Recap: DP   2 kinds of problems → Optimization (superlative) and Combinatorial (#ways)

Combinatorial → 1. Direct pattern. 2. Recurrence Relation.

Optimization:

1. Greedy? Yes → Proof. No → Counter case

→ Think of recurrence relation. Q. Does recursion make sense?

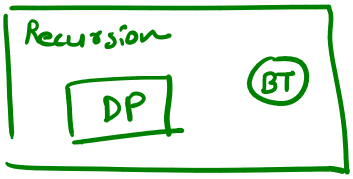
2. Ad hoc? → Intuitive Recursion?

Yes. ↓ DP.   No ↓ Ad hoc techniques.

DP → repetitive calls.

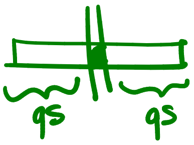
Caching → memoization

DP = Recursion + Memo.



$$\text{Recursion.} \xrightarrow[\text{Ensure Repetitive calls.}]{\text{fixed pattern}} \text{DP}$$

Note: Quicksort.

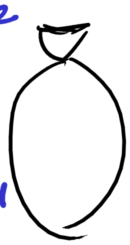


Classic template: Knapsack Problem

1. n items → price[]
→ weight[]

$$\begin{bmatrix} 10 & 6 & 18 & 12 & 20 \\ 2 & 1 & 7 & 4 & 5 \end{bmatrix}$$

maximize
constrained



c=12kg.

Q. What items should I put in the bag to maximize the price of the bag?

greedy → price↑ weight↓ $\frac{\text{price}}{\text{weight}}$ → price per unit } Counter case.

DOESNT WORK.

p: [ ]
w: [ ]

Reason: you don't know the actual weight.

C=12kg   p: 23$ 20$ 4$
w: 11kg 10kg 2kg

$\frac{p}{w}$: 2.09 2 2
X

→ Greedy: C = 1 kg. | O/P: 23

C = 2kg | Price = 20

0kg | O/P: 24

n items, C capacity, ans = 0

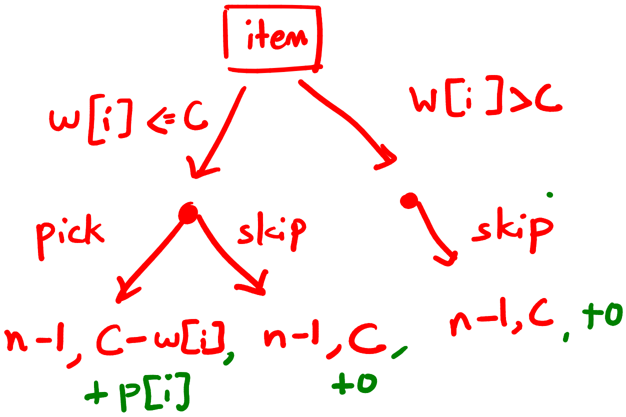
Repeat.

1. Greedy X

2nd: Recurrence relation.

Optimize ↓ price of bag?

what happens? items bag.



