Dynamic Programming

Technique used in computes programming to efficiently by overlapping sub problems.

→ Breaking down problems into smaller sub problems and solving them independently and saves the solution avoiding redundant computations.

Example :-

Calculation of fibonnaci sequence.

Algorithm :-

$F(0) = 0 \Rightarrow 0$

$F(1) = 1 \Rightarrow 1$

$F(2) = F(1) + F(0) \Rightarrow 1$

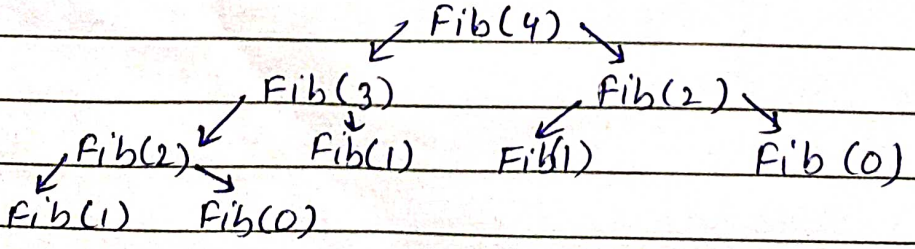
$F(3) = F(2) + F(1) \Rightarrow 2$

$F(4) = F(3) + F(2) \Rightarrow 3$

Dynamic Programming

- Top down approach : Recursive
- Bottom up approach : Iterative

Top down approach of Recursion in DP :-



Bottom Up approach :-

Fib(0) = 0 => 0
Fib(1) = 1 => 1
Fib(2) = Fib(1) + Fib(0) => 1
Fib(3) = fib(2) + fib(1) => 2
Fib(4) = Fib(3) + Fib(2) => 3

* DP finds global optimum.

Elements of Dynamic Programming :-

1. Optimal Sub structure : Solutions of larger problems built from smaller subproblems
2. Memoization : store and reuse solutions to avoid re-calculation
3. Recurrence Relations : express solution in terms of subproblems.

State Representation:

P.T.O

Longest Common Subsequence :-

→ LCS is defined as the longest subsequence that is common to all given sequences.

→ Order should be maintained

→ Longest Subsequence means maximum length among the sets.

Algorithm :-

1. Create a table (n x m) where n is the size of X and Y is the m.
   ↳ excluding 0th index
2. Set zeroth row and column to zero.
3.0 Increment counter for common elements.
3.1 If no common elements, fill T[i, j] with max (T[i-1, j] and T[i, j-1]

Example :-

X = A C A D B

Y = C B D A

1. Create a table (n+1) * (m+1) and assign zero to zeroth index.
2. Fill the cells using the logic :-
   2.1 If characters of current row and column matches, fill the current cell by adding one of the diagonal element. Point the arrow towards the selected diagonal cell.
   2.2 Else take the maximum value from the previous row and column. Point the arrow to selected maximum.
   2.2.1 If they are equal point to any.

3. The value in the last row and column is the length of longest common subsequence.

4. In order to find the longest common sequence follow the arrow from the bottom right and the common elements are the longest common subsequence in X and Y.

		C	B	D	A
	0	0	0	0	0
A	0	↖ 0	← 0	← 0	↖ 1
C	0	↖ 1	← 1	← 1	1 ↑
A	0	1 ↑	1 ↑	1 ↑	↖ 2
D	0	1 ↑	1 ↑	↖ 2	2 ↑
B	0	1 ↑	↖ 2	2 ↑	2 ↑ ← length of common subsequence

* If different character then max of above row and left column.
* If same then fill 1 + diagonal value (upper diagonal value)

* follow the arrow and all the common elements inside this path form LCS.

