Knapsack

N objects ⇒ value & weight
A bag of capacity W kg
You have to fill the bag with max value while remaining inside weight limit

2, 3, 4 $\quad W = 5$

10 $\quad$ 100 $\quad$ 1 a

Fractional Knapsack ⇒ You can break the item down

Sweets

value	3	8	10	2	5
weight	10	4	20	8	15

Bag capacity $W = 40$

Idea: 1) Max ~~total~~ value
2) Max per kg value

value	3	8	10	2	5
weight	10	4	20	8	15
	·3	2	·5	·25	·33

⇒

8	10	5	3	2
4	20	15	10	8

val = 8 + 10 + 5 + 0·3

weight = 4 + 20 + 15 = 39

1) Sort acc to value/weight desc order

2) Iterate on the sorted array

if you can all of the weight

take the whole thing

else take whatever you can

TC: $O(n \log n)$

SC: $O(1)$

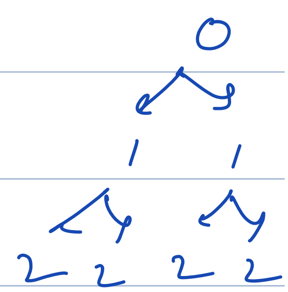
0/1 Knapsack

N items → weight
N items → value

Pick some items such that total weight $\leq W$ & max total value
Cannot pick same item multiple times.

weight	20	10	30	40	$W = 50$
values	100	60	120	150	

__Brute:__ Consider all possibilities
How? Recursion
TC: $O(2^n)$



Recursive relation

Parameters → 1) Which item is under consideration.

2) Weight till now

	0	1	2	3
weight	20	10	30	40
values	100	60	120	150

W=50

0, 0

take 0th item

100+(1,20)

dont take 0th item

(1, 0)

take 1th

100+60+(2,30)

no 1th

100+(2,20)

i, W

take leave

$val_i + (i+1, W+w_i)$ $(i+1, W)$

take max

$dp(i,j) \rightarrow$ max value by picking till i items with weight $= j$

$$dp(i,j) = \max(val_i + dp(i+1, j+weight_i), dp(i+1, j))$$

$70 + 100 + dp(20, 40)$ $W = 30$

Code

```
dp[n][W]    // = -1
int calc( int i, int j){
  if( i==n ){
     if (j<=W)   return 0
     else     return   INT_MIN
  }
  if( j>W )
       return INT_MIN
  if( dp[i][j] != -1 )
       return dp[i][j]
  int ans= calc (i+1, j)
  ans= max ( ans, val_i + calc(i+1, j+w_i))
  dp[i][j] = ans
  return ans
}
```

TC: } O(NW)
SC:

Unbounded Knapsack 0/N

Now you take any item multiple times

weight	2	1	5	W= 5
value	1000	10	50	

Idea: After taking i^{th} item, I dont need to move ahead. I'll still be at same index

(0, 0)

leave

$val_0 + (0, wt_0)$ (1, 0)

$2val_0 + (0, 2wt_0)$

↳ $1, 2wt_0$

Parameters → 1) Which item is under consideration.
2) Weight till now

$0, 0$

take ↙

$val_0 + (0, wt_0)$

take ↙

$2val_0 + (0, 2wt_0)$

↘ leave

$2val_0 + (1, 2wt_0)$

i, W

take ↙ ↘ leave

$val_i + (i, W + wt_i)$ $\qquad$ $(i+1, W)$

take max

$dp(i,j) \rightarrow$ max value by picking till i items with weight $= j$

$$dp(i,j) = \max(val_i + dp(\quad\quad),\ dp(i+1, j))$$

Code

```
dp[n][W]        // = -1
int calc( int i, int j) {
  if( i==n) {
     if (j<=W)    return 0
     else     return    INT_MIN
  }
  if( j > W )
        return INT_MIN
  if( dp[i][j] != -1 )
        return dp[i][j]
  int ans = calc (i+1, j)
  ans = max ( ans, val_i + calc (i, j+wt_i))
  dp[i][j] = ans
  return ans
}
```

TC, SC } O(NW)

$\rightarrow 0$

toffees $\rightarrow 1$

$\rightarrow 2$

$\rightarrow 3$

$\vdots$

toffees

take   leave

toffees   {done}

take   leave

toffees

take   leave

val	30	1	$W = 100$
weight	50	1	

$\Rightarrow 100 \quad \Rightarrow 80 \quad \Rightarrow 60$