2 kinds of problems : 1. Optimization → maximum, minimum, shortest, longest, highest

2. Combinatorial → #ways, how many ways, can you count...

① Minimum no of coins to make the change.

denominations/coins = [1, 2, 5, 10, 20, 50]

change = 786 Rs.

Sol^n: 750 + 20 + 10 + 5 + 1

50×15 1 1 1 1

min # coins = 15 + 4 = 19.

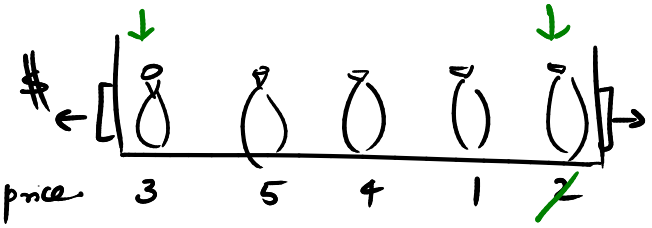
Greedy approach.

whatever is leftover, you try to get maximum value out of it.

② Shelf of wines.

price of wine → doubles every year

consume 1 bottle per year



Consume these bottles in such a order that I drink wine of max value.

Sol^n:

1 year → $2

2 year → $2

3 year → $12

4 year → $32

5 year → $80

O/P: 128 $

6	10	8	2
12	20	16	
	40	32	
	80		

Greedy

↳ minimum so that time increases the value of remaining bottles.

I want to do something n times.

— — — — — ...

max max max max max ...

Greedy principle :

If I do optimal decisioning at every single step, I will have the optimal answer overall.

At each step you make optimal choice.

Interesting :- Does greedy always work?

1 : Greedy works because every next coin is atleast double the previous coin. $C_{i+1} \geq 2 * C_i$

Coins = {1, 3, 4} change = 6

Greedy:

'6' - 4
'2' - 1
'1' - 1
—
0

} 3 coins. (1, 1, 4)

min coins that from this change.

2 coins {3, 3}. GREEDY FAILED!

Eg: Minimum steps to 1.

User: input 'n' → number.

3 operations:
1. if n%3 == 0 : n → n/3
2. if n%2 == 0 : n → n/2
3. n → n-1

Min. operations to reduce n to 1.

Eg:
n = 10 ??
greedy
÷2 → 5 → -1 → 4 → ÷2 → 2 → -1/÷2 → 1 (4 ops.)
-1 → 9 → ÷3 → 3 → ÷3 → 1 (3 ops)

Greedy fails!!

** n = 10^5 → nlogn

* Thinking for a approach:-

① Given question is optimization problem.

↳ 1. Is it greedy? → Searching + Sorting + Common sense logic.

Eg: n elements of array. You want to find min/smallest absolute diff of any 2 elements.

Sol: sort it & find consecutive ele diff : O(nlogn) → Why did you sort?

form all pairs & minimize the diff : $O(n^2)$

Greedy. → Sure it works. Why? → Logic:

sorted: a b c

c-a > b-a, c-b

Is it going to fail?
- No → Prove: greedy will work always. This is the ans.
- Yes → Come with the counter example.

2. Is this ad-hoc problem? ⟶ Eg. Max area under histogram (stacks), min/max substring (2 ptr and sliding window)

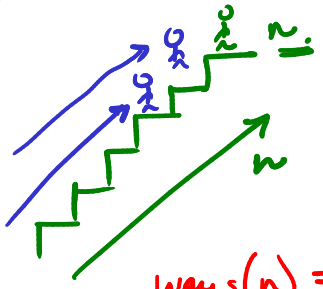
this specific/particular.

If you can't come up with a recurrence relation → high chances it's an adhoc problem.

3. Recurrence Relations → Recursion. | Looping vs. Recursion.

Myth breaker: Staircase problem.

Eg: climb {1, 2} steps on the staircase.



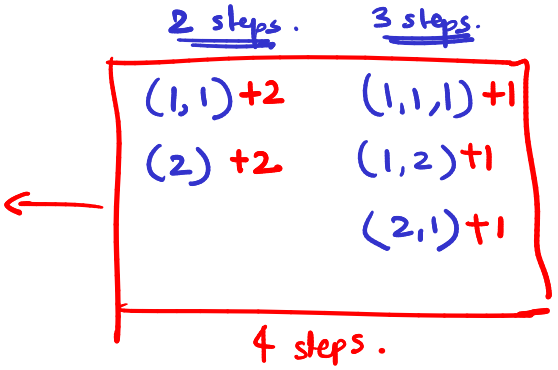
Recursion was natural.

