11/01/22 Prequisites → functions & memory management

(from Recursion video)

How fn calls works internally :-

```
psvm(String[] args){
    print1(1);}
static void print1(int n){
    sout(n);
    print2(2);}
static void print2(int n){
    sout(n);
    print3(3);}
static void print3(int n){
    sout(n);
    print4(4);}
static void print4(int n){
    sout(n);}
```

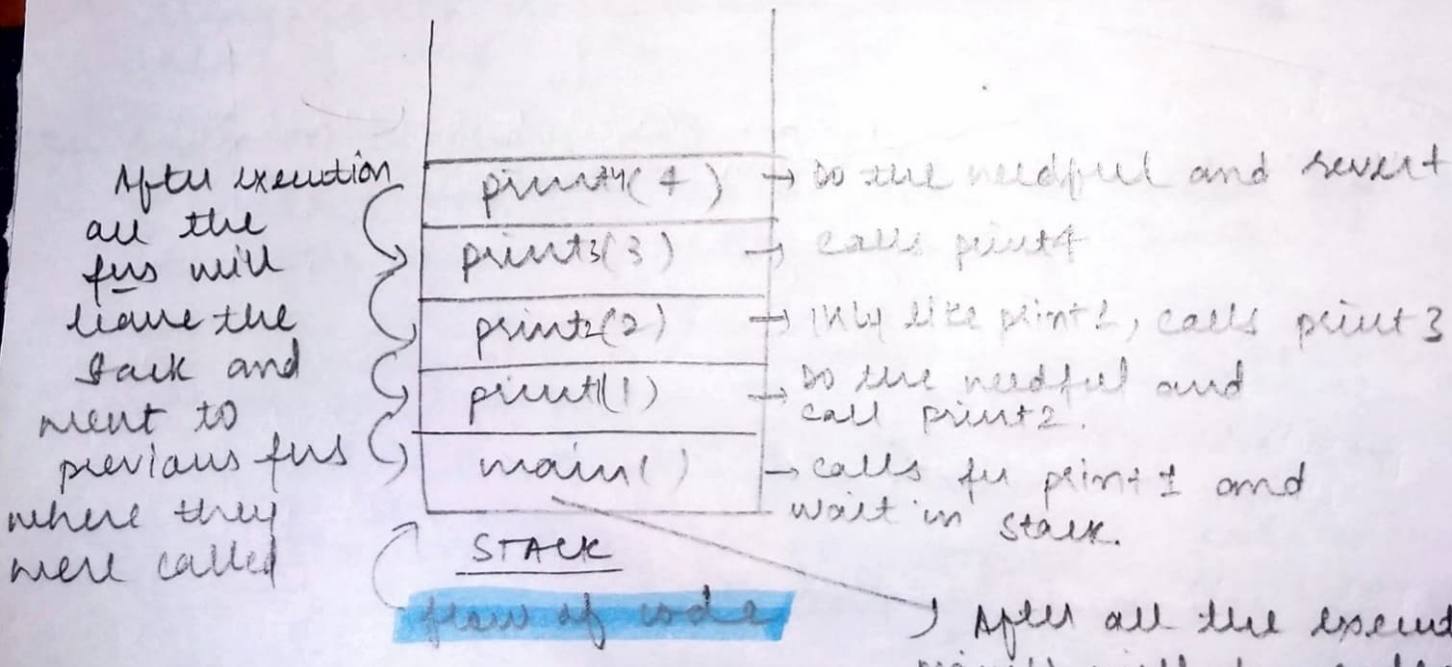
1st to go & last to leave in stack

primitives are also stored in stack

★ while the fn is not finished executing, it will remain in stack.

★ when a fn, finishes executing, it is removed from stack and the flow of program is restored to where the fn is called.

★ execution of fns are occurs in STACK.



[Program over]

13-01-22 | Prequisites - fn & memory mngmnt

RECURSION :- fn that calls itself.

