Knapsack Problems

Given N objects with their values v_i and their weights w_i.
A bag with capacity W that can be used to carry some objects such that →

total sum of object weights ≤ W, and.
sum of values in the bag is maximised.

Fractional Knapsack.

Given N cakes with their happiness and weight.
Find max total happiness that can be kept in a bag with capacity = W. (cakes can be divided)

N=5. happiness[] → [4 8 10 2 5]

W=40 weight[] → [4 4 20 8 16]

ideas → Select the cake with the maximum happiness ✗

Select the cake with minimum/ maximum weight first. ✗

Idea → Sort the cakes on the basis of $\frac{\text{happiness}}{\text{wt.}}$ ratio

N=5.

W=40

happiness[] → [4 8 10 2 5]

weight[] → [4 4 20 8 16]

$\frac{\text{happiness}}{\text{wt.}}$ → 1 2 0.5 0.25 0.3125

↓ Sort on the basis of $\frac{\text{happiness}}{\text{wt.}}$ in desc. order

$$\begin{array}{l} \text{happiness} \rightarrow \\ \text{wt.} \rightarrow \end{array} \begin{bmatrix} 8 & 4 & 10 & 5 & 2 \\ 4 & 4 & 20 & 16 & 8 \end{bmatrix}$$

W = ~~40~~ ~~36~~ ~~32~~ ~~12~~ 0

ans → $8 + 4 + 10 + \left(\frac{5}{16} * 12\right)$

ans = 25.75

#code· →

```
→ Create pair arr[N];

→ Sort( arr, desc);    // $\frac{happiness*1.0}{wt * 1.0}$
```

```
Pair {
    int happiness;
    int wt;
}
```

```
double ans = 0;

for( i=0; i<N; i++) {
    if( arr[i].wt ≤ W){
        ans += arr[i].happiness;
        W -= arr[i].wt;
    }
    else{
        ans += $\frac{arr[i].happiness}{arr[i].wt * 1.0}$ * W

        break;
    }
}
return ans;
```

T.C → O(NlogN)

S.C → O(N)

0-1 KnapSack

Given N toys with their happiness and weight. Find max total happiness that can be kept in a bag with capacity W. [toys can't be divided]

N = 4

W = 7

	0	1	2	3
h →	[4	1	5	7]
w →	[3	2	4	5]
h/w →	1.33	0.5	1.25	1.4

~~total happiness → 8~~

ans → 9.

maxHappiness(N, W)

N → no. of toys

W → capacity of bag.

✗ → 0 + maxHappiness(N-1, W)

✓ → happiness[N-1] + maxHappiness(N-1, W - wt[N-1])

maxHappiness(i, j)

i → no. of toys

j → capacity.

✗ → 0 + maxHappiness(i-1, j)

✓ → $\underline{\text{happiness}[i-1]}$ + maxHappiness(i-1, j - $\underline{\text{wt}[i-1]}$)

happiness of ith toy

wt. of ith toy.

code:→

```
                                            N       W
                                            ↑       ↑
int  Knapsack ( happ [N], wt[N], i, j ) {
        if( i ==0 || j ==0) { return 0 }
        f1 = Knapsack( happ[], wt[], i-1, j);        //not selecting ith toy
        f2 = 0
        if( wt[i-1] ≤ j ) {
            f2 = happ[i-1] + Knapsack( happ[], wt[], i-1, j - wt[i-1]);
        }                                                  ↑
                                                 //selecting ith toy.
        ans = Max( f1, f2);
        return ans;
}
```

T.C → O(2^N)
S.C → O(N)

optimal substructure ✓
overlapping sub-problems ✓ } D.P.

top-down

```
int dp[N+1][W+1]    //initialse dp[i][j] = -1

int Knapsack( happ[N], wt[N], i, j, dp[][] ){
    if( i == 0 || j == 0 ){ return 0 }
    if( dp[i][j] != -1 ){ return dp[i][j] }
    f1 = Knapsack( happ[], wt[], i-1, j, dp[][] );
    f2 = 0
    if( wt[i-1] ≤ j ){
        f2 = happ[i-1] + Knapsack( happ[], wt[], i-1, j - wt[i-1], dp[][] );
    }
    ans = Max( f1, f2 );
    dp[i][j] = ans;
    return ans;
}
```

T.C → O(N*W)

S.C → O(N*W)

