Dynamic Programming

Fibonacci Number

$F_0 = 0$

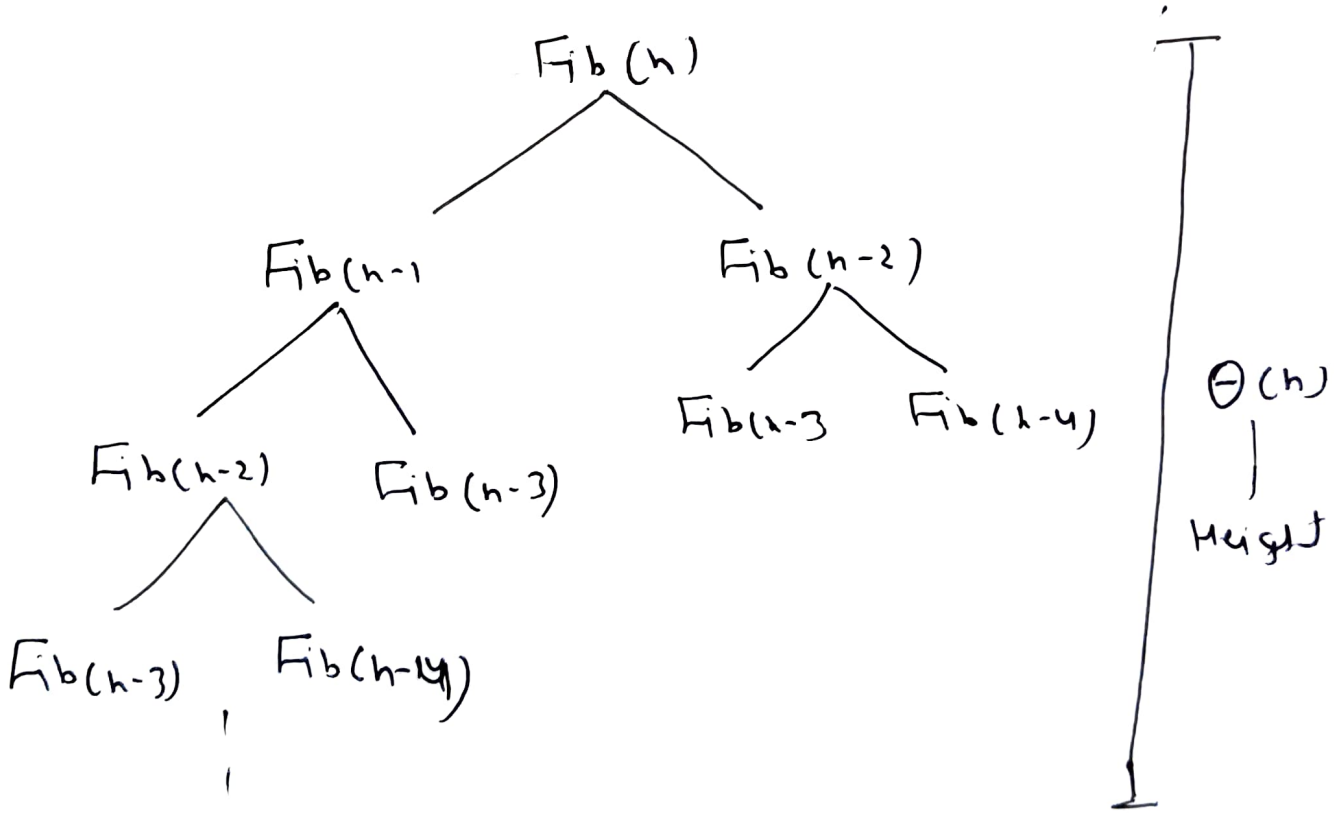
$F_1 = 1$

$F_2 = F_0 + F_1$

⋮

$F_n = F_{n-1} + F_{n-2}$

```
Fib ( long n )
{
    if (n==0 || n==1)
        return n
    return Fib(n-1) + Fib(n-2)
}
```



Assumig it is a complete binary tree

$$\text{nodes} = 2^{h-1} - 1$$

nodes an exponential function of height

Why it is exponential? its solving same problem solving again - again. Overlapping subproblem.

