A SMOOTH SEA NEVER MADE A SKILLED SAILOR

Topics:

Recurtion Introduction

Function call Tracing

Factorial of a number

Increasing

Decreasing

Print Decreasing & Increasing

Fab onachi no.

Recursion => function calling itself

=> solve a problem by using a subproblem.

smaller instance of same problem.

why it is need?

1. It is pre-requesties for
    1. Back Tracking
    2. DP
    3. Trees
    4. Graphs

2. Sorting algorithms!

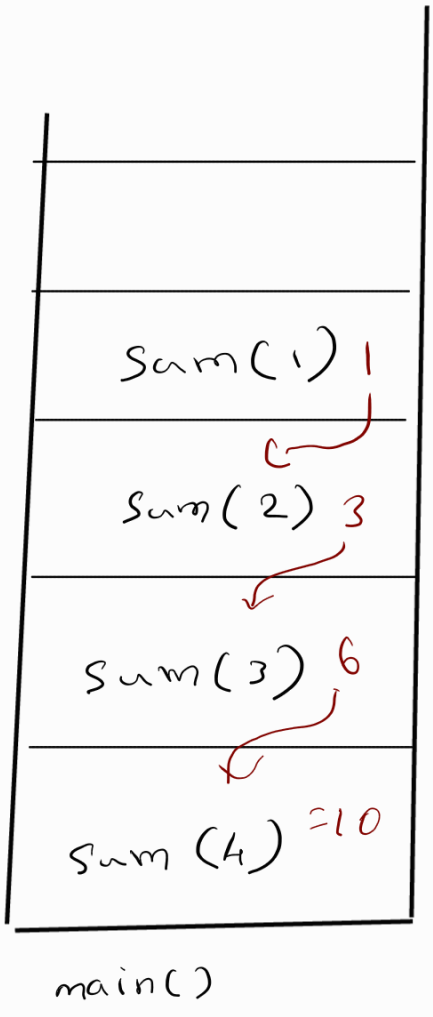
mergesort & quick sort.

Example: Sum of first 5 natural numbers

$$Sum(5) = \underbrace{1+2+3+4}_{Sum(4)} + 5 \Rightarrow 15 \Rightarrow \frac{N(N+1)}{2}$$

$$\underbrace{1+2+3}_{sum(3)}+4$$

```
def sum(N):
    if N == 1: return 1
    return sum(N-1) + N
```



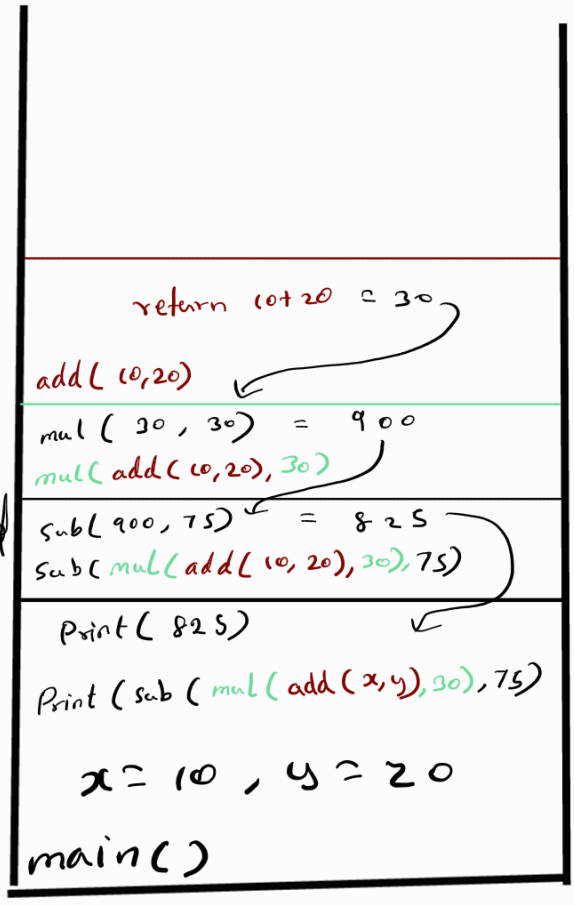
Steps for recursive code:

1. Expectation from the function
2. writing main logic:

   Breaking problem to Sub problem
3. Base Case : How the recursion has to stop

Function call Tracing

```
main (){
    x=10, y=20;
    Print (sub(mul(add(x,y),30),75)
```



1. First In Last Out
The top most call gets excicuted

2. The child function will always return its answer to its parent function and parent function will excute only when it has received the answer from its child.

