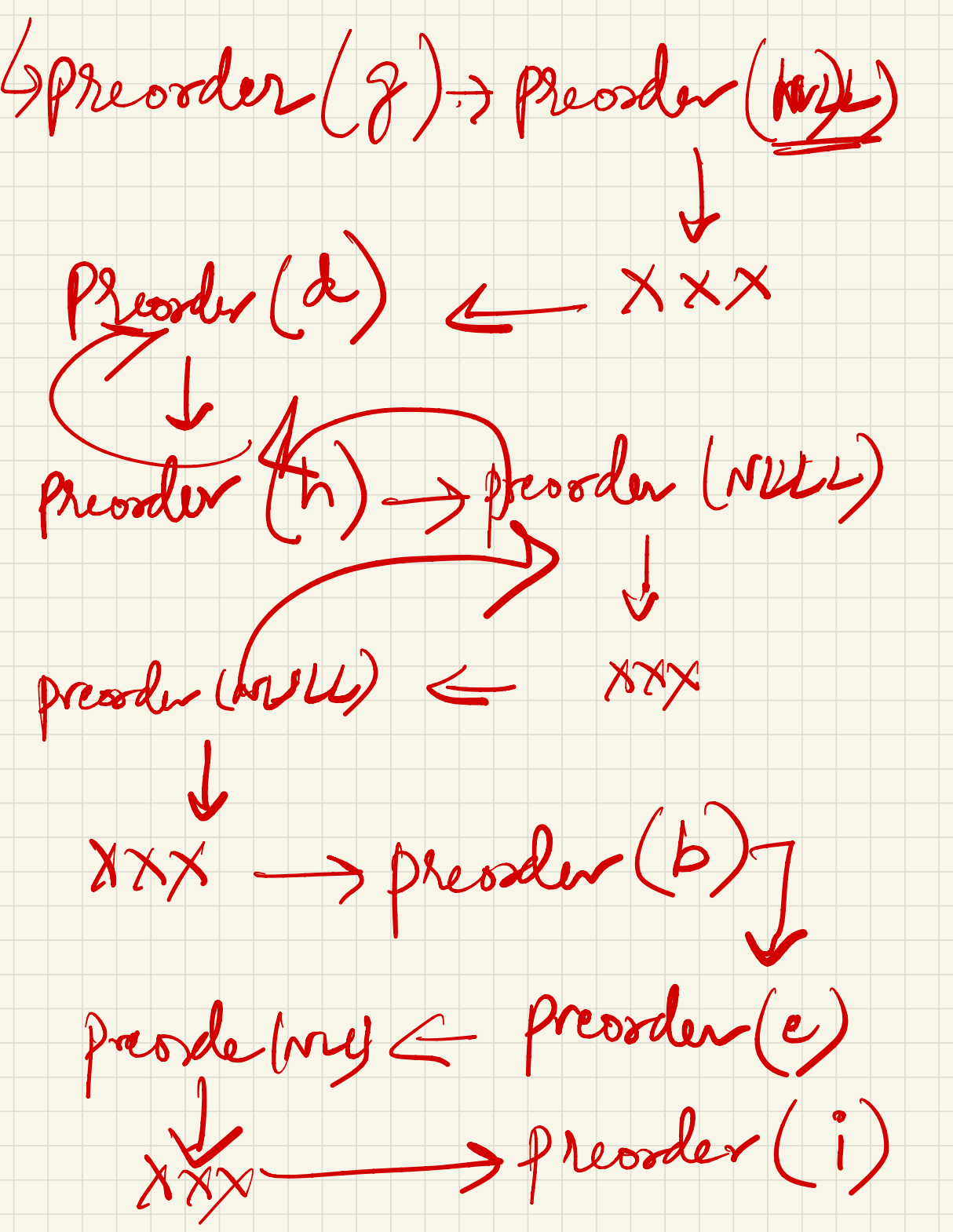
Output → a b d g h e i c f j

Pre order (b)

Preordr (d)

Preordr (g)

preordr (NULL)


preorder(g) → preorder(NULL)
XXX
preorder(d) ← XXX
preorder(h) → preorder(NULL)
XXX
preorder(NULL) ← XXX
XXX → preorder(b)
preorder(NULL) ← preorder(e)
XXX → preorder(i)

(NULL) — xxx

(NULL) — xxxx.

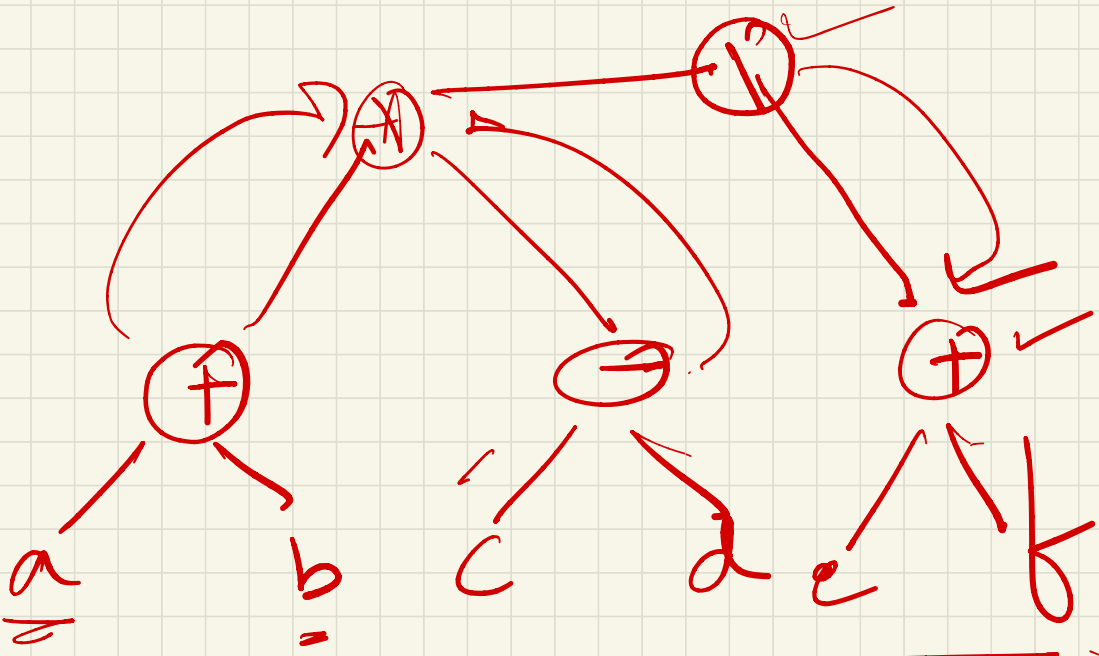
preorder(i) → preorder(c) → (b)

preorder(c) ← (a)

preorder(f) → NULL → xxx

preorder(g) ← (f)

Expression → ((a+b) * (c-d))/(e+f)

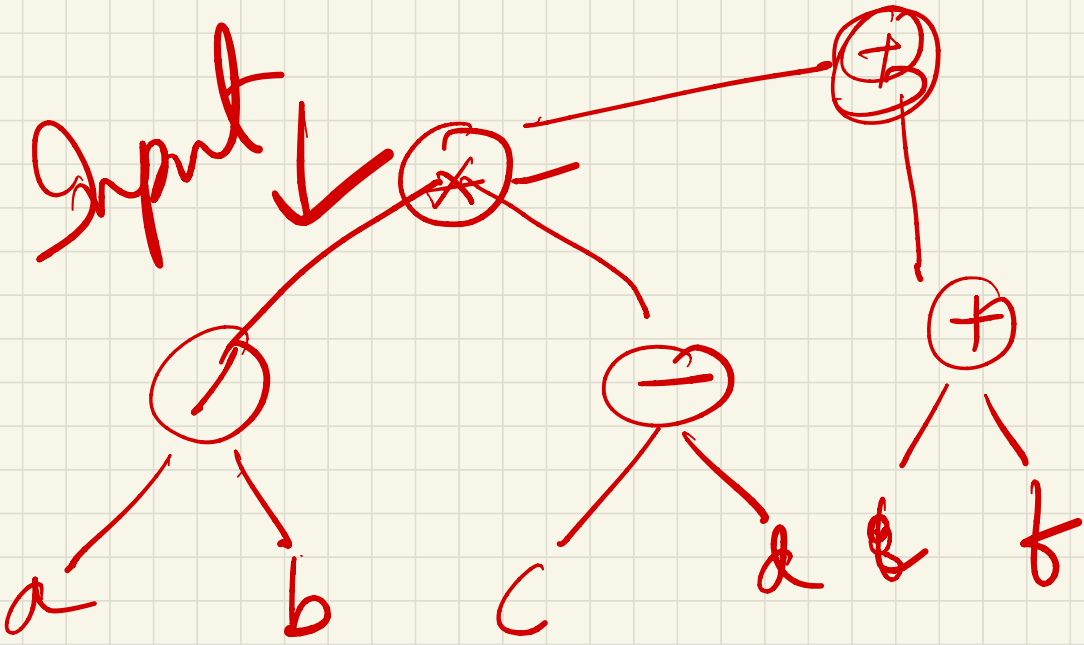


/ * + a b - c d + e f

Prefix form:

/ * + + - a b c d e f

$$((a/b)*(c-d))+(e+f))$$

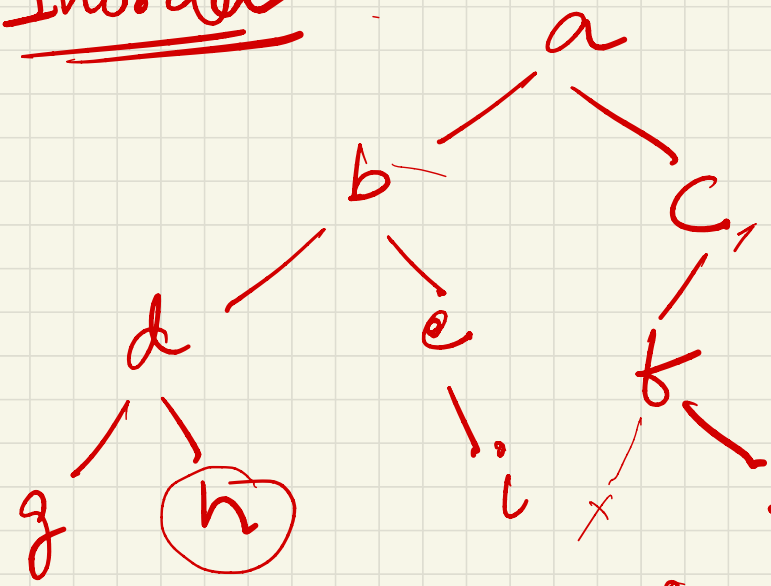


$$+*/ab-cd+ef$$

↳ prefix.