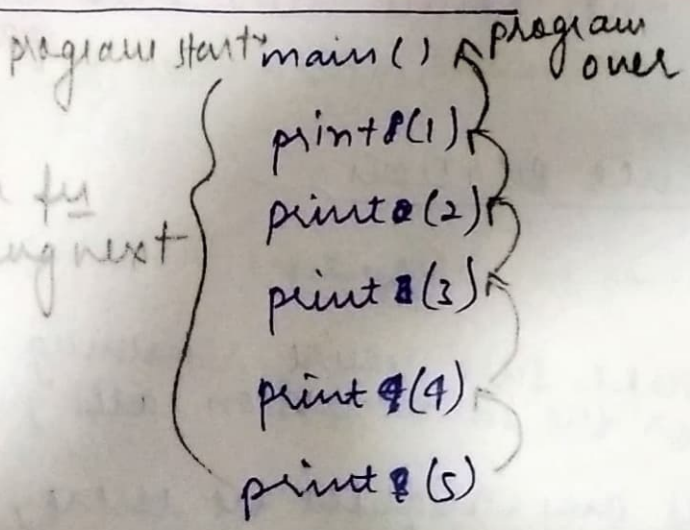
→ **Base Condition:** condition where our recursion will stop making new calls. It needs to be returned.
Answers that already provided you

→ Why we use Base condition?
If we don't use it, fn calls will keep executing, and with every call stack is filled separately. And due to ~~it~~ repeatedly calling stack memory exceeds its limit and will throw error, termed as STACKOVERFLOW ERROR.

→ Why do we need Recursion
→ helps us in solving biggest/complex problems in a simpler way.
→ can convert recursion solutions into iteration and vice-versa.

→ Space complexity is not const because of recursive calls.
→ Helps us in breaking down bigger problems to smaller.

→ VISUALIZATION Recursion



RECURSION TREE

→ Identify the solution can be solved via Recursion.

- If you can break it down into smaller problems.
- when you write formula for recursion, called recursive relation / reccurence relation
- Base condition is what we already know about the question or the answers we already have.

→ APPROACH

1. → Breaking down
2. → recursion relation
3. → Recursive tree
   - @ → see the flow of fns, how they are storing in stack
   - b → identify and focus on left & right side calls in the tree.
   - c → Draw tree & ptr again & again using pen & paper.
   - d → use debugger for better understanding.
4. → See how values are returned at each step.
5. → see where the fn call has come out of.

(key area for recursion)

key area for recursion → variables, data types and, return value and what to pass.
(arguments, parameters, return value / type.)

→ Types of Recurrence Relation

1. Linear → Not efficient as it calls same task repeatedly.
2. Divide & conquer

→ make sure you're returning whatever the subrecursion calls giving.

→ In any recursive fn, if any variables are there, you need to pass in future fn calls put it inside the argument.
agar koi variable ki value kar fn call ke sath change ho rahi to fn parameters me rakho usko.

Example -

fibonacci Series

0 1 2 3 4 5 6 7

0, 1, 1, 2, 3, 5, 8, 13 ....

