Space complexity :-

$S(n) = I/P + Extra$

- I/P → constant (2-Byte)
- $Extra$ → stack size, Table size

$= 2 + O(n) + O(n)$

$S(n) = O(n)$

⇒ Longest Common Subsequence :-

- Subsequence of a given sequence is just the given sequence only in which 0 or more symbols left out.

eg:

$S = (A\ B\ B\ A\ B\ B)$

positions: 1 2 3 4 5 6 → increasing order

$SS_1 = (A\ A)$ at positions 1, 4

$SS_2 = (B\ B\ B\ B)$ at positions 2, 3, 5, 6

increasing order

$SS_3 = ( )$ → 0 elements

$SS_4 = (B\ A\ B\ A)$ (2 1 3 4) ✗ not a subsequence

- Common subsequence is the common sequence to X and Y.

eg: $X = (A\ B\ B\ A\ B\ B)$

$Y = (B\ A\ A\ B\ A\ A)$

$CS_0 = ()$ ✓

$CS_1 = (A)$ ✓

$CS_2 = (AB)$ ✓

$CS_3 = (AAB)$ ✓ ← (LCS)

$CS_4 = (ABAB)$ ✗ no possible case

eg: $X = (A\ B\ A\ B\ A\ B\ B)$

$Y = (B\ A\ B\ A\ B\ B)$

$CS_0 = ()$

$CS_1 = (A)$

$CS_2 = (AB)$

$CS_3 = (ABA)$

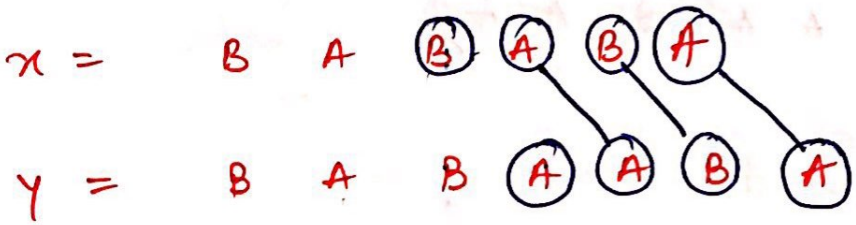
$CS_4 = (ABAB)$

$CS_5 = (BABAB)$

$CS_6 = (BABABB)$ → [LCS]

LCS(m,n) → it returns length of common subsequence of X and Y where X contains m symbols and Y contains n symbol.

Algorithm for LCS(m,n) :-



LCS(6,7) = 1 + LCS(5,6) match A–A

↓

1 + LCS(4,5) B–B

↓

0 + max (LCS(3,3), LCS(2,4))

↓ 2 ↓ 3

Pseudocode :- (Recursive Program)

```
LCS(m,n)  — T(m,n)
{   if( m==0 || n==0)
        return 0;

    else
    {
```

```
if( x[m] == y[n] )
    return ( 1 + LCS(m-1, n-1));
```

Best case

1 func^n call

```
else
{
    a = LCS(m-1, n);
    b = LCS( m, n-1);
    c = max( a, b);
    return c;
}
}
}
```

$T(m-1,n)$ — a = LCS(m-1, n);

$T(m,n-1)$ — b = LCS(m, n-1);

2 func^n call

Worst case

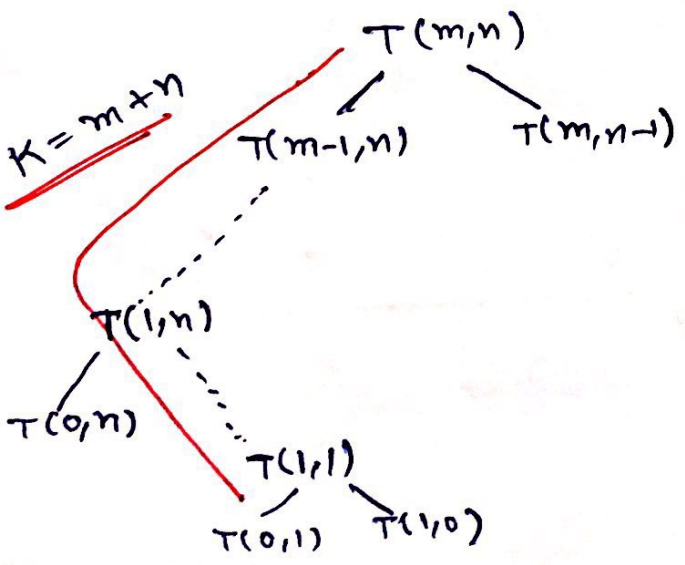
$O(1)$ — c = max(a, b); return c;