Expression Tree → prefix (with help of preorder)

Inorder:



Result :- g d h b e i a f j c

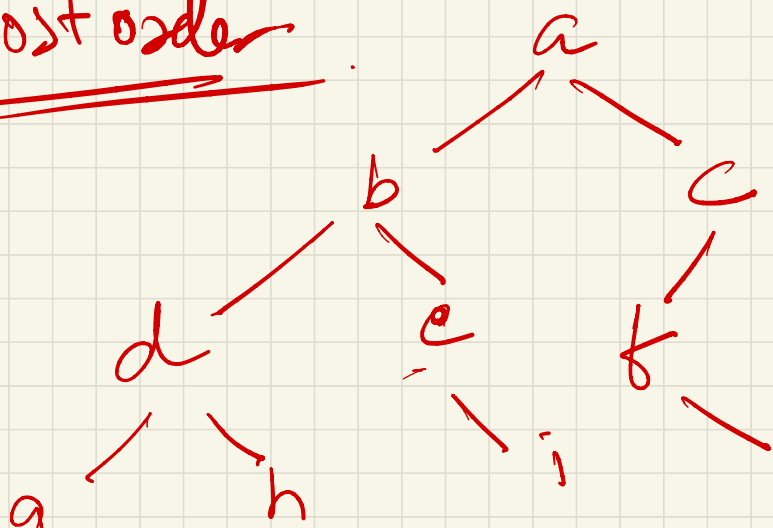
In(a) → In(b) → In(d) → In(g)

NULL ← print(g) ← XXX ← NULL

↓

XXX → In(d) → In(h)

Post order



Output :- g h d i e b j f c a

Result :- a → b → d → g

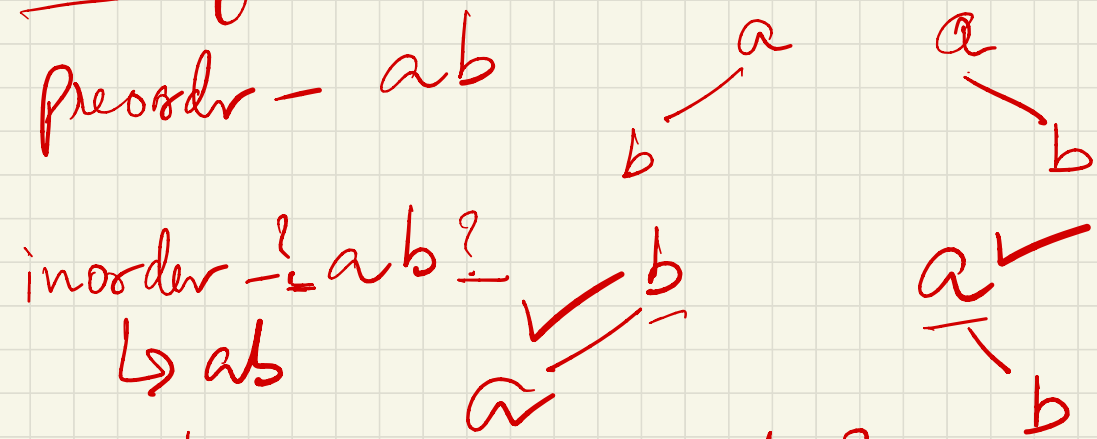
b ← d ← h ← d

→ e → i → e → b → a

a

↑ c ← f ← j ← f ← c

Binary Tree Construction



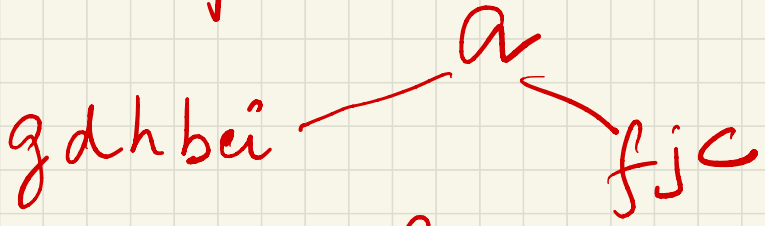
Inorder = g d h b e i @ f j c

Preorder = @ b d g h e i c f j

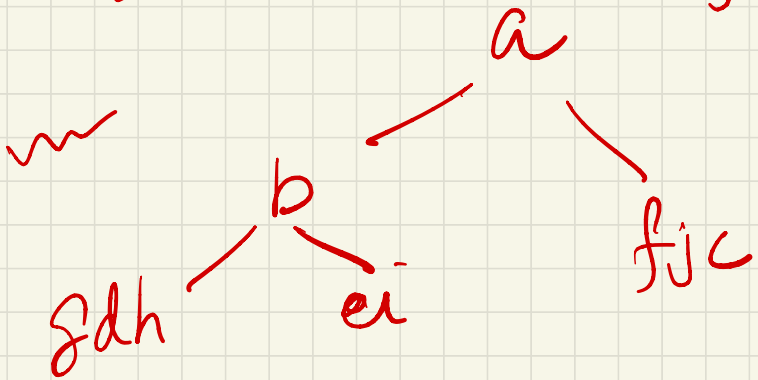
→ As per preorder → Root = a

→ Scan the Inorder; and divide the tree as per root found above.

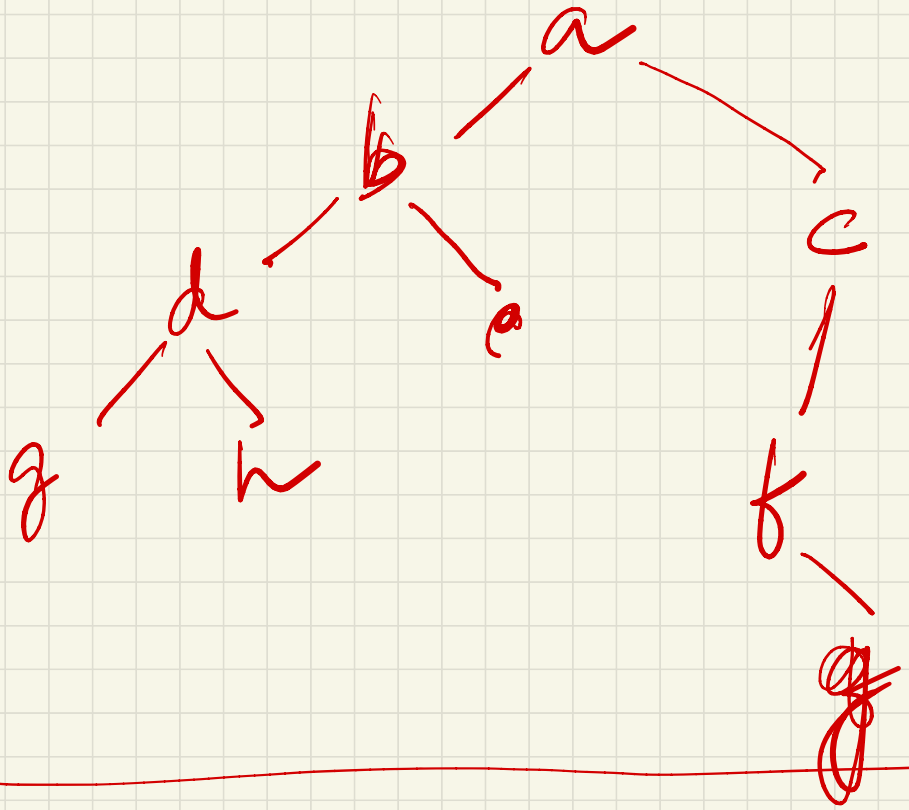
Inorder - left, Root, Right



✓ (1)



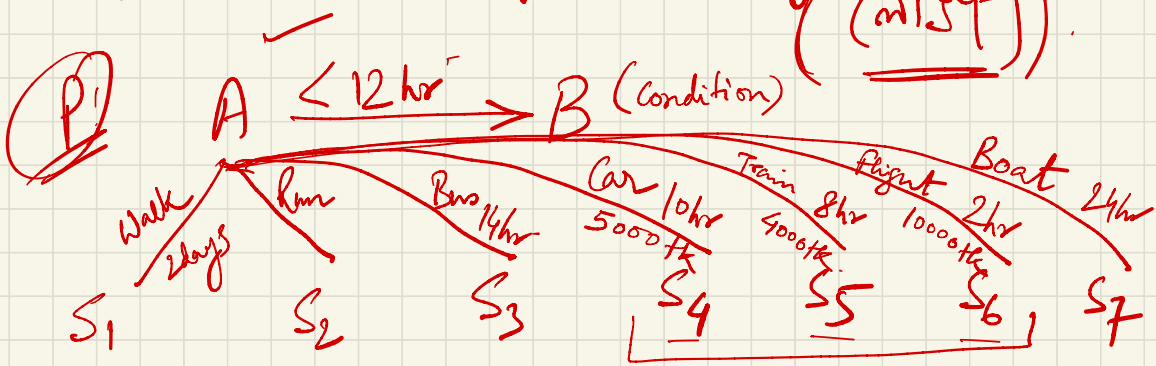
✓ (2)



13 Aug 20:

Greedy Method. (Algorithm technique)

(Control)



Out of 7 solutions, S_4, S_5 and S_6 meet the conditions.

$S_4, S_5, S_6 \rightarrow$ Feasible Solution ✓✓

Conditions. 1) 12 hr. 2) Minimum cost.

Out of S_4, S_5, S_6, S_5 has minimum cost.

$S_5 \rightarrow$ Optimal Solution. ✓✓

We can have multiple feasible solutions, but only one optimal solution

So, these type of problems are called optimization problems. ✓✓

Optimization problem (Either Minimum or Maximum).

1) Greedy Method. ✓ } Kruskal, Prim, Dijkstra, Bellman
2) Dynamic Programming. ✓ } MCM, LCS, 0/1 knap.
3) Branch & Bound. ✓ }

Greedy Method.

```
Algorithm Greedy (a, n)
{ for i=1 to n do
  {
    x = Select (a)
      if Feasible (x) then
        Solution = Solution + x
  }
}
```

a

a_1	a_2	a_3	a_4	a_5

n

$x = a_1$ → walking ✗

a_2 → Run ✗

a_3 → Train ✓

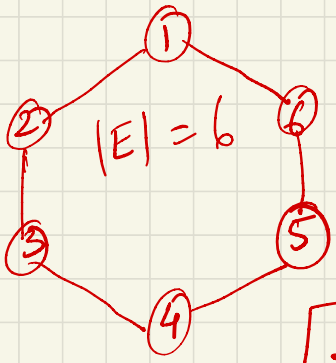
a_4 → Flight ✓

a_5 → Car ✓

Solution Set = $\{a_3, a_4, a_5\}$.

Optimal Solution = a_3 (Train). Optimal/Best.

Minimum Spanning Tree.



$G = (V, E)$.

$V = \{1, 2, 3, 4, 5, 6\}$ ✓

$E = \{(1,2), (2,3), (3,4) \ldots\}$.

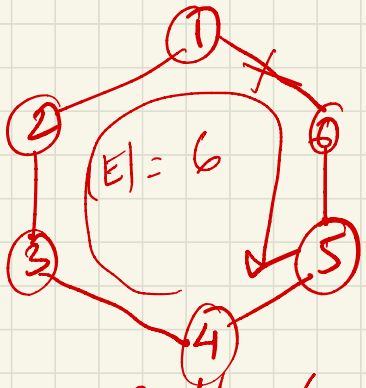
$S \subseteq G_2$ (S → Spanning Tree).

S is subset of G_2.

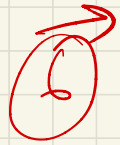
$S = \{V', E'\}$.

✓ $V' = V$, $|E'| = |V| - 1$

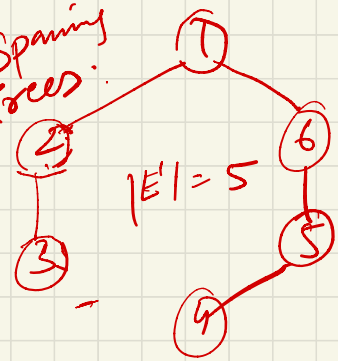
Spans across all vertices.



Spanning Tree →



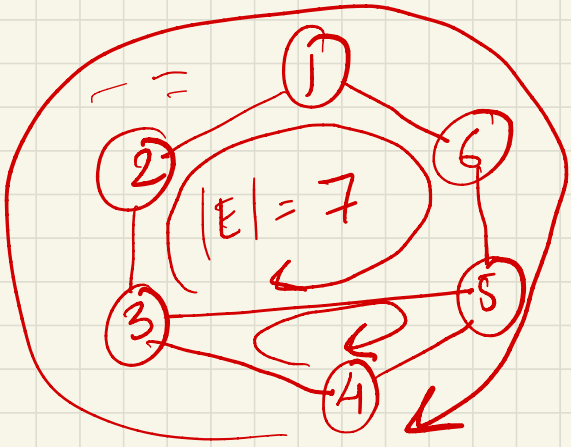
6 Spanning trees:



From a graph G, how many possible spanning trees can be formed ?? = 6

Combination → $^{6}C_{5} = \frac{n!}{r! \times (n-r)!} = \frac{6!}{5! \times 1!}$

$= \frac{6 \times 5!}{5!} = 6$



$|E'| = {}^{7}C_{5} = \frac{7!}{5! \times 2!}$

$= 21$

2) How many cycles with less than total no. of vertices ? 2

No. of Spanning Trees = $|E| C_{|V|-1}$ − no. of cycles with $< |V|$ no. vertices

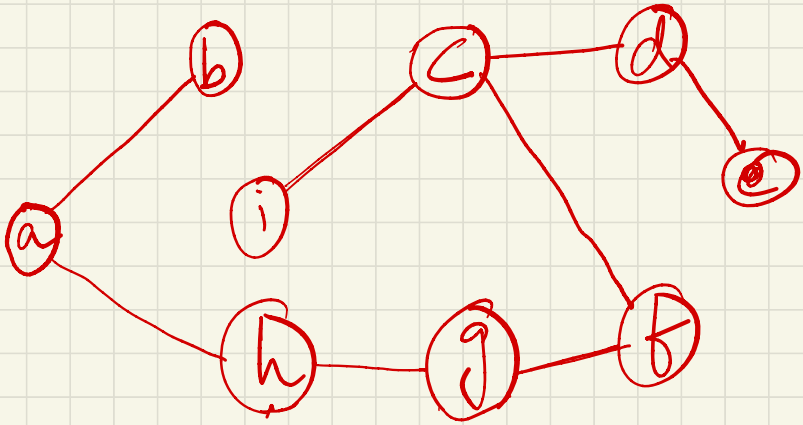
$= |E| C_{|V|-1}$ − no. of cycles with $< |V|$

9mp.