3. once function call returns answer it is going to be removed from the call stack.

Quiz: Factorial of 5 ? = 120

Factorial (N!)

Factorial is the product of all the possitive numbers less than or equal to a number.

ex: $3! = 3 \times 2 \times 1 = 6$

$5! = 5 \times 4 \times 3 \times 2 \times 1 = 120$

$1! = 1$

$0! = 1$

$7! = 7 \times 6 \times 5 \times 4 \times 3 \times 2 \times 1 =$

$(a, b, c) = 3! = 6$

abc bca cab
acb bac cba
} 6 different way it can be wirten

$(a, b) = 2! = 2 \times 1 = 2$

ab
bc
} 2 different pairs can be formed

Time Complexity:

O (Number of function calls * Time per function call)

Space Complexity:

O (maximum depth of recursive stack space
↓↓
(Num. function calls) (+ space per function calls)
*

Problem 1 : Factorial of N

Given int N, find the factorial of N

N = 5

fact(5) = 5 x 4 x 3 x 2 x 1

fact(4)

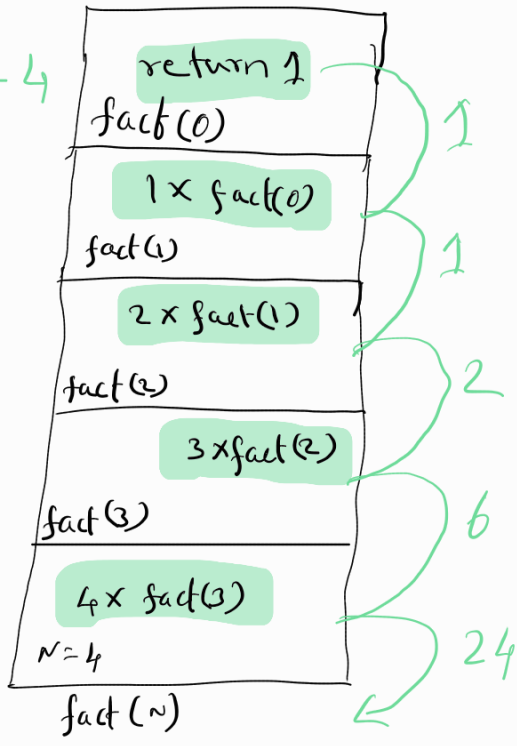
fact(N) = N x fact(N-1)

```
def fact(N):
    if (N==0) : return 1   # Base case
    return (N x fact(N-1))
```

Ex: N = 4

4
3
2
1 =>

output = 24



Time Complexity:

```
def fact(N):
    if (N==0): return 1   # Base case
    return (N x fact(N-1))
```

$T(N)$ = Time taken by recursive function to calculate fact(N)

$T(N-1)$ = Time taken to calculate fat(N-1)

$T(N) = T(N-1) + 1$ ⇒ Recurrence Relation

$(N-1)$ $\quad$ $(N-1)-1$ $\quad$ $N-2$

$T(N-1) = T(N-2) + 1$

$T(N) = T(N-2) + 1 + 1$

$T(N) \Rightarrow T(N-2) + 2 \rightarrow (2)$

$T(N-2) = T(N-3) + 1$

$T(N) = T(N-3) + 1 + 2$

$T(N) = T(N-3) + 3 \rightarrow (3)$

After (k) substitution:

$T(N) = T(N-K) + K$

$= T(N-N) + N$

$= T(N-N) + N$

$T(N) = N$

$T(0) = 1$

$\Rightarrow T(N-K) = T(0)$

$\Rightarrow N-K = 0$

$\Rightarrow K = N$

SC:

N → function calls

1 → space per function

$\Rightarrow N * 1$

$\Rightarrow O(N)$

Problem 2: Print 1 to N

⇒ Given N, Print all the numbers from 1 to N in increasing order.

