Today's Agenda:- Starting 7:05

1. Understanding Knapsack.
2. 3 different Types of Knapsack Problems.
3. 1 Question

A combination of following entities in a question constitute the knapsack problems.

1. No. of objects — N
2. Two attributes of the object
   Value — v_i
   Weight — w_i
3. Capacity of my bag (knapsack) — W

We are trying to maximise the Value that can be kept in a bag.

1. Fractional Knapsack (Greedy Algo'n)

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

h[] - [4 8 10 2 5]
w[] - [4 4 20 8 16]

W = 40

Idea: Pick the cake with max happiness / weight ratio first.

h[] - [4 8 10 2 5]
w[] - [4 4 20 8 16]

h/w → [1. 2 0.5 0.25 5/16]

W = 40 36 32 12 0

Happiness = 0 8 12 22 + 5/16 × 12³
4.
3.75

= 25.75

5/16 × 12.

16 unit → 5 unit
1 unit → 5/16
12 unit → 5/16 × 12

Algorithm

① Sort the data based on decreasing order of h/wt ratio.

[4 8 10 2 5]
[4 4 20 8 16]

⇓ sort

[8 4 10 5 2]
[4 4 20 16 8]

h/w → [2 1 0.5 5/16 0.25].

Code: # After sorting is done.

```
double ans = 0, selectedwt = 0;
for (i = 0; i < N; i++) {
    if (wt[i] + selectedwt
            <= capacity) {
        ans += h[i];
        selectedwt += wt[i];
    }
    else {
        ans += (capacity - selectedwt)
                * h[i] / wt[i];
        break;
    }
}
return ans;
```

O(N lg N).

S.C. O(N);

2. 0 / 1 Knapsack

Given N toys with their happiness & weight. Find max total happiness that can be kept in a bag with capacity = W

(toys can't be divided)

N = 4
W = 7

h → { 4 1 5 7 }
w → { 3 2 4 5 }

h/wt = { 1.33 0.5 1.25 1.4 }. 9.

W = 7 · 20
N = 7 8

∴ Greedy is not going to be helpful.

B. F idea:- Consider every subset of toys & check if they give the max h.

O(2^N).

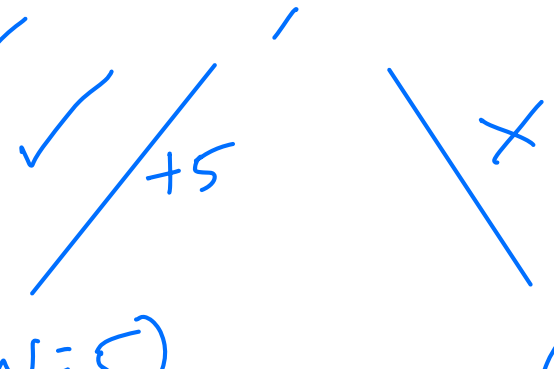
$h[] = [8 \quad 3 \quad 6 \quad 4 \quad 5]$ $W = 8, \quad N = 5$

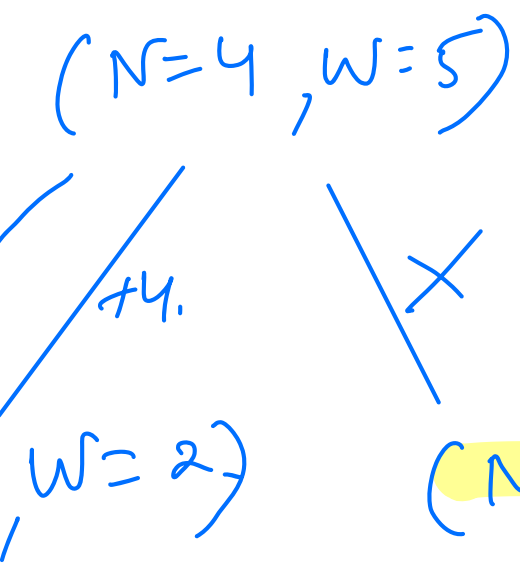
$wt[] = [5 \quad 2 \quad 4 \quad 3 \quad 3]$

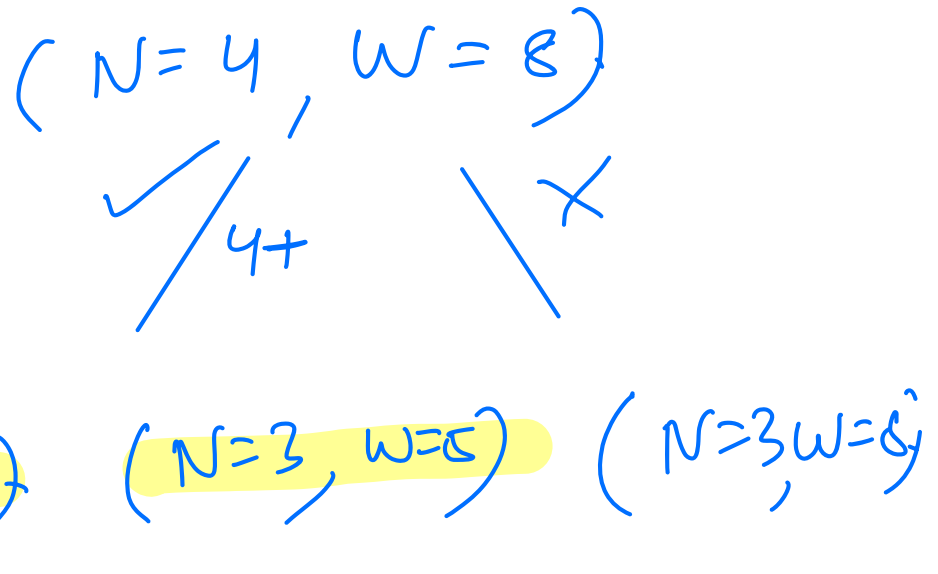
(dry run B.F. idea)

