→ Its more of category

DP 3 : Knapsack

↳ 50% of time DP questions asked is for knapsack.

Given N objects with their values V_i profit / loss their weight w_i. A bag is given with capacity W that can be used to carry some objects such that the total sum of objects weights W & sum of profit in the bag is maximized or sum of loss in the bag is minimized.

★ Generally, we will try knapsack when these combinations are given :-

- ↳ number of objects will be N.
- ↳ every object will have 2 attributes namely value & weight.
- ↳ And capacity will be given.

Question 1 :- FRACTIONAL KNAPSACK

[you can pick every item only 1 time & you can pick the item fractionally as well]

N items → Weight
→ Profit

N = 3

Weights →	10	20	30
Value →	60	100	120

Cap = 50

IDEA :- Should we use profit Per unit (PPU).

N=3

Weights →	10	20	30
Value →	60	100	120
PPU →	6	5	4

Cap = 50

Cap = 50 - 10 = 40 - 20 = 20 - 20 = 0

Profit 60 + 100 80[4 * 20] = 240

CONCLUSION :- Sort the Given data based on PPU & then start with one which provides max Value. In End we might need to pick partially.

PSEUDO CODE

```
class item {                    → Implement Comparable
    int weight;
    int value;
    double ppu;
    Public item ( x , y , z) {
        weight = x ;
        value = y;
        ppu = z;
    }
}
```

N = len

```
int fractionalKnapsack(int[] weights, int[] values,
                       int cap)
    Items[] items = new Items[N];
    for (i = 0; i < n; i++){
        double ppu = values[i] / weights[i]; * 1.0
        items[i] = new Item(weights[i],
                            values[i],
                            ppu);
    }
    int ans = 0
    Array.sort(items) // Ascending order
    for (i = n-1; i >= 0; i--){
        Item item = items[i];
        if (item.weight <= cap){
            cap -= item.weight;
            ans += item.value;
        } else {
            ans += item.ppu * cap;
            cap = 0;
            break;
        }
    }
    return ans;
}
```

Question 2 [0 1 : KNAPSACK]

Flipkart is planning a special promotional event where they need to create an exclusive combo offer. The goal is to create a combination of individual items that together offer the highest possible level of customer satisfaction (indicating its popularity and customer ratings) while ensuring the total cost of the items in the combo does not exceed a predefined combo price.

Simplified version :- Same as above (01: knapsack)

(1) you can pick every item only one time.

(2) you will either pick an item or not.

Quiz 1 :- In the Fractional knapsack problem, what is the key difference compared to the 0/1 knapsack?

↳ Items can be partially included, allowing fractions

Ex :- N = 4 items, Cap = 50

		0	1	2	3
weights []	⇒	20	10	30	40
Val []	⇒	100	60	120	150
PPU	⇒	5	6	4	3.7

IDEA :- ① By choosing max Value.

Items picked = { 1, 3 }

profit / Value = { 210 } ✗

② choosing based on PPU.

Item picked = { 1, 0 }

Profit / Value = { 160 } ✗

③ EXPLORE ALL CASES

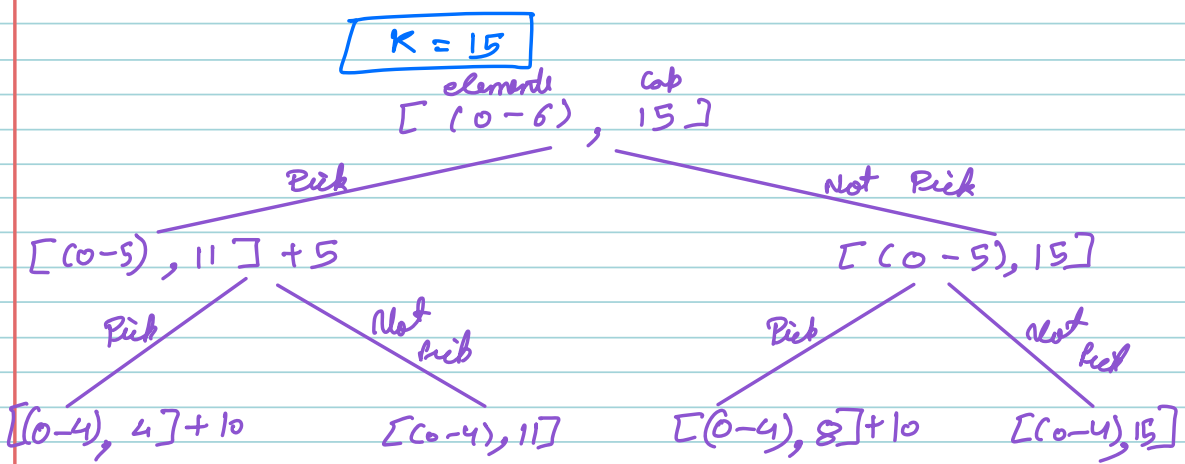
Items picked ⇒ { 0, 2 } | Value = { 220 }