For best case :- $T(m,n) = \min(m,n)$

$$T(m,n) = T(m-1, n-1) + c$$

For worst case :-

$$T(m,n) = T(m-1, n) + T(m, n-1) + c$$



$$T(m,n) = c + 2c + 2^2c + \cdots + 2^k c$$

$$= c(1 + 2 + 2^2 + \cdots 2^k)$$

$$= c(2^{m+n})$$

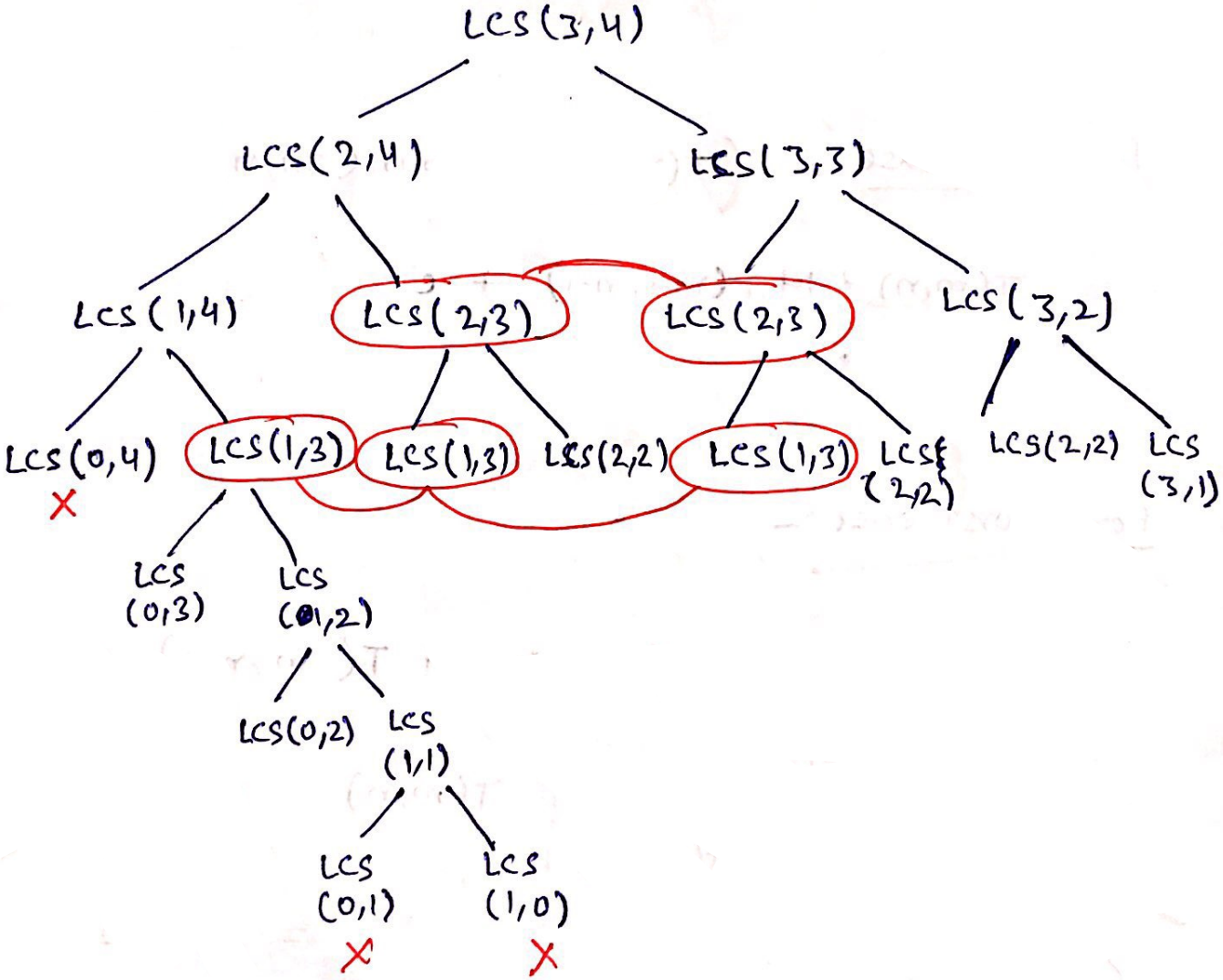
$$T(m,n) = O(2^{m+n})$$

eg:- LCS(3,4)

