Agenda

- Target Sum
- Knapsack
  - Fractional
  - 0-1
  - Unbounded

Given an array of non negative integers, and a target sum K, check if there is a subset with sum K.

Ex : [3, 34, 4, 12, 5, 2] K = 9

<4, 5>
<3, 4, 2>

ans = true

BF : Generate all subsets, maintain cur sum, compare with K

TC : $O(2^N)$

SC : O(N) → stack space

Optimized :

N = 6

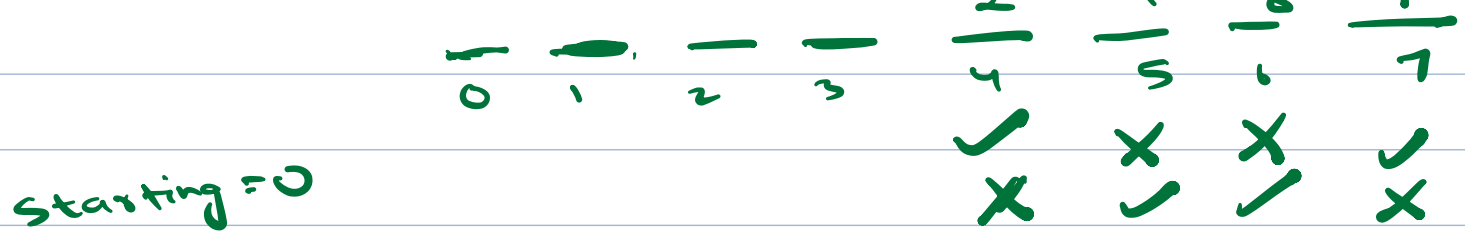
0 1 2 3 4 5
3, 34, 4, 12, 5, 2 K = 9

isPossible (0-5, 9) (sum)

isPossible (0-4, 7) OR isPossible (0-4, 9)

isPossible (e, s) → is it possible to find a subset in 0-e with sum = s

K = 10



isPossible(end s
3, 1)

isPossible(end s
3, 1)

int dp[N][K+1] = {-1}

```
int isPossible(int e, int s) {
    if (s == 0)   return true/1
    if (s < 0)    return false/0
    if (e == -1)  // s>0   return false/0

    if (dp[e][s] != -1) return dp[e][s]

    // include arr[e]
    bool i = isPossible(e-1, s-arr[e])
    //exclude arr[e]
    bool e = isPossible(e-1, s)

    dp[e][s] = i || e
    return dp[e][s]
}
```

TC: O(N×K)

SC: O(N×K + N+K) = O(N×K)

// arr, N, K → I/P

return isPossible(N-1, K)

K = 25

[4 6 8 10 20]

dp → -1 initial value, prob not solved

→ 1 isPossible = true

→ 0 isPossible = false

Customized Shopping Recommendations

N items → Price
→ Happiness value

Find max happiness value within budget

	Price	Happiness
1	110	39
2	180	57
3	50	13
4	120	44
5	100	24

Budget = 300

Cost = 300 Val = 57 + 44
= 101

Price	Happiness
20	10
10	8
10	4
10	1

Budget = 20

Greedy : sort based on happiness does not work

N , Budget

maxH(5, 300)

price ≤ budget

4th idx

MAX

x
5th idx

24 + maxH(4, 200) maxH(4, 300)

maxH(N, B) = max happiness I can achieving by buying some items out of first N items (0 → N-1) given buyt B