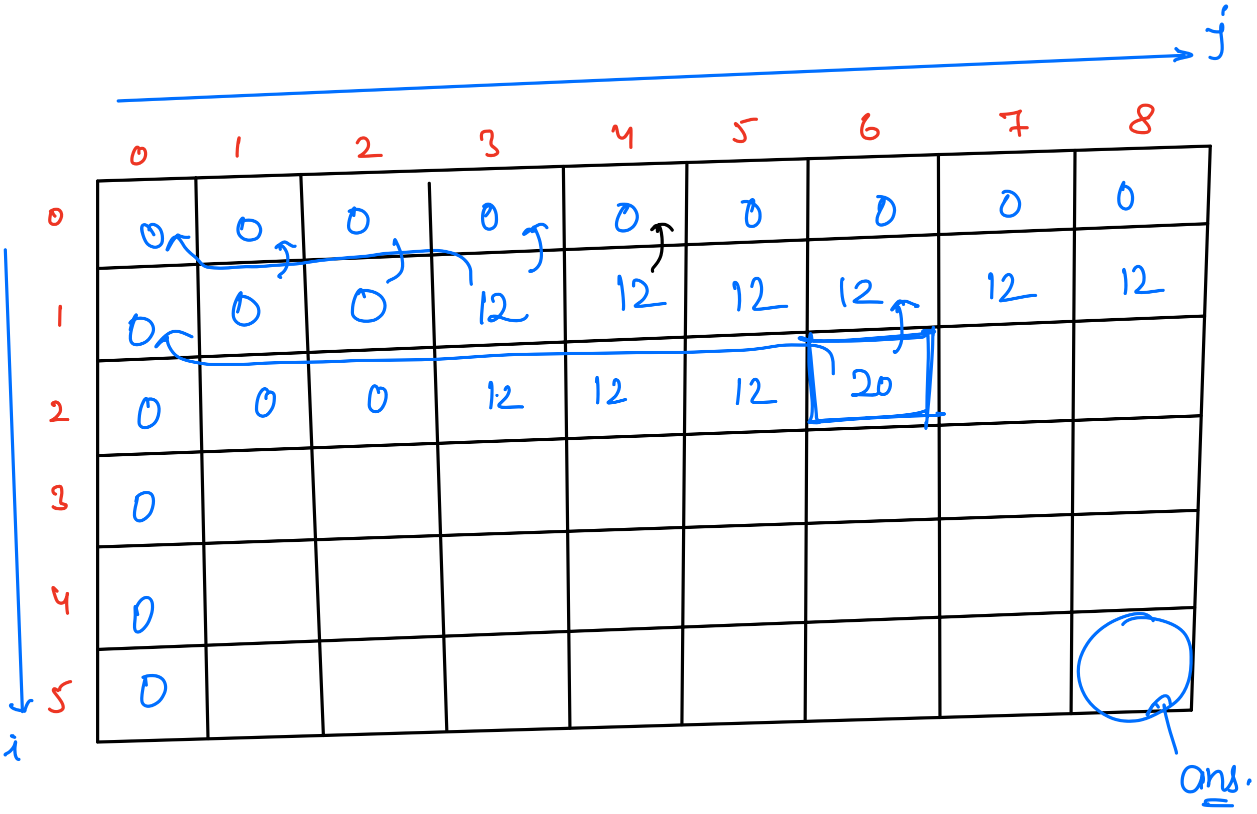
Bottom-up.

h[] → [12 20 15 6 10]
0 1 2 3 4
wt[] → [3 6 5 2 4]

W=8
N=5.

int dp[N+1][W+1]

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



	0	1	2	3	4	5	6	7	8
0	0	0	0	0	0	0	0	0	0
1	0	0	0	12	12	12	12	12	12
2	0	0	0	12	12	12	20		
3	0								
4	0								
5	0								

dp[i][j] → Maximum happiness first i-toys and j → capacity.

f1 → dp[i-1][j]

wt[i-1] ≤ j ✓ f2 → happ(i-1)+dp[i-1][j-wt[i-1]]

code:→

```
dp[N+1][W+1];

// initialise 0th row & 0th column with 0.

for( i = 1; i ≤ N; i++){
    for( j = 1; j ≤ W; j++){
        f1 = dp[i-1][j]
        f2 = 0;
        if( wt[i-1] ≤ j){
            f2 = happ[i-1] + dp[i-1][j-wt[i-1]];
        }
        dp[i][j] = Max(f1, f2);
    }
}

return dp[N][W];
```

T.C → $O(N*W)$
S.C → $O(N*W)$

↓

Since the answer of a row depends only on the previous row.
SC can be further optimised.