---

19 Aug 20. Sessional - Prims.
- kruskal.

Kruskal from Cormen book.



M-S-T.

A = list of final edges. initially A = ∅

if, Findset(u) ≠ Findset(v).

A = A ∪ {u, v}. A = (h,g) {i,g}

Next edge (i, c). Find(i) = i
Find(c) = c

Next edge = (g, f). Find (g) = {h, g}
Find (f) = f

Next (c, f) → Find (c) = {c, i}.
Find {f} = h, g, f.

Next (i, g) = Find (i) = i, c, f, g, h
Find (g) = i, c, f, g, h.

is find(i) = find(g). ?
If Yes, then dont include that edge.

→ Djikstra
→ Bell Man Ford } Lee Swapnil.

× × —— × × ~

31/08/20 Dynamic Programming. — MCM
— LCS

→ Greedy Methods vs DP.

→ Kruskal – Select the shortest edge & proceed.

// Before starting, we decided that we will select the shortest/minimum value edge & then proceed.

// Dynamic Programming → Principle of optimality.
At every stage, we will decide the optimal solution.

// Dynamic Programming generally deals with recursive formulae.

// Example → To find the n^{th} fibonacci term.

(Formula)

$$fib(n) = \begin{cases} 0 & \text{if } n = 0 \\ 1 & \text{if } n = 1 \\ fib(n-2) + fib(n-1) & \text{if } n > 1 \end{cases}$$

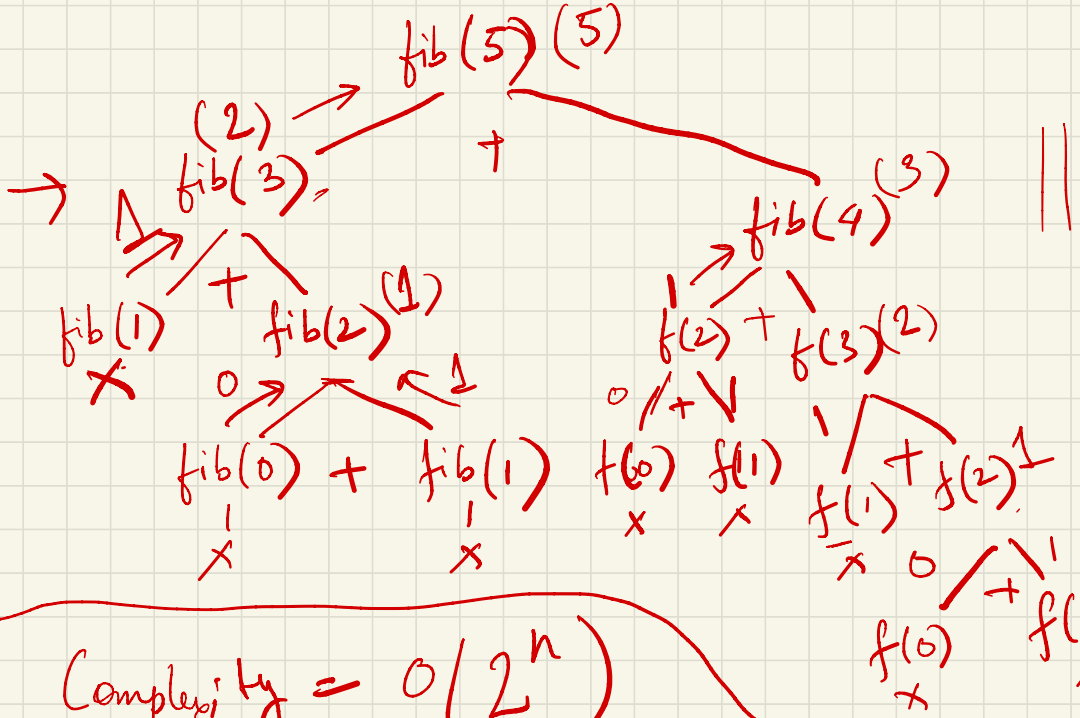
fib(0)

Fib → 0, 1, 1, 2, 3, 5, 8, 13 .....

0 1 2 3 4 5 fib(5) = 5.

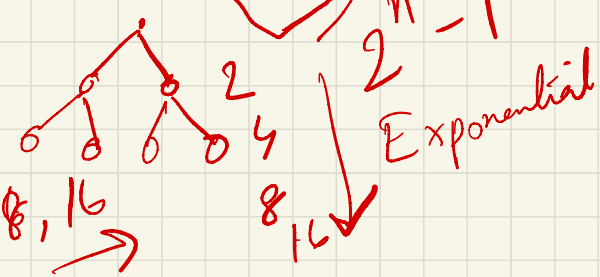
```
int fib(int n)
{ if (n<=1)   {0,1}
    return n;
  return fib(n-2) + fib(n-1);
}
```

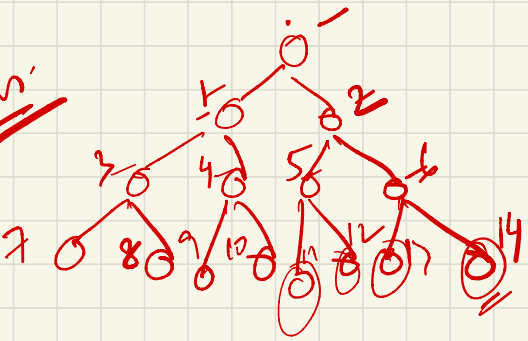
Find, the **5th term** in Fibonacci series



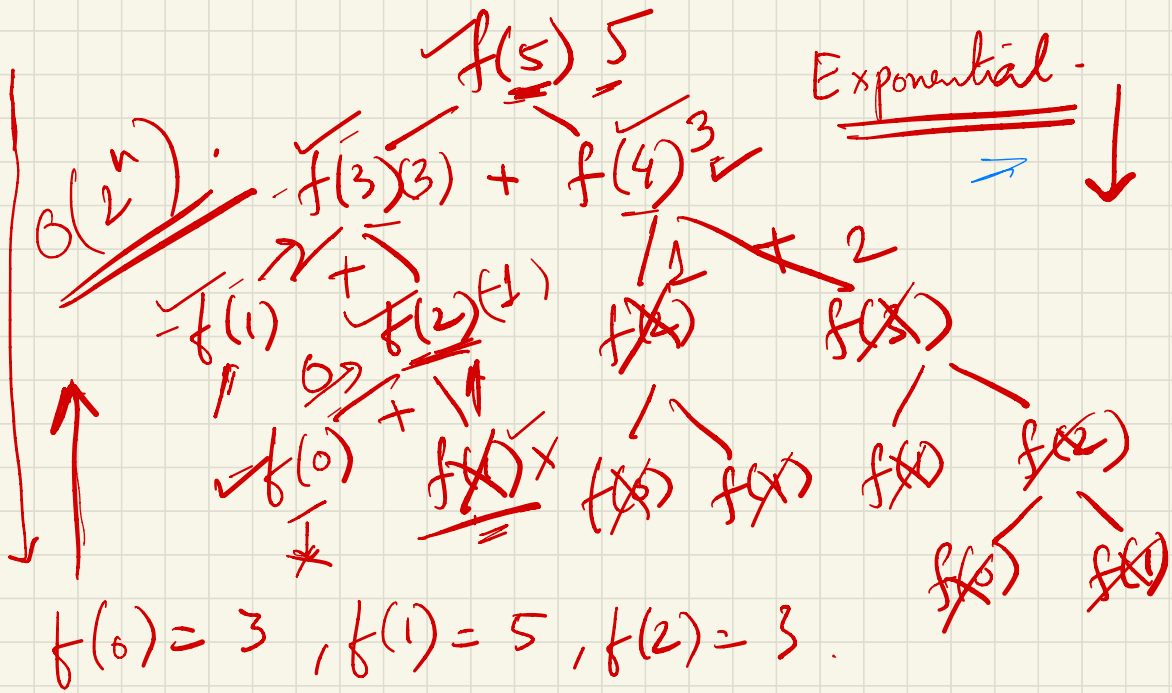
Complexity = $O(2^n)$

Level fib(5) → 2
1 level 2 → 4
lvl 3 → 6





Fn calls: $n \log n$.



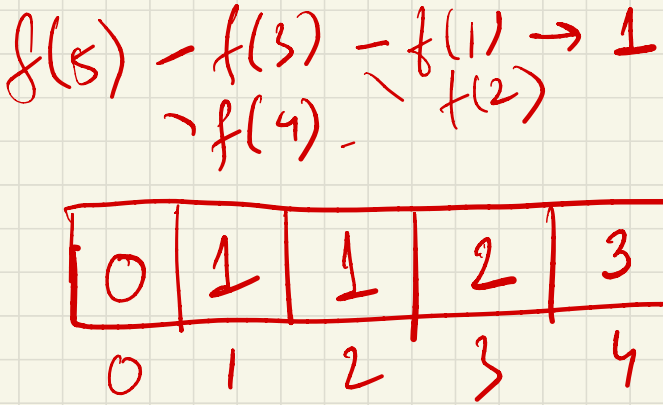
$f(0) = 3$, $f(1) = 5$, $f(2) = 3$.

Can the complexity be reduced ?.

// What if we take a global array.

$a[5] =$

-1	-1	-1	-1	-1	-1
0	1	2	3	4	5



$(n+1)$ – calls.

So, from 2^n calls, we reduced to $(n+1)$ calls

$2^n \rightarrow n$, Exponential → linear.

This above Concept is known as **Memoization**

⇒ Memoization (Memoize)

Dynamic Programming.

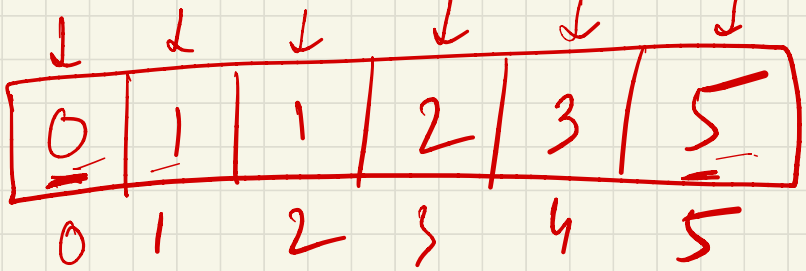
Memoization (Recursive)

Tabulation (Iterative).

Tabulation Method / (Iterative Method)

```
int fib(int n)
{
    if (n<=1)
        return n;
    F[0]= 0; F[1] = 1;
for(int i=2; i<=n ; i++)
  { F[i]= F[i-2] + F[i-1]
  }
return F[n];
}
```

Bottom up.



Memoization	vs	Tabulation
1) Recursive		1) Iterative
2) Top down Approach We started fib(5)		2) Bottom up Approach We started with F[0]

MCM: Matrix Chain Multiplication.

$A_1 \times A_2 =$ Conditions ??

