Recurrence Relation

- A recurrence is an equation that describes a function in terms of its values on smaller inputs.

- When an algorithm takes a recursive call to itself, its running time can often be described by a recurrence.

- A recurrence is a recursive description of a function or in other words, a description of a function in terms of itself. Like all recursive structures, a recurrence consist of one or more base cases and one or more recursive cases.

- Eg: $f(n) = n$ can be described as —

$$f(n) = \begin{cases} 0 & \text{if } n = 0 \quad f(0) = 0 \\ f(n-1) + 1 & \text{otherwise} \quad f(n) = f(n-1) + 1 \end{cases}$$

$\forall n > 0$

(1) line is base case.

(2) is recursive case.

- We usually formulate a recurrence relation as a function that is written in terms of itself (recursive case) and constant value when i/p is small (base case)

3 ways to solve Recurrences —

(1) Recursion Tree method

(2) ~~substitution~~ Substitution method

(3) Master Theorem.

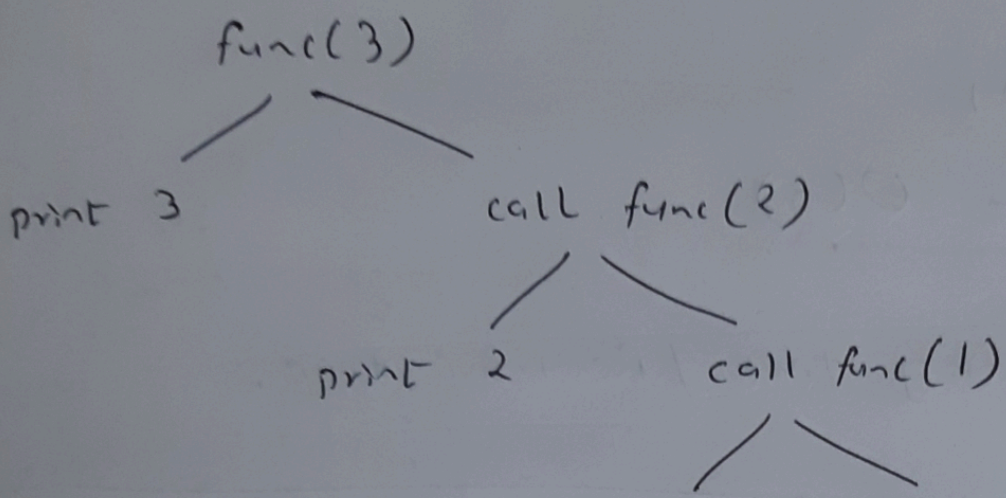
---

1) Consider an example —

```
void func(int n)
{
   if (n > 0)
   {
     printf("%d", n);
     ~~fntn~~
     func(n-1);
   }
}
```

- Here the function "func." is calling itself for (n-1) times. If we trace the function for some sample value say n=3, the recursion tree will be —

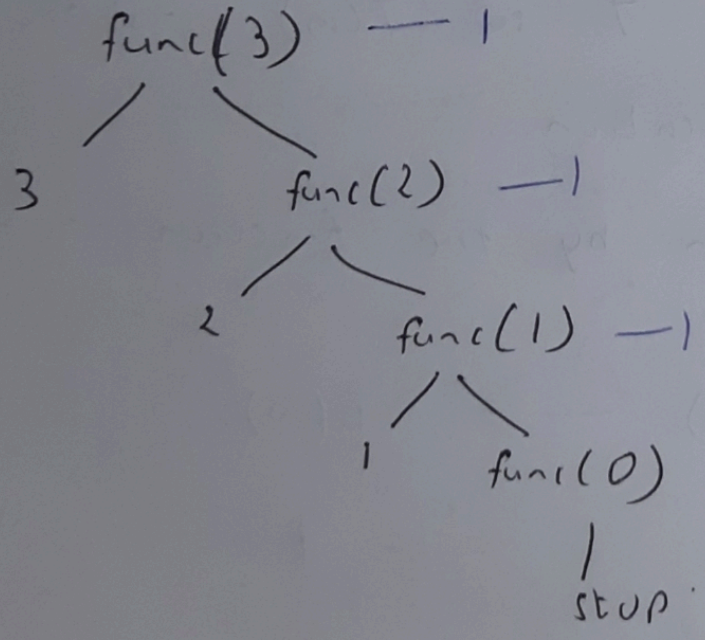
func(3)



→ Here cond. becomes false. Then it will stop.

⇓

True



- The function is printing 3 times and calling itself for 4 times. At every step it is printing and calling. which is constant say 1.

- For n as input, it will make n+1 calls. and will ~~print~~ execute print =) n times

Time depends on no. of calls.

So time function can be written as

$f(n) = n+1$

$\therefore \quad f(n) = O(n)$

This is a tracing tree / Recursive tree.

- How to prepare a recurrence relation of any function

Note: For recurrence relation we use func. name as T so $T(n)$ instead of $f(n)$ as per convention.

$T(n)$ = time taken by the function.

```
void func(int n)          ——  T(n)
{
  if (n > 0)              ——  1
  {
    printf("%d", n);      ——  1 unit of time.
    func(n-1);            ——  takes T(n-1) time.
  }
}
```

$\therefore T(n) = T(n-1) + 2$ (constant only. ~~e.g.~~ If more than 1, round to 1 only or write const. c)

or

$T(n) = T(n-1) + 1$

Recurrence relation – for above

$$T(n) = \begin{cases} 1 & n = 0 \quad -① \\ T(n-1)+1 & n > 0 \quad -② \end{cases}$$

If $n=0$, $T(n)=0$ but instead of 0 we write constant c or 1. we don't write 0

~~Recurrence Tree –~~

- Solving Recurrence relation by substitution –

① is already solved. we have to solve ②

$T(n) = T(n-1) + 1$ – to solve.

Since $T(n) = T(n-1)+1$

$\therefore T(n-1) = T(n-1-1)+1$

$= T(n-2)+1$

{substitute n by n-1}

Also $T(n-2) = T(n-2-1)+1 = T(n-3)+1$

$\therefore$ $T(n) = T(n-1) + 1$ {Question}

In question substitute $T(n-1)$ to

$T(n-2) + 1$ — ☆

$\therefore$ $T(n) = [T(n-2)+1] + 1$

$T(n) = T(n-2) + 2$

If we substitute again then — ☆

$T(n) = [T(n-3)+1] + 2$

$\therefore$ $T(n) = T(n-3) + 3$

⋮

⋮ continue for k times

$\therefore$ $T(n) = T(n-k) + k$

We know when $n = 0$, $T(n) = 1$ (given)

Thus if we go on substituting further at some point since we are subtracting it will become 0. ~~Reo~~

Assume we reach $n = 0$.

$\therefore$ Assume $n-k=0$

$\therefore$ $n=k$

$\therefore$ $T(n) = T(n-n) + n$ (substitute $n=k$)

$T(n) = T(0) + n$

$T(n) = 1 + n$ which is $O(n)$.

(order of linear class n)

- So for given func → prepared Recurrence Tree Recurrence Relation.

and by solving through both tree and relation method we got the answer $O(n)$. Total time $T(n) = n+1$