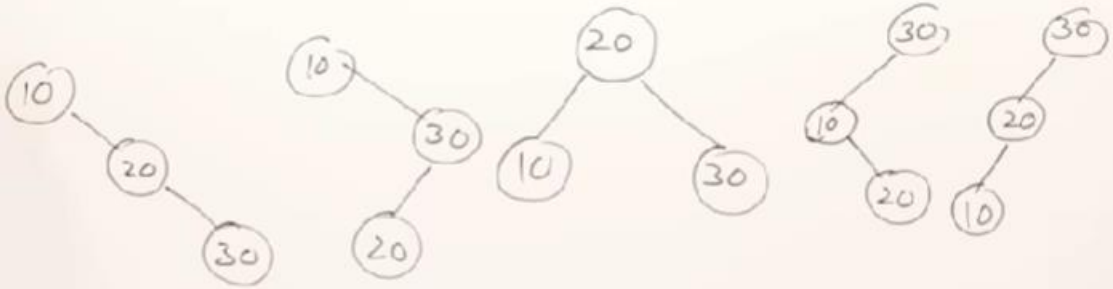
Optimal Binary Search Tree

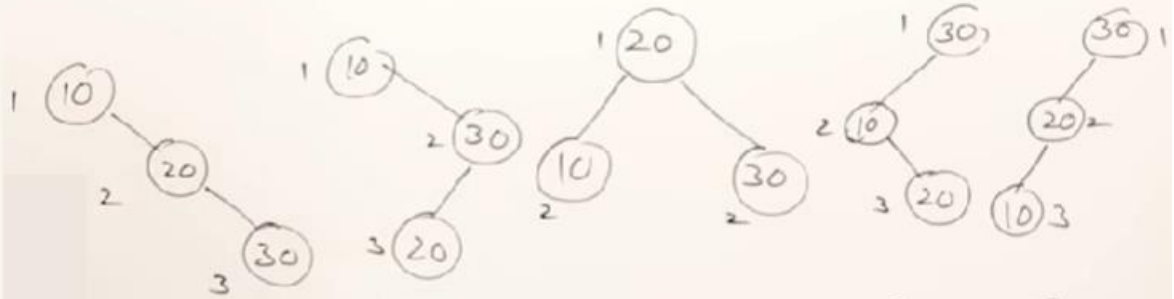
Keys → 10, 20, 30



$$\frac{2nC_n}{n+1} \qquad \frac{2\times3C_3}{3+1}=5$$

Optimal Binary Search Tree

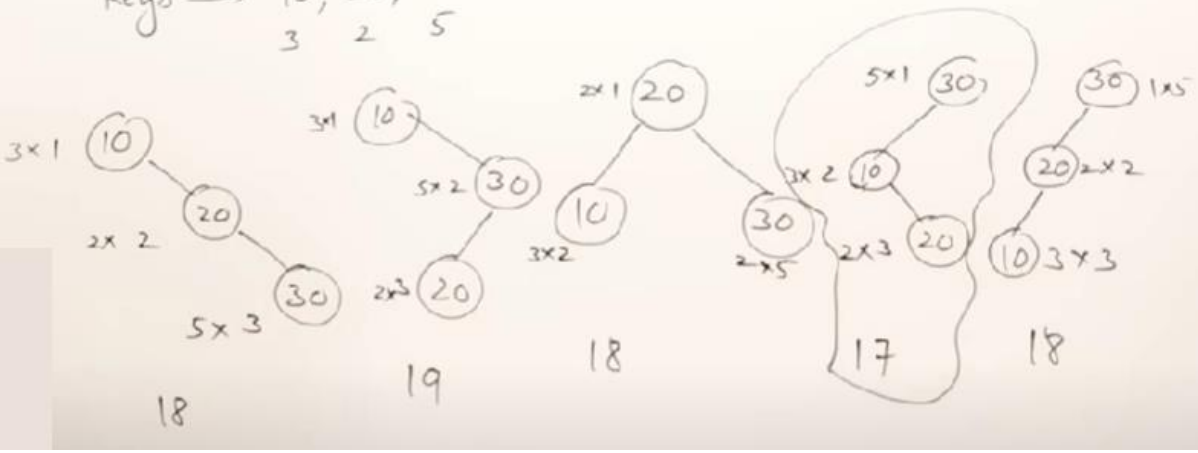
Keys → 10, 20, 30



$\frac{1+2+3}{3}=\frac{6}{3}=2$ $\frac{1+2+3}{3}=\frac{6}{3}=2$ $\frac{5}{3}$ $\frac{6}{3}$ $\frac{6}{3}$

Optimal Binary Search Tree

Keys → 10, 20, 30
3 2 5



Optimal Binary Search Tree

	1	2	3	4
Keys →	10	20	30	40
frequency →	4	2	6	3

$l = j - i = 0$

0-0=0