Divided into two smaller problems,

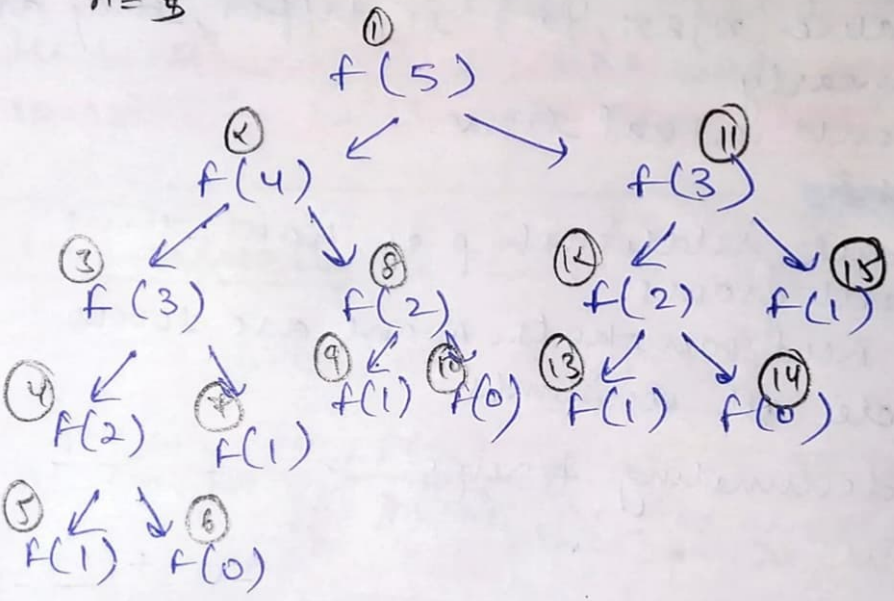
$f(n) = f(n-1) + f(n-2)$

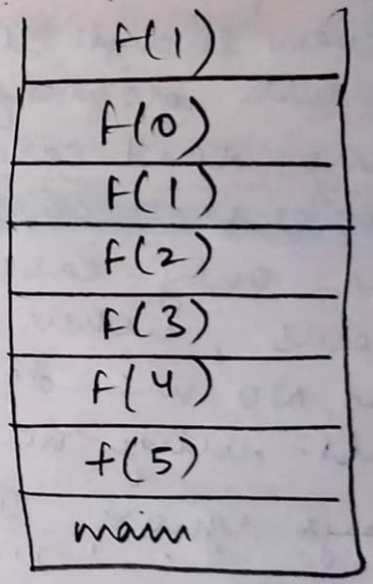
Code -

```
int f(int n){
if n=0,1 if (n < 2) { // base condition
        return n; }
    return f(n-1) + f(n-2);
}
```