It also have the property of "optimal substructure" in the sense that sub problem use to solve main problem.

We can reduce it by solving in bottom-up approach and store the solution of sub problem to avoid recomputation.

```
Fib(log n)
{
    create an array A of size n+1
    A[0] = 0
    A[1] = 1
    for i = 2 to n
        A[i] = A[i-1] + A[i-2]
    return A[n];
}
```

$\Theta(n)$

0	0
1	1
2	1
3	2
4	3
n	

(3) 0-1 Knapsack

	Item1	Item2	Item3
W(kg)	2	3	5
₹	3	4	8

Knapsack size $C = 8$ kg

Base case. (bottom-up)

(Item 1, $C=1$) $\longrightarrow$ 0 (Item1 weight is 2kg)

This problem have two dimension

Item and capacity and

we can increase in both dimension.

So lets go with C :-

(Item1, $C=2$) $\longrightarrow$ 3

⋮

Now with Item1.

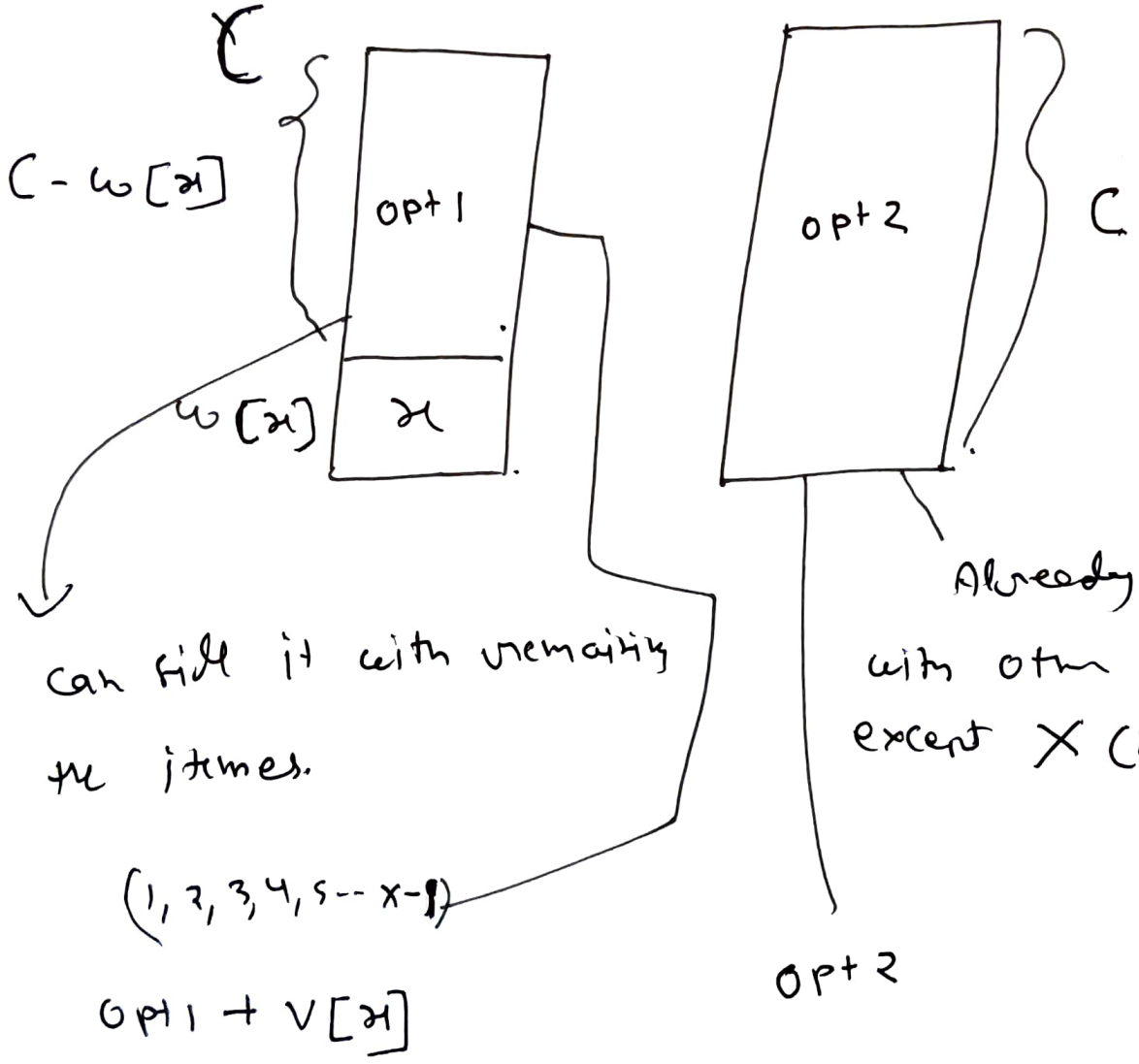
$((I_1 \& I_2), C=1) \longrightarrow 0$

$((I_1 \& I_2), C=2) \longrightarrow 3$

Given item X we have two possibility

taken X

Not take X



Let opt1 = 30, opt2 = 40, V[x] = 15

then taking x is better deal

(5)

	I_1	I_2	I_3
w →	2	3	5
R -	3	4	8

C = 8 kg

	0	1	2	3	4	5	6	7	8
No Item.	0	0	0	0	0	0	0	0	0
Item 1	0	0	3	3	3	3	3	3	3
Item 1 & Item 2	0	0	3	4	4	7	7	7	7
Item 1 & Item 2 & Item 3	0	0	3	4	4	7	7	9	10

← Result

Optimal Solution taking item 1 & Item 2 with, capacity 4.

knapsack

So 0/1 knapsack have "optimal substructure" property but not have greedy choice property.

f[i] [j] : optimal Solution for a knapsack of capacity j Using items (1, 2 -- i)