1-1=0
2-2=0
3-3=0
4-4=0

i \ j	0	1	2	3	4
0	0				
1		0			
2			0		
3				0	
4					0

Optimal Binary Search Tree

	1	2	3	4
Keys →	10	20	30	40
frequency →	4	2	6	3

c[0,1]=4 c[1,2]=2 c[2,3]=6 c[3,4]=3

10	20	30	40
4	2	6	3

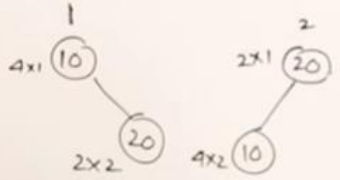
$l = j - i = 1$

1-0=1 (0,1)
2-1=1 (1,2)
3-2=1 (2,3)
4-3=1 (3,4)

i \ j	0	1	2	3	4
0	0	4			
1		0	2		
2			0	6	
3				0	3
4					0

c[0,2]

10 20
4 2



4+4=8 8+2=10

$w(0,4) = \sum_{i=1}^{4} f(i)$

$l = j - i = 2$

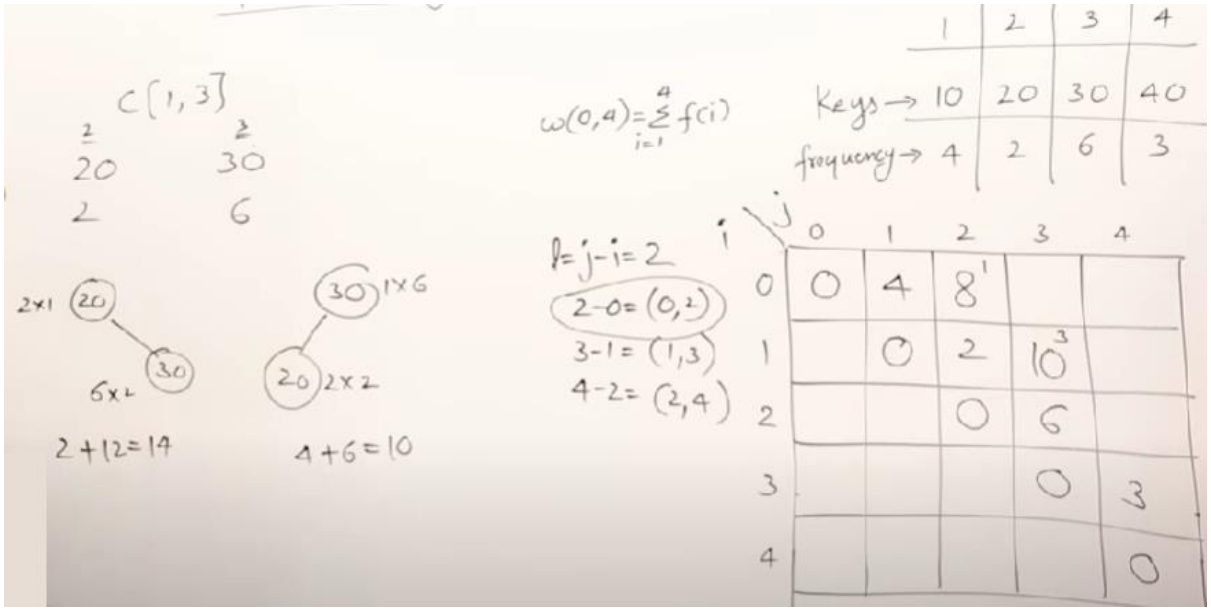
2-0=(0,2)
3-1=(1,3)
4-2=(2,4)