Algorithm :-

→ X and Y be two sequences
→ Initialize two table X.length * Y.length
→ LCS[0][ ] = 0
→ LCS[ ][0] = 0
→ Start from LCS[1][1]
→ Compare X[i] and Y[j]

If X[i] = Y[j]
LCS[i][j] = 1 + LCS[i-1, j-1]
Point an arrow to LCS[i][j]

Else
LCS[i][j] = max (LCS[i-1][j], LCS[i][j-1])
Point an arrow to max (LCS[i-1][j], LCS[i][j-1])

Time Complexity :-

O(

Space Complexity :-

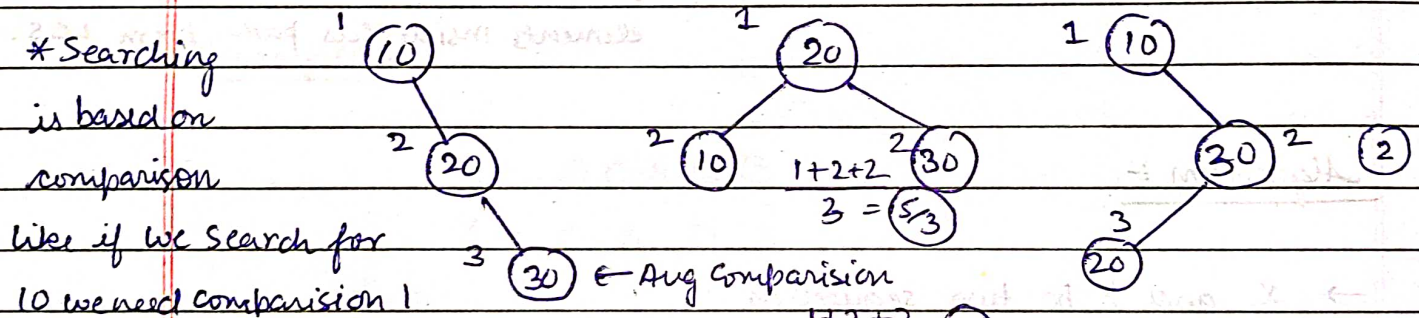
O(1)

Optimal Binary Search Tree (OBST) :-

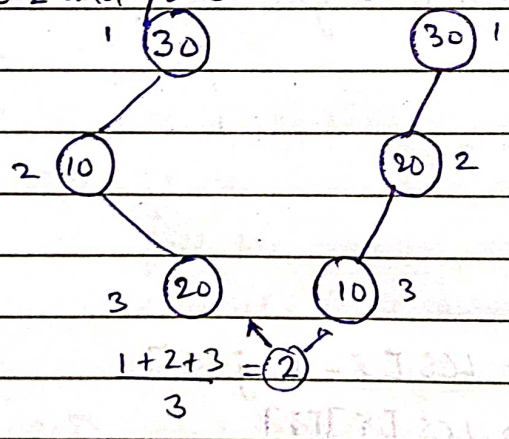
→ Binary Search Tree have fewer level, reducing search time.
→ OBST further optimizes by considerations key frequencies

Keys → 10, 20, 30

For, given keys how many binary search trees can be constructed. :-



*Searching is based on comparison like if we search for 10 we need comparision 1 and 20 we need 2 and for 30 we need 3.



No. of binary trees possible

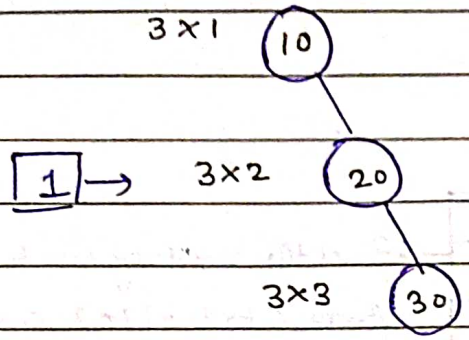
$\Rightarrow \frac{2^n C_n}{n+1} \Rightarrow$ 5