=

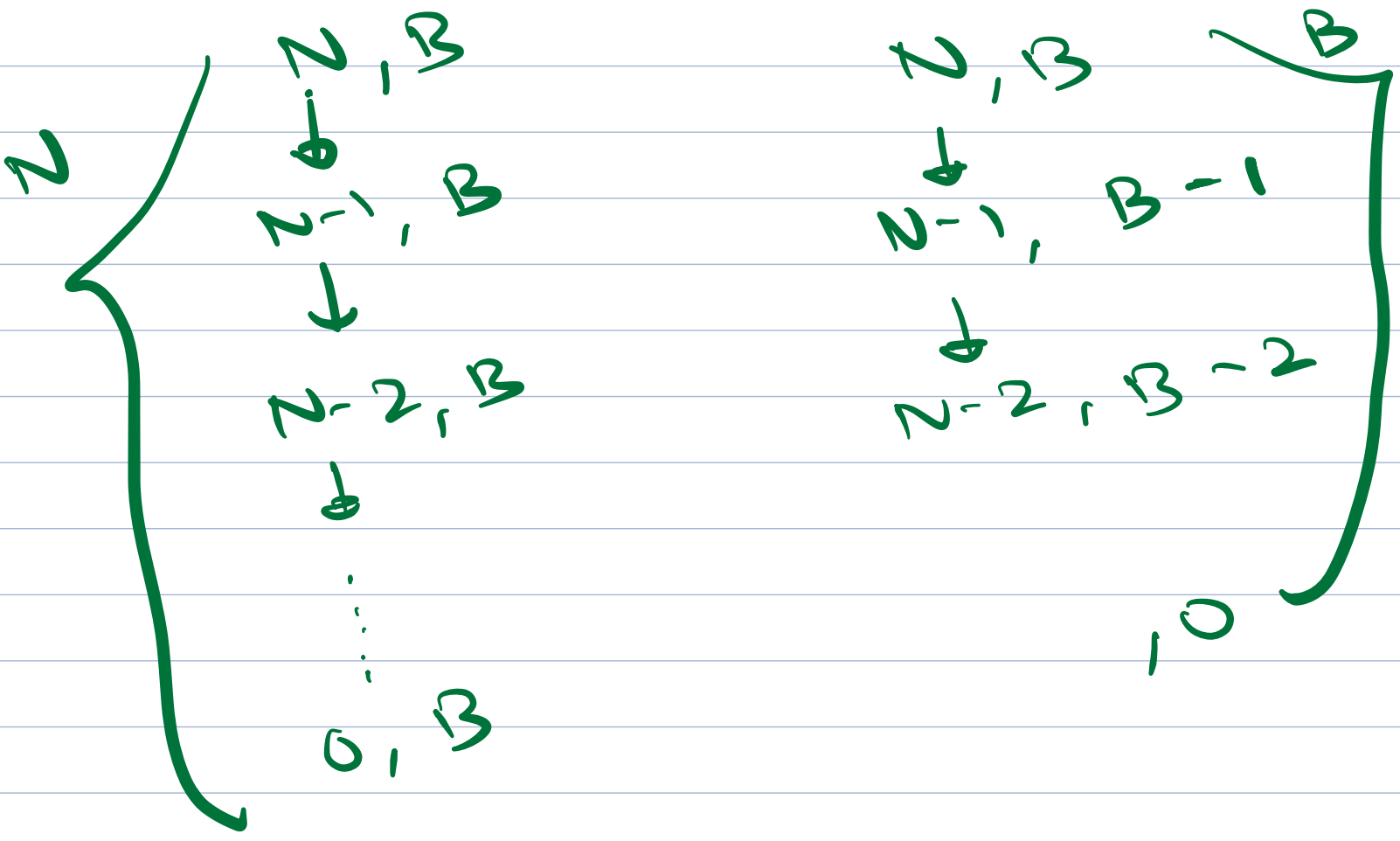
N 5
0-4

```
int dp[N+1][B+1] = {-1}

int maxH(int N, int B) {
    if (B == 0) return 0
    if (B < 0) return INT_MIN / -2
    if (N == 0) return 0

    if (dp[N][B] != -1) return dp[N][B]

    // include last item
    int i = happiness[N-1] + maxH(N-1, B-price[N-1])

    // exclude last
    int e = maxH(N-1, B)
    dp[N][B] = max(i, e)
    return dp[N][B]
}
```

TC: O((N+1)(B+1))

SC: O((N+1)(B+1))

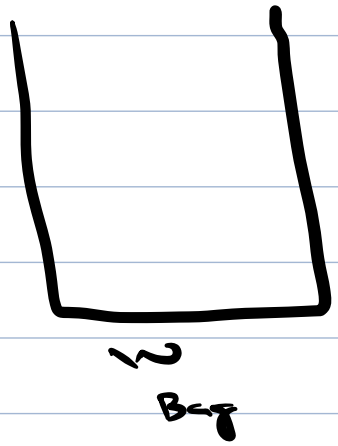
// price[], happiness[], N, B



Stack space : min (N, B)