x = A A A

Y = B B B B

Recursion tree

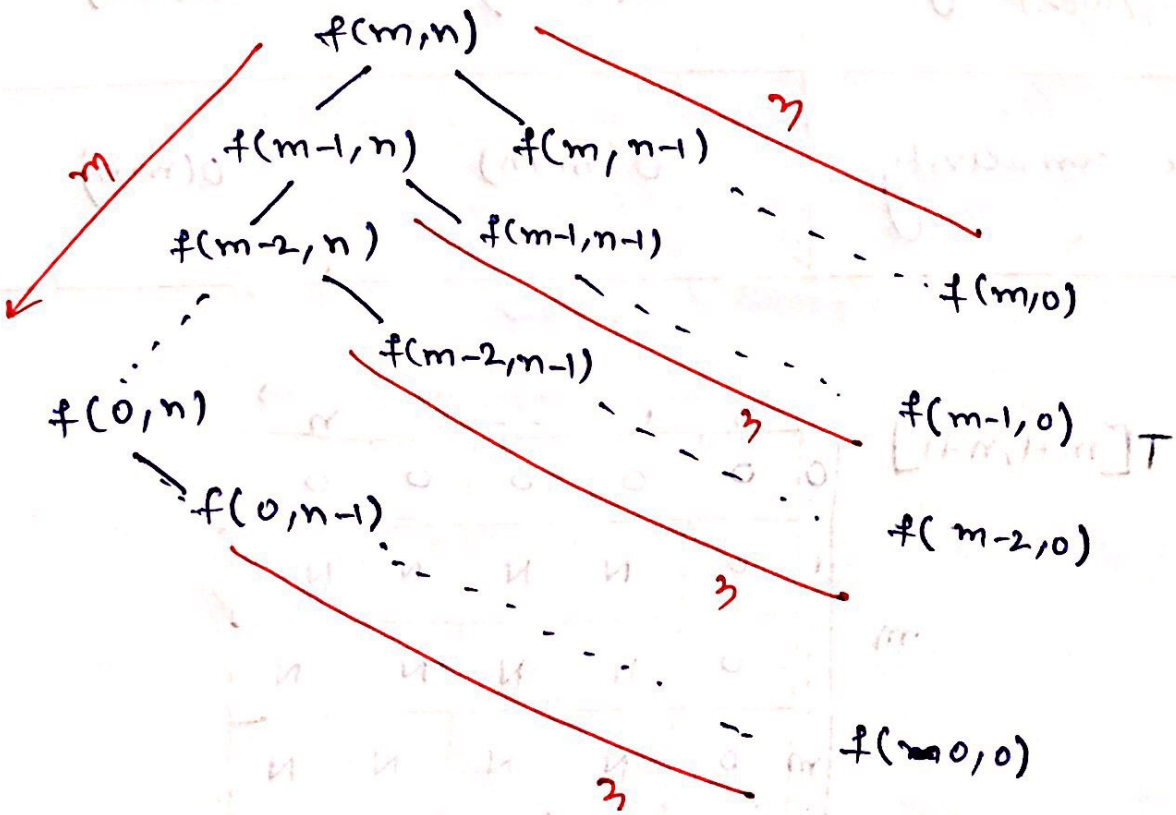


Stack space = No: of level = k

= O(m+n)

- In the above recursive tree many func^n calls are repeating, so we can use dynamic programming which ~~ea~~ will compute only dinstinct func^n call.

Distinct func calls = $\binom{m}{+1} * (n+1)$ = O(mn)



$T(n) = O(m,n)$ — Time complexity with dynamic programming

Space complexity :- I/P + Extra

I/P → m+n

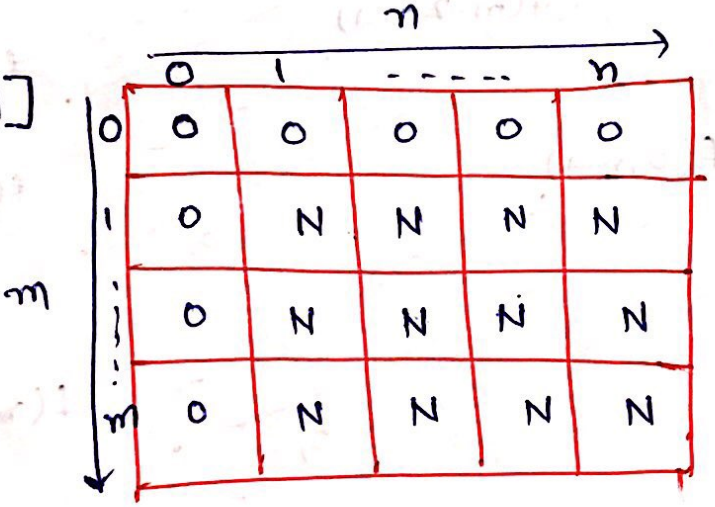
Extra → Stack space → m+n

Extra → ~~Time~~ Table (2D Array) → O(mn) [(m+1)(n+1)]

= O(mn)

	without Dynamic Prog	with Dynamic Prog
Time complexity Avg/wost	$O(2^{m+n})$	O(mn)
Space complexity	O(m+n)	O(mn)

T[m+1, n+1]



In dynamic programming, add conditional statement before each recursive call.

```
DP_LCS(m,n)
{
    if ( m==0 || n==0)
        return 0;
    else
    {
        if( x[m] == y[n])
        {
            if(T[m-1,n-1] == NULL)
                T[m-1,n-1] = DP_LCS(m-1,n-1)
            else
                return (1 + T[m-1,n-1]);
        }
        else
        {
            if ( T[m-1,n] == NULL)
                T[m-1,n] = DP_LCS( m-1,n);
            if ( T[m,n-1] == NULL)
                T[m,n-1] = DP_LCS ( m,n-1);
            return max( T[m-1,n] , T[m,n-1]);
        }
    }
}
```

⇒ 0/1 Knapsack :- (Fraction not allowed)

eg:- $n=3$ $M=10$

Objects	O_1	O_2	O_3
Profits	70	95	30
Height	6	7	4

Feasible solutions

$O_1\ O_2\ O_3$	Profit
0 0 0	0
0 0 1	30
0 1 0	95
1 0 0	70
1 0 1	100

Best solution

0/1 using Greedy knapsack :-

	O_1	O_2	O_3
P/W	11.66	13.57	7.5

0	1	0

Total Profit = 95

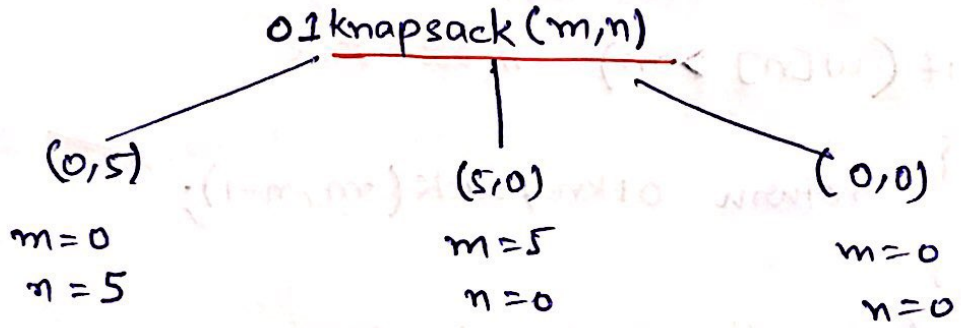
this is wrong as we calculate using Brute force, maximum profit = 100

NOTE:- so, Greedy techniane will not give always right answer, as it always finds local best.