Try all the possible subsequence, for the case where weights fits capacity & generate max profit will be the Ans.

N = 7

	0	1	2	3	4	5	6
wt [] →	4	1	5	4	3	7	4
Val [] →	3	2	8	3	7	10	5

K = 15



Q what will be Ans knowing Ans of Left & Right Side ?

↳ Max (Left Ans, Right Ans)

PSEUDO CODE

```
int  01Knapsack ( int[] Weight, int[] Value, int Cap,
                                                int end ){
    // Base Case
    if ( Cap >= Weight[end] ){
        int include = 01Knapsack ( Weight, Val,
                                   Cap - Weight[end],
                                   end - 1)      + Value[end];
    }
    int exclude = 01Knapsack ( Weight, Value, Cap,
                                                end - 1);
    return  Max ( include , exclude );
}
```

Q Can we optimizing using DP?

↳ yes

Q What will be the dimention of DP array?

↳ 2D array because 2 variables are changing.

PSEUDO CODE [Memoization]

```
int dp[N+1][m+1] = {-1};
int  01Knapsack ( int[] Weight, int[] Value, int Cap,
                                                int end ){
    // Base Case
    if ( dp[end][cap] != -1){ return dp[end][cap];}
    if ( Cap >= Weight[end] ){
        int include = 01Knapsack ( Weight, Val,
                                   Cap - Weight[end], end - 1) + Value[end];
    }
    int exclude = 01Knapsack ( Weight, Value, Cap,
                                                end - 1);
    dp[end][cap] = Max ( include , exclude );
}
```

return dp[n][cap];

TABULATION

↳ 2D DP array is Required

Q what we will be storing in dp[i][j]?

↳ dp[i][j] = Max profit we can get in a bag of capacity 'j' if we can choose till first "i" items.

n = 5, k = 8

	0	1	2	3	4
wt[] =	3	6	5	2	4
val[] =	12	20	15	6	10

val	wt	ele ↓ / cap →	0	1	2	3	4	5	6	7	8
		0	0	0	0	0	0	0	0	0	0
12	3	1	0	0	0	12	12	12	12	12	12
20	6	2	0	0	0	12	12	15	20	20	20
15	5	3	0	0	0	12	12	15	20	20	27
6	2	4	0	0	6	12	12	18	20	21	27
10	4	5	0	0	6	12	12	18	20	22	27

(2, 8)

(1-1), 2 (1-1), 8

Q How much profit you can make with Cap = 0 i.e. [No Space]

↳ 0

Q How much profit I can make with No element present i.e. Nothing to choose from.

↳ 0

PSEUDO CODE

```
dp[N+1][K+1];
for( i=0; i<=N; i++){
    for( j=0; j<=K; j++){
        if(i==0 || j==0){
            dp[i][j] = 0;
        }else{
            notpick = dp[i-1][j];
            pick = 0;
            if(j >= wt[i-1]){
                pick = val[i-1]
                     + dp[i-1][j-wt[i-1]];
            }
            dp[i][j] = max(pick, notpick);
        }
    }
}
return dp[n][K];
```

TC — O(N*K) | SC — O(N*K)

10:52 ← RESUME

★★★★
★ TRY TO optimize over Space [Hint:- use 1D array]
↳ HW

Question 3 :- UNBOUNDED KNAPSACK

Note 1 :> We can select 1 element as many times as we want.

