All Data Structures are built on top of

① Arrays
② Linked Lists (DS)

① loops | recursion

② Recursion → Trees
Heap
Dynamic Programming
Graph
Backtracking
} Heavily use Recursion

---

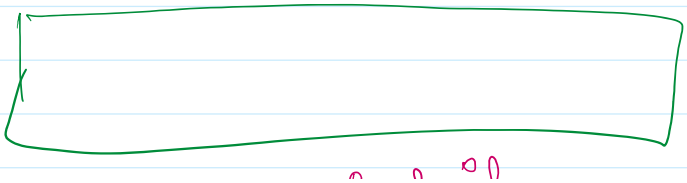
## Recursion trees

→ A function call itself

Types direct Recursion · → f → f

Indirect Recursion f → g → f

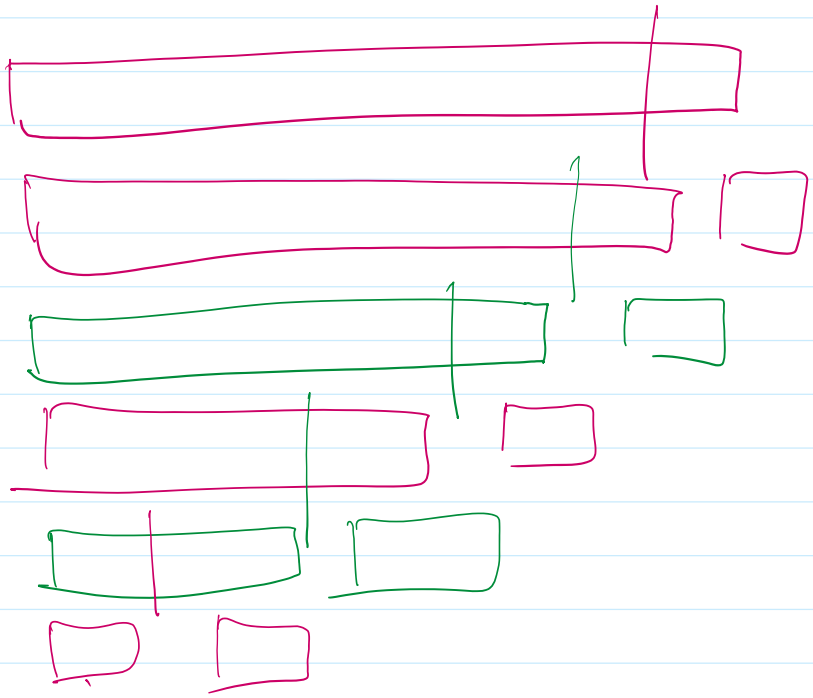
Recursion



problem

Break it → smaller problem

Recursive Case



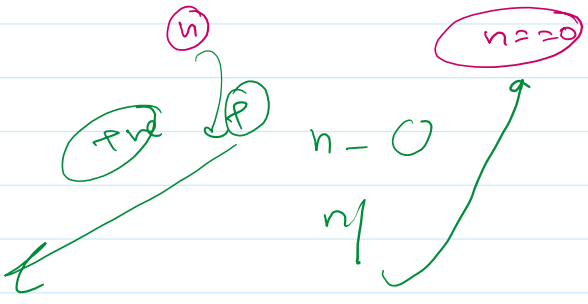
Obvious answer → Base Case

problem → small that we have an obvious answer

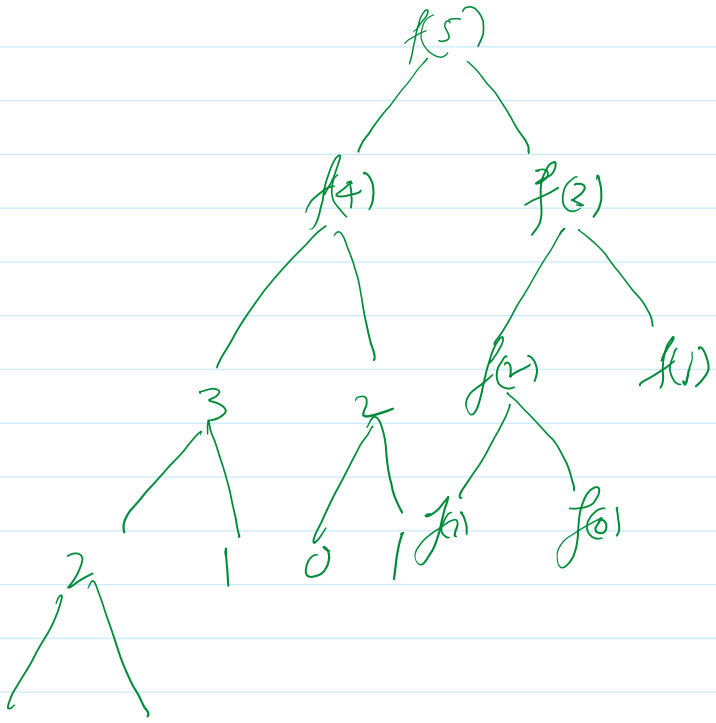
① Base case → Known Answe // Know As

② Recursive case → Breaking — How you break

move towards base case



fibonacci 0, 1, 1, 2, 3, 5, 8, 12 - - - -

