Lecture : Recursion - 1

- Trees
- DP
- Graph
- Backtracking
- Trie.

Agenda

- Introduction of recursion
- factorial of a number.
- fibonacci of a number.
- Power of a number.
- Time complexity of recursive function.

Recursion

- function calling itself
- solving a problem, using smaller instances of same problem.

Example:

Qu Calculate sum of first n natural numbers.

$$sum(n) = \underbrace{1+2+3+4+5 \text{ - - - - - - - - } n-1}_{sum(n-1)} + n$$

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

$$\downarrow$$

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

$$\downarrow$$

$$sum(n) = sum(n-3) + (n-2) + (n-1) + n$$

* How to write recursive codes?

1. Assumption :- what your function should do?
2. Main logic :- Solving problem using smaller instances
3. Base case :- smaller problem you can solve.

Qu: Given a number n, find 1 + 2 + 3 + 4 + 5 - - - - - - - - - n. using recursion.

```
int sum( int n) {
    if ( n == 1) {
        return 1;
    }
    int sa = sum(n-1);
    return sa + n;
}
```

1. Assumption

Given a number n. find & return sum of first n natural no.

2. Main logic

sum(n)

↓

sa = sum(n-1)

return $\boxed{sa + n}$

3. Base case

Stopping pt of recursion.

$n = 1 \longrightarrow sum = 1$

```
if(n==1) {
    return 1;
}
```

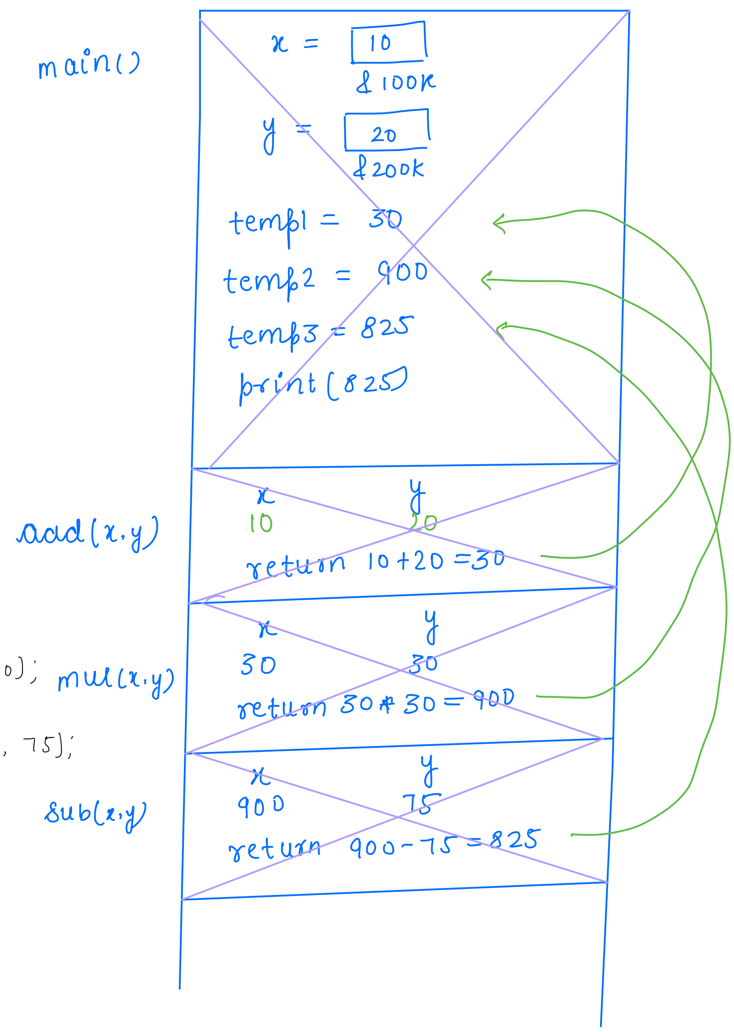
function call tracing

```
int add(x, y) {
    return x + y;
}

int sub(x, y) {
    return x - y;
}

int prod(x, y) {
    return x * y;
}

main() {
    int x = 10;
    int y = 20;
    int temp1 = add(x, y)
    int temp2 = mul(temp1, 30);
    int temp3 = sub(temp2, 75);
    print(temp3);
}
```