$(c_1 = r_2)$ ✓

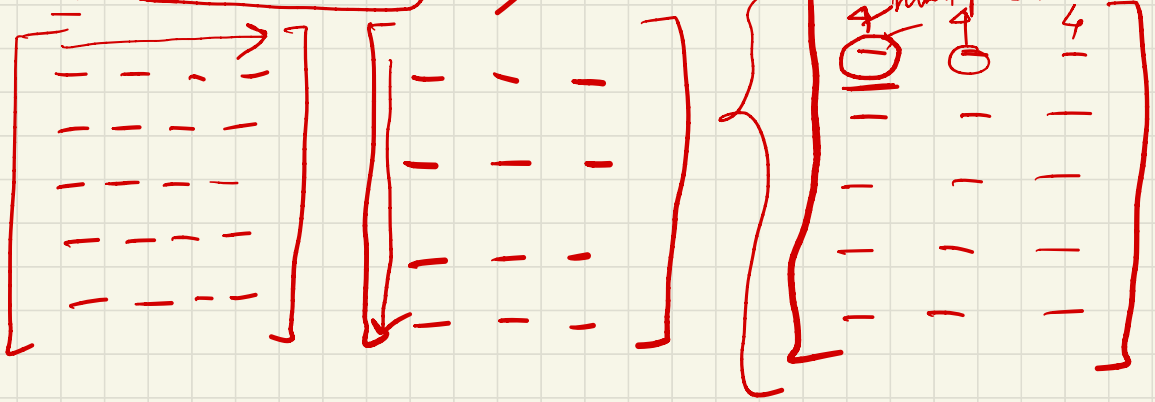
A_1: $r_1 \times c_1$, 5×4

A_2: $r_2 \times c_2$, 4×3

Product = final $(r_1 \times c_2)$

$A \times B = C$

$(5 \times 4) \quad (4 \times 3)$ ✓



$C = 5 \times 3$, Total no. of multiplications

$= 5 \times 3 \times 4$

$A_{10 \times 10} \times B_{10 \times 5} = C_{10 \times 5}$

Total Multiplication $= 10 \times 10 \times 5$

What if, we need to multiply more than two matrices..

MCM, its goal is to find the minimum cost of multiplication and not the result.

$$\underbrace{A_1}_{5\times4} \cdot \underbrace{A_2}_{4\times6} \cdot \underbrace{A_3}_{6\times2} \cdot \underbrace{A_4}_{2\times7}$$

$$(A_1 \cdot A_2) \cdot (A_3 \cdot A_4) \quad \ldots (1)$$

$$(5\times6) \cdot (6\times7) \rightarrow (5\times7)$$

$$A_1 \cdot (A_2 \cdot A_3) \cdot A_4 \quad \ldots (2)$$

$$((A_1 \cdot A_2) \cdot A_3) \cdot A_4 \quad \ldots (3)$$

$$10 \times 10 \times 5 \times 20$$

$(10\times10) \cdot (5\times20) = R_1$

$10 \times (10\times5) \times 20 \rightarrow R_2$

$((10\times10) \cdot 5) \times 20 \rightarrow R_3$

Equal?? Yes they will be equal.

A_1	A_2	A_3
2×2	2×3	3×1

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \cdot \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} \cdot \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix} = \begin{bmatrix} \quad \\ \quad \end{bmatrix}$$

① $((A_1 \cdot A_2) \cdot A_3)$

$(2 \times 3) \cdot (3 \times 1)$

(2×1)

② $(A_1 \cdot (A_2 \cdot A_3))$

$(2 \times 2) \cdot (2 \times 1)$

(2×1)

$$\begin{bmatrix} 78 \\ 170 \end{bmatrix} \quad \text{Equal} \quad \begin{bmatrix} 78 \\ 170 \end{bmatrix}$$

Whether ① is better than ② ??

It will depend on the minimum no. of multiplications.

$(A_1 \cdot A_2 \cdot A_3) = \frac{4C_2}{3} = 2$

A_1 (2×2), A_2 (2×3), A_3 (3×1)

① $((A_1 \cdot A_2) \cdot A_3) = [(2\times2\times3) + (2\times3\times1)]$

$(A_1 \cdot A_2)$ (2×3) B, $(B \cdot A_3)$ (3×1)

$= 12 + 6 = 18$ X

② $(A_1 \cdot (A_2 \cdot A_3))$ ✓ $= [(2\times2\times1) + (2\times3\times1)]$

(2×2) (2×1)

$= 10$ X

So, ② is minimum,

MCM using DP, will guide us how to parenthesize the chain of matrices.

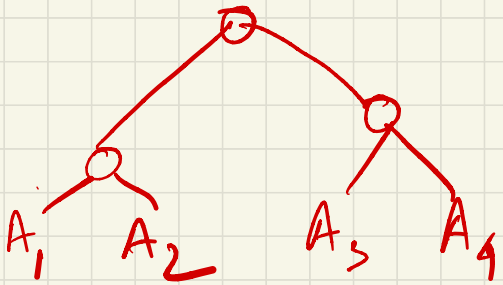
Now, $[A_1 \cdot A_2 \cdot A_3 \cdot A_4]$.

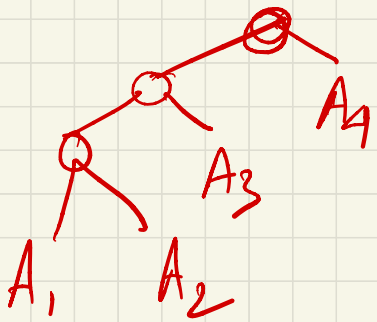
How many ways we can parenthesize ???

No. of parenthesis $= \frac{2nC_n}{n+1}$ ✓

n = no. of node.

Tree.





No. of nodes = 3.

// No. of parenthesis (P) = $\frac{6C_3}{3+1} = \frac{20}{4} = 5$

MCM Process:

$(A_1 \cdot A_2) : (A_3 \cdot A_4)$

$5\times4 \quad 4\times6 \quad 6\times2 \quad 2\times7$

Total 5 distinct ways to multiply.

$A_1 (A_2 \cdot A_3) \cdot A_4$

$A_1 - 5\times\underline{4}$, $A_2 - \underline{4}\times6$, $A_3 = \underline{6}\times\underline{2}$, $A_4 = \underline{2}\times7$

$P_0\ P_1 \quad P_1\ P_2 \quad P_2\ P_3 \quad P_3\ P_4$

$P_0 = 5,\ P_1 = 4,\ P_2 = 6,\ P_3 = 2,\ P_4 = 7$

$5\times4,\ 4\times6,\ 6\times2,\ 2\times7$

General Formula $A_1 = P_0 \times P_1$ $A_3 = P_2 \times P_3$

$A_2 = P_1 \times P_2$ $A_4 = P_3 \times P_4$

$$A_i = P_{i-1} \times P_i$$

$(A_1 \cdot (A_2 \cdot A_3) \cdot A_4))$

A_1: 5×4 A_2: 4×6 A_3: 6×2 A_4: 2×7

$m[1,1] = 0$

$A_1 \times A_1 = m(2,2) = 0$

$m[1,2] = 5 \times 4 \times 6 = 120$

$m[2,3] = 4 \times 6 \times 2 = 48$

$m[3,4] = 6 \times 2 \times 7 = 84$

	1	2	3	4
1	0	120	88	158
2		0	48	104
3			0	84
4				0

	1	2	3	4
1	0	1	1	3
2		0	2	3
3			0	3
4				0

$m[1,3] = (A_1 \cdot A_2 \cdot A_3)$

$m[2,4] = (A_2 \cdot A_3 \cdot A_4)$

$A_1 \cdot (A_2 \cdot A_3)$

$(A_1 \cdot A_2) \cdot A_3$

$A_2 \cdot A_3 A_4$

$A_2 A_3 \cdot A_4$

Select min

General Formula

$$m[i,j] = \begin{cases} 0 & \text{if } (i = j) \\ \min_{i \le k < j} \{ m[i,k] + m[k+1,j] + p_{i-1} \cdot p_k \cdot p_j \} & \end{cases}$$

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

Now,

$m[1,3]$ ⇒ Here k ranges from (1, 2).

For, $k = 1$.

$m[1,1] + m[2,3] + p_0 \times p_1 \times p_3$

$= 0 + 48 + 40 = 88$ ... (1).

For $k = 2$.

$m[1,2] + m[3,3] + p_0 \cdot p_2 \cdot p_3$

$= 120 + 0 + 60 = 180$ ... (2).

$m[2,4] \Rightarrow k \to 2, 3.$

For, $k=2$;

$m[2,2] + m[3,4] + p_1 \cdot p_2 \cdot p_4$

$= 0 + 84 + 168 = 252 \ldots (1)$

For $k=3$.

$m[2,3] + m[4,4] + p_1 \cdot p_3 \cdot p_4$

$= 48 + 0 + 56 = 104 \ldots (2)$

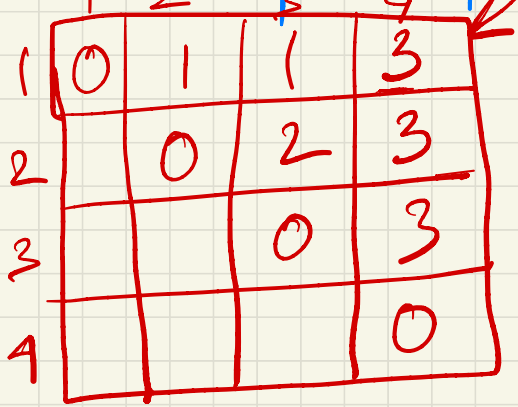
$m[1,4] = k \to 1, 2, 3.$ $k=3$

For $k=1$, $k=2$

$m[1,1] + m[2,4] + p_0 p_1 p_4$

$= 0 + 104 + 140$

How to form the optimal parenthesis ??



	1	2	3	4
1	0	1	1	3
2		0	2	3
3			0	3
4				0

5

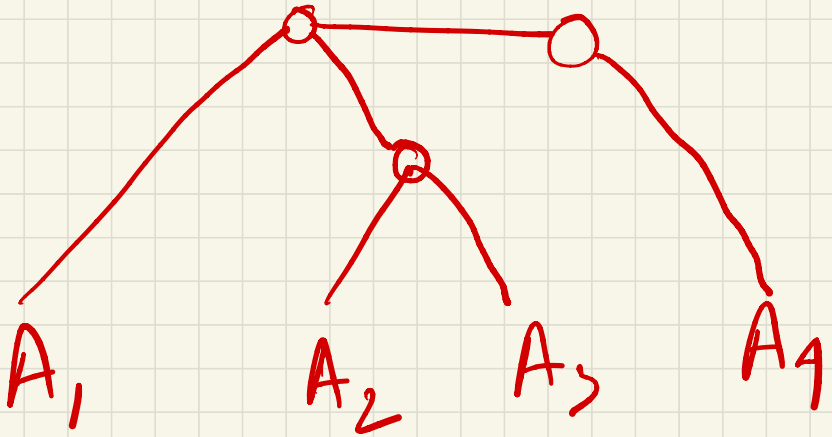
$(A_1 \cdot A_2 \cdot A_3) \cdot (A_4)$ (5).

(1,4) → 3 (Here K was 3).

(1,3) → 1 (Here K was 1).

$((A_1) \cdot (A_2 \cdot A_3)) \cdot (A_4)$.

$((A_1 \cdot (A_2 \cdot A_3)) \cdot A_4)$.



time complexity.

MCM complexity $\Rightarrow n^2 \times n$

$= O(n^3)$.

space complexity $\Rightarrow (n^2 \times 2)$.

How can you reduce the space complexity

Ans: Vacant spaces of (m) can be filled for (s) matrix

Longest Common Subsequence (LCS)

— 09 Sep 20

LCS is like string matching algorithm.

✓ Str 1 :- a b c d e f g h i j

✓ Str 2 :- c d g i

Are these strings equal → No.

What is a sequence ?.

↳ A particular fixed pattern.

DNA Sequencing.