01knapsack(m,n) → m is capacity of knapsack
n is no. of objects.

eg:- n = 5, m = 17

Objects :	O_1	O_2	O_3	O_4	O_5
Profit :	25	75	55	35	45
Weight :	3	5	4	3	5



So, termination condition (m,n = 0) then return 0.

(i) if m=0 or n = 0 01knapsack(m,n) → 0

(ii) if w[n] > m

01knapsack(m,n)

↓

01knapsack(m, n-1)

(iii) if $w[n] \le m$

$$01knapsack(m,n) = \max \begin{pmatrix} 01knapsack(m-w[n],n-1) + p[n], \\ 01knapsack(m,n-1) \end{pmatrix}$$

Psudocodi:-

```
01knapsack (m,n)                                   — T(m,n)
{   if (m==0 || n==0)
        return 0;
    else
    {   if (w[n] > m)        // best case
        {   return 01knapsack(m,n-1);              — O(n)
        }
        else    // Worst or Avg case
        {   a = 01knapsack (m-w[n],n-1) + p[n];
            b = 01knapsack (m,n-1);
            c = max (a,b);                         — O(1)
            return c;
        }
    }
}
```

Best case :-

$$T(m,n) = T(m,n-1) + c$$

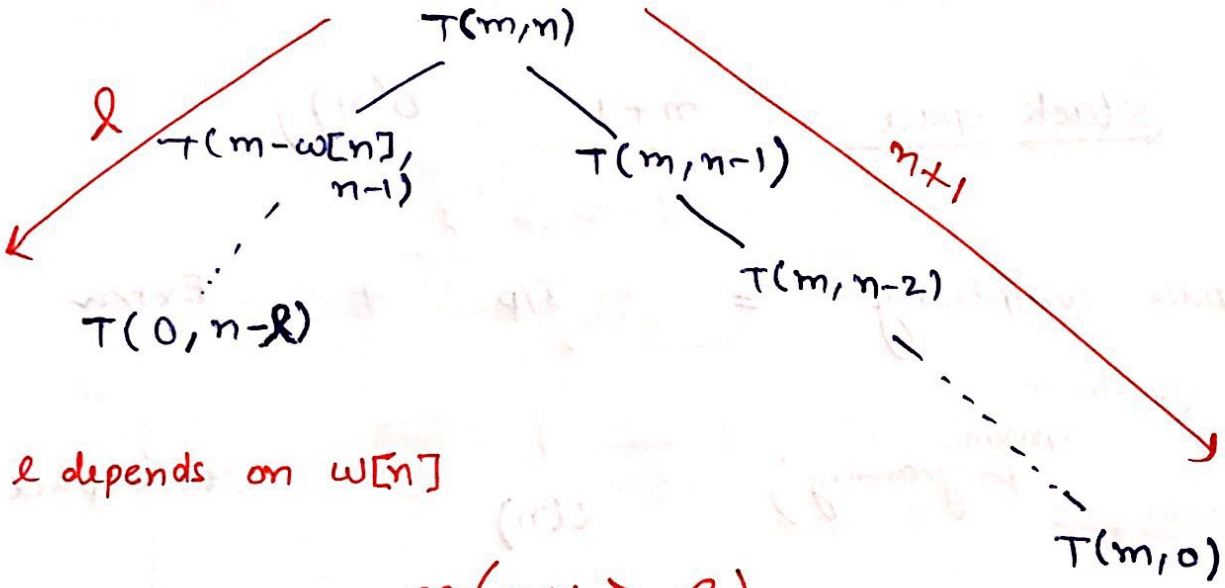
$$= n \ (c)$$

$$T(m,n) = O(n)$$

no object is selected or kept in knapsack

Worst case :-

$$T(m,n) = T(m-w[n], n-1) + T(m, n-1) + c$$



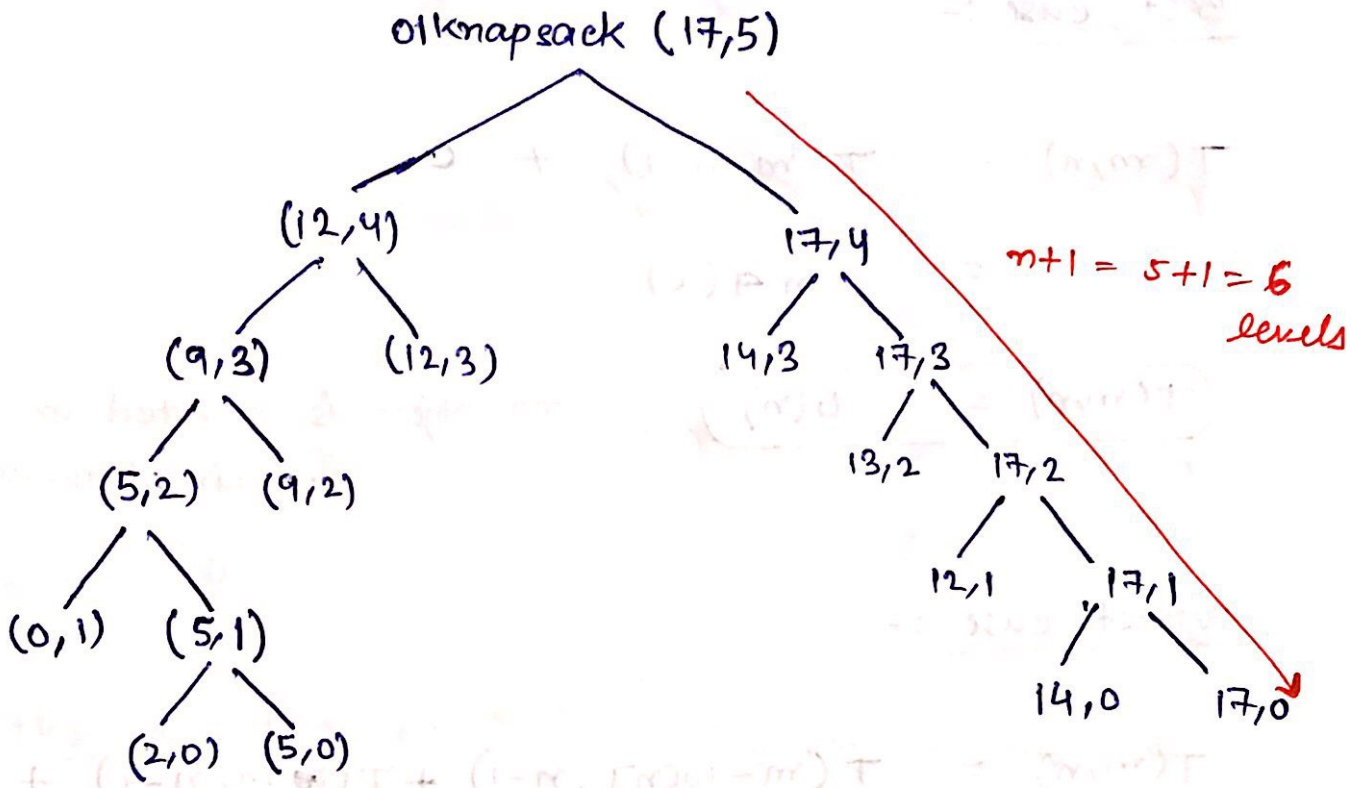
ℓ depends on w[n]