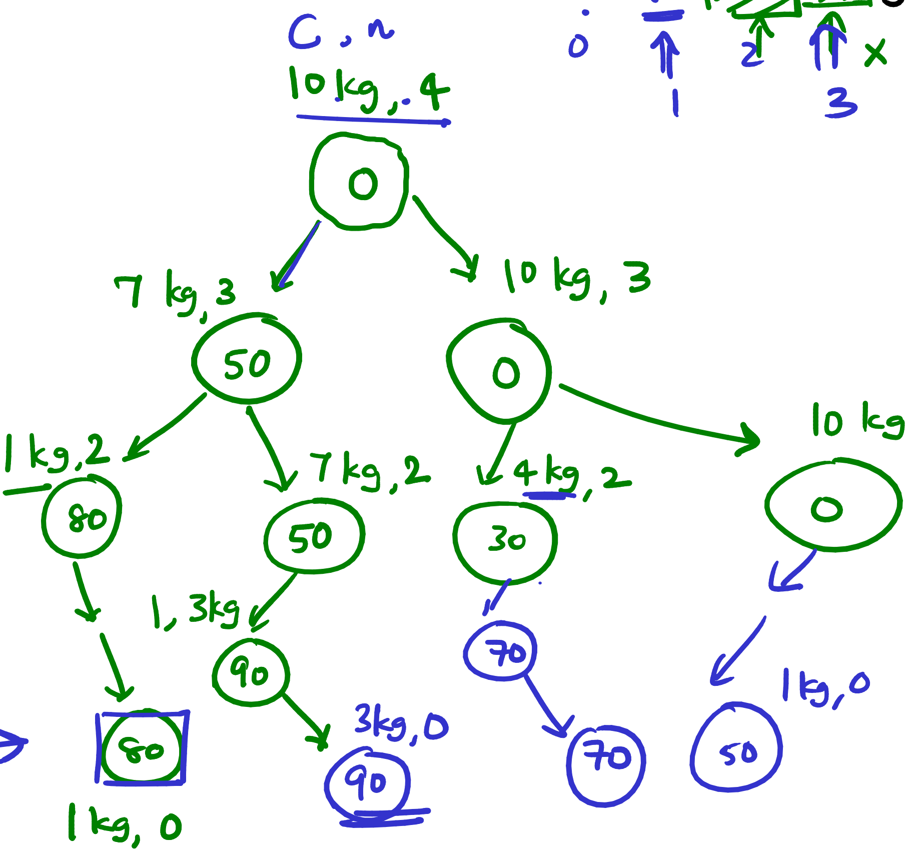
Dry run : C = 10 kg

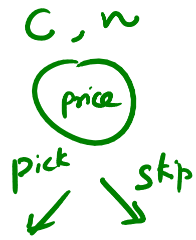
n = 4

items: $\begin{array}{c|cccc} w = & 5 & 4 & 6 & 3 \\ p = & 10 & 40 & 30 & 50 \\ & 0 & 1 & 2 & 3 \end{array}$ x

O/P: 90

Decomposition





maximum value : 90

Call : Knapsack(w, p, n-1, 0)

```
Code !       max value
             of bag
Idea         int  knapsack (int weight[], int price[], int n, int C):
                 if C <= 0 or n == -1 :    return 0

                 if w[n] > C :
                     return  knapsack(weight, price, n-1, C)

                 else:
return  max {           knapsack (weight, price, n-1, C),
             price[n] + knapsack(weight, price, n-1, C - weight[n]) }
```

TC. ~ $O(2^n)$ exponential.

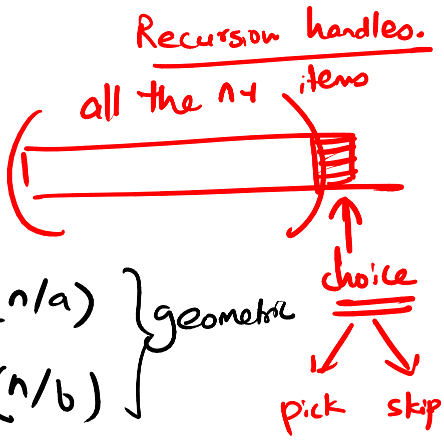
$f^n(n)$: → $(2^{n/a})$

$f^n(n-a)$, $f^n(n-b)$ } reducing arithmetically

fibonacci

$(\log_a n)$ $f^n(n/a)$, $f^n(n/b)$ } geometric

Mergesort Quicksort



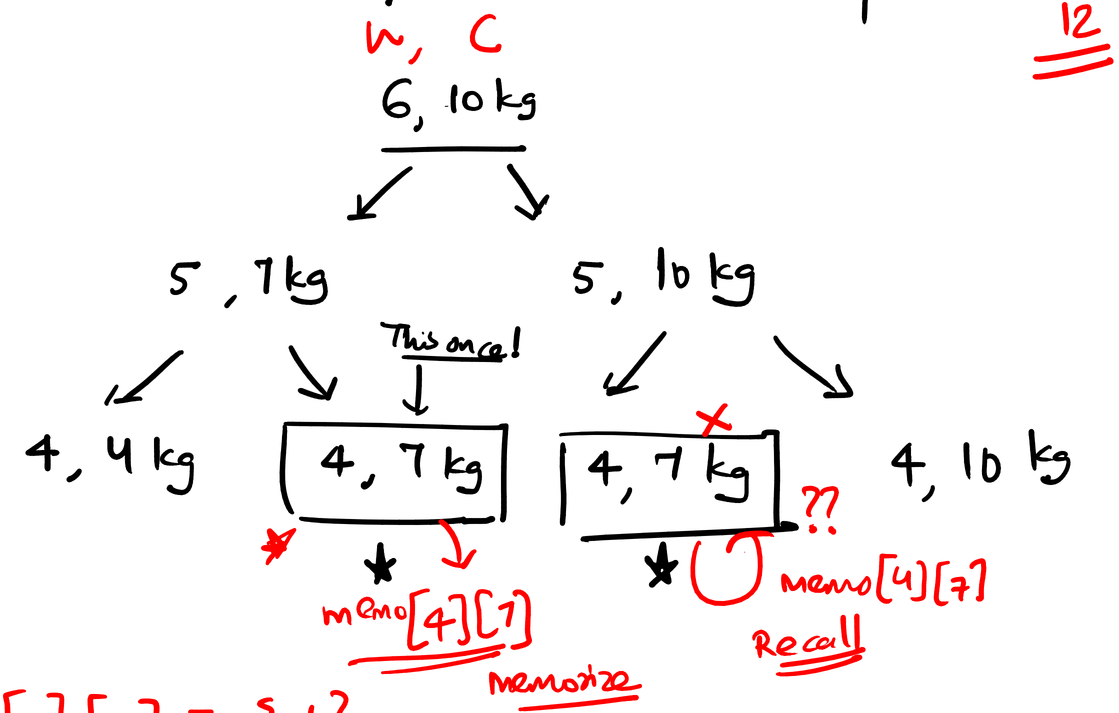
Do we need to memoize it ? → Are there repetitive calls ?