↓

$O(W)$

$1 \le N \le 10^3$
$1 \le W \le 10^5$
→ S.C → $O(N*W)$

↓

M.L.E

code:→ S.C optimised

```
int dp1[W+1], dp2[W+1];

for( i = 1; i ≤ N; i++) {
    for( j = 1; j ≤ W; j++){
            f1 = dp1[j];
            f2 = 0;
            if( w[i-1] ≤ j){
            [     f2 = happ(i-1) + dp1[j - wt[i-1]];
            }
             dp2[j] = Max(f1, f2);
    }
    dp1 = dp2;
    dp2 = new int[W+1];
}
return dp1[W];
```

[T.C → O(N*W)
S.C → O(W)]

Unbounded KnapSack (0-∞ KnapSack)

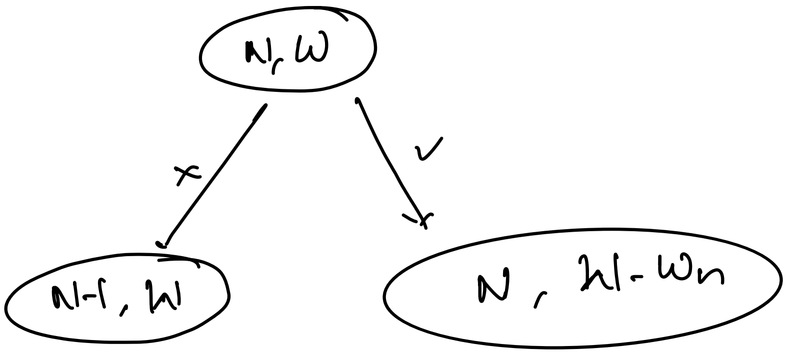
→ You can pick any element any no. of times.

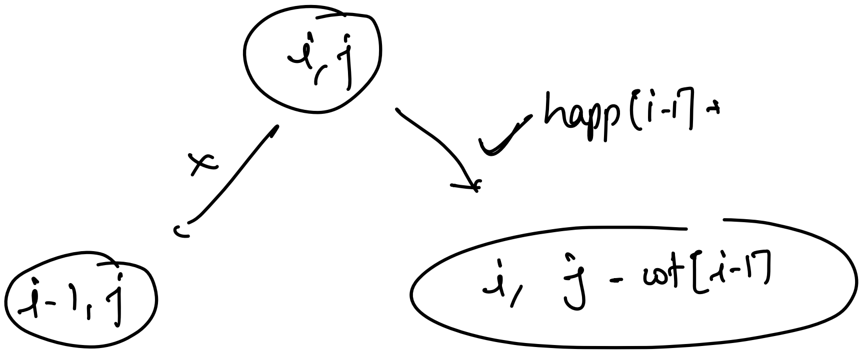
value[] → [2 3 5]

W=8, N=3

wt[] → [3 4 7]

ans=6.





idea-1.

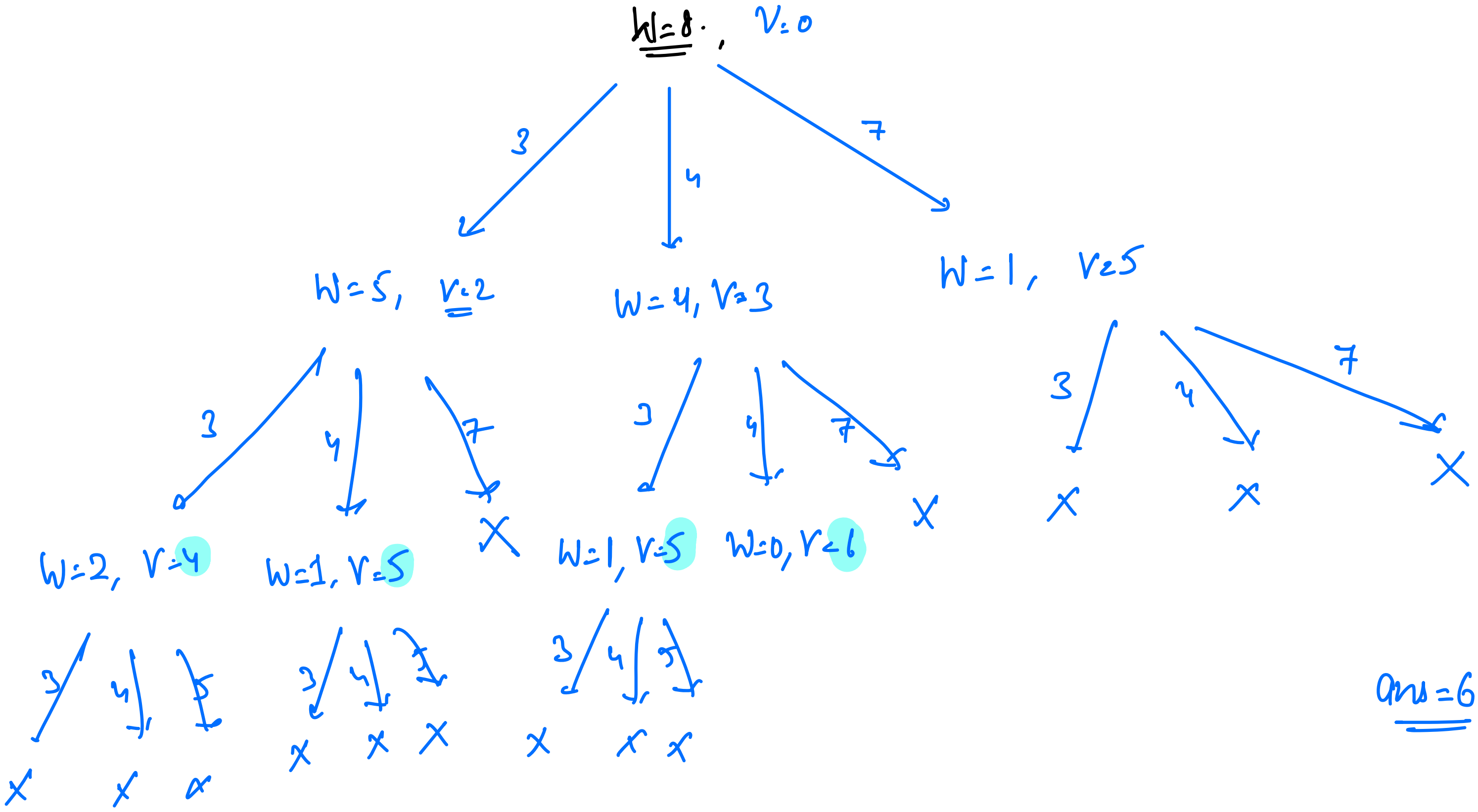
```
dp[N+1][W+1];

// initialise 0th row & 0th column with 0.

for( i = 1; i ≤ N; i++){
    for( j = 1; j ≤ W; j++){
        f1 = dp[i-1][j]
        f2 = 0;
        if( wt[i-1] ≤ j){
            f2 = happ[i-1] + dp[i][j-wt[i-1]];
        }
        dp[i][j] = Max(f1, f2);
    }
}
return dp[N][W];
```