so, $(n+1 \geq \ell)$

$$T(m,n) = c(1 + 2 + 2^2 + 2^3 + \cdots\cdots 2^n)$$

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

without dynamic programming



(stack space = n+1 = O(n))

space complexity = I/p + Extra

(without Dynamic programming)

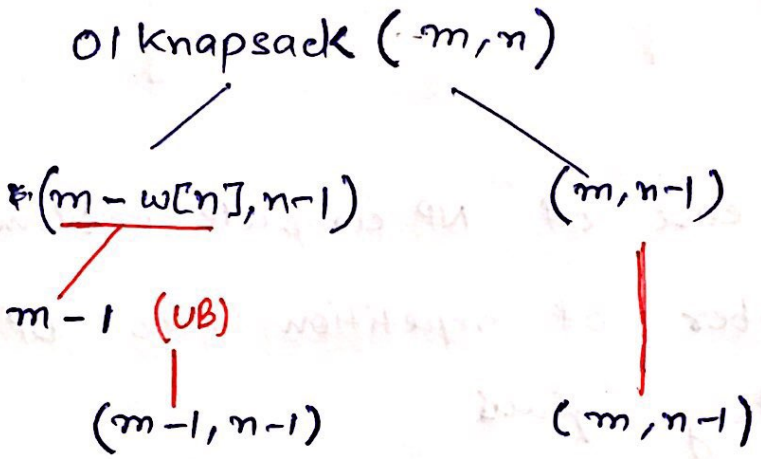
I/p → O(n)

Extra → stack space → O(n)

S(n) = O(n)

In the above recursive tree some func^n calls are repeating so we can apply dynamic programming which will compute only distinct func^n calls.

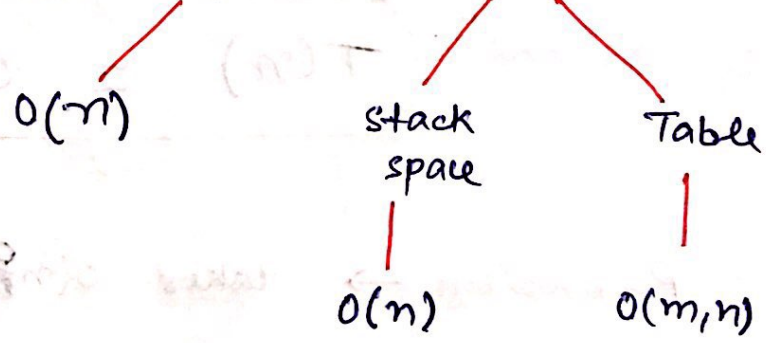
Distinct func^n calls:-



so, Distinct func calls = $O(mn)$

$T(m,n) = O(mn)$ with dynamic programming

Space complexity = I/p + Extra



$S(n) = O(mn)$

	without DP	with DP (Theoritically)
Time complixity	$O(2^n)$	$O(mn)$
Space complixity	$O(n)$	$O(mn)$

NOTE:- In case of NP complete problem which has less number of repetition, so DP will approximately given

$$T(n) \cong O(2^m)$$

Emphirically 0/1 knapsack problem is a NP complete problem so,

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

P-problem → takes $O(n)$ time (polynomial or less)

⇒ Sum of Subsets :-

I/p :- Set of n +ve integer , integer m

O/p :- Find subset such that its sum of elements is m.

eg:-

$$S = \begin{pmatrix} 50 & 80 & 20 & 30 & 70 & 100 & 45 \\ 1 & 2 & 3 & 4 & 5 & 6 & 7 \end{pmatrix}$$

M = 300

SOS(m,n) → finding a subset from given array of n elements ; so that its sum is m.

cases :-

(i) if m=0 — sos(m,n) = SS

sum = 0
so no element
(empty subset)

where (SS = ∅) initially

(ii) if $n = 0$ $\quad S(M,N) = \text{error}$

(300,0)

(iii) if $S[N] > M$ then,

$SOS(m,n) = SOS(m, n-1)$

skip n

(iv) if $S(N) \leq m$ then,

$$SOS(m,n) = \begin{cases} SOS(m - S[n], n-1) & SS = SS \cup S[n] \\ SOS(m, n-1) \end{cases}$$

m n

SOS(300, 7)

(300, 6)

(200, 5)

(130, 4)

(100, 3)

(80, 2)

(0, 1)

Return SS

- Similar to 0/1 knapsack

	without DP	with DP
Time complexity	$O(2^m)$	$O(mn)$
Space complexity	$O(n)$	$O(mn)$

NOTE:- 0/1 knapsack problem polynomially reduced to sum of subsets problem.