String matching

a b c d.

b c d

1) c d g i

2) d g i

3) g i

4) i

LCS = cdgi

$S_1 \rightarrow a\ b\ c\ d\ e\ f\ g\ h\ i\ j$

$S_2 \rightarrow e\ c\ d\ g\ i$

1) e g i 2) c d g i

3) d g i 4) g i 5) i

LCS = c d g i

$S_1 = a\ b\ d\ a\ c\ e$

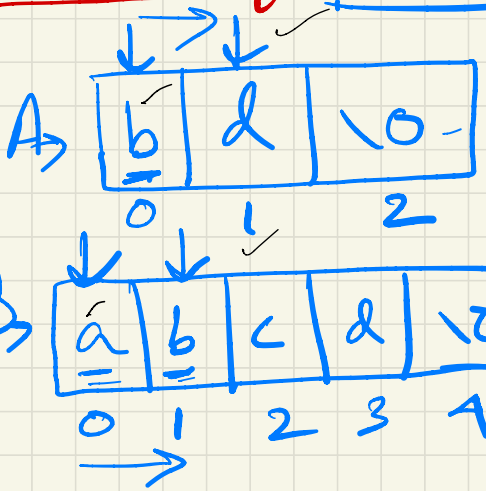
$S_2 = b\ a\ b\ c\ e$

1) b a c e 2) a b c e

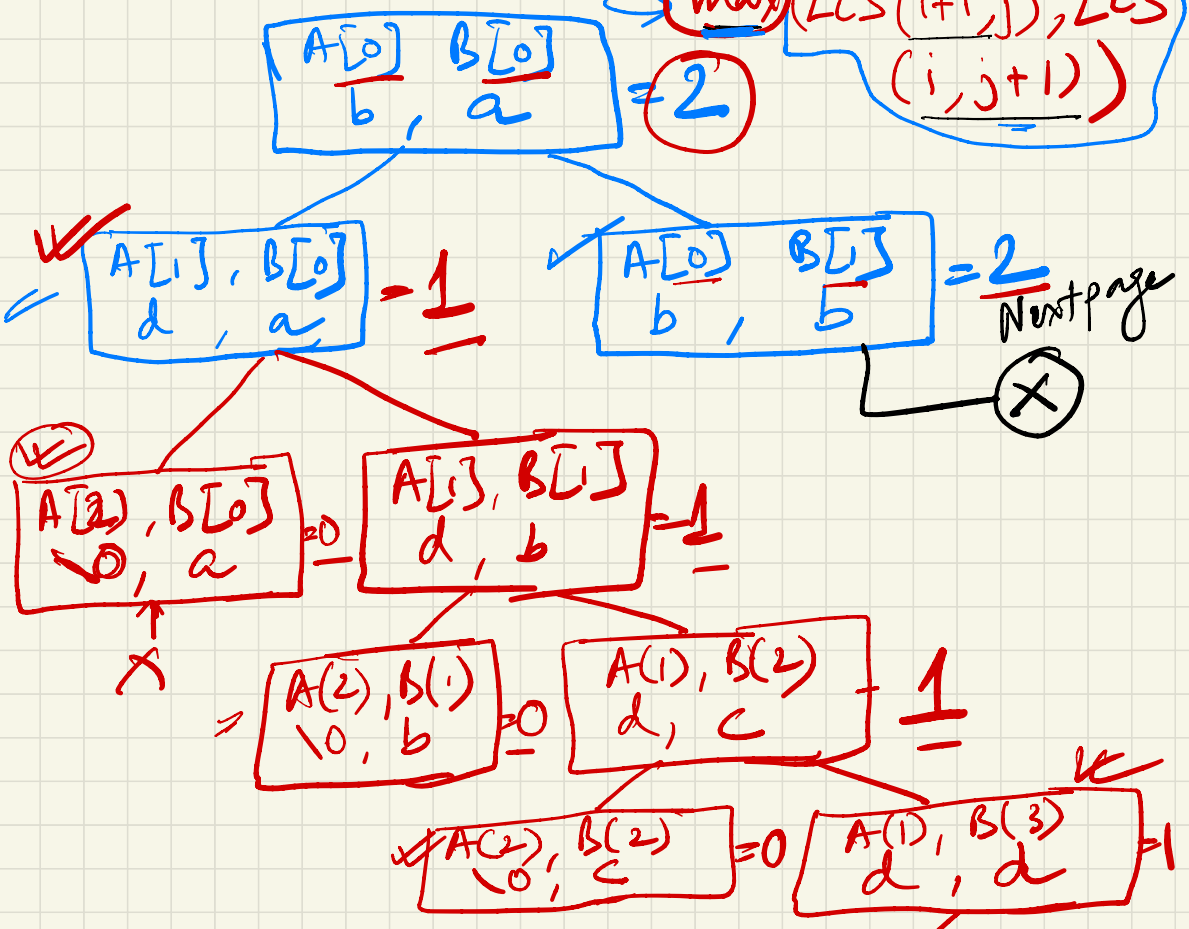
LCS = bace, abce.

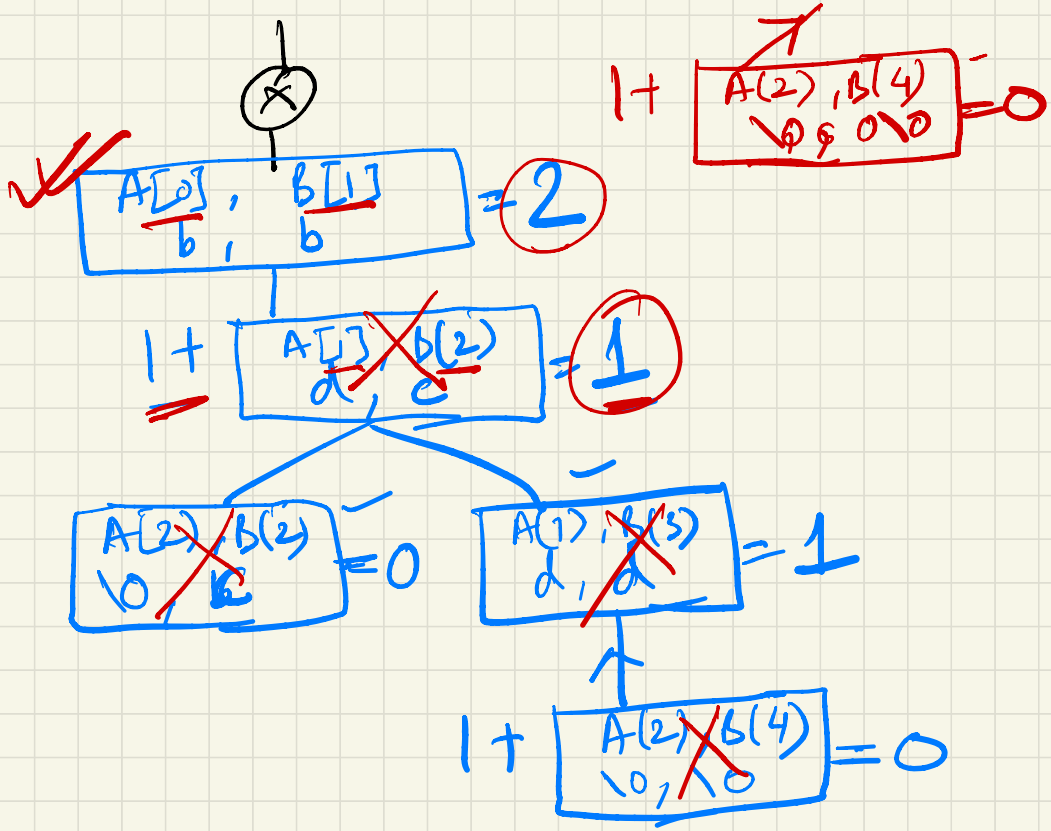
LCS using Recursion

ALGO




```
int LCS(i,j)
{
  if(A[i]=='\0' || B[j]=='\0')
    return 0;
  else if(A[i]==B[j])
    ret (1+LCS(i+1,j+1))
  else return
    max(LCS(i+1,j), LCS(i,j+1))
}
```





LCS Length = 2

LCS using memoization.

A		b	a	b	c	d	a
		0	0	0	0	0	0
b	0	2	2				
d	1	1	1	1	1		
\0	2	0	0	0		0	

A(2), B(0) = 0

→ Time complexity using recursion = $O(2^n)$

→ " " using memoisation = $O(m \times n)$

So, using memoisation time complexity reduced from $O(2^n) \rightarrow O(m \times n)$.

LCS using Recursion ✓
LCS using Memoisation ✓
} Top down approach.

LCS using Iterative approach → Bottom up.

Algo → if (A[i] == B[j])

$Lcs[i,j] = 1 + Lcs[i-1, j-1]$

initial values $(i=1, j=1)$

else

$Lcs[i,j] = \max\{[Lcs(i-1,j)]$, array

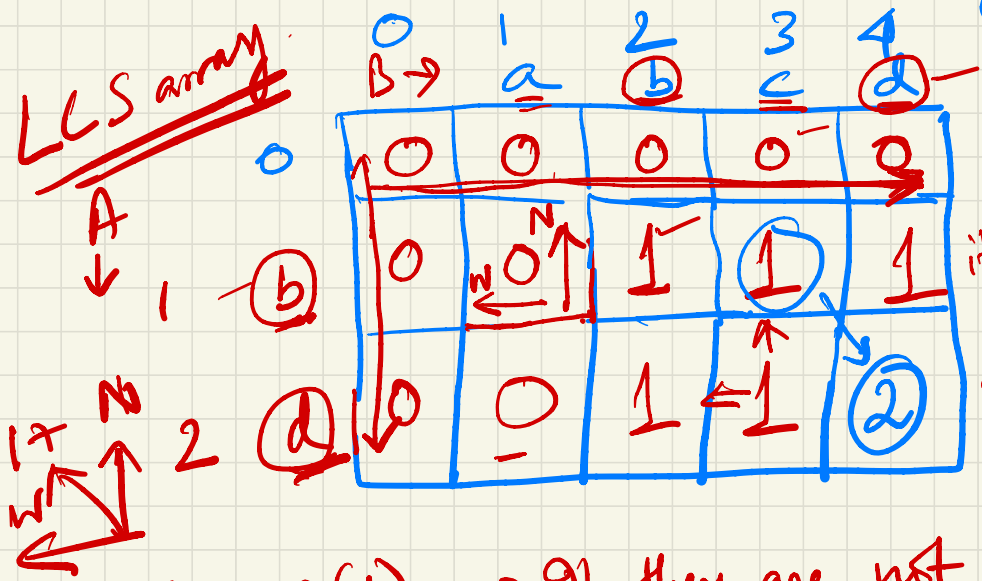
array → $Lcs(i, j-1)]\}$.

m → A: | b | d | (2)

n → B: | a | b | c | d | (1 2 3 4)

[[Changes
↳ NO NULL
↳ Index starts with 1 and not 0.

LCS array



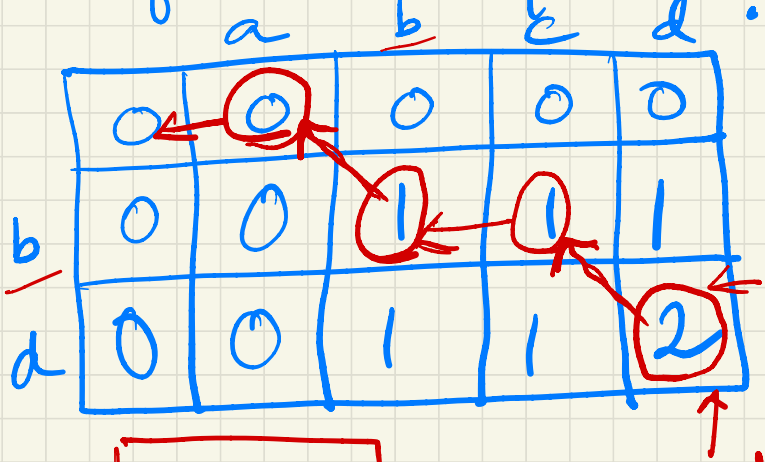
A(1), B(1) → If they are not equal, find the max value from North & West (cells)

North value = 0, West Value = 0.

A(1), B(2), → (b, b) → If they are equal add 1 to the North West diagonal element.

So, LCS Length = 2

How, to find the sequence ?.



LCS = b d

Str 1 :- Stone.

Str 2 :- Longest

o n e

LCS = 3, pattern = one.

Stone
Longest = one

---

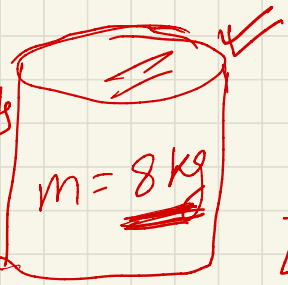
24 Sep 20

0/1 knapsack Problem

Fibon ✓ MCM ✓
LCS ✓
0/1 knapsack.