Suppose given →

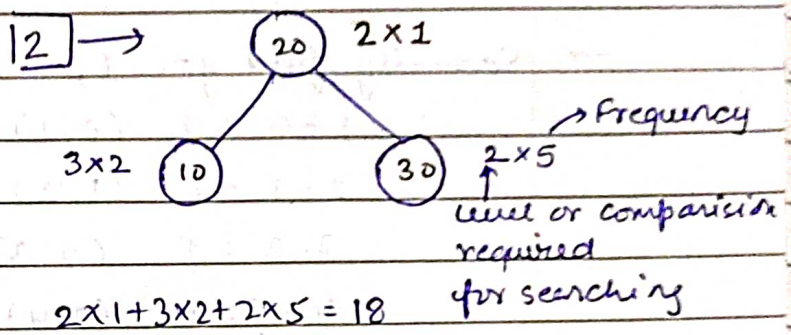
Keys → 10, 20, 30

Frequency → 3, 2, 5

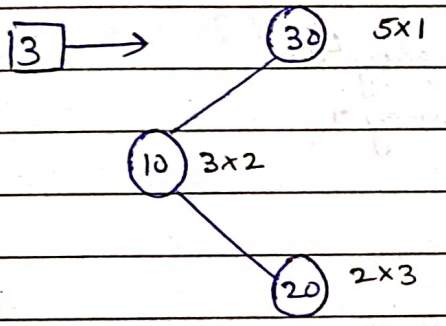
Calculation of cost with frequency



$3\times1+3\times2+3\times3=18$



$2\times1+3\times2+2\times5=18$



$5\times1+3\times2+3\times3 \Rightarrow 17$ * since the tree [2] has less comparision and also balanced but the frequency calculation [3] has minimum

To find out which tree search construction is optimal by the given frequency :-

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

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

$w(i,j)$ = sum of all frequencies from i, j

k = selected Key value or index number

i = row, j = column.

P.T.O

→ First find $j-i=0$,

$0-0=0$

$1-1=0$

$2-2=0$

$3-3=0$

$4-4=0$

rows i \ j columns	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

→ Secondly find $j-i=1$,

$1-0=1$ $(0,1)$ →

$2-1=1$ $(1,2)$ → Find the cost of each vertex from 0 to 1, 1 to 2 and so on

$3-2=1$ $(2,3)$ →

$4-3=1$ $(3,4)$ →

Cost :-

C[0,1] = (10) Key with frequency 4.

C[1,2] = (20) Key with frequency 2

C[2,3] = (30) ⇒ 6

C[3,4] = (40) ⇒ 3

→ Find $j-1=2$,

~~0-2~~ $2-0=(0,2)$

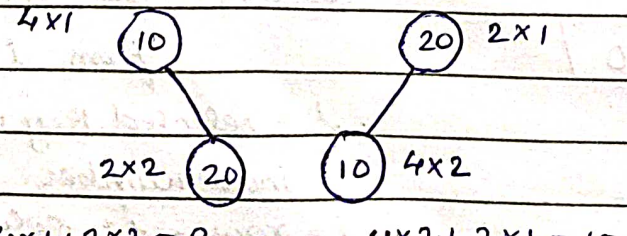
~~1-3~~ $3-1=(1,3)$

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

Cost :-

C[0,2] = (10), (20)

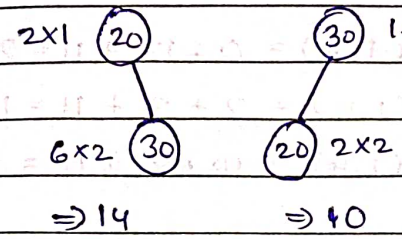
(10) = 4 , (20) = 2



* take minimum and add to matrix. 8^1 → Index

c[1,3] = (20), (30)

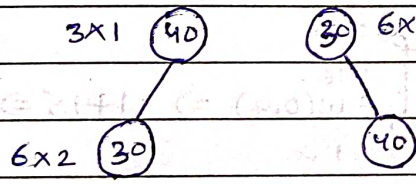
(20) = 2 , (30) = 6



Fill the table → 1 0 3 → Key no.

c[2,4] ⇒ (20), (40)

(20) = 2 , (40) = 3



Fill with → 4 12 3

Find j-i=3

3-0 = (0,3)

4-1 = (1,4)

Cost :-

c[0,3] = (10), (20), (30)

(10) = 4, (20) = 2, (30) = 6

i=0, j=3.