Because of 0/1 knapsack is NP complete so SOS is also NP complete.

⇒ Matrix chain Multiplication :-

eg: $[A]_{3\times4} \times [B]_{4\times2}$ $[C]_{3\times2} = [A] \times [B]$

↓

3X2 element

Each element ⟶ 4 multiplication

$$C_{11} = A_{11}B_{11} + A_{12}B_{21} + A_{13}B_{31} + A_{14}B_{41}$$

Total Multiplication for C = 3X2X4 = 24

eg:- $[A]_{5\times 7} \times [B]_{7\times y} = [e]_{5\times 4}$

$= 5\times 4\times 7$

$= 140$ multiplications

eg:- $= ([A]_{3\times 2} \times [B]_{2\times y}) \times [C]_{4\times 5}$

$[D]_{3\times 4}$

$= 3\times 4\times 2$

$= 24$

$[E]_{3\times 5}$

$= 3\times 5\times 4$

$= 60$

$= 60 + 24$

$= 84$

OR

$[A]_{3\times 2} \times ([B]_{2\times 4} \times [C]_{4\times 5})$

$[BC]_{2\times 5} \rightarrow 2\times 5\times 4$

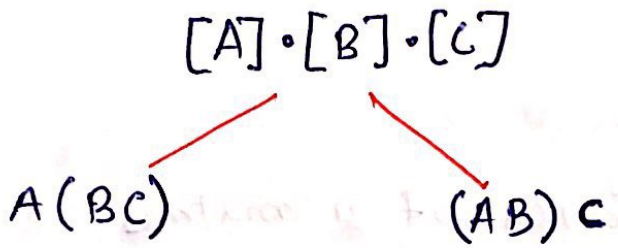
$= 40$

$[ABC]_{3\times 5} \rightarrow 3\times 5\times 2 = 30$

$= 40 + 30 = 70$

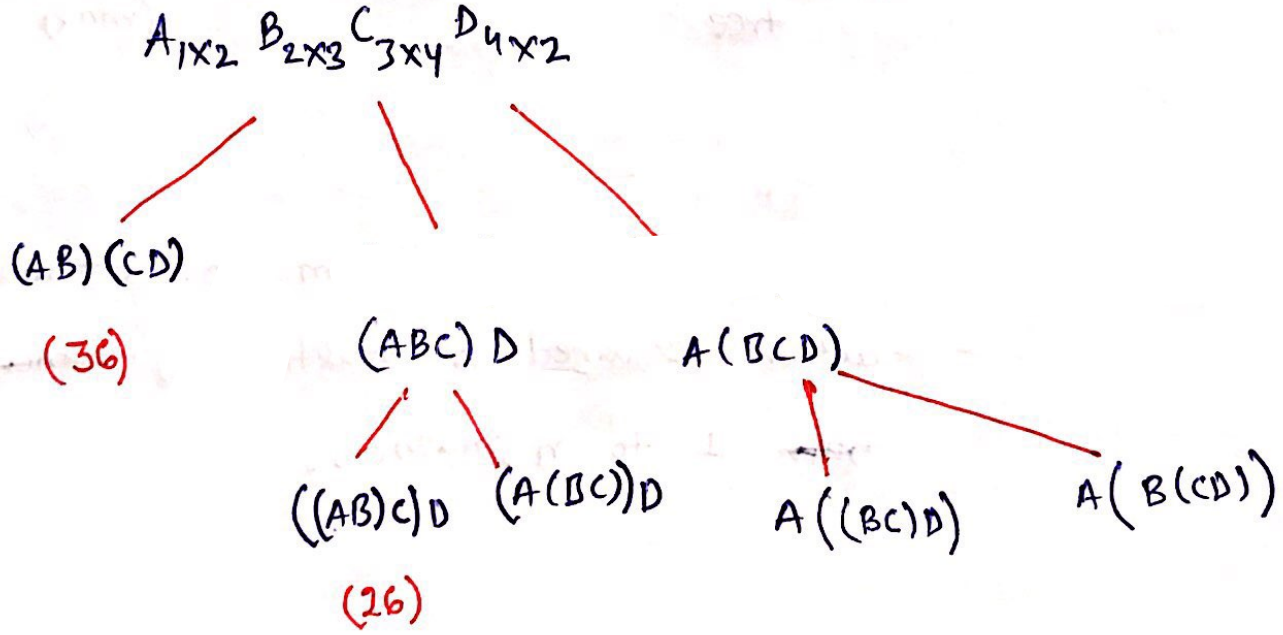
- We can see, in chain matrix multiplication we have different solutions for different possibility.

- So here in "matrix chain multiplication" problem we find the best way where no. of multiplication is minimum.