Optimal SS ✓ $N=5, W=8$

Overlapping S.P. ✓







$$\max \text{Happiness}(N, W) = \max \begin{cases} \max H(N-1, W) \\ h_N + \max H(N-1, W-w_N) \end{cases}$$

$dp[i][j]$

$$dp[i][j] = \max \begin{bmatrix} dp[i-1][j]; \\ dp[i-1][j-wt[i-1]] + h[i-1]; \end{bmatrix}$$

$wt[i-1] \leq j$

↗ dp[N][W].

① DP state

dp[i][j] → max. happiness for first i toys & j weight.

② DP expression

$$dp[i][j] = \max\begin{bmatrix} dp[i-1][j]; \\ dp[i-1][j-wt[i-1]] + h[i-1]; \end{bmatrix}$$

③ Base conditions.

dp[i][0] = 0;
dp[0][j] = 0;

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

V = 5
W = 8

dp[N+1][W+1]

dp[2][1].

dp[2][6]

20 + dp[2-1][6-6]

20 + dp[1][0]

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

	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								

ans:-

Code:

```
int dp[N+1][W+1];
# Initialise 0th row & 0th col with 0
for(i=1; i<=N; i++){
    for(j=1; j<=W; j++){
        # Not selecting
        f1 = dp[i-1][j];
        f2 = 0; # By selecting.
        if(wt[i-1] <= j){
            f2 = val[i-1] + dp[i-1][j-wt[i-1]];  # only change for 0/∞.
        }
        dp[i][j] = max(f1, f2);
    }
}
return dp[N][W];
```

T.C → O(N*W)
S.C → O(N*W)
↓
O(W)
(N·W).

S.C. → O(W) approach:

```
int dp[W+1];
for (int i=1; i<=n ; i++){
    for (int j= W; j >= 0 ; j--){
        if (w[i-1] <= j ) {
            dp[j] = max (dp[j],
                dp[j-w[i-1]]
                + h[i-1])
        }
    }
}
return dp[W];
```

Top - Down

```
int dp[N+1][W+1] // initialize every cell with -1
                 // N, W

int maxH(h[], wt[], i, j, int[][] dp){
  if(i==0 || j==0) {return 0;}
  if(dp[i][j] != -1) {return dp;}
  f1 = 0 + maxH(h, wt, i-1, j, dp);
  f2 = 0;
  if(wt[i-1] <= j){
    f2 = h[i-1] + maxH(h, wt, i-1,
                       j-wt[i-1], dp);
  }
  dp[i][j] = max(f1, f2);
  return max(f1, f2);
}
```

T.C → O(N*W)
S.C → O(N*W)

3. Unbounded Knapsack (0 / ∞).

(0 / ∞ Knapsack)

→ You can pick any element any no. of times

2 min.

value [] — [2 3 5] W = 8

wt [] — [3 4 7] or = 6.

✓ ✓

value [] — [8 5 12] W = 5

wt [] — [3 2 4] or = 13.

v/wt — [2.66 2.5 3]

Greedy can not be applied..

$$maxH(N,W) = max\begin{cases} h[N-1] + maxH(N, W-w+[N-1]) \\ 0 + maxH(N-1, W). \end{cases}$$

Code.

① DP State

$dp[i][j] \rightarrow$ max. happiness for first i toys & j weight.

② DP expression

$$dp[i][j] = max\begin{bmatrix} dp[i-1][j]; \\ dp[i][j-w[i-1]] \\ \uparrow + h[i-1]; \end{bmatrix}$$

only change.

③ Base conditions.

$dp[i][0] = 0;$
$dp[0][j] = 0;$

$\nearrow dp[N][W].$

T.C → $O(N*W)$
S.C → $O(N*W).$

S.C: $O(W).$

```
dp[W+1] ; // dp[i] = 0;

for(i=1; i<=W; i++) {
    for(j=1; j<=N; j++) {
        if(w[j-1] <= i) {
            dp[i] = max(dp[i], h[j-1]
                        + dp[i-w[j-1]])
        }
    }
}
```

Q:- Given an array with the positive integers. Flip sign (multiply by -1) some of its elements such that sum of elements of final array is non. non-negative integer. Find min. elements to flip.

Ex:- [10 15 6 5 3 3]

= [10 15 -6 5 -3 3].

= 24.

= [-10 -15 6 5 3 3].

= -8.

= [10 -15 -6 5 3 3]

= 0.

or = 2.

Obs:-

① Two choices → flip
→ not be flipped.

value → 1.

w → value of that elem.

max. value of the flipped elems.

= sum/2.

S. T.C. → O(S/2 * N).

S. Sum of flipped elements = x.

S - 2x ≥ 0

x ≤ S/2.

(5 2 3). - 2 - 2.