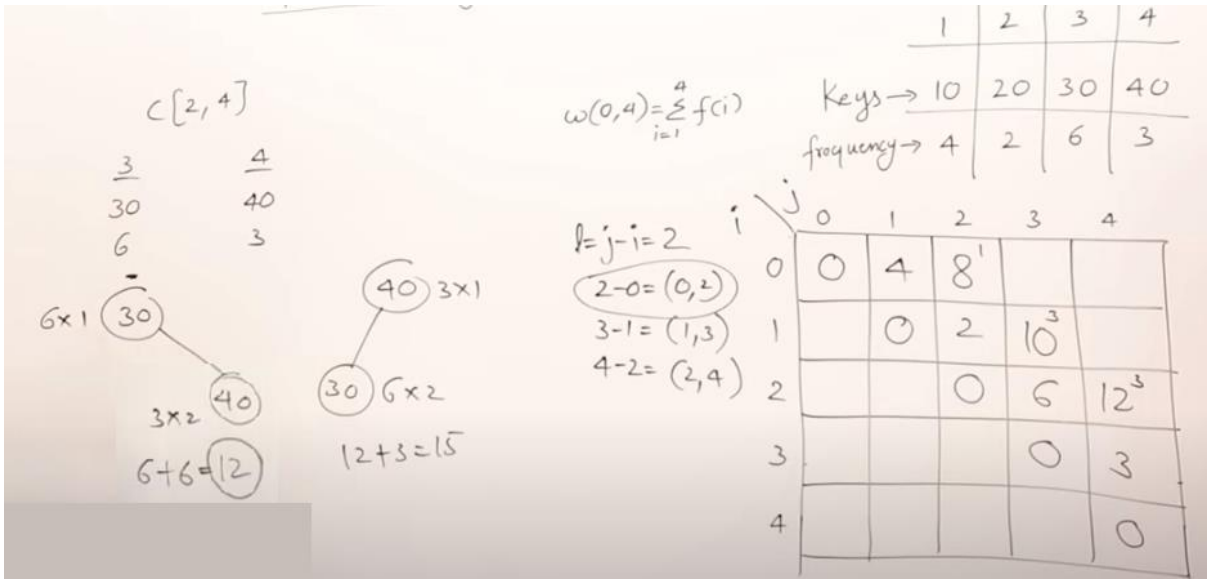
i \ j	0	1	2	3	4
0	0	4	8		
1		0	2		
2			0	6	
3				0	3
4					0