In item 1 & item 2 row at capacity 3 we have choice

take item 2

Capacity 3 → 4 + 0

4 → 4 + 0

5 → 4 + 3

Don't take item 2

3

3

3

for item1 & item2 & item3 we have to copy the previous solution till capacity 5 because item 3 have weight 5.

	taken item3	Not taken item3
Capacity 5	6 + 0	7
6	6 + 0	7
7	6 + 3	7
8	6 + 4	7

which item has been taken and which had not been taken?

~~For V[i, j]~~ V(+[3, 8]) V(+[2, 8])

Compare V(+[i, j]) with V(+[i-1, j])

if value is changed then item i is taken

For remain item we have to check the remaining capacity. (in this case) (8 - 5) = 3

check V(+[i-1, 3]) with V(+[i-2, 3])

V(+[2, 3]) with V(+[1, 3])

V = Array of values
w = " " weights
n = # Item
C = Capacity

DP_0/1_knapsack (w, v, n, c)
{

Create a 2D array t n+1 rows & C+1 columns.

for (J=0 to C)
    f [0] [J] = 0   // fill first row with 0

for (i=1 to n)   // i is item number
    for (J=0 to C)   // J is capacity
        if w[i] > J   // item does not fit
            f[i][J] = f[i-1][J]   // take from previous row
        else
            f[i][J] = Max (V[i] + f[i-1] [J - w[i]] , f[i-1][J])

take i: V[i] + f[i-1][J - w[i]]

Remaining capacity: J - w[i]

Don't take i: f[i-1][J]

}

Decide Items
{
    i = n ; J = C
    for (i = n to 1)
        if f [i] [J] > f [i-1] [J]
            take item i
            J = J - w[i]
}

$\Theta(n)$

Running time

$T(n) = \Theta(nc)$