* use formula because construction of tree will take lot of complexities.

min { Select Key ⇒1 = ~~min~~ c(i,K-1]+c[K,j]+w(i,j) = c[0,0]+c[1,3]+12 ⇒0+10+12 = 22
Select Key ⇒2 = c[0,1]+c[2,3]+12 ⇒ 22
Select Key ⇒3 = c[0,2]+c[3,3]+12 ⇒ 8+0+12 = 20 ✓

Fill with 20^3

$* c(i,R-1)+c(K,j)+w(i,j)$

$c[1,4]$

$$\min \begin{cases} \text{Key} \Rightarrow 2 = c(1,1)+c(2,4)+w(1,4) = 0+12+11 = 23 \\ \text{Key} \Rightarrow 3 = c(1,2)+c(3,4)+w(1,4) = 2+3+11 = 16 \\ \text{Key} \Rightarrow 4 = c(1,3)+c(4,4)+w(1,4) = 10+0+11 = 21 \end{cases}$$

find $j-i=4$

$4-0=4 \rightarrow (0,4)$

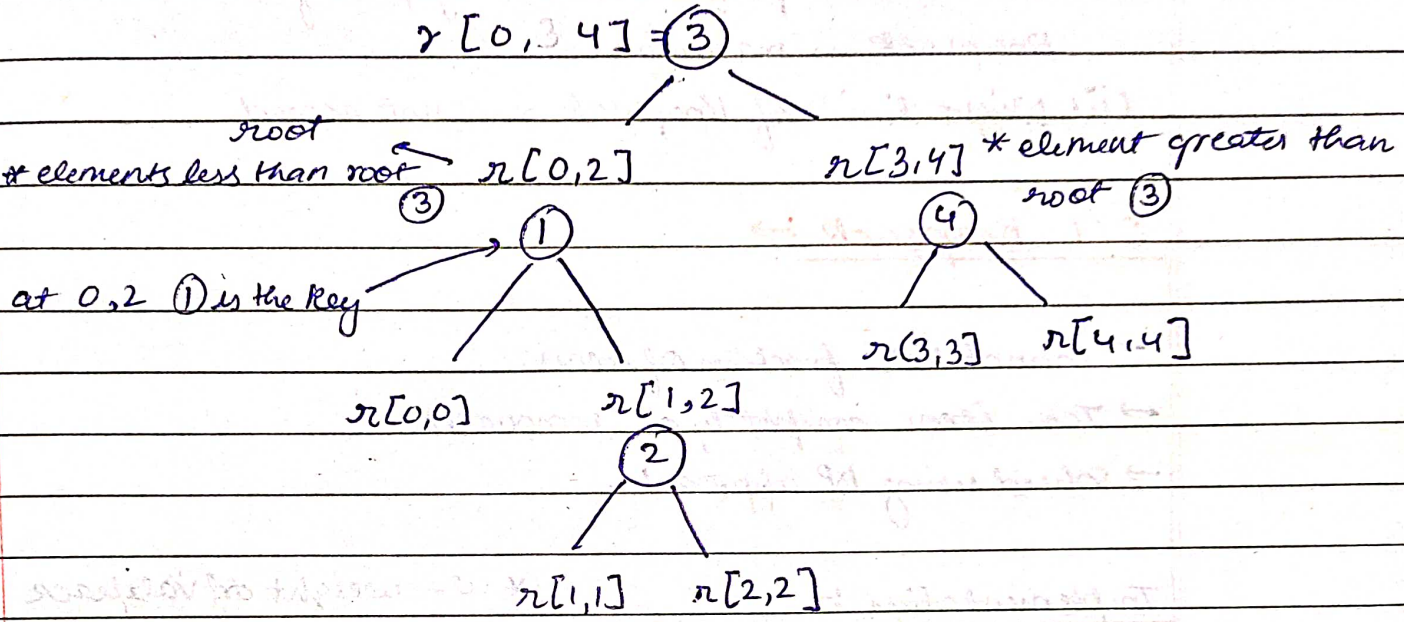
$c(0,4) \Rightarrow$

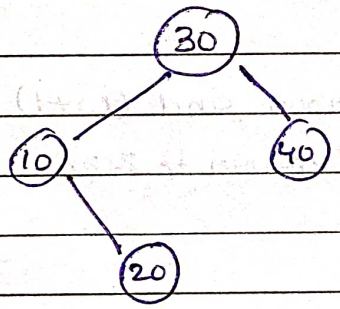
$$\text{Key} \Rightarrow 1 = \begin{cases} c(0,0)+c(1,4) \rightarrow 16 \\ \text{Key} \Rightarrow 2 = \min \begin{cases} c(0,1)+c(2,4) \rightarrow 16 \\ c(0,2)+c(3,4) \rightarrow 11 \checkmark \end{cases} \\ \text{Key} \Rightarrow 3 = \\ \text{Key} \Rightarrow 4 = c(0,3)+c(4,4) \rightarrow 20 \end{cases} + w(0,4) \Rightarrow 11+15 \Rightarrow 26^3$$

i \ j	0	1	2	3	4
0	0	4	8	20^3	26^3 → Key number
1		0	2	10^3	16^3
2			0	6	12^3
3				0	3
4					0

Construct a tree from the table :-

At 0,4 root is 3.





Time Complexity :- $O(n^2 \times n) \Rightarrow O(n^3)$

↑ table creation. ↑ no. of nodes

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

Knapsack problem :-

→ Which item should be placed in bag such that
