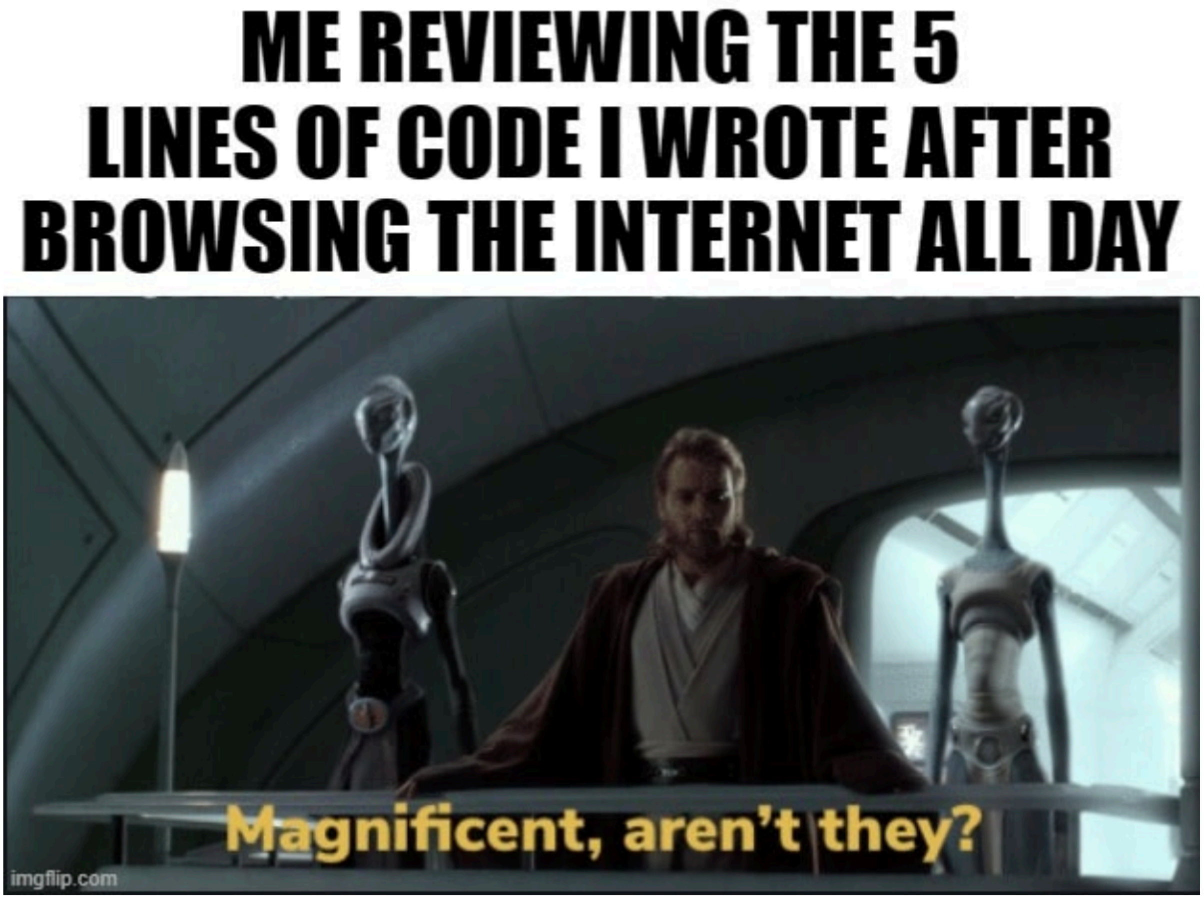
SCALER

DP 3 - KnapSack

TABLE OF CONTENTS

1. General statement of KnapSack problems
2. Fractional KnapSack
3. 0 - 1 KnapSack (Bounded KnapSack)
4. 0 - ∞ KnapSack (Unbounded KnapSack)


ME REVIEWING THE 5 LINES OF CODE I WROTE AFTER BROWSING THE INTERNET ALL DAY
Magnificent, aren't they?
imgflip.com

Q → Given an integer array & a target sum K.

Check if there exists a subset sum = K. (A[i] > 0)

subset → may / may not be continuous

$$A = \begin{array}{c} \begin{array}{ccccc} 0 & 1 & 2 & 3 & 4 \end{array} \\ [\begin{array}{ccccc} 3 & 4 & 11 & 5 & 2 \end{array}] \end{array} \qquad K = 8$$

A[0] + A[3] = 8

$$A = \begin{array}{c} \begin{array}{ccccc} 0 & 1 & 2 & 3 & 4 \end{array} \\ [\begin{array}{ccccc} 1 & 2 & 1 & 3 & 5 \end{array}] \end{array} \qquad K = 4$$

A[0] + A[1] + A[2] = 4

Bruteforce → check sum for every subset.