Optimal solution
↳ Greedy Method ✗
↳ Dynamic Progr, MCM, LCS, 0/1.
↳ Branch & Bound → 0/1 ✗
↳ Backtracking — N Queens
DS & Algo
↳ KMP String Relation / Master theorem
↳ Recurrence
↳ NP completeness.

0/1 Knapsack problem (Dynamic Programing)



n = no. of weights
= 4.
m = max weight
= 8

$W = \{2, 3, 4, 5\}$

$P = \{1, 2, 5, 6\}$

↳ Profit. $x = \{1, 0, 0, 1\}$.

Problem Definition: Move sack from one place to another so that the profit is maximum.

2 → 6 Tk, 5,2 → 7 Tk.

5,3 → 8 Tk, 5,4 → ✗ > m.

Solution/objective ⇒ Maximum profit.

$x_i \to 0, 1$ $x = \{1, 0, 0, 1\}$, $x = \{0, 1, 0, 1\}$.

Maximum profit ⇒ $\sum p_i x_i$ should be max.

↳ $(1 \times 1) + (0 \times 2) + (0 \times 5) + (1 \times 6) = 7$

↳ $(0 \times 1) + (1 \times 2) + (0 \times 5) + (1 \times 6) = 8$.

Total no. of possible combinations = ??.

[0,1].

0 0 0 0
0 0 0 1
0 0 1 0
0 0 1 1

④ $\Rightarrow 2^n = 16$.

(Brute force combinations)

General form = $O(2^n)$ $\Rightarrow$ Exponential.

Costly time.

So, Solution is optimisation problem — Greedy (DP) Branch & Bound.

So, 0/1 knapsack using DP → Tabulation.

P_i	W_i	0	1	2	3	4	5	6	7	8	rows cols (weight)
		0	0	0	0	0	0	0	0	0	
1 ← 2	1	0	0	1	1	1	1	1	1	1	
2	3	0	0	1	2	2	3	3	3	3	
5	4	0	0	1	2	5	5	6	7	7	
6	5	0	0	1	2	5	6	6	7	8	(6,8)

(4,1)

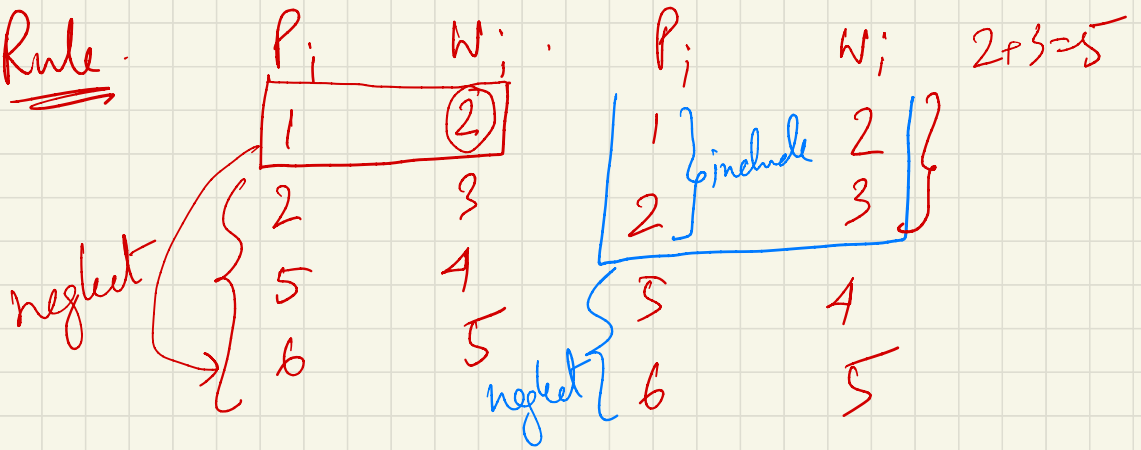
V(4,6)

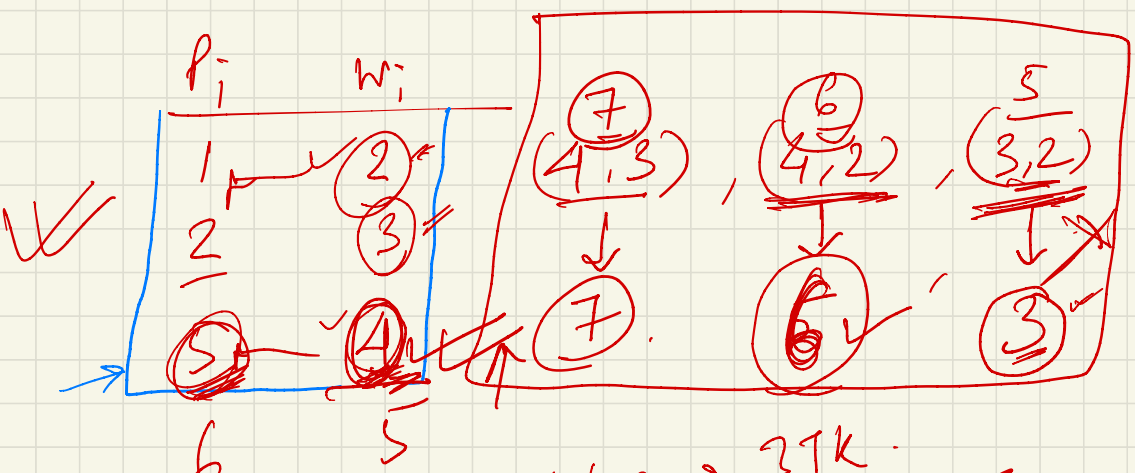
rows (items)

$n = 8$

Inside Table → Profit

Rule.





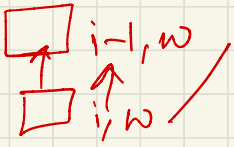
5kg → 3Tk.

4kg → 5Tk.

Formula →

$$V[i, w] = \max\{V[i-1, w], V[i-1, w-w[i]] + P[i]\}$$

rows, Column



$V[4,1] = \max\{V[4-1,1], V[3,1-5]+6\}$

$= \max[0, V[3,-4]+6]$ $\quad w[1] := 5$

$V[3,-4]$ → undefined

$5 - w[1]$ $\quad 5 \Rightarrow$

$V[4,5] = \max[V(3,5), V[3,0]+6]$

$= \max[5, 0+6] = 6$

$V[4,6] = \max\{V[3,6], V[3,6-5]+6\}$ $\quad 2$

$= \max\{6, V(3,1)+6\} = \max\{6,6\}$

$= 6$

$V[6,8] = 8$ → Total profit earned.

x_1	x_2	x_3	x_4 (Profit)
✗	✓	✗	✓ (6)
↓	↓ (2)	↓	↓
0	1	0	1

Balance Profit $= 8 - 6 = 2$

Prft Remaining = 2-2 = 0.

$x = \{0, 1, 0, 1\}$. $P = \{1, 2, 5, 6\}$

$\sum p_i x_i$ should be maximum (optimal solution).

$\sum p_i x_i = 2 + 6 = 8$ (Max profit).

0/1 knapsack Problem.

	0	1	2	3	4
P →	0	1	2	5	6
wt	0	2	3	4	5

$n = 4$.

$m = 8$.

Program main()

```
{
int P[5] = {0, 1, 2, 5, 6};
int wt[5] = {0, 2, 3, 4, 5};
int m=8, n=4;
int K[5][9];
```

```
for (int i=0; i<=n; i++)
{
  for(int w=0; w<=m; w++)
  {
    if (i==0 || w==0)
      K[i][w] = 0;
    else if (wt[i] <= w)
      K[i][w] = max(P[i] + K[i-1]
                [w- wt[i]], K[i-1][w]);
    else
      K[i][w] = K[i-1][w];
  }
}
cout << K[n][w];
}
```

KMP String Matching Algorithm. (V. Imp)

Basic String Matching Method / Process.

String :- a b c d e f g h (\0)
1 2 3 4

Pattern :- d e f (\0) → 4th position.

(a, d) ≠ → (b, d) ≠ → (c, d) ≠ →
(d, d) = → (e, e) = → (f, f) = → \0

If pattern reaches the last character, or
if pattern position is NULL, then it is a match.

n i

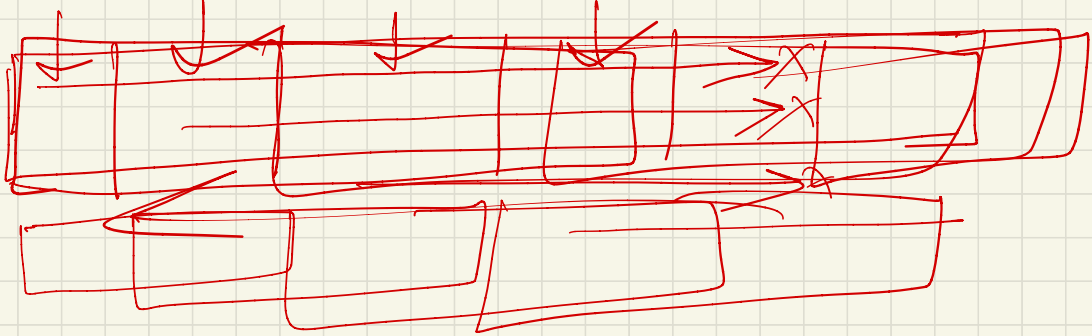
String :- a b c d a b c a b c d f g
1 2 3 4 5 6 7 8 9 10 11 12

Pat (m) j a b c d f

i → 5th to 2nd ; 8th to 3rd ; 8th to 4th

j → 0, 0.

i.



String :-

```
for (i -> 1 to n)
    j (1 -> n-m
```

Pattern :- | a | b | c | d | ... | g |

String :- | a | a | a | a | a | a | a | b |
1 2 3 4 5 6 7 8

Pattern :- | a | a | a | b |
1 2 3 4

Complexity.

Worst | a | b | $O(m \times n)$

$\rightarrow$ Naiive String Matching

1) On the existing Naiive Algo, We did not study the pattern.

a a a b.

2) We backtracked the position pointer in main string or searched string. (This is a wastage of time) :=

KMP:

We need to study the pattern and form a π or LPS table. (LPS → longest prefix which is also suffix).

Pattern :- a b c d a b c

1 2 3 4 5 6 7

Prefix :- a, ab, abc, abcd, ...

Suffix :- c, bc, abc, dabc, ...

P_1 :-

1	2	3	4	5	6	7	8	9	10
a	b	c	d	a	b	e	a	b	f
0	0	0	0	1	2	0	1	2	0

P_2 :-

1	2	3	4	5	6	7	8	9	10	11
a	b	c	d	e	a	b	f	a	b	c
0	0	0	0	0	1	2	0	1	2	3

P3 :-

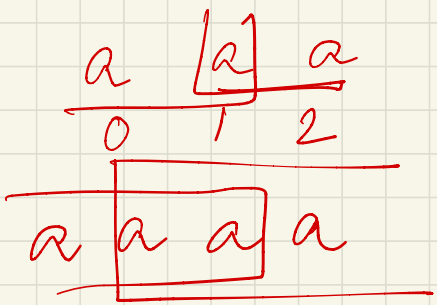
1	2	3	4	5	6	7	8	9	10
a	a	b	c	a	d	a	a	b	e
0	1	0	0	1	0	1	2	3	0

P4 :-

1	2	3	4	5	6	7	8	9
a	a	a	a	b	a	a	c	d
0	1	2	3	0	1	2	0	0

P4 :-

1	2	3	4	5	6	7	8
a	a	a	a	a	a	a	b
0	1	(2)	3	4	5	6	0



aa, aa.

Length of LPS → longest prefix as suffix.

Same

P3 :-

1	2	3	4	5	6	7	8	9	10
a	a	b	c	a	d	a	a	b	e
0	1	0	0	1	0	1	2	3	0

[a a b c @] 2

Suffix - a
Prefix - a

KMP Algorithms / Process

(i, j)

i → string

j → Pattern

String :-

a	b	a	b	c	a	b	c	a	b	a	b	a	b	d
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

Pattern :-

a	b	a	b	d
0	0	1	2	0

NULL

initially

i = 1.

j = 0.

if (String[i], Pattern[j+1]) equals.