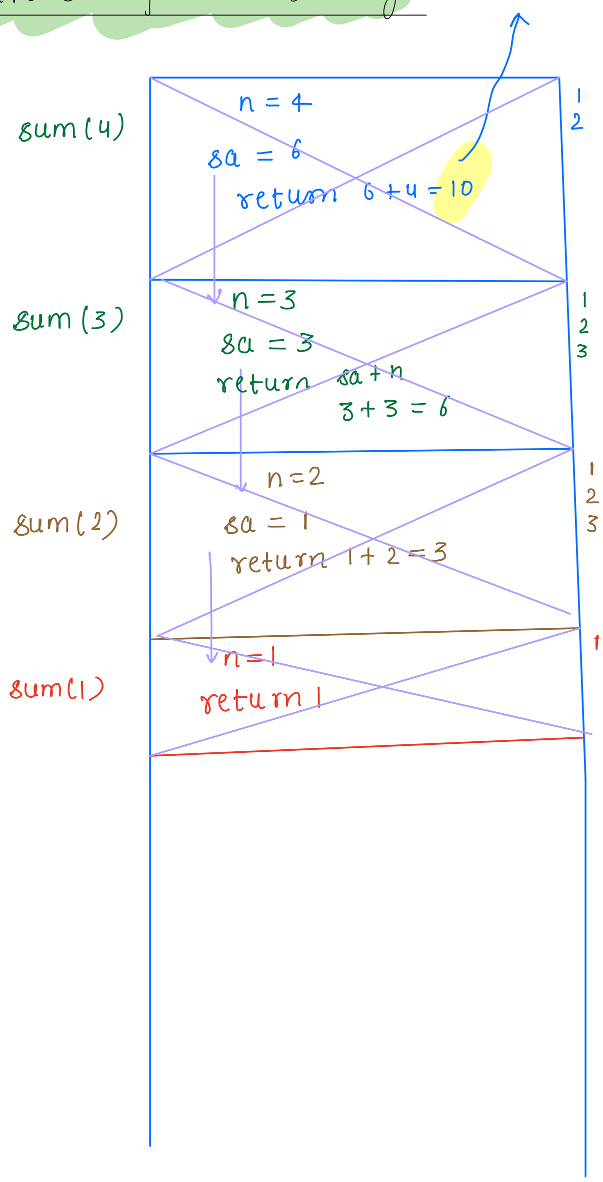


Recursive call function tracing

```
int sum(int n) {
1.    if(n==1) {
          return 1;
      }
2.    int sa = sum(n-1);
3.    return sa + n;
}
```

TC: $\overset{x}{n} * \overset{y}{1} \simeq O(n)$

SC: $n * 1 \simeq O(n)$



Qu: Given a number (+ve) n, find factorial of n. using recursion

1! ⇒ 1

2! ⇒ 2

3! ⇒ 6

5! ⇒ 5 * 4 * 3 * 2 * 1 = 120 ⇒ 5 * fact(4)

0! ⇒ 1

Approach

```
int fact(int n){
    if(n==0) {
        return 1;
    }
    int sa = fact(n-1);
    return sa*n;
}
```

1. Assumption

Given a no. n, find & return factorial of n.

fact(5) = 120

fact(2) = 2

2. Main logic

fact (n)

↓ return sa*n;

sa = fact(n-1)

3. Base case

n==0, return = 1.

fact(1)

↓ ↑ return sa*n

1*1 = 1

sa = fact(0)

Recursive call tracing of factorial

```
int fact(int n){
  1  if(n==0) {
        return 1;
     }
  2  int sa = fact(n-1);
  3  return sa *n;
}
```

Step1: x = How many times you are generating this function?

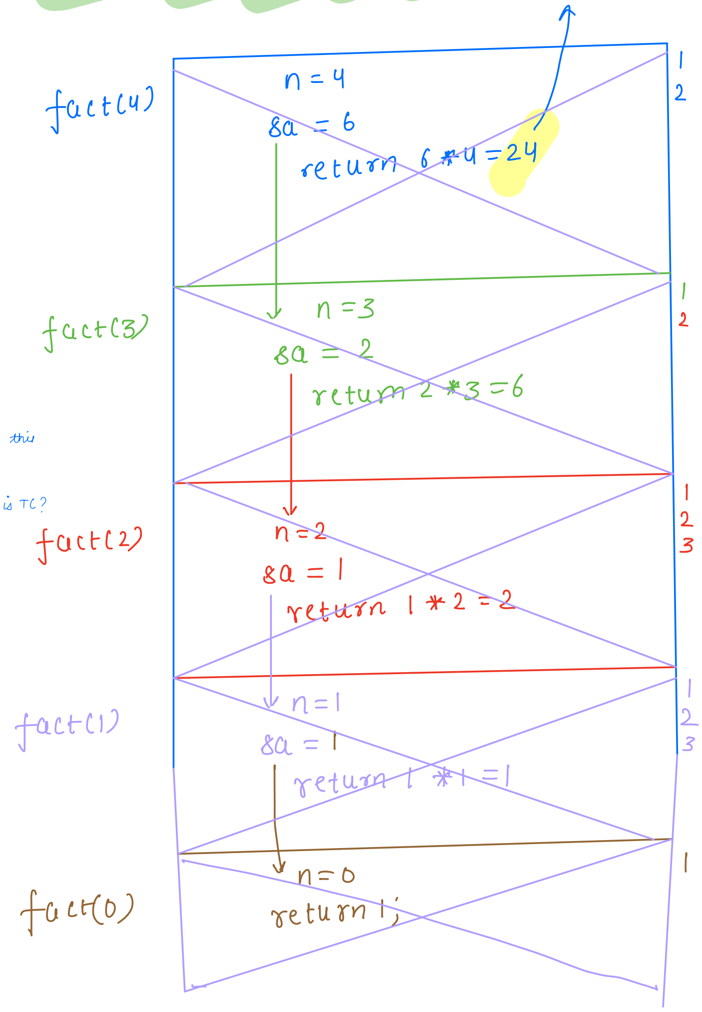
Step2: y = Inside that fun in stack space. what is TC?