Knapsack

N objects → Value → Profit
→ Weight

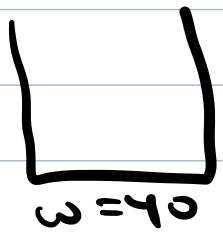


1. Fractional Knapsack

N Cakes → Happiness
→ Weight

	0	1	2	3	4
H =	3	8	10	2	5
W =	10	4	20	8	15

N = 5



Max Happiness such that overall weight $\leq W$

	0	1	2	3	4
H =	3	8	10	2	5
W =	10	4	20	8	15

0 item
10 kg → 3
1 kg → 3/10

3 item
8 kg → 2
1 kg → 2/8

Overall H = 8 + 10 + 5 = 23
Overall W = 4 + 20 + 15 = 39 $\leq W$ 40

	0	1	2	3	4
H =	3	8	10	2	5
W =	10	4	20	8	15

15 kg → 5
1 kg → 5/15
10 kg → 1/3 × 10

Overall H = 3 + 10 + 0.3 = 13.3
Overall W = 10 + 20 + 10 = 40 $\leq W$ 40

	0	1	2	3	4
H =	3	8	10	2	5
W =	10	4	20	8	15

ans = 23.3

0 item
10 kg → 3
1 kg → 3/10

3 item
8 kg → 2
1 kg → 2/8

overall H = 8 + 10 + 5 + 0.3 = 23.3
overall W = 4 + 20 + 15 + 1 = 40 ≤ W

Sol: Find happiness/weight ratio of every cake, sort the cakes based on ratio

	0	1	2	3	4
H =	3	8	10	2	5
W =	10	4	20	8	15

H/W = 0.3, 2, 0.5, 0.25, 0.33

id →	1	2	4	0	3
H →	8	10	5	3	2
W →	4	20	15	10	8

2, 0.5, 0.33, 0.3, 0.25

Fractional

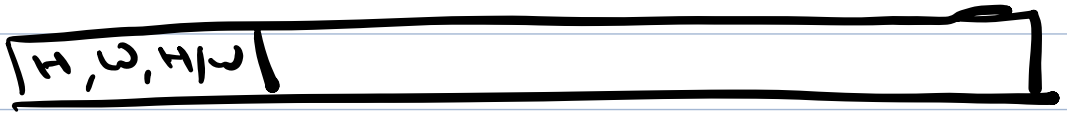
W = 40

W = ~~40~~ ~~30~~ 10 + 0

ans = 8 $\xrightarrow{+10}$ 18 $\xrightarrow{+5}$ 23 $\xrightarrow{+0.3}$ 23.3

// H[], wt[], N, W

// list of triplets, list of list



// calculate H/W of every item
// sort 'items' list based on H/W
in descending order

```
capacity = W
int ans = 0
for (i=0 ; i<N ; i++) {
    if (weight[i] ≤ capacity) {
        ans += happiness[i]
        capacity -= weight[i]
    }
    else {  // fractionally
        ans += (capacity) * ratio[i]
        capacity = 0
        break
    }
}
return ans
```

```
class cake {
    int happiness
    int weight
    int ratio
```

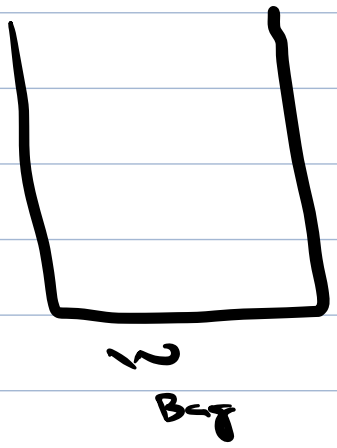
TC: O(N log N)

SC: O(N)

10:45

Knapsack (0-1)

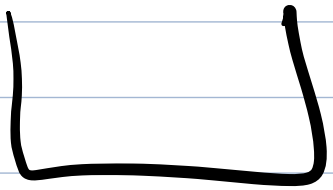
N objects → Value → Profit
→ Weight



Toys (division is not allowed)