(i) The value or profit obtained by putting the items into the knapsack is maximum.
(ii) Weight limit of knapsack does not exceed.

0/1 Knapsack :→

→ Cannot take fraction of items.
→ Take item completely or remove it.
→ solved using DP approach.

Implementation :-

* w = weight of knapsack
* n = no. of items.

→ Initialize Table :-
1. Create Table (T) with (n+1) rows and (w+1) columns.
2. Set all boxes of 0^{th} row and column to zero.

→ Fill Table :-
3. Iterate from i,j start from (1,1)
Formula :- $T(i,j) = \max\{T(i-1,j), value_i + T(i-1, j-w_i)\}$
4. Fill completely table using above formula till last.

Time Complexity :-
↳ $O(n \times w)$

→ Entry in table requires $O(1)$.

Example :-

$n = 4$

Item	Weight	Value
$i = 1$	2	3
$i = 2$	3	4
$i = 3$	4	5
$i = 4$	5	6

$W = 5$

$n = 4$

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

Formula $\Rightarrow T(i,j) = \max\{T(i-1,j), value_i + T(i-1, j-weight_i)\}$

→ $T(1,1) \Rightarrow$

$i=1, j=1, w=2, v=3$

$T(1,1) = \max\{T(0,1), 3 + T(0,1-2)\}$

$= \max\{T(0,1), 3 + T(0,-1)\}$; -1 neglect

$= \max\{T(0,1)\}$

$= 0$

→ $T(1,2) \Rightarrow$

$i=1, j=2, w=2, v=3$

$T(1,2) = \max\{T(0,2), 3 + T(0,0)\}$

$= 3$

Similarly find for all and fill the table (T).

⟹ So the maximum possible value that can be put into knapsack is 7. the last most element.

The best profit we get from putting item 1 and 2.

P.T.O