T.C → O(N*W)
S.C → O(N*W)

value[] => [2 3 5]

wt[] => [3 4 7]

W=8, N=3



ans = 6.

optimal substructure ✓

overlapping sub-problems ✓

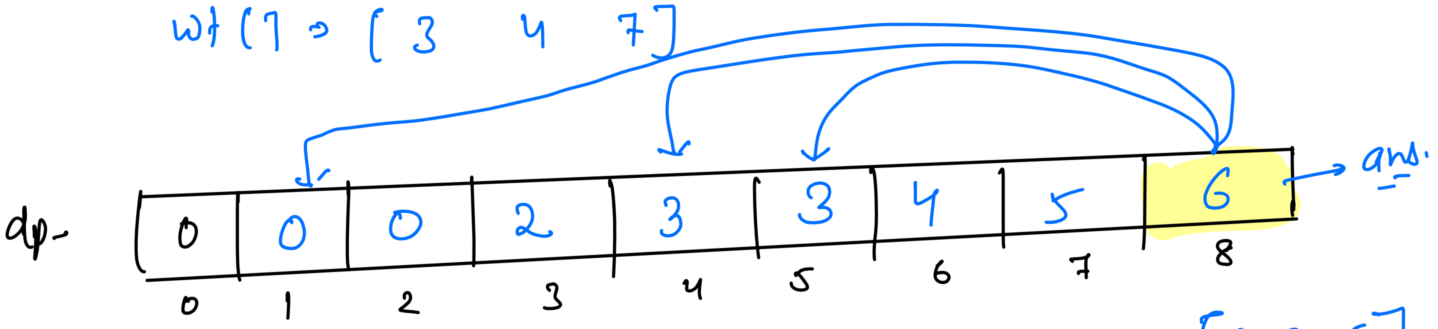
D.P

Only the Capacity is varying.

value[] → [2 3 5]

wt[] → [3 4 7]

W=8, N=3



$\max\begin{pmatrix}2+3\\3+2\\5+0\end{pmatrix}$

$\begin{pmatrix}2+3\to5\\3+3\to6\\5+0\to5\end{pmatrix}\max$

```
#
dp[W+1]; , ∀i, dp[i]=0;

for( i=1; i <= W; i++){
    for( j=0; j < N; j++){  → Considering all the toys.
        if ( wt[j] <= i){
            dp[i] = Max( dp[i], happ[j] + dp[i - wt[j-1]]);
        }
    }
}
return dp[W];
```

T.C → O(N*W)
S.C → O(W)

Flip. → 0-1 Knapsack [Hint].