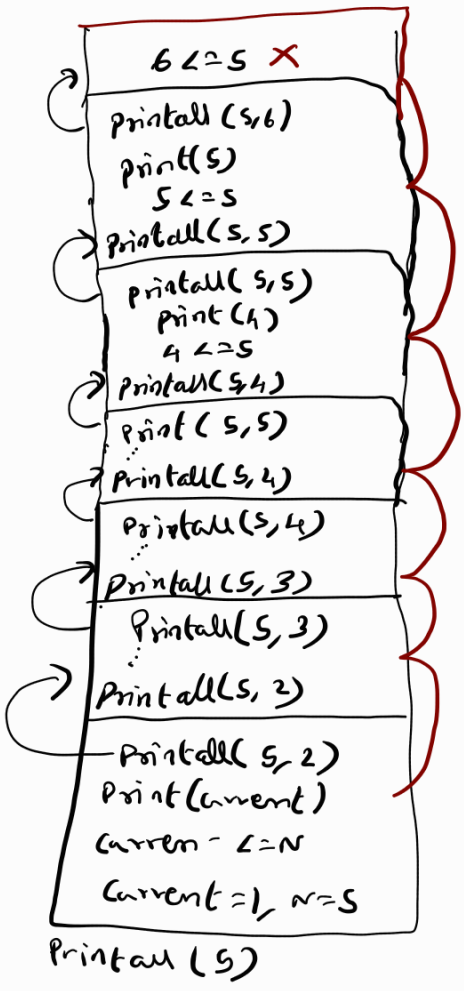
```
def printall(N, current=1)
    if current <= N: #Base case
        Print(current)
        printall(N, (current+1))
```

```
def inc(N):
    if N == 0: #Base case
        Print(N)
    else:
        inc(N-1)
        print(N)
```

TC: O(N)

SC: O(1)

N = 5



inc(5)	inc(4)
1	1
2	2
3	3
4	4
5	

logic should be

inc(4)

Print(5) Print 5 manually

```
def inc(N):
    if N == 0:
        Print(N)
    else:
        inc(N-1)
        Print(N)
```

1 5

```
def inc(start, end)
    # increase the start
    if start ≤ end:
        Print(start)
        inc((start+1), end)
    else: return
```

1 _printed

2 ↗

3 ↗

4 ↗

5 ↗

6 ≤ end X

Problem: Whirlpool's Countdown Timer

=> given int A. Print out each minutes until zero

=> decrement one by one until zero.

A = 5

```
def decrement(N):
    if N<0: # Base Case
        return
    print(N)
    decrement(N-1)
```

5
4
3
2
1
0

if N<0 return.
print(N)
decrement(N-1)

Decrease time 10:00 one by one.

10 : 60
09 : 59
09 : 58
09 : 59
08 : 60

x = 12 y = 60

```
dec(x, y):
    if y == 0:
        y = 60
        x -= 1
    print(x, y)
    dec(x, y-1)
```

Time Complexity:

O (Number of function calls * Time per function call)

Space Complexity:

O (maximum depth of recursive stack space
(Num. function calls) (+ space per function calls)
*

Problem: Fibunachi numbers

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

0	1	2	3	4	5	6	7	8	9	10
0	1	1	2	3	5	8	13	21	34	55 ...

what is Fibunachi of N?