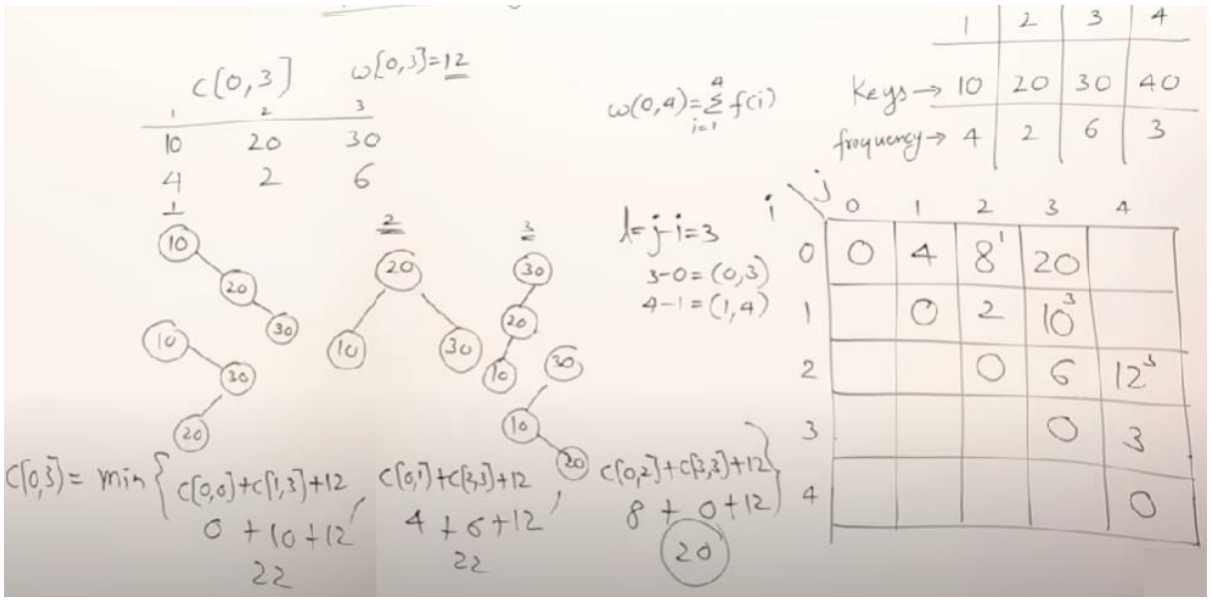
c(0,0)+c[1,2]+w[0,2] c[0,1]+c[2,2]+w[0,2]

0 + 2 +4+2 4 +0 + 6 = 10

8


c[1,3]
2 3
20 30
2 6
2×1 20 30 6×2
2+12=14
30 1×6 20 2×2
4+6=10
ω(0,4)=Σ f(i) i=1 to 4
Keys → 10 20 30 40
frequency → 4 2 6 3
l=j-i=2
2-0=(0,2)
3-1=(1,3)
4-2=(2,4)
i j 0 1 2 3 4
0 0 4 8¹
1 0 2 10³
2 0 6
3 0 3
4 0


c[2,4]
3 4
30 40
6 3
6×1 30 40 3×2
6+6=12
40 3×1 30 6×2
12+3=15
ω(0,4)=Σ f(i) i=1 to 4
Keys → 10 20 30 40
frequency → 4 2 6 3
l=j-i=2
2-0=(0,2)
3-1=(1,3)
4-2=(2,4)
i j 0 1 2 3 4
0 0 4 8¹
1 0 2 10³
2 0 6 12⁵
3 0 3
4 0


c[0,3] ω[0,3]=12
1 2 3
10 20 30
4 2 6
c[0,3]= min { c[0,0]+c[1,3]+12 0+10+12 22
c[0,1]+c[2,3]+12 4+6+12 22
c[0,2]+c[3,3]+12 8+0+12 20
ω(0,4)=Σ f(i) i=1 to 4
Keys → 10 20 30 40
frequency → 4 2 6 3
l=j-i=3
3-0=(0,3)
4-1=(1,4)
i j 0 1 2 3 4
0 0 4 8¹ 20
1 0 2 10³
2 0 6 12⁵
3 0 3
4 0

	1	2	3	4
Keys →	10	20	30	40
frequency →	4	2	6	3

$C[1,4]=16 \quad w[1,4]=11$

$w(0,4)=\sum_{i=1}^{4} f(i)$

2	3	4
20	30	40
2	6	3

$l=j-i=3$

$3-0=(0,3)$

$4-1=(1,4)$

$$C[1,4]=\min\begin{cases} C[1,1]+C[2,4] & 0+12 \\ C[1,2]+C[3,4] & 2+3 \\ C[1,3]+C[4,4] & 10+0 \end{cases} +11=16$$

