0/1 Knapsack

Que.1 Can we apply Greedy approch?

⇒ Wt → [3, 2, 5] W = 6
Val → [30, 40, 60]

In Greedy approch we choose first most valued item then 2nd most valued item till knap_weight <= capacity

so By Greedy
thief stole first = item 3 → val = 60
wt = 5

but after that he can't stole any thing else;

but ans = 30+40 = 70
↓
3+2 = 5 weight

so Greedy fails bcz uniformaity is not there

uniformatiy means the item whose weight is more but value is low or item whose value is more weight is moderate & and value is more

so on.

So we try all possible combination & in all the combination we got the best value combination

all possible way done using Recursion

Rules for recurrance :-

(1) Express ~~every~~ every thing index so for indexing consider item no. as array idx. Also find required parameter

required parameter → (idx, knap_weight)

write all the Base cases

(2) Explore all possibilites. Here two possibilites

(i) take
(ii) Not take

(3) return max (take, not take)

wt = [3, 2, 4] W = 6

val = [30, 40, 60]

f(idx, knap_weight)

or

f(n-1, w)

~~f(2,6) : it mean till the idx~~

~~2 what is the maximum that you will generate~~

f(2,6) ⟶ It means that till index 2 what is the max value you get with weight of bag = 6

```
f(idx, W)
{ //Base cases
    if(idx==0)
    {  if( wt[0] <= W) return val[0];
       return 0;
    }
// Explore all possibilities
notTake = f(idx-1, W);
take = 0
if(wt[idx] <= W)
        take = val[idx] + f(idx-1, W- wt[idx]);
return max(take, notTake);
}
```

T.C. = $O(2^n)$

S.C. = $O(N)$

Memoization

changing parameters → idx , W

idx → (0 to n-1) / (n-1 to 0)

W → (0 to W) / (W to 0)

dp[N][W+1]

T.C. = O(N×W)

S.C. = O(N×W) + O(N)

Tabulation

Rules

1. Base Case analyses
2. Nested loops based on changing parameters their are two changing parameters so two nested loops & inside ~~the~~ loop write the recurrance copied ~~from~~ memoization.
3.

```
//Base case
for( i = wt[0] to W)
        dp[0][i] = val[0];

//Nested loop
// In memoization idx goes from (n-1) to 0
  so here  0 to n-1
  also  W ⟶ W to 0   here  0 to W

for(idx=1 to n-1)
 {
    for( w=0 to W)
    {
        Copy the recurrance

    }
 }
```