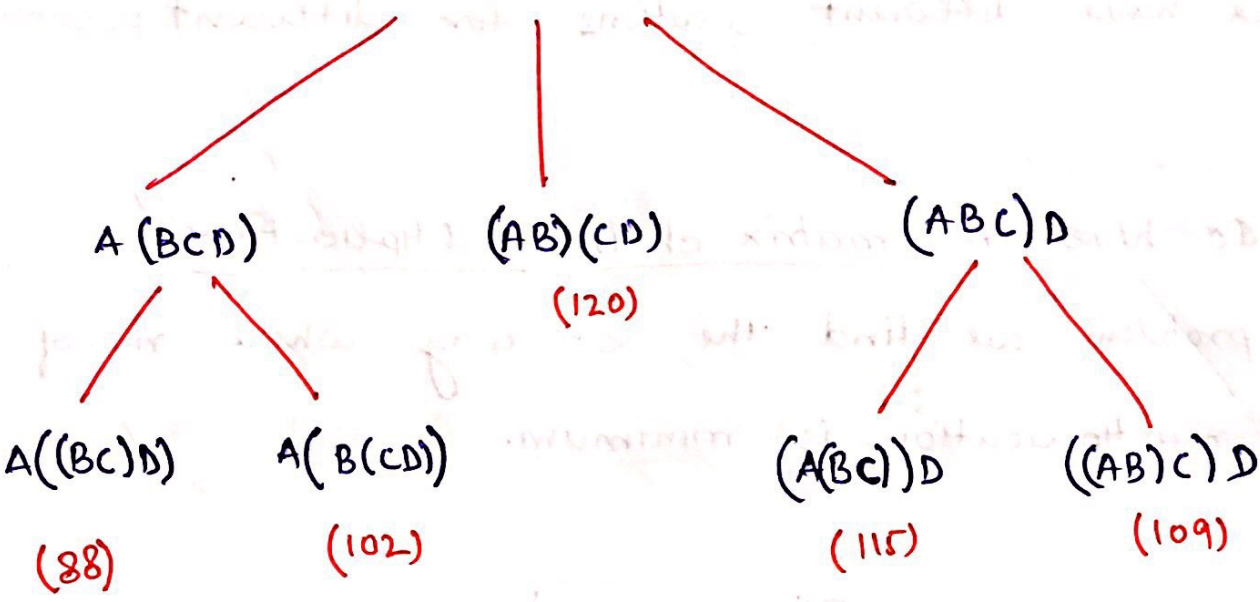


two ways according to associativity.

eg:



eg:- $A_{3\times2}$ $B_{2\times4}$ $C_{4\times5}$ $D_{5\times3}$



NOTE:- Multiplication of n matrices

n-level tree

n-1 child possible (max)

MCM (1,n) → it gives minimum no. of multiplication required to multiply 1 to n matrices.

MCM (1,4) { minimum no. of MUL required to multiply M_1, M_2, M_3, M_4 matrices }

$$MCM(1,4) = \begin{cases} 88 \Leftarrow MCM(1,1) + MCM(2,4) + (P_0 \times P_4 \times P_1) & (P_0P_1 \quad P_1P_4) \\ 120 \Leftarrow MCM(1,2) + MCM(3,4) + (P_0 \times P_4 \times P_2) & (P_0P_2 \quad P_2P_4) \\ 109 \Leftarrow MCM(1,3) + MCM(4,4) + (P_0 \times P_4 \times P_3) & (P_0P_3 \quad P_3P_4) \end{cases}$$

$\min(88, 120, 109)$

$= 88$

Assume, $M_5 = P_4 \times P_5$ $\qquad M_7 \text{ ---- } M_{13} = P_6 \times P_{13}$

$M_{100} = P_{99} \times P_{100}$ $\qquad M_i \text{ ---- } M_j = P_{i-1} \times P_j$