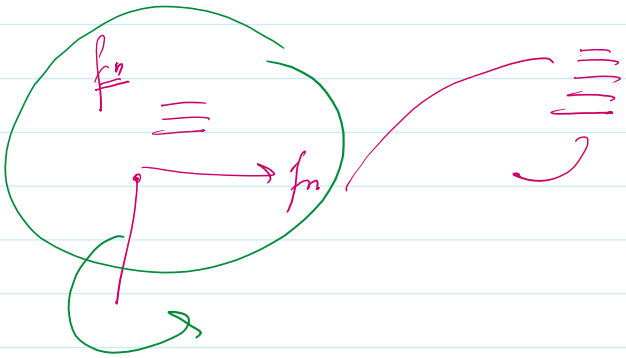



① for all recursive functions you must draw recursion tree

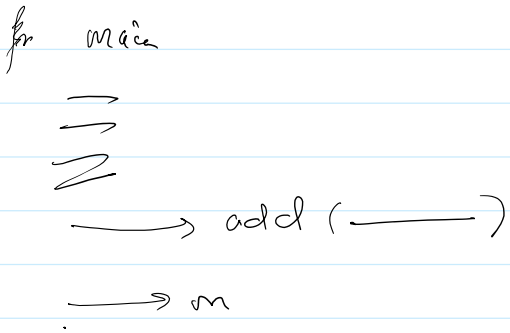
# Complexity Analysis

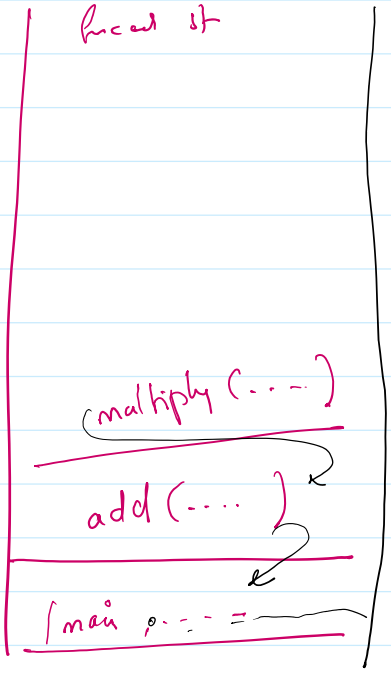
Recursion — ① function call stack space
② function call overhead

fn calling fn




Execution always start from main




function call stack space

function call overhead — extra time to save context and create variables

cpp [ ]

loop

Recursion

① fn call ↑

② fn call overhead ]

Tail Recursion

last call is recursive call

→ Read about

easily converted to loop | without any stack

Solve chapter 1

→ Recursion [ ]

Asymptotic Notations

↳ Time & Space Complexity

DOC doubt we can discuss

① at least 1 question each day:

- Sheet [ ] { daily update }