TC = $\underline{O(2^n)}$

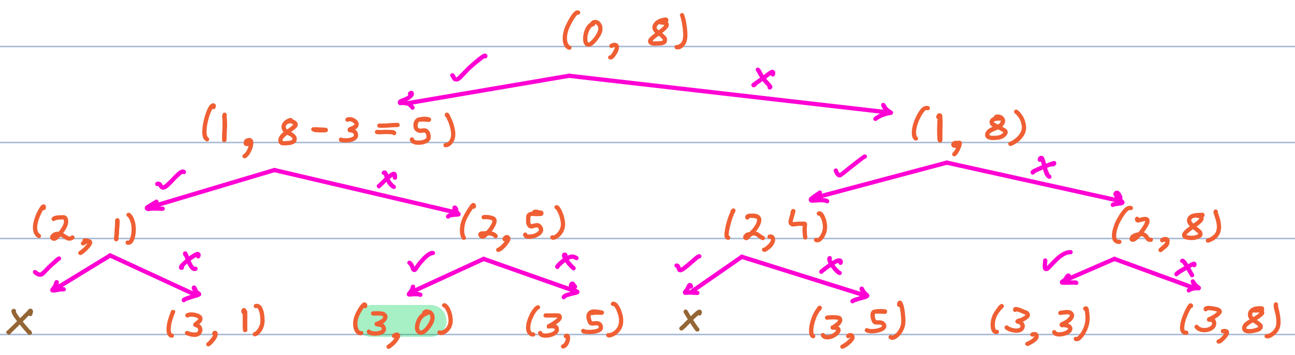
∀ A[i] → select
→ reject

$$A = \begin{array}{c} \begin{array}{cccc} 0 & 1 & 2 & 3 \end{array} \\ [\begin{array}{cccc} 3 & 4 & 5 & 2 \end{array}] \end{array} \qquad K = 8$$

Ans = true

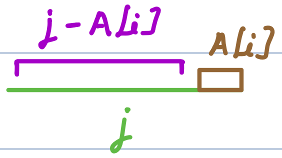
↑

(index, K)



optimal substructure
overlapping subproblems } use DP

dp[i][j] → check if subset sum = j considering elements from 0 to i.



$$\begin{tikzcd} dp[i][j] \arrow[r, "\checkmark"] \arrow[dr, "\times"'] & dp[i-1][j-A[i]] \text{ OR} \\ & dp[i-1][j] \end{tikzcd}$$

∀i, dp[i][0] = true (empty subset)

```
for i → 0 to (N-1)                    0—(N-1) , 0—K
        dp[i][0] = true
if ( K >= A[0] )   dp[0][A[0]] = true
for i → 1 to (N-1) {
    for j → 1 to K {                          select                  reject
            if ( j >= A[i] )   dp[i][j] = dp[i-1][j - A[i]] || dp[i-1][j]
            else      dp[i][j] = dp[i-1][j]
    }
}  return dp[N-1][K]
```

TC = O(N * K)   SC = O(N * K)

O(2K) = O(K) ✓

KnapSack Problem

profit / loss

Given N objects with their values Vi and their weights Wi. (cost)

A bag with capacity W that can be used to carry some objects such that →

budjet

total sum of objects weights ≤ W , and

sum of values in the bag is maximised.

Fractional KnapSack

Given N cakes with happiness and weight.

Find max total happiness that can be kept in a bag with capacity = W

(cakes can be divided)

N = 5 happiness[ ] → [3 8 10 2 5] sum = 23

W = 40 weight[ ] → [10 4 20 8 15]

20 - 4 = 16 40 - 20 = 20 16 - 15 = 1

→ use bag completely

→ How to select ⇒ h[i] / w[i]

N = 5

W = 40

	0	1	2	3	4
happiness[ ] →	[3	8	10	2	5]
weight[ ] →	[10	4	20	8	15]
h/w →	0.3	2	0.5	0.25	0.33
# parts →	10	4	20	8	15
	4	1	2		3

Sum = 2 * 4 ✓ + 0.5 * 20 ✓ + 0.33 * 15 ✓ + 0.3 * 1 ✓

= 8 + 10 + 5 + 0.3 = <u>23.3</u> (Ans)

<u>Sol</u> → select items in descending order wrt h[i]/w[i] till complete capacity is used. (Greedy)

TC = <u>O(N log(N))</u> SC = <u>O(N)</u>

Practical Scenario

Flipkart is planning a special promotional event where they need to create an exclusive combo offer. The goal is to create 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 item in the combo doesn't exceed a predefined.