N = 4

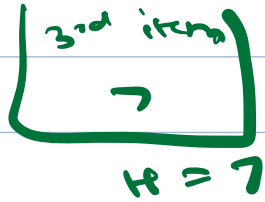
	0	1	2	3
H =	4	1	5	7
Wt =	3	2	4	7
H/W =	1.33	0.5	1.25	1.4



Overall H = 4 + 5 = 9

Overall W = 3 + 4 = 7 ≤ 7

Item 0 and 2



W = 7

N = 4

	0	1	2	3
H =	4	1	5	7
W =	3	2	4	7

maxH(4, 7)

N, B

3rd idx — MAX — 3rd idx

7 + maxH(3, 0)

maxH(3, 7)

dp[N][B] =

maxH(N, B) = max Happiness I can achieving by buying some items out of first N items (0 → N-1) ,given buget B

dp[i][j] = max Happiness I can get by buying some items out of first i items (0 → i-1), given capacity is j

dp[i][j]

(i-1) item / MAX / x

dp[i-1][j]

H[i-1] +

dp[i-1][j-wt[i-1]]

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

row → cnt of items

col → capacity of bag

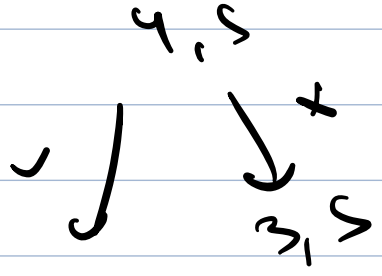
```
for (j=0 ; j <= W ; j++){
    dp[0][j] = 0
// (i==0) ; no item available
}
```

```
for (i=0 ; i <= N ; i++){
    // j==0 ; capacity of bag is 0
    dp[i][0] = 0
}
```

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

TC : $O((N+1)(W+1))$

SC : $O((N+1)(W+1))$



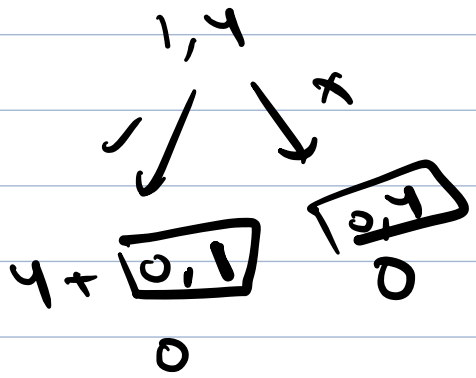
N = 4 W = 7

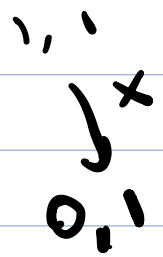
	0	1	2	3
P =	4	1	5	7
Wt =	3	2	4	5

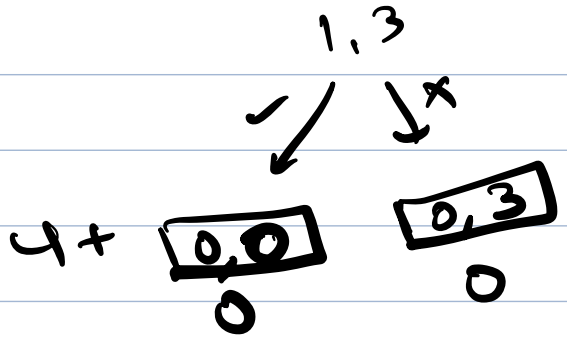
dp[N+1][W+1]
5 8

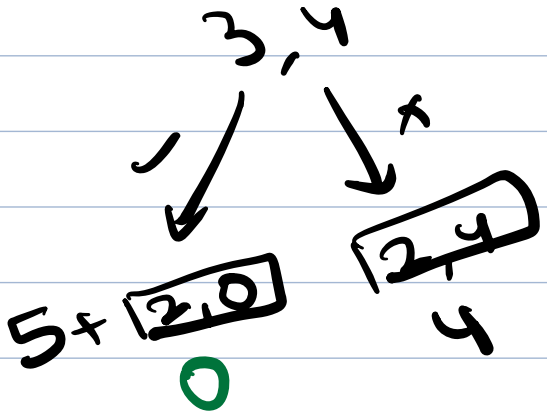
capacity

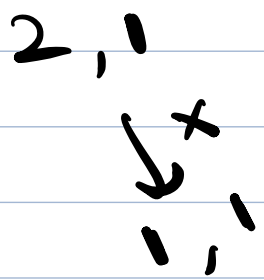
cnt of items	0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0	0
1	0	0	0	4	4	4	4	4
2	0	0	1	4	4	5	5	5
3	0	0	1	4	5	5	6	9
4	0	0	1	4	5	7	7	9

