Todays Content:

→ knap Sack 0/1

→ knap Sack ∞

Knapsack: 0/1

Given N items each with a weight & value, find max value which can be obtained by picking items such that total weight of all items <= k

Note1: Every item can be picked at max 1 time

Note2: We cannot take a part of item

Ex: N = 4 items, k = 50

N =	1	2	3	4
w[] =	20	10	30	40
v[] =	100	60	120	150
$\frac{v}{w}$ =	5	6	4	3.75

Idea1: Pick greedily based on value

Pick 4 & 2 → 210

Idea2: Greedy based on v/w ration

Pick 2, 1 → 160

↳ Greedy fails

Correct ans = Pick 1 & 3 = 220

idea1: Generate all subsets with weight <= k & get max value out of all subsets

TC: (2^N) SC: O(N)

↳ stack size

Since it's recursion → optimal substructure

Constraints:

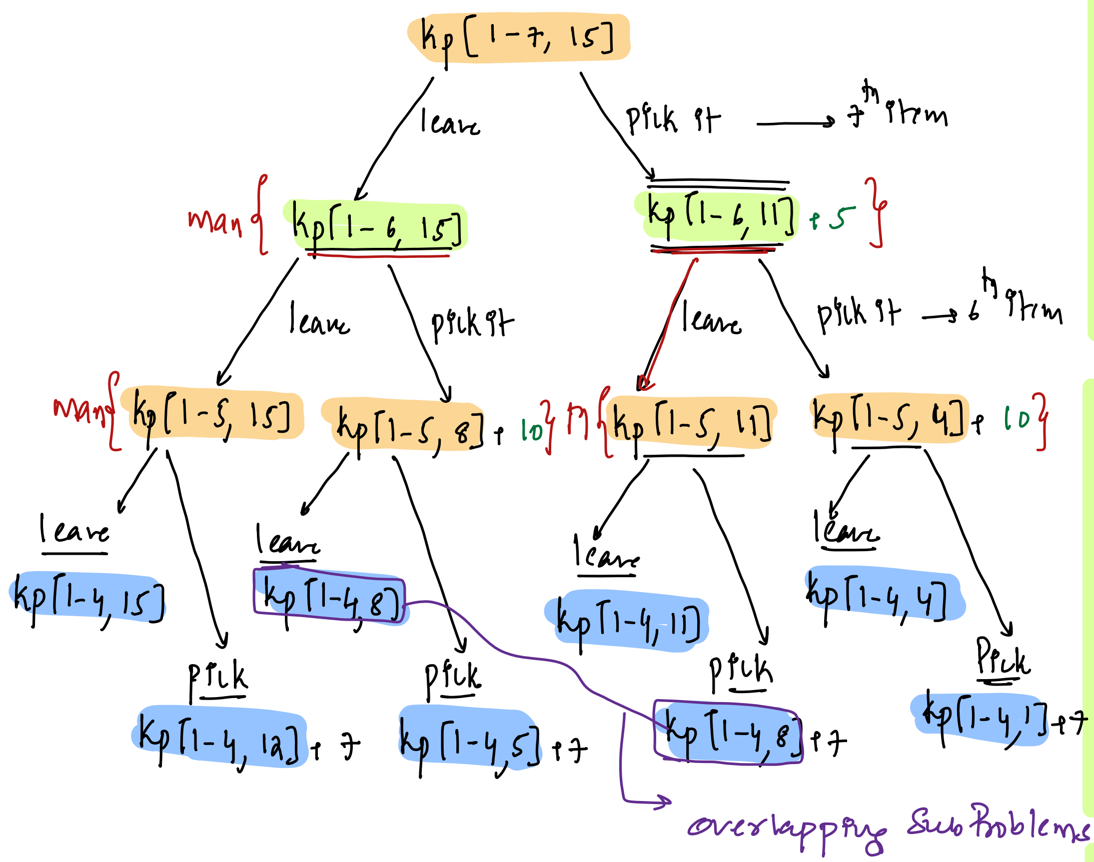
$1 <= N <= 10^3$
$1 <= k <= 10^3$
$1 <= w[i] <= 10^5$
$1 <= v[i] <= 10^5$

