AA Mod 3

- **Dynamic Programming**

Complex problems are broken down into simpler sub-problems. By solving each subproblem only once and storing results, it avoids redundant computations, leading to more efficient solutions.

1. **Top-down Approach (Memoization)**
   Start with final solution and recursively break it down into smaller subproblems.
2. **Bottom-up Approach (Tabulation)**
   Start with the smallest subproblems and gradually build up the final solution.

- **Longest Common Subsequence**

Recursive solution $c[i,j] = \begin{cases} c[i-1, j-1] + 1 & x[i] = y[j] \\ \max(c[i, j-1], c[i-1, j]) & \text{otherwise} \end{cases}$

```
LCS-Length(X, Y)          Initialize table c of size (m+1)×(n+1)
m = length(X)
n = length(Y)
for i = 1 to m            c[i,0] = 0      Set base cases for 1st row
for j = 1 to n            c[0,j] = 0      and 1st column to 0
for i = 1 to m            Iterate through each symbol x_i in sequence X
for j = 1 to n            For each symbols in X, iterate through each symbol
    if (X_i == Y_j)       Y_j in sequence Y.
        c[i,j] = c[i-1, j-1] + 1
current symbols
contribute to LCS length  else c[i,j] = max(c[i-1,j], c[i,j-1])
                                          ↓ left   ↓ above
    return c
```

This means that LCS doesn't include current symbols

Time Complexity: $O(m \times n)$
Space Complexity: $O(m \times n)$

Example: X = ABCB
Y = BDCAB
LCS (X, Y) = BCB

	j	0	1	2	3	4	5
i		Y_j	B	D	C	A	B
0	x_i	0	0	0	0	0	0
1	A	0	0	0	0	1	1
2	B	0	1	1	1	1	2
3	C	0	1	1	2	2	2
4	B	0	1	1	2	2	3

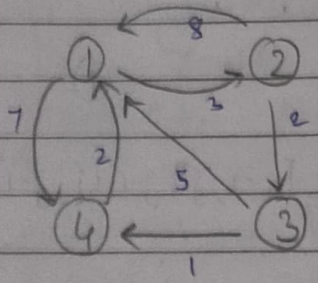
for (i, 4)
if ($X_i == Y_j$)
c[i, j] = c[i-1, j-1] + 1
else c[i, j] = max (c[i-1, j], c[i, j-1])
for i, j

Length: 3 LCS: BCB : reversed
LCS: BCB : straight

- **Floyd Warshall Algo**

Used to find the shortest path b/w all pairs of nodes in a weighted graph. Highly efficient and can handle graphs with both +ve and -ve edge weights suitable for solving wide range of network & connectivity problems

Time Complexity: $O(V^3)$ Space Complexity: $O(V^2)$



A^0 =

	1	2	3	4
1	0	3	∞	7
2	8	0	2	∞
3	5	∞	0	1
4	2	∞	∞	0

A^1 =

	1	2	3	4
1	0	3	∞	7
2	8	0	2	15
3	5	8	0	1
4	2	5	∞	0

smallest cost when going thru 1

A^2 =

	1	2	3	4
1	0	3	5	7
2	8	0	2	15
3	5	8	0	1
4	2	5	7	0

A^3 =

	1	2	3	4
1	0	3	5	6
2	7	0	2	3
3	5	8	0	1
4	2	5	7	0

A^4 =

	1	2	3	4
1	0	3	5	6
2	5	0	2	3
3	3	6	0	1
4	2	5	7	0

adjacency matrix

```
function FloydWarshall(W, n):        → no. of vertices
    for k ← 1 to n do
        for i ← 1 to n do
            for j ← 1 to n do
                D[i][j] = min(D[i][j], D[i][k] + D[k][j])
return D
```