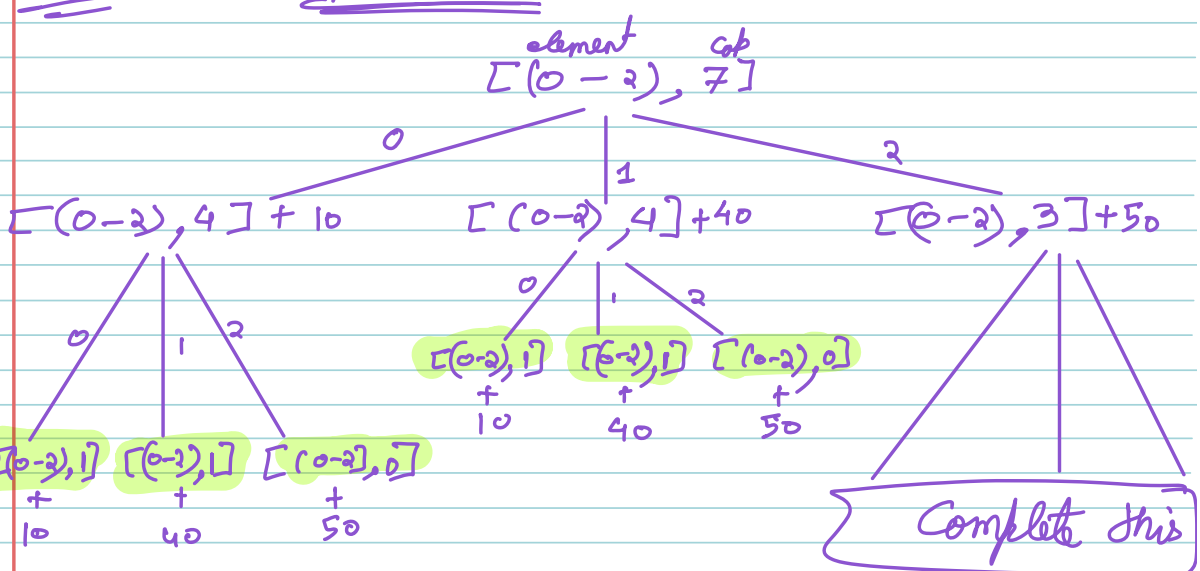
Note 2 :> you can't select any item partially.

Ex: wts[] = 20 (0) 13 (1) 10 (2) 40 (3)
K = 50 val[] = 100 66 40 150

Ans → {0, 0, 2} → 240

Ex2 val[] = {10 (0), 40 (1), 50 (2)}
wts[] = {3, 3, 4}
K = 7

IDEA :- Explore all cases



Q Can we solve it using DP?
↳ yes

Q what will be the dimention of DP array?
↳ 1D DP array is required

Q what we are storing in dp[i]?
↳ dp[i] = Max profit we can get in a bag of capacity "i"

TABULATION

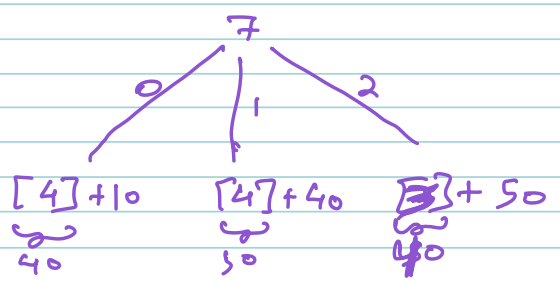
K = 7,

val[] = { 10 (0), 40 (1), 50 (2) }

wt[] = { 3, 3, 4 }

dp[i] = Max profit we can get in a bag of capacity "i"

0	1	2	3	4	5	6	7
0	0	0	40	50	50	80	90
			{40}	{50}	{50}	{40, 40}	{50, 40}



PSEUDO CODE

```
int dp[k+1];
dp[0] = 0;
for (i=1; i<=k; i++){
    for(j=0; j<n; j++){
        if (i >= wt[j]){
            currAns = val[j] + dp[i - wt[j]];
            ans = max(ans, currAns);
        }
    }
}
return dp[k];
```

TC - O(N * K)

SC - O(K)

Quiz? We have weight of 100

* Values = {1, 30}
* Weights = {1, 50}

what is the maximum value you can have?

↳ 100 [weight 1, 100 times]