(6) n items =

C = 10 kg

P_1	P_2	P_3	P_4	10 $	~~12 $~~
W_1	W_2	W_3	W_4	3 kg	~~3 kg~~

12



Memoize :

```
int memo[n][c] = {-1}

int knapsack ( int w[], int p[], int n, int c) :
    if n==-1 || c <= 0 :    return 0

    if memo[n][c] != -1 :   return memo[n][c]

DP.
    int ans = 0

    if w[n] > c :
        ans = knapsack (w, p, n-1, c)

    else
  * {  ans = max( knapsack (w, p, n-1, c) ,
                  p[n] + knapsack (w, p, n-1, c-w[n]) )

memoization →  memo [n][c] = ans
    return ans
```

T.C O(n.c)

Problem : Deeper!   n items = { ___ }

i^{th} item

pick   skip

Q. Subset Sum Problem

sum = 11   arr = [1, 2, 3, 5, 7, 12]

are there some elements that sum to the given sum? → Two sum. pair

→ Yes/No.

set = {1, 3, 7}   sum(set) = 11.

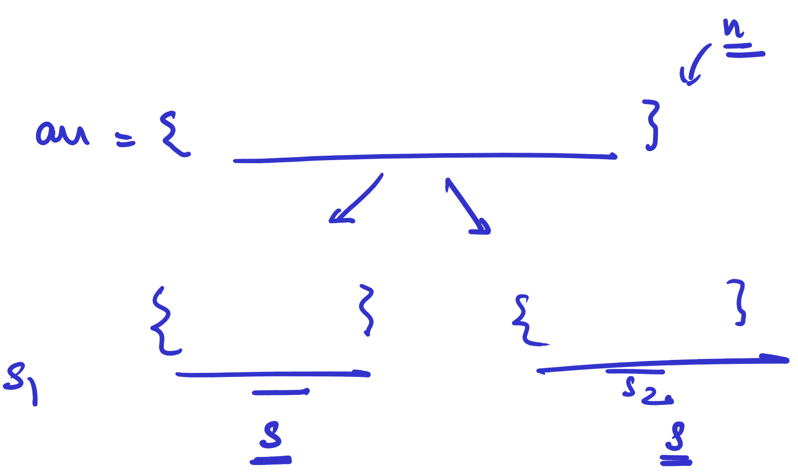
Code:

```
bool ss(int arr[], int n, int sum):
    if sum == 0 : return true
    if n == -1 : return false

    if arr[n] > sum:
        return ss(arr, n-1, sum)
    else:
        return ss(arr, n-1, sum) ||
               ss(arr, n-1, sum - arr[n]);
```

memo[n][sum]

Eg. Equal sum partition.



arr = [1, 3, 4, 5, 5, 8]

pick   skip

[1, 3, 4, 5]   [5, 8]

A   NA

sum(arr) = 2s $s = \frac{sum(arr)}{2}$

ss(arr, n-1, sum(arr)/2)

Eg. Target sum. arr = { 1, 3, 5 } sum = 3

pick

+1 -3 +5 skip ⇒ 3

sum{ } - sum{ } = sum (target)

+ve -ve

A ~A

Importance : item arr[i] → Knapsack Q.

pick skip

Why it works ? $O(2^n) \longrightarrow O(n.C)$

Pruning of tree.

Is it that I am missing out on some possibilities ? If yes, why so sure ?

Redundant recursion calls are not made.

If not, there there are no 2^n possibilities you imply ?

0/1

_ _ _ _ _ _
0 1 2 3 4 5

Intuition : I never asked for the actual subset.

6 places

0 to $2^6 - 1$

Knapsack → max price possible → But cant tell me which item did I add in bag ?

Subset sum problem → Yes/No to be able to make the sum. → what elements are making that sum ? ×

Actual subset → Backtracking !! $O(2^n)$