Fib(7) = 0 1 1 2 3 5 8 **13**

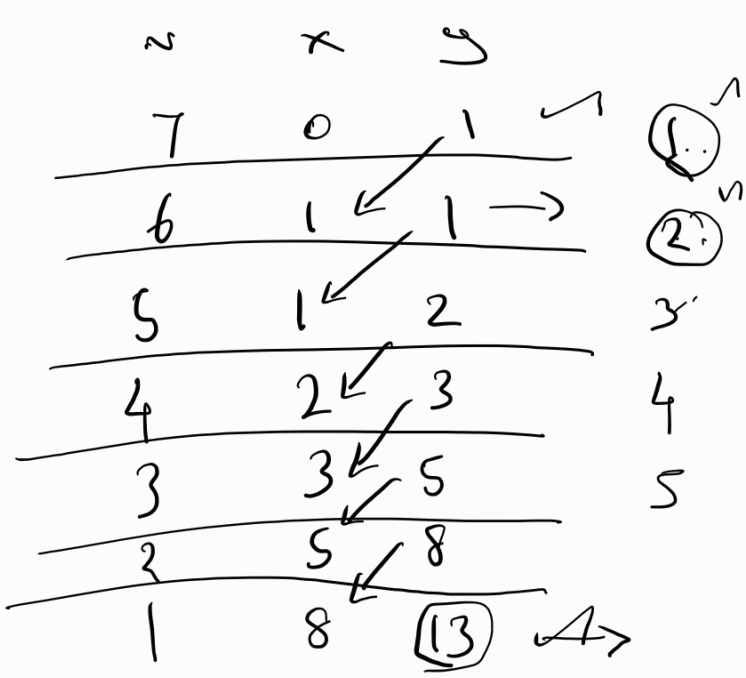
```
ans = [0,1]   (0 1 2 3 4 5)
for i in range(1, 7)
    ans.append(ans[i] + ans[i-1])
return ans
```

```
def fib(N)
    x = 0   y = 1
    for i in range(1, N)
        temp = (x+y)
        x = y
        y = temp
    return y
```

```
def fib(N, x=0, y=1, index=1)
    if index < N:
        temp = x+y
        x = y
        y = temp, index+=1
        fib(N, x, y)
    else:
        return y
```

x	y
0	1
1	1
1	2
2	3
3	5
5	8
8	13 ✓

```
def fib(N, x=0, y=1)
    if N==1: return y
    fib(N-1, y, (x+y))
```



Fib number = sum of previous two fib numbers

$fib(5) = fib(4) + fib(3)$

$fib(N) = fib(N-1) + fib(N-2)$

Expectation: calculate & return fibunaci of N

~~N=0 fib(0-1) + fib(0-2)~~

~~N=1 fib(1-1) + fib(1-2)~~

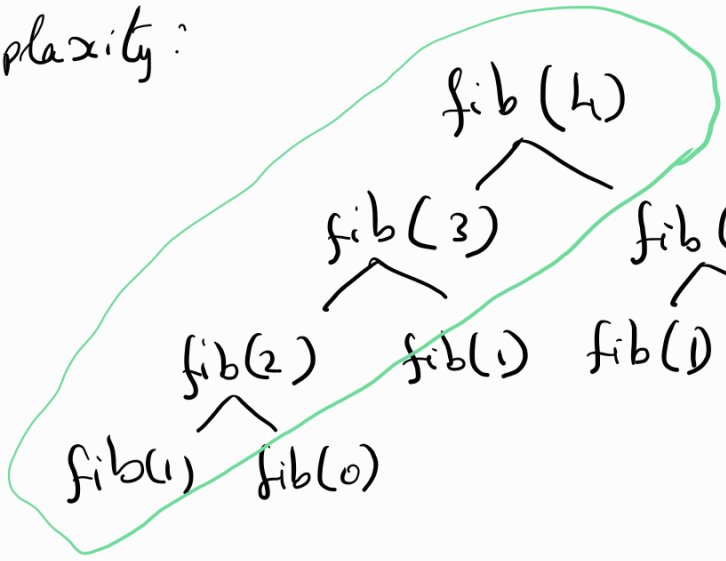
N=2 fib(2-1) + fib(2-2)

Code:

```
fib(N):
    if N==1: return 1
    if N==2: return 1
    if N==0: return 0
    return fib(N-1) + fib(N-2)
```

if $N \leq 1$: return N

Time Complexity:



only 4 calls in space

$2^0 = 1$

$2^1 = 2$

$2^2 = 4$

$2^3 = 8$

2^N

Total number of calls $= 2^0 + 2^1 + 2^2 \ldots + 2^N = 1 + N$

$=$

what is the sum of this series (GP) $= \dfrac{a(r^n - 1)}{r - 1}$

ratio $= 2$

$n = (1 + N)$

$\Rightarrow \dfrac{a(2^{(1+N)} - 1)}{2 - 1} \Rightarrow a(2^{(1+N)} - 1)$

$\Rightarrow 1 * (2^{(N+1)} - 1)$

$= (2^{N+1} - 1)$

$TC \approx 2^N$

Space Complexity

Number of function calls = 4

SC: O(N)