Recurrence Relation for no. of multiplication :-

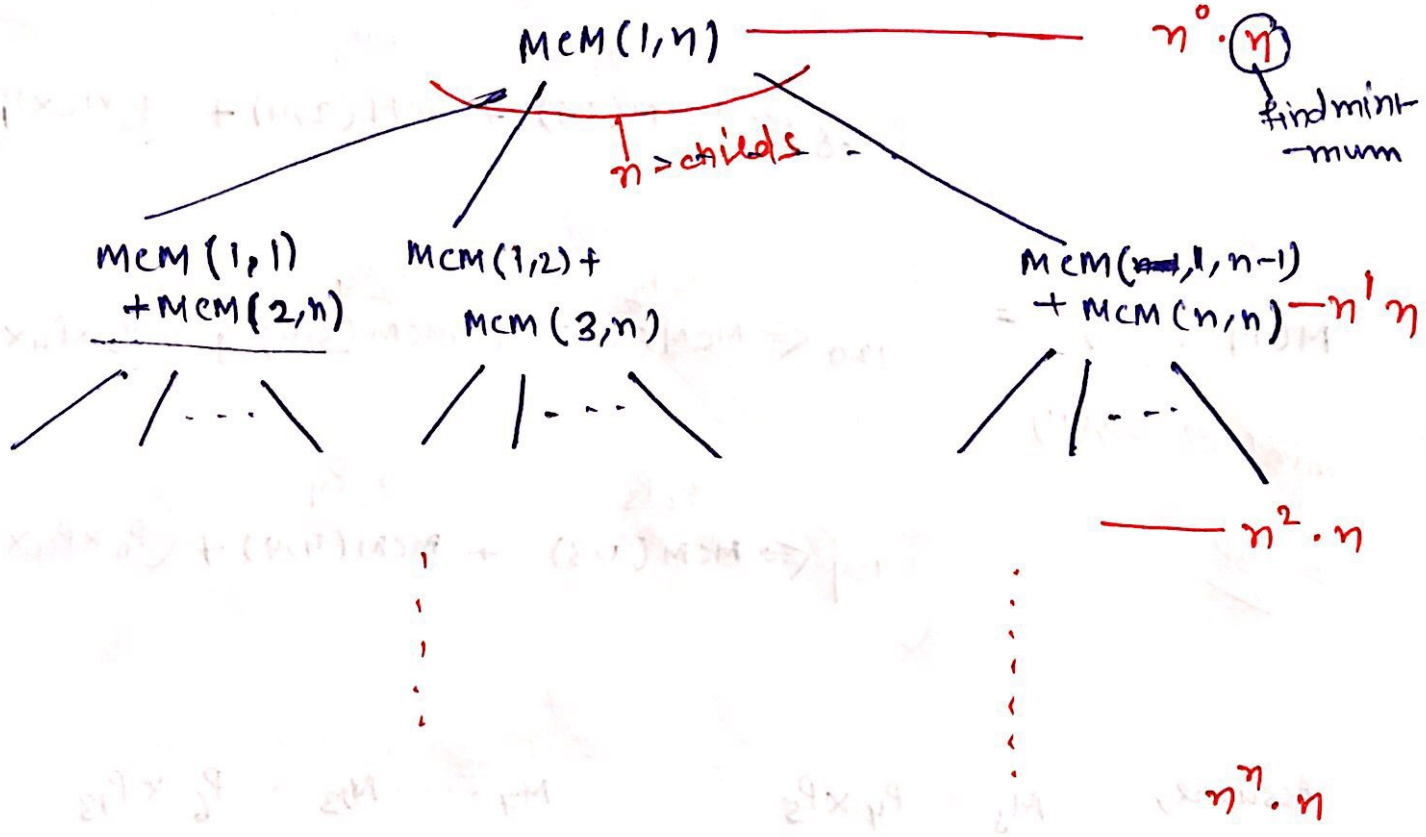
$$MCM(i,j) = \begin{cases} 0 & \text{if } i == j \\ \min\left(\underbrace{MCM(i,K) + MCM(K+1,j) + P_{i-1}P_KP_j \quad \forall \ (i \le K \le j-1)}_{n-1 \text{ terms}} \right) & \text{if } i < j \end{cases}$$

$1, n$

$O(n)$

to find minimum from $n-1$ terms

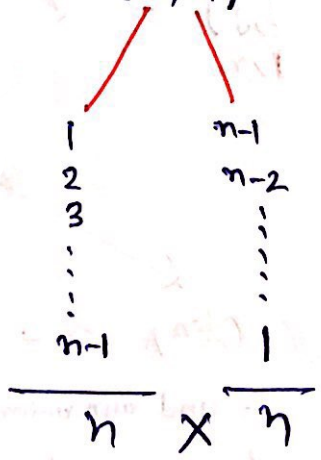
selection sort 1 pass



$T(n) = O(n^n)$ without dynamic programming

Since recursion tree contain same func^n call so apply dynamic programming.

Distinct func^n calls - MCM(1,n)



$= n^2$

Time complexity = $O(n) \times O(n^2)$

$O(n)$ — find minimum

$O(n^2)$ — distinct calls

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

Space complexity = I/P + Extra

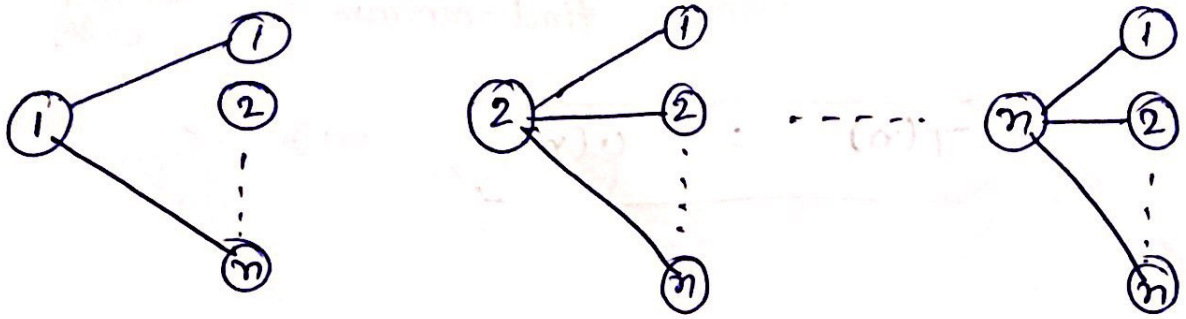
I/P = n

Extra: Stack = n (n-level tree), Table = n^2

$S(n) = O(n^2)$ with DP

	without DP	with DP
Time complexity	$O(n^n)$	$O(n^3)$
Space complexity	$O(n)$	$O(n^2)$

⇛ All Pair Shortest Path :-



$(n^2$ - pairs)

+ve - edge weights :-

$|V| \times$ Dijkstra $\Rightarrow$ $(V * (V+E) \log V)$

apply dijkstra
V times
for each vertex
as source

-ve edge weights :-

$|V| \times$ Bellmen's ford $\Rightarrow$ $(V(VE))$

Unweighted weights :-

$|V| *$ BFT $\Rightarrow$ $(V(V+E))$

Floyd Warshall's algo:- $O(V^3)$

(+ve, -ve, weighted, un-weighted)

15 Jul 2019