For above constraints, Brute force code won't work

$2^N \longrightarrow O(N * k)$

N = 7	1	2	3	4	5	6	7	K = 15
w[]	4	1	5	4	3	7	4	
v[]	3	2	8	3	7	10	5	

{max value which can be obtained using 1 - 7, total wt = 15



// dp state

$dp[i, j]$ = max value using [1 to i] items such total weight $\le j$

// dp Expression

$$dp[i, j] = \max \left\{ \underbrace{dp[i-1, j]}_{\text{leave } i^{th} \text{ item}},\ \underbrace{dp[i-1, j - w[i]] + v[i]}_{\text{pick it if } j >= w[i]} \right\}$$

// dp table , final ans = $dp[N][K]$

$dp[N+1][K+1] = \{-1\}$

Pseudocode :

```
dp[N+1][K+1] = {-1}  // In main & pass as reference

int kp(int dp[][], int i, int j, int w[], int v[]){
    if(i==0 || j==0) { return 0 }
    if(dp[i][j] == -1){
        int a = kp(dp, i-1, j, w, v)  // leave i^th element
        if(j >= w[i]){  // pick i^th element
            a = max(a, kp(dp, i-1, j-w[i], w, v) + v[i])
        }
        dp[i][j] = a
    }
    return dp[i,j]
}
```

At max overall weight d = j

TC: #dp states * TC for each

(N*k) * (1) ⟶ O(N*k)

SC: O(N, k)

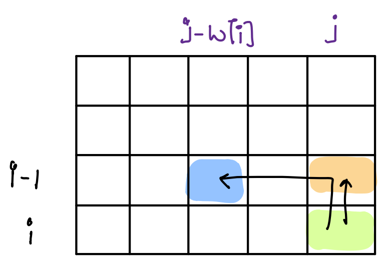
dp Expression :

$$dp[i,j] = \max\begin{cases} \underbrace{dp[i-1,j]}_{\text{leave } i^{th} \text{ item}}, & \underbrace{dp[i-1,\ j-w[i]] + v[i]}_{\text{pick it}} \quad \text{if } j >= w[i] \end{cases}$$

Edge Cases: if i==0, Edge Case

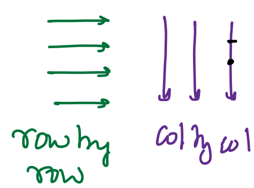
↳ j==0, Expression will hold

```
int kpIterative( int N, int k, int w[], int v[]){

    int dp[N+1][k+1]
                                        Note: i^th item weight at w[i]
                                              i^th item value at v[i]
    // Base Conditions, i=0 -> 0 items,   in our class, please
                                          follow above discussion
    for( int j = 0 ; j <= k ; j++){       1-based index
        dp[0][j] = 0
    }

    // How to fill Matrix ?
```



Fill the matrix :
- → top to down and
- → Left to right

possible ways to fill Matrix




```
    i = 1 ; i <= N ; i++){
        j = 0 ; j <= k ; j++){
            // dp[i, j]
            int a = dp[i-1, j]
            if( j >= w[i]){
                a = max (a, dp[i-1, j-w[i]] + v[i])
            }
            dp[i, j] = a
        }
    }
    return dp[N][k]
}
```

10:38pm - 10:48pm

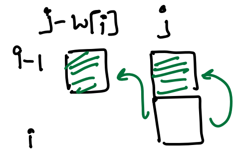
TC: O(N*k) SC: O(N*k)