↳ Distance matrix containing shortest distances b/w vertices

$A^3[1,2] = \min(A^2[1,2], A^2[1,3] + A^2[3,2])$

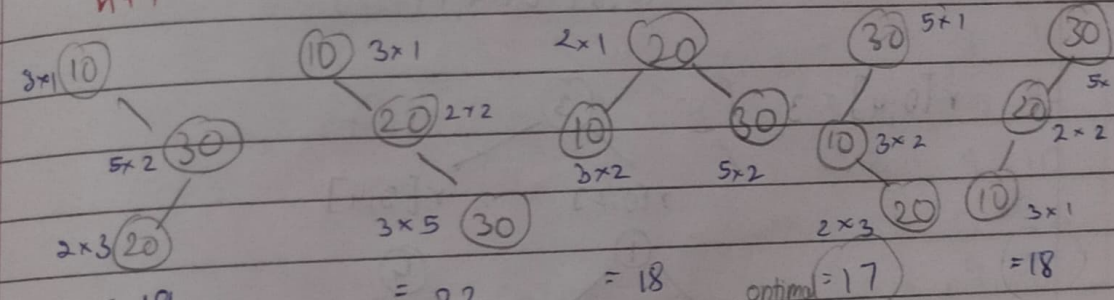
i: 1, j: 2, k: 3

• Optimal Binary Search Tree

$\frac{^{2n}C_n}{n+1}$: no. of trees

Example:- keys: 10 20 30
freq: 3 2 5

$^{2\times3}C_3 / 4 = 5$



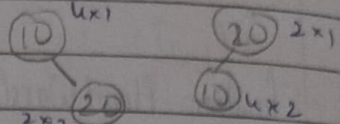
	1	2	3	4
keys:	10	20	30	40
freq:	4	2	6	3

$10^3 \rightarrow 3^{rd}$ graph

	0	1	2	3	4
0	0	4	8^1	20^3	26^3
1		0	2	10^3	16^3
2			0	6	12^3
3				0	3
4					0

Page No.
Date

$l = j - i = 0$	$l = j - i = 1$	$c[0,1] = 4$
$0 - 0 = 0$	$1 - 0 = 1$ $(0,1)$	$c[1,2] = 2$
$1 - 1 = 0$	$2 - 1 = 1$ $(1,2)$	$c[2,3] = 6$
$2 - 2 = 0$	$3 - 2 = 1$ $(2,3)$	$c[3,4] = 3$
$3 - 3 = 0$	$4 - 3 = 1$ $(3,4)$	
$4 - 4 = 0$		

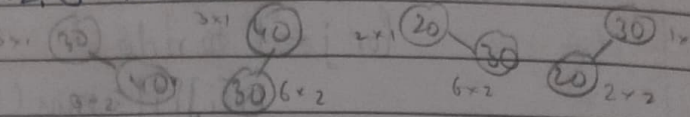


$l = j - i = 2$ $\quad c[0,2] = 8$

$2 - 0 = 2$ $(0,2)$ $\quad c[1,3] = 6 + 4 = 10$

$3 - 1 = 2$ $(1,3)$ $\quad c[2,4] = 12$

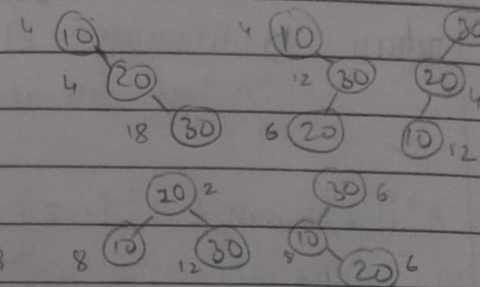
$4 - 2 = 2$ $(2,4)$



$l = j - i = 3$ $\quad c[0,3] = 20$

$3 - 0 = 3$ $(0,3)$ $\quad c[1,4] = 16$

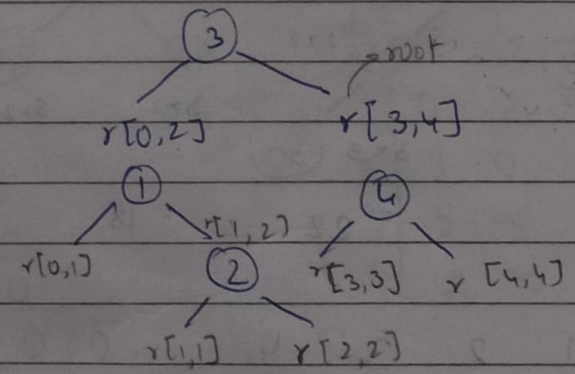
$4 - 1 = 3$ $(1,4)$



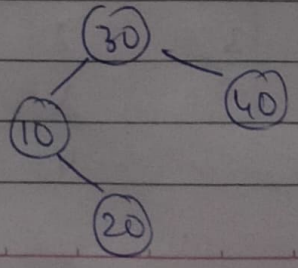
$$c[0,4] = \min \begin{cases} c[0,0] + c[1,4] & 0+16 \\ c[0,1] + c[2,4] & 4+12 \\ c[0,2] + c[3,4] & 8+3 \\ c[0,3] + c[4,4] & 20+0 \end{cases} + 15$$

→ total freq.