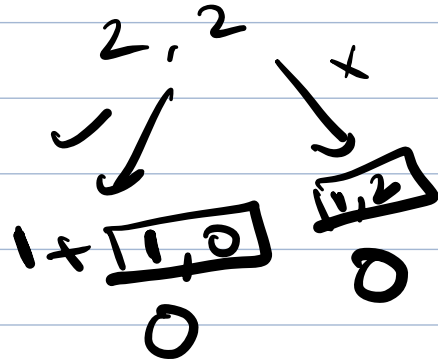


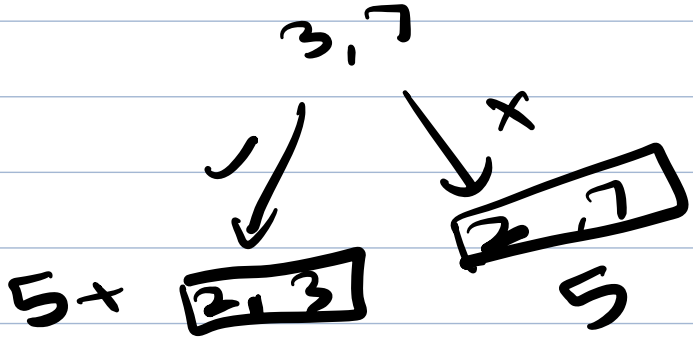












Maintain prev and cur row
dp[i-1] dp[i]

int prev[W+1] = <0>, cur[W+1]

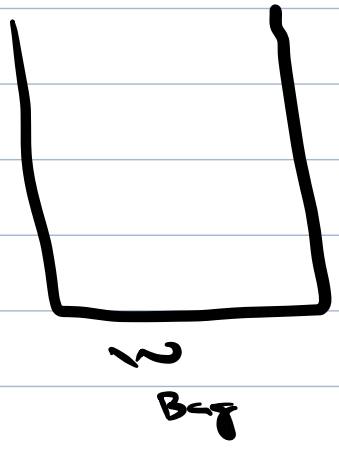
SC: O(2×W) = O(W)

Unbounded Knapsack / 0-N Knapsack

0-∞

N categories → Value → Profit
→ Weight

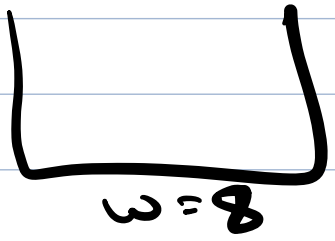
Toys (division is not allowed)



N=3

	0	1	2
H =	2	3	5
Wt =	3	4	7

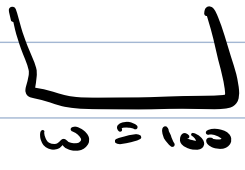
ans = 6



Toy 1 + Toy 1
Overall H = 3 + 3 = 6
Overall wt = 4 + 4 = 8 ≤ W

N = 2

	0	1
H =	1	30
wt	1	50



Toy 0 100 rimy H = 100

N = 3 W = 8

	0	1	2
H =	2	3	5
Wt =	3	4	7

maxH(3, 22, cnt)

leave the toy ✗ → maxH(2, 22)

pick the toy once or more ✓ → 5 + maxH(3, 15)

✗ → maxH(2, 15)

✓ → 5 + maxH(3, 8)

```
int dp[N+1][B+1] = {-1}

int maxH(int N, int B){
    if(B==0) return 0
    if(B<0) return INT_MIN / -6
    if(N==0) return 0

    if(dp[N][B]!=-1) return dp[N][B]

    // include last item
    int i = happiness[N-1] + maxH(N, B-price[N-1])

    // exclude last
    int e = maxH(N-1, B)
    dp[N][B] = max(i, e)
    return dp[N][B]
}
```

TC : O(NB)

SC : O(NB)