Tracing :

items: 1 2 3 4 5✓

w[] : 3 6 5 2 4

N = 5, k = 7   v[] : 12 20 15 6 10



w[]	0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0	0
1	0	0	0	12	12	12	12	12
2	0	0	0	12	12	12	20	20
3	0	0	0	12	12	15	20	20
4	0	0	6	12	12	18	20	21
5	0	0	6	12	12	18	20	22

22 → final ans

dp[5][7] != dp[4][7]
→ pick 5th element
→ dp[4][7-4] = dp[4][3]

dp[1][3] != dp[0][3]
→ pick 1st element
→ dp[0][3-3] = dp[0][0]

$$dp[i, j] = max\{ dp[i-1, j] , dp[i-1, j - w[i]] + v[i] \}$$

dp[2,6] = max(dp[1,6], dp[1,0] + 20) = max(12, 20) = 20

dp[2,7] = max(dp[1,7], dp[1,1] + 20) = max (12, 20) = 20

dp[3,5] = max (dp[2,5], dp[2,0] + 15) = max(12, 15) = 15

dp[4,2] = max (dp[3,2], dp[3,0] + 6) = max (0, 6) = 6

dp[4,5] = max (dp[3,5], dp[3,3] + 6) = max(15, 12+6) = 18

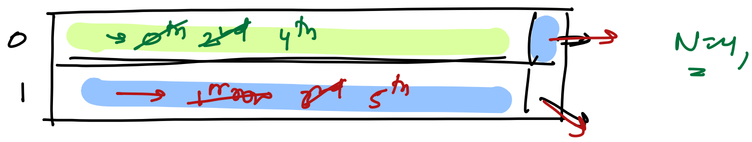
dp[5,7] = max(dp[4,7], dp[4,3] + 10) = max(21, 12+10) = 22

// get items :

```
i = N, j = k, list<int> ans;
while( i>0 && j>0 ){
    if ( dp[i,j] == dp[i-1,j] ){
        i = i-1;
    }
    else { // we are picking ith element
        ans.insert(i)
        i = i-1, j = j - w[i]
    }
}
```

Space Optimization :

⇒ At any given we only need 2 rows



row	→	fill
0th	→	0th
1st	→	1st
2nd	→	0th
3rd	→	1st
4th	→	0th
5th	→	1st

ith row data we fill at i%2nd row
(i-1)th row data we fill at (i-1)%2 row

// Code :

```
int kpiterativeSpaceOp( int N, int k, int w[], int v[]){

    int dp[2][k+1] = 0
    Base Conditions, i=0 → 0 item,
    for( int j=0; j<=k; j++){
        dp[0][j] = 0
    }
    i=1; i<=N; i++){
        j=0; j<=k; j++){
            //dp[i,j]
            int a = dp[ (i-1)%2, j]
            if( j >= w[i]){
                a = max (a, dp[(i-1)%2, j-w[i]] + v[i])
            }
            dp[ i%2, j] = a
        }
    }
    return dp[N%2] [k]
}
```

Disadvantage:
We can no longer get list of items to store

TC: $O(N^*k)$

SC: $O(2^*k) \rightarrow O(k)$

Subsets :	→ Dp Recursion	→ Dp Iterative	→ Dp Iterative Space Optimization
TC: $O(2^N)$	TC $O(N^*k)$	TC $O(N^*k)$	TC $O(N^*k)$
SC: $O(N)$	SC $O(N^*k)$ + Stack space	SC $O(N^*k)$	SC $O(k)$

2Q) Exactly Same above Problem, k is given, ∞ KnapSack

Note: A single item can be picked as many times as we want?

Ex: $k=50$

N =	1	2	3	4
w[i] =	20	13	10	40
v[i] =	100	66	40	150

→ overlapping
→ optimal substructure

// dp state

dp[i, j] = max value using [1 to i] items such total weight <= j

// dp express

$$dp[i,j] = \max\begin{cases} dp[i-1, j] & \text{leave it} \\ dp[i, j-w[i]] + v[i] & \text{pick it if } (j >= w[i]) \end{cases}$$