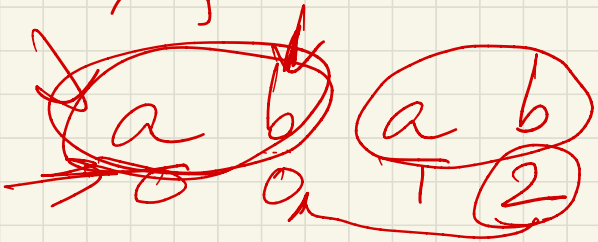
then i++; j++;

if (string[i], Pattern(j+1)) is not equal

then move j to corresponding (π) index.

table.

Catch ↪ if π index is NULL, then increment i



Stop when $(j+1) == m$ (m is length of pattern).

$i = 15$, $j = 4$., Pattern found at $(i-j)$ location.

Complexity :- String parse $\rightarrow O(n)$.

π table computation $\rightarrow O(m)$.

Complexity = String Parsing + π table compute

$$\boxed{O(m+n).}$$

Naiive $\rightarrow O(mn)$.

KMP $\rightarrow O(m+n)$

Recurrence Relation

Master Theorem.

NP-Hard / NP Complete

Branch & Bound $\rightarrow$ D.P.

$\rightarrow$ Greedy.

$\rightarrow$ B.B (Branch & Bound)

Recurrence Relation (Class Notes).

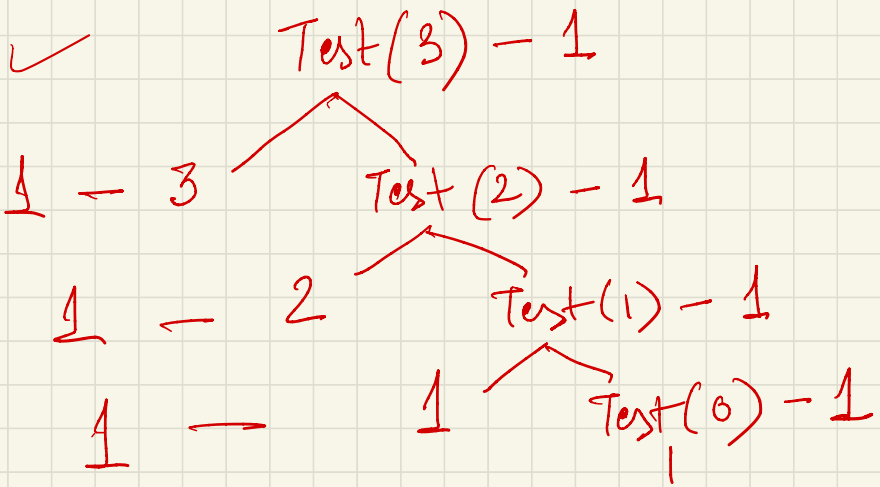
```
void Test(int n).
{ if (n>0)
  {
    printf("%d", n);   - 1
    Test(n-1);         - 1
  }
}
```

- $O(n)$.
  - Prove ??

Unit time
Some Time

- Decreasing.

Pass value 3 in n. (Test(3))



Total time - printf (5). - n
Test() → 4. - n+1.
2n+1.

Recursive Tree approach → O(n).

Let total time taken by a fun of n. be called let it be $T(n)$

```
Void Test (int n)  → T(n)
{
  if (n>0)          — unit time. - Neglect
  {
    printf("%d", n);   — 1
    Test(n-1);         — T(n-1)
  }
}
```

Main function — Test(n) → $T(n)$

Inside Recursive call — Test(n-1) → $T(n-1)$

Total time, $T(n) = T(n-1) + 1$ ... (1)

$$T(n) = \begin{cases} 1 \text{ or } c & \text{if } n=0 \\ T(n-1)+1 & \text{if } (n>0) \end{cases}$$ function.

We will find the complexity of $T(n)$ by Substitution method :—

$$T(n) = T(n-1) + 1 \quad \ldots (1)$$

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

Substitute $T(n-1)$ in eq. (1).

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

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

Find: $T(n-2)$

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

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

Substitute (4) in (3).

$$T(n) = T(n-3) + 3. \quad \text{---- (5)}.$$

(continue for 'k' times).

$$T(n) = T(n-k) + k \quad \text{---- (6)}.$$

↳ One time will come, when k will be almost equal to n.

Assume $\boxed{n \cong k}$ (approximately equal). ---- (7)

↳ Substitute (7) in (6).

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

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

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

↳ $T(n) \Rightarrow \boxed{O(n)}$.

Problem 2.

```
T(n) → Void Test(int n)
1    — {
           if (n>0)
n+1  —     {
               for(i=0; i<n; i++)
n    —         {
               printf("%d", n);
           }
T(n-1) →   Test(n-1)
       }
```

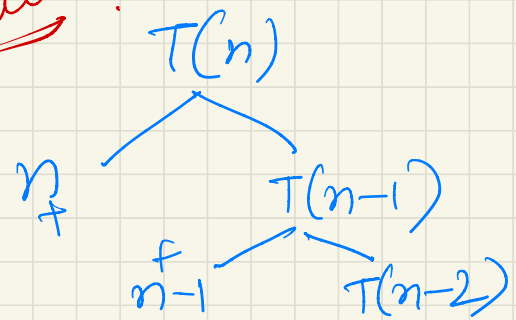
$T(n) = T(n-1) + n + n + 1 + 1$.

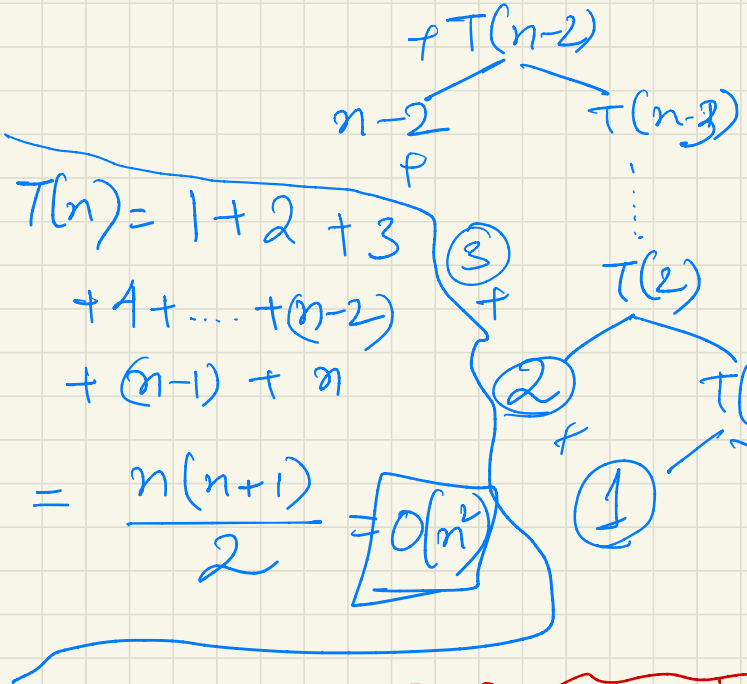
$= T(n-1) + 2n + 2 = \boxed{T(n-1) + n}$.

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

↳ Recurrence Relation

Recursive Tree Method.





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

$\Rightarrow O(n^2)$

$T(n) = 1 + 2 + 3 + 4 + \dots + (n-2) + (n-1) + n$

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

Substitution Method

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

$T(n) = T(n-1) + n \quad \dots (1)$

Substitute n by $(n-1)$

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

Substitute (2) in (1)

$T(n) = T(n-2) + (n-1) + n \quad \dots (3)$

$T(n-2) = T(n-3) + (n-2) \quad \dots (4)$

Substitute (4) in (3)

$T(n) = T(n-3) + (n-2) + (n-1) + (n)$ .... (5)

⋮ Contine till k - $n-1 \to n-k-1$

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

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

✓ Imp.

✓ V. Imp. (6)

Assume $n = k$.

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

$\dots + (n-1) + n$.

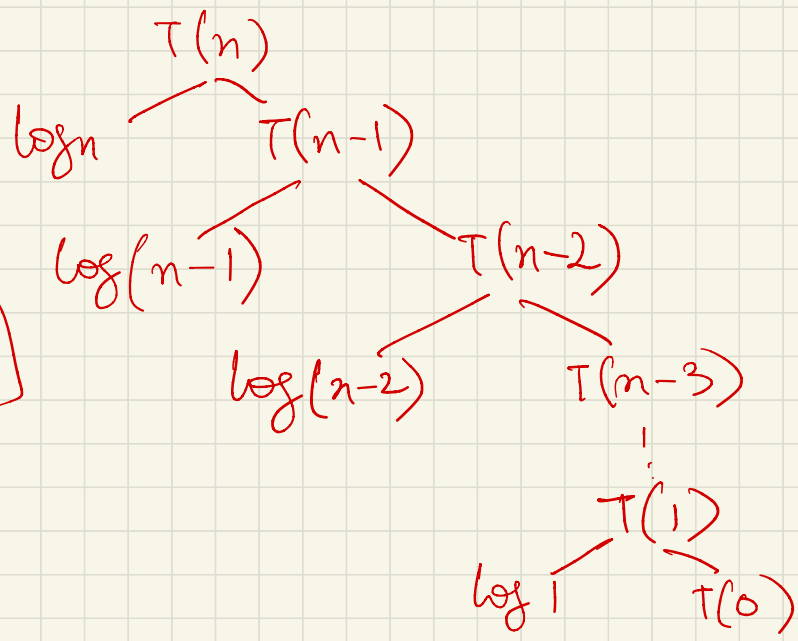
$= 1 + (1+2+3+\dots+n) = 1 + \frac{n(n+1)}{2}$

$$\Rightarrow O(n^2)$$

Problem 3:-

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

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



$\log 1 = 0$

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

$= \log [n \times (n-1) \times (n-2) \times \cdots \times 1]$

$= \boxed{\log(n!)} \Rightarrow$

$n^n, 2^n, n \log n, \log n^n$

Upper bound of $(n! \rightarrow n^n)$.

$\Rightarrow \log n^n \Rightarrow \boxed{O(n \log n)}$.

Substitution method.

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

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

$= T(n-3) + \log(n-2) + \log(n-1) + \log n$.

$\vdots$ Continue till $k = (n-k=0)$

$= T(n-k) + \log 1 + \log 2 + \log 3 + \cdots \log(n-1) + \log n$.

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

$+ \log n! \Rightarrow \boxed{O(n \log n)}$

General forms:

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

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

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

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

5) $T(n) = T(n-2) + 1$ — $n/2$ ... $O(n)$

6) $T(n) = 2T(n-100) + n$ — $O(n^2)$.

7) $\boxed{T(n) = \boxed{2T(n-1) + 1}}$ Next class

& Next week.