Recursion tree

$n = 5$

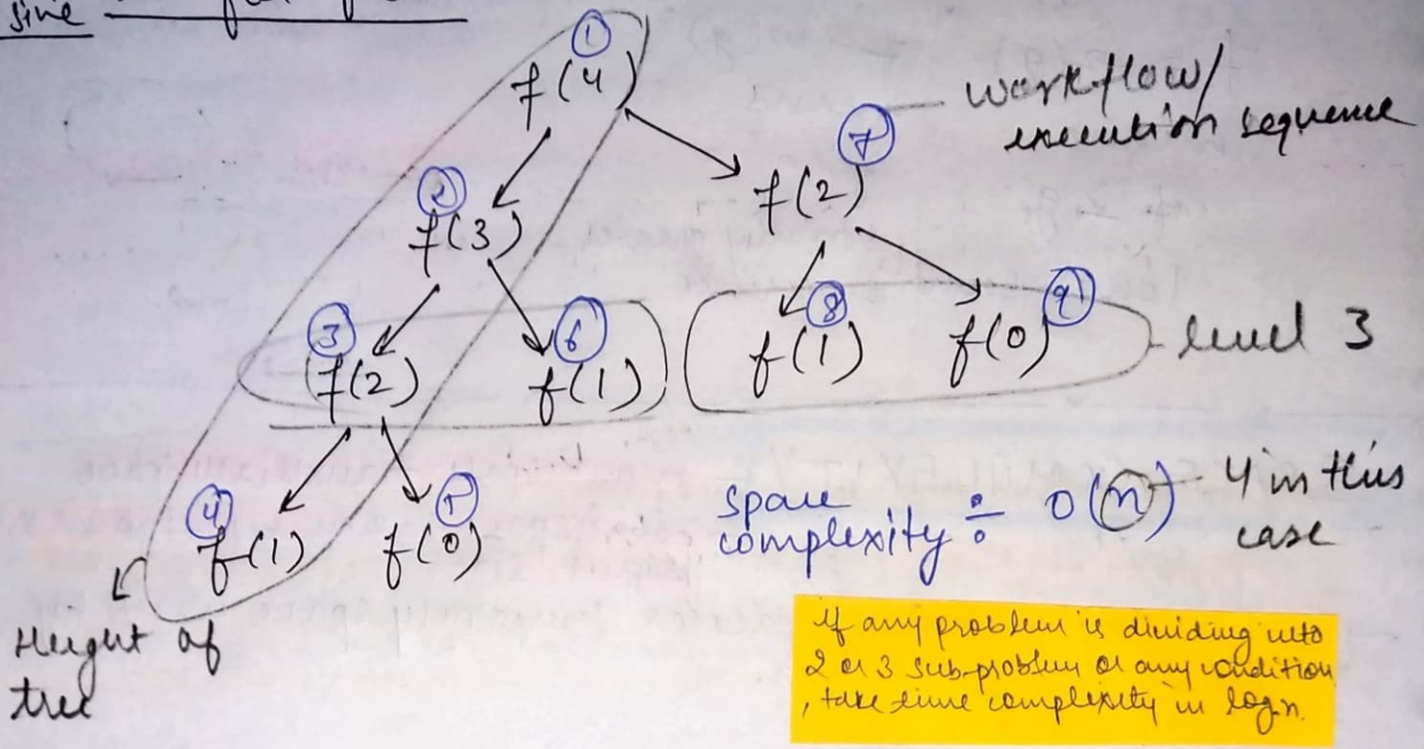




Recursive Algorithms :-

→ In recursion programs, space complexity will not be constant, because fn calls stored in stack.

Recursive tree for fibo



space complexity :- O(n) 4 in this case

If any problem is dividing into 2 or 3 sub-problem or any condition, take time complexity in log n.

→ Space complexity in recursive programs or in tree is actually the height of tree.

→ At any particular time, no two fns call at same level of recursion will be in stack.

ek time pe bas linked fns stack me rahenge or jab wo execute hojayenge to stack se remove hojayenge. Or tree me same nahi ek level ke fns stack me ho ke kyunki same nahi wo ek dusre se link ho. (see level 3 of tree)

→ interlinked calls of fn will be in stack at same time.

→ space complexity = height of tree or pattern

longest chain → root to last leaf of tree or any other pattern.

How to actually solve to get complexity

1. Plug & chug → substitution method
2. Master's theorem
3. Akra Bazzi (MIT → 1996) → skipped from Kunal Kushwa video

→ AKRA BAZZI

Bit manipulation → Practice Questions
Bitwise video in Kunal

→ See Abdul Bari video for computing Time Complexity on youtube and some topics of complexity

21-01-22

→ RECURRENCE RELATION (we can find time complexity from these relations)

Abdul Bari → Playlist → Algorithms

→ We write recurrence relation to see the next step of recursive calls in our program.

→ we can say that it the eqn form of recursive tree.

→ can find complexity from tree method too.

22-01-23

for Decreasing function recurrence relation

* $T(n) = T(n-1)+1 \longrightarrow O(n)$
* $T(n) = T(n-1)+n \longrightarrow O(n^2)$
* $T(n) = T(n-1)+\log n \longrightarrow O(n \log n)$
* $T(n) = T(n-1)+n^2 \longrightarrow O(n^3)$
* $T(n) = T(n-2)+1 \longrightarrow \frac{n}{2}\, O(n) \rightarrow O(n)$
* $T(n) = 2T(n-1)+1 \longrightarrow O(2^n)$
* $T(n) = 2T(n-1)+n \longrightarrow O(n2^n)$
* $T(n) = 2T(n-2)+1 \longrightarrow O(2^{n/2})$

MASTER THEOREM FOR DECREASING FUNCTIONS

General form: $T(n) = aT(n-b) + f(n)$

$a > 0$, $b > 0$ and $f(n) = O(n^k)$ where $k \geq 0$

→ case 1 - if $a < 1$ → $O(n^k)$
$O(f(n))$

→ case 2 - if $a = 1$ → $O(n^{k+1})$
$\{O(n * f(n))$

→ case 3 - if $a > 1$ → $O(n^k a^{n/b})$
$O(f(n)\, a^{n/b})$