#ways can I climb staircase of n steps?

n = 4

op: 5

(1,1,1,1)
(1,1,2)
(1,2,1)
(2,1,1)
(2,2)



ways(n) = ?
ways(n-1) + ?
ways(n-2)

n', n'' < n

Recurrence relation. $f(n) = f(n') + f(n'') + \dots$

Recall: quicksort.: arr = { } ↓ after partition

partition( ) pivot.

Think inductively !!

3 shirts
2 tshirts
4 jeans
} # ways you can get dressed?

(2) Combinatorial problem → #ways

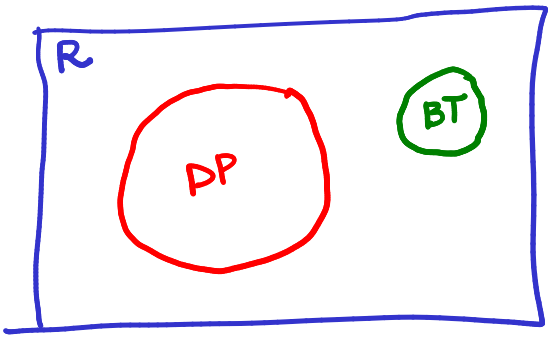
1. Is there a pattern? → Using a formula involving P&C.

and → x OR → +

shirt x jean + tshirt x jean
3 x 4 + 2 x 4 = 5 x 4
= 20
(shirt + tshirt) x jean

2. Always a recurrence relation.
(rare exception)

Greedy → Recursion → DP.
✗ ✓ 99%.



Problems solved using recursion

DP → 1. Combinatorial / Optimization
2. Foresight (ability to see something in future).

Coin change. coins = {1, 3, 4} change = 6

optimal choice : 4, 1, 1

DP : 3, 3

#coins to make change

change = 0

BT → Backtracking
game solving.
levels (win cond^n.) possibilities (moves.)

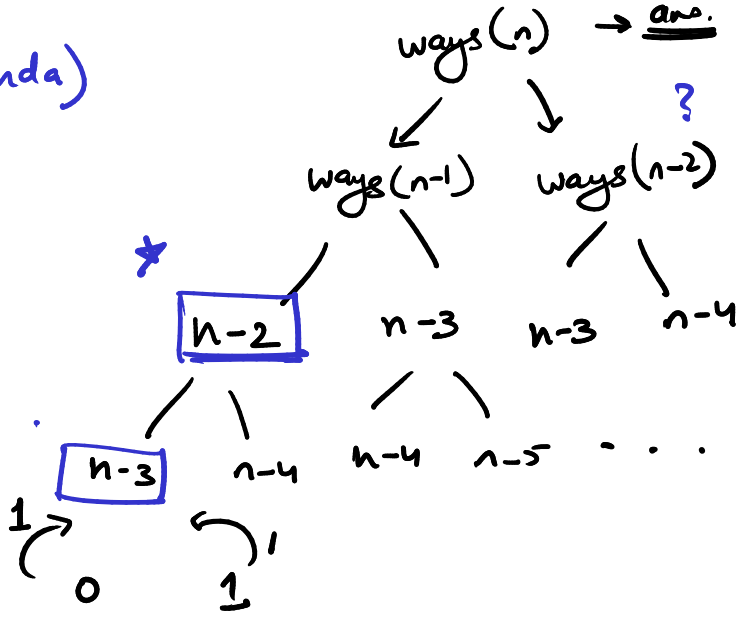
DP → Dynamic Programming

"Foresight": You take suboptimal decision sm, so that you can take more optimal decision later.

"Optimal decisions every step doesn't mean optimal result overall. all the times." ~ DP/Life

"Code with memory": "Remember the mistakes/lessons from the past, otherwise you'll repeat." ~ DP.

Optimization → Caching (kinda)



Recursion.

```
fib(n):
Base | if n==0 || n==1:
         return 1
Decomp | int a = fib(n-1)
       | int b = fib(n-2)
Recomp | return a+b
```

~ $O(2^n)$

Ensure: Repetitive calls

Mechanical Conversion ↓ Memoization

DP

```
int memo[n] = {-1}
fib(n):
  if n==0 || n==1:
      return 1
Recall | if memo[n] != -1:
       |     return memo[n]
  int a = fib(n-1)
  int b = fib(n-2)
Memorization | memo[n] = a+b
             | return memo[n]
```

Optimize → ~ $O(n)$

DP = Recursion + Memoization.

Conversion steps:

1. Look at the args that are changing. Make a memo table of that size.
2. Recall: check if ans for that 'n' is precomputed. Return that ans directly.
3. If you have computed the ans → store it now, so that you can recall it later.

Framework: Φ.

1. Understand the question.
2. Why greedy fails? OR Can I sense foresight?
3. I want to come up with natural/intuitive recursions. → game over !!
4. Mechanical Conversion.