0 - 1 KnapSack

Given N toys with their happiness and weight.

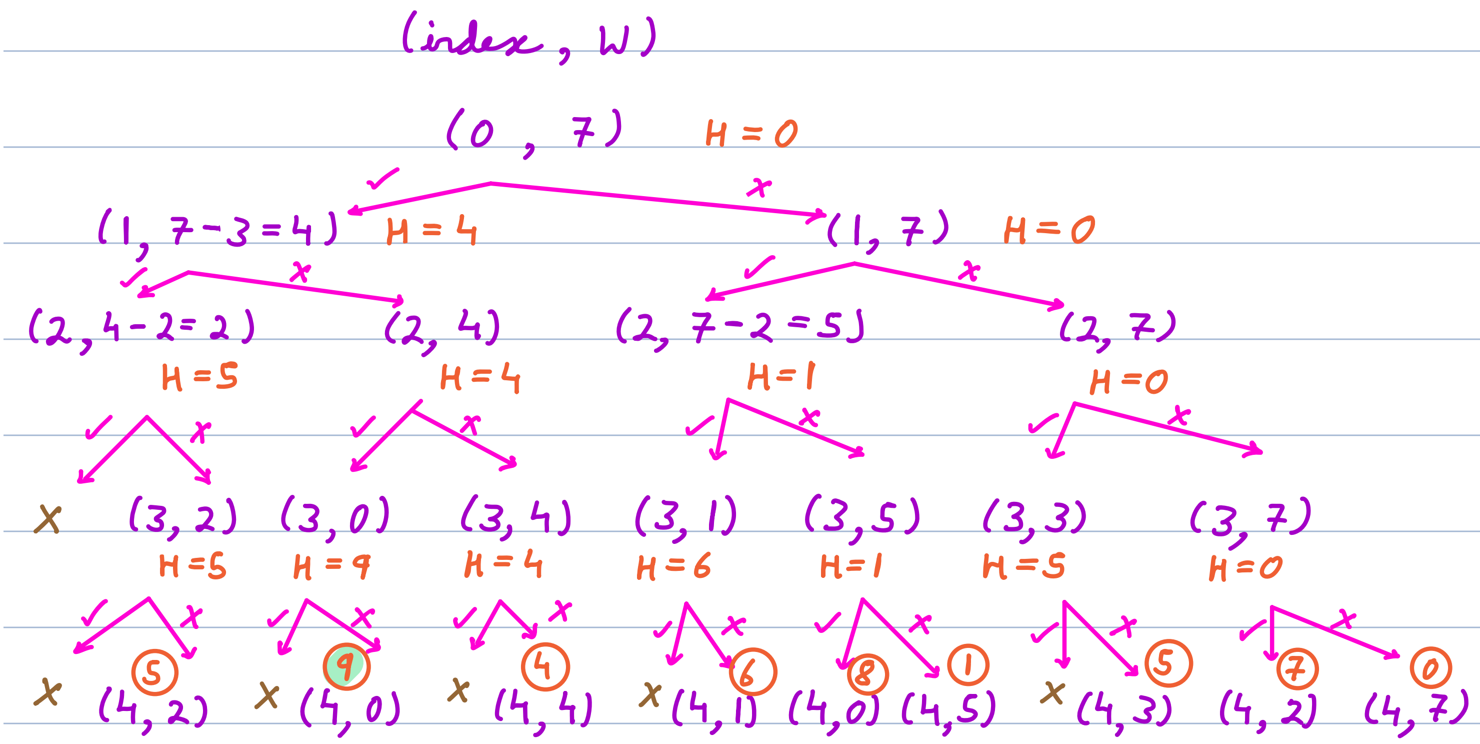
Find max total happiness that can be kept in a bag with capacity W.