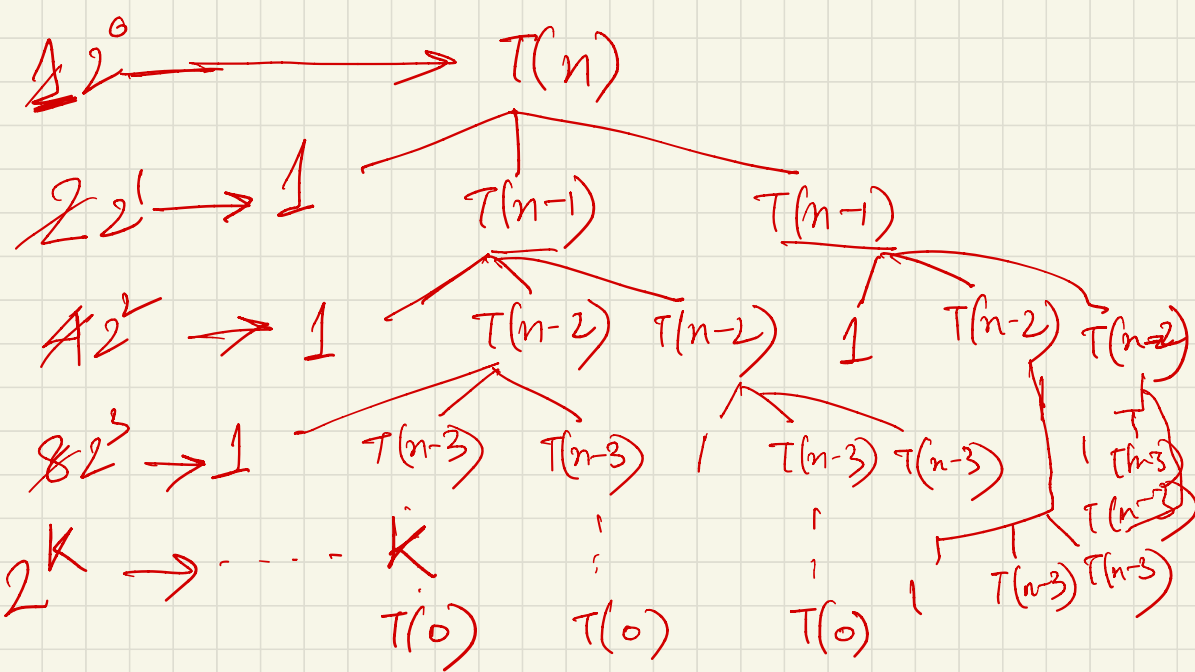
Problem 4:-

```
Test(n)                                  — T(n)
{
  if (n>0)
  {
    printf(n);                           — 1
    Test(n-1);                           — T(n-1)
    Test(n-1);                           — T(n-1)
  }
}
```

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

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

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



Total time $= 1 + 2^1 + 2^2 + 2^3 + \ldots + 2^k$

Geometric Progression :- G.P series :

$a + ar + ar^2 + ar^3 + ar^4 \ldots + ar^{n-1}$

$$= \frac{a(r^{k+1} - 1)}{r - 1}$$

In our case, Total time $= 1 + 2^1 + 2^2 + 2^3 \ldots + 2^k$.

Here $a = 1$, $r = 2$.

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

Assume, $n - k = 0$, $\therefore$, $k = n$. (A

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

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

Substitution Method :-

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

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

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

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

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

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

↓ Continue till k.

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

→ (4)

Assume $n - k = 0$; $n = k$.

$\therefore 2^n \underline{T(n-n)} + \boxed{2^{n-1} + 2^{n-2} + \ldots + 2^2 + 2 + 1}$

$1 + 2 + 4 + \ldots + 2^{n-1}$

$= 2^n \times 1 + \boxed{2^n - 1}$

$= 2^n + 2^n - 1 = 2^{n+1} - 1. \Rightarrow \boxed{O(2^n)}$

General Form

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

$3T(n-1) + \log n \Rightarrow O(3^n \cdot \log n)$

Master Theorem for Decreasing Functions - 6.

1) $T(n) = 1\,T(n-1) + 1$ — $f(n)\cdot n \times 1$ — $O(n)$ — $n \times \frac{1}{n} = n^0$
2) $T(n) = 0.5\,T(n-1) + n$ — $f(n)$ — $O(n^2)$ — $n \times n$
3) $T(n) = 0.5\,T(n-1) + \log n$ — $O(n \log n)$
4) $T(n) = 2\,T(n-1) + 1$ — $O(2^n)$
5) $T(n) = 3\,T(n-1) + 1$ — $O(3^n)$
6) $T(n) = 2\,T(n-1) + n$ — $O(n \cdot 2^n)$

$$T(n) = \boxed{a\,T(n-b)} + f(n)$$

$1 \;\; \frac{1}{n} \;\; \log n \;\; n^2$ } $f(n)$

$n^0 \;\; n^1, n^2, n^3$

$\log n$

$\sqrt{n} \;\; n/2$

$a > 0, \quad b > 0$

$\boxed{f(n) = O(n^k)}$ where $\boxed{k \geq 0}$

if (1) $\boxed{a = 1}$, $f(n) = O(n^k)$

$O(n^{k+1}) \Rightarrow O(n \cdot f(n))$

$a^b = c$

$\boxed{b = \log_a c}$

② if $a > 1$,

↳ $O(a^n \cdot f(n))$

↳ $O(a^{n/b} \cdot f(n))$

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

$= O(2^n)$.

$a^n \cdot f(n)$.

if $T(n-2)$, or

$T(n-3) \ldots T(n-b)$.

③ if $a < 1$; → Neglect.

↳ $O(f(n)) + C$. Neglect.

↳ $O(f(n))$

Neglect $aT(n-b)$

$2T(n-1)$.

$= 2^n$.

$\frac{1}{2} T(n-1)$.

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

$\frac{1}{100} T(n-1)$.

$= \left(\frac{1}{100}\right)^n$

Case 1 :- if $a < 1 \rightarrow O(f(n))$

Case 2 :- if $a = 1 \rightarrow O(n \cdot f(n))$

Case 3 :- if $a > 1 \rightarrow O(a^{n/b} \cdot f(n))$

Master's Theorem for Decreasing Functions

Recurrence Relations for Dividing functions.

(1) Algorithm Test(int n).

```
{
  if (n>1)
  {
    printf (n);
    Test (n/2);
  }
}
```

10
5
2
1

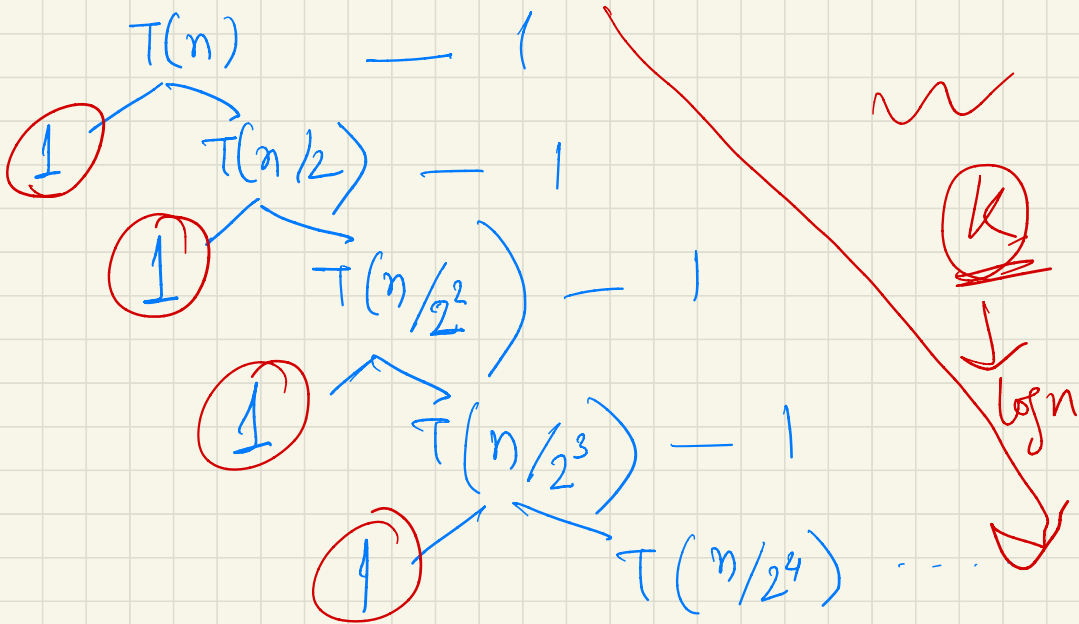
1
T(n/2)

decreasing
n-1
n-100
n/2
√2, n/100

O(n/2) = O(n)

Recurrence Relation :-

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



$\frac{n}{2^k} \simeq 1$ (after k steps)

After k steps:

$n/2^k = 1$

$T(n/2^k)$ ... $-k$

$\frac{n}{2^k} = 1$ (Assume) — after k steps.

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

Check by substitution method :-

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

$$T(n) = [T(n/2^2) + 1] + 1.$$

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

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

⋮ After k steps.

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

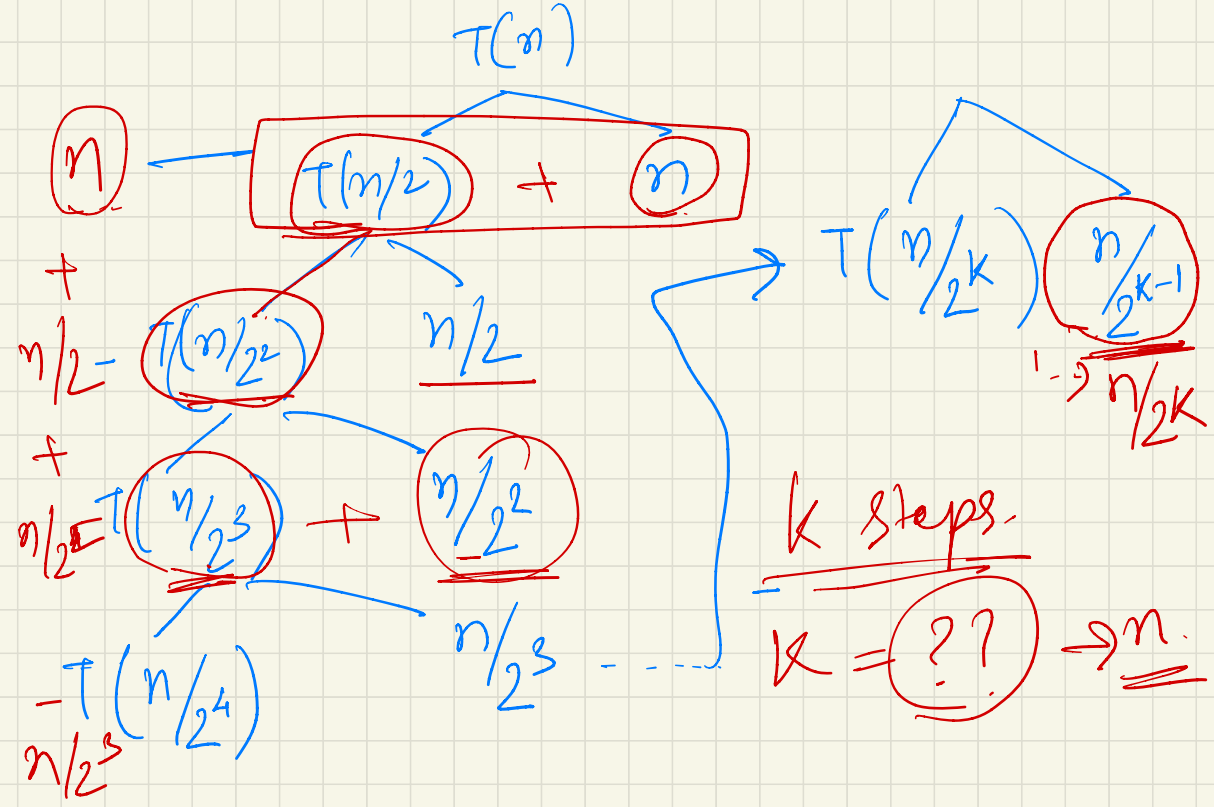
Assume $\frac{n}{2^k} = 1$, $k = \log n$.

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

$= 1 + \log n \Rightarrow O(\log n)$.

---

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



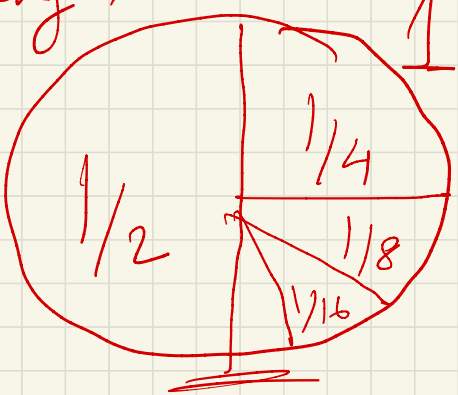
Total time $T(n) = n + n/2 + n/2^2 + \cdots + n/2^k$

$= n\left[1 + \frac{1}{2} + \frac{1}{2^2} + \frac{1}{2^3} + \cdots + \frac{1}{2^k}\right] = 1$

$= n \times \sum_{i=0}^{k} \frac{1}{2^i} = 1.$

$1 + \left[\frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \frac{1}{16} + \frac{1}{33} \cdots \frac{1}{2^k}\right].$

→ Value is decreasing.



$\Rightarrow 1 + 1 = 2.$

$T(n) = 2n.$

$O(n)$