i \ j	0	1	2	3	4
0	0	4	8^1	20^3	
1		0	2	10^3	16^3
2			0	6	12^3
3				0	3
4					0

	1	2	3	4
Keys →	10	20	30	40
frequency →	4	2	6	3

$C[0,4]=26 \quad w[0,4]=15$

$w(0,4)=\sum_{i=1}^{4} f(i)$

1	2	3	4
10	20	30	40
4	2	6	3

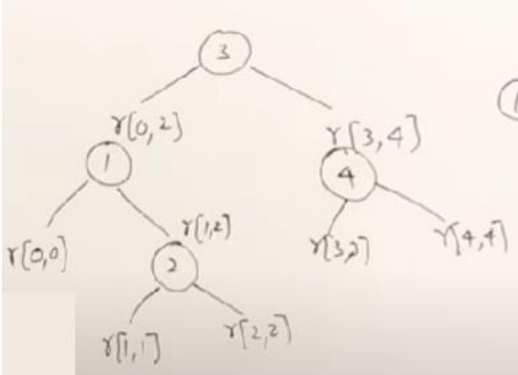
$l=j-i=4$

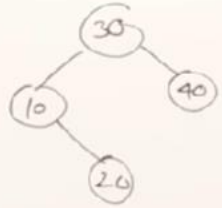
$4-0=(0,4)$

$$C[0,4]=\min\begin{cases} C[0,0]+C[1,4] & 0+16 \\ C[0,1]+C[2,4] & 4+12 \\ C[0,2]+C[3,4] & 8+3 \\ C[0,3]+C[4,4] & 20+0 \end{cases} +15 = 11+15$$

i \ j	0	1	2	3	4
0	0	4	8^1	20^3	
1		0	2	10^3	16^3
2			0	6	12^3
3				0	3
4					0

$$C[i,j]=\min_{i<k\le j}\{C[i,k-1]+C[k,j]\}+w(i,j)$$





	1	2	3	4
Keys →	10	20	30	40
frequency →	4	2	6	3

i \ j	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

Optimal Binary Search Tree (BST) using Dynamic Programming

Binary Search Tree (BST): A binary tree in which every node follows the property that the key in the node's left subtree is less than the node's key, and the key in the node's right subtree is greater than the node's key.

Optimal Binary Search Tree: A BST that provides the minimum possible search time for a given sequence of keys.

Dynamic Programming Approach:

Dynamic programming is used to find the optimal BST efficiently by avoiding the exponential time complexity associated with the brute-force approach. The approach involves breaking down the problem into smaller subproblems and storing solutions to those subproblems to avoid redundant computations.

Key Concepts:

Subproblem Definition: Define the problem recursively, breaking it down into smaller subproblems.

Memorization: Store the solutions to subproblems in a table to avoid redundant computations.

Bottom-up Approach: Solve smaller subproblems first and build up solutions to larger subproblems iteratively.

Cost Function: Define a cost function that measures the overall cost of a BST, typically considering search time and probabilities of accessing keys.

Algorithm Steps:

Initialization: Initialize a table to store solutions to subproblems.

Bottom-up Calculation: Iterate through all possible subtrees and compute the optimal cost for each subtree.

Optimal Substructure: Define the optimal cost for a subtree in terms of optimal costs of its child subtrees.

Final Solution: Compute the optimal cost of the entire BST by considering all possible root nodes.

Step 1: Read n symbols with probability p_i.
Step 2: Create the table $C[i, j]$, $1 \leq i \leq j + 1 \leq n$.
Step 3: Set $C[i, i] = p_i$ and $C[i - 1, j] = 0$ for all $i \in [n]$.
Step 4: Recursively compute the following relation:

$$C[i, j] = C[1 \ldots k + 1] + C[k + 1 \ldots j] + \sum_{m=1}^{n} p_m, \quad \text{for all i and j}$$

Step 5: Return $C[1 \ldots n]$ as the maximum cost of constructing a BST.
Step 6: End.

Time Complexity:

The time complexity of constructing an OBST is O(n^3), where n is the number of keys. However, with some optimizations, we can reduce the time complexity to O(n^2). Once the OBST is constructed, the time complexity of searching for a key is O(log n), the same as for a regular binary search tree.