[Toys can't be divided]

N = 4

W = 7

$$\begin{array}{llcccc} & & 0 & 1 & 2 & 3 \\ \text{happiness[ ]} & \rightarrow & [\ 4 & 1 & 5 & 7\ ] \\ \text{weight[ ]} & \rightarrow & [\ 3 & 2 & 4 & 5\ ] \\ & & & & & \uparrow \end{array}$$



dp[i][j] → max total happiness considering elements till index i & capacity j.

$$\begin{tikzcd} dp[i][j] \arrow[r, "\checkmark"] \arrow[dr, "\times"'] & h[i] + dp[i-1][j-w[i]] \\ & dp[i-1][j] \end{tikzcd}$$

$\forall i,\ dp[i][0] = 0$ (no capacity)

$\forall j,\ dp[0][j] = 0$ (nothing to buy)

// 1-based index

```
∀i, dp[i][0] = 0
∀j, dp[0][j] = 0
for i → 1 to N {
    for j → 1 to W {
        if ( j >= w[i])
            dp[i][j] = max (h[i] + dp[i-1][j-w[i]], dp[i-1][j])
        else    dp[i][j] = dp[i-1][j]
    }
} return dp[N][W]
```

TC = $\underline{O(N*W)}$ SC = $\underline{O(N*W)}$

$O(2W) = \underline{O(W)}$

Bottom - up

0 1 2 3 4

weight[ ] → [3 6 5 2 4]

W = 8

N = 5

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

i \ j	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	20	20
✓3	0	0	0	12	12	15	20	20	27
4	0	0	6	12	12	18	20	21	27
5	0	0	6	12	12	18	20	22	27

(Ans)

Find the items selected.

```
i = N        j = W
if ( dp[i][j] == dp[i-1][j] ) → i th item is rejected
        i = i - 1
else → i th item is selected
        i = i - 1        j = j - w[i]
```

Unbound KnapSack (0 - ∞ 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]

Items can be selected ∞ times.

$$\begin{array}{lcccc} & 0 & 1 & 2 & \\ \text{value[ ]} \rightarrow [& 2 & 3 & 5 &] \\ \text{weight[ ]} \rightarrow [& 3 & 4 & 7 &] \end{array}$$

W = 8 , N = 3

2 times Ans = 6

$$\begin{array}{lccc} & 0 & 1 & \\ val = [& 1 & 30 &] \\ wt = [& 1 & 50 &] \end{array}$$

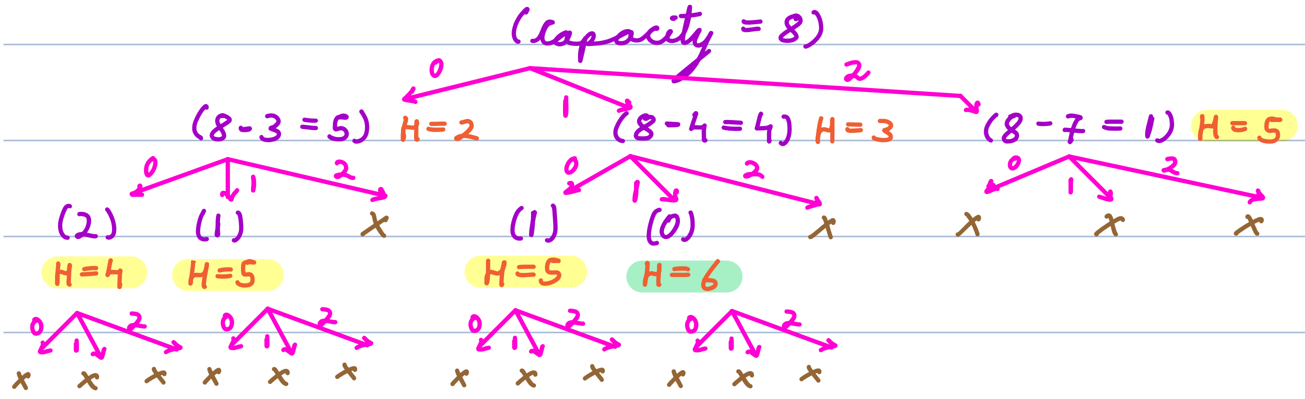
W = 100

Ans = 1 * 100 = 100

100 times

$$\begin{array}{lcccc} & 0 & 1 & 2 & \\ \text{value[ ]} \rightarrow [& 2 & 3 & 5 &] \\ \text{weight[ ]} \rightarrow [& 3 & 4 & 7 &] \end{array}$$

W = 8 , N = 3



dp[i] → Max happiness with capacity i.

$dp[0] = 0$

$$dp[i] = \underset{0 \le j \le N-1}{\max} \left(h[j] + dp[i - w[j]] \right)$$

```
dp[0] = 0
for i → 1 to W {
    for j → 0 to (N-1) {
        if (i >= w[j])
            dp[i] = max (dp[i], h[j] + dp[i- w[j]])
    }
} return dp[W]
```

TC = $\underline{O(N * W)}$ SC = $\underline{O(W)}$

0 1 2

W = 8 , N = 3

dp →									
	0	1	2	3	4	5	6	7	8