$r[0,4] = 3$



OBST is



Time Complexity: $O(n^3)$

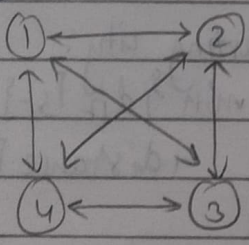
Space Complexity: $O(n^2)$

function optimalBST(keys, freq, n):
1. Initialize cost and root arrays
2. Initialize diagonal elements of cost array with frequencies cost[i][i] = freq[i]
3. Compute optimal cost and root for each subproblem
4. for length from 1 to n:
5. for i from 1 to n - length + 1:
6. j = i + length - 1
7. Compute cost[i][j] and root[i][j] using nested loops
8. Return the root array and optimal cost

7: cost[i][j] = INF
 for r from i to j: for each sub problem, it computes optimal cost
 ← temp = cost[i][r-1] + cost[r+1][j] + sum(freq[i..j]) sum of costs of left & right subtrees along with sum of frequencies of the keys in range i to j
 if temp < cost[i][j]:
  cost[i][j] = temp it updates cost array & root array accordingly
  root[i][j] = r

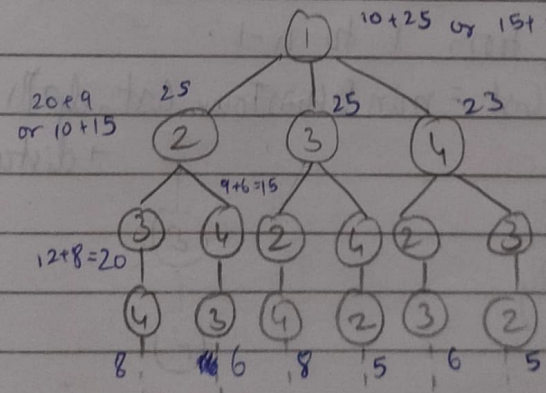
- **Travelling Salesman Problem**

Example:-



A =

	1	2	3	4
1	0	10	15	20
2	5	0	9	10
3	6	13	0	12
4	8	8	9	0



Time Complexity:-
$O(2^n n^2)$

Space complexity:-
$O(n\,2^n)$

$g(1, \{2,3,4\}) = \min \{c_{1k} + g(k, \{2,3,4\} - \{k\})\}$

$g(i, s) = \min \{c_{ik} + g(k, s - \{k\})\}$

$g(2, \phi) = 5$ $g(3, \phi) = 6$ $g(4, \phi) = 8$

$g(2, \{3\}) = 15$ $g(2, \{4\}) = 8$ $g(3, \{2\}) = 5$

$g(3, \{4\}) = 8$ $g(4, \{2\}) = 5$ $g(4, \{3\}) = 6$

$g(2, \{3,4\}) = 25$ $g(3, \{2,4\}) = 25$

$g(4, \{2,3\}) = 25$

$g(1, \{2,3,4\}) = \min \{c_{12} + g(2, \{3,4\}), c_{13} + g(3, \{2,4\}), c_{14} + g(4, \{2,3\})\}$

35 $(10 + 25)$ $15 + 25$ $20 + 25$

→ b/w pair of cities [matrix]

algo initialize dp array to store min cost of visiting subset of cities exactly once & ending at city j

```
function TSP(distance):
    n := number of cities
    Initialize a 2D array dp of size 2^n × n to store subproblem sol^n
    Initialize dp[0][0] to 0 (base case)
    for subset size from 1 to n-1: excluding empty & full set of cities
        for each subset s of cities of size subset size:
            for each city j in s:
                if j is not the starting city:
                    dp[s][j] = min{dp[s-{j}][k] + distance[k][j]}
    Set minTourCost = INF
    for each city j from 1 to n-1:
        minTourCost = min(minTourCost, dp[(1<<n)-1][j] + distance[j][0])
    return minTourCost
```

it computes the cost of reaching city k + dist from k to j

after that it finds min TC by adding dist from last city back to starting city