$x = n+1 \simeq O(n)$

$y = O(1)$

TC: $O(n)$

SC: $n * O(1) \simeq O(n)$



Qu find nth fibonacci number using recursion

0th	1st	2nd	3rd	4th	5th	6th	7th			
0	1	1	2	3	5	8	13	21	34	....

```
int fib( int n)  {
    if(n==0 || n==1) {
        return n;
    }
    int sa1 = fib(n-1);
    int sa2 = fib(n-2);
    return sa1 + sa2;
}
```

1. Assumption

Given a no n. find & return nth fibonacci no

fib(2) =1

fib(6) = 8

2. Main logic

fib(n)

sa1=fib(n-1) sa2=fib(n-2)

return sa1+sa2;

3. Base case

n=0 → return 0

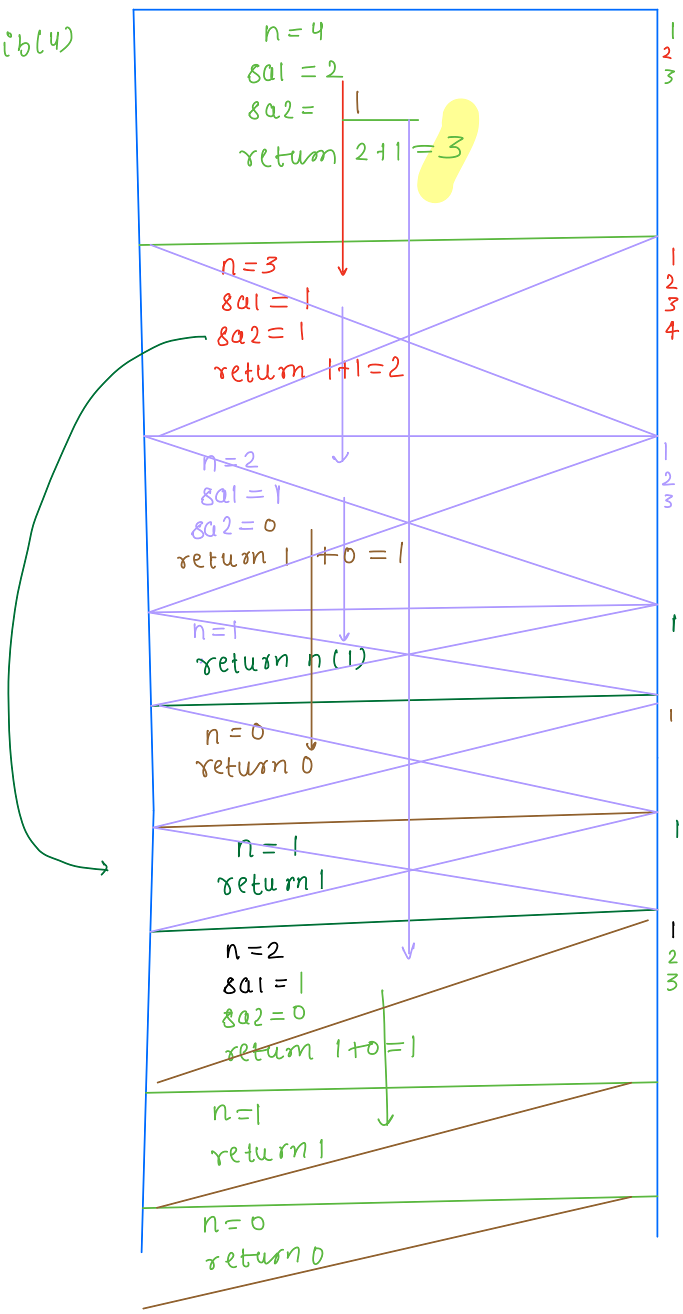
n==1 → return 1.

```
if(n==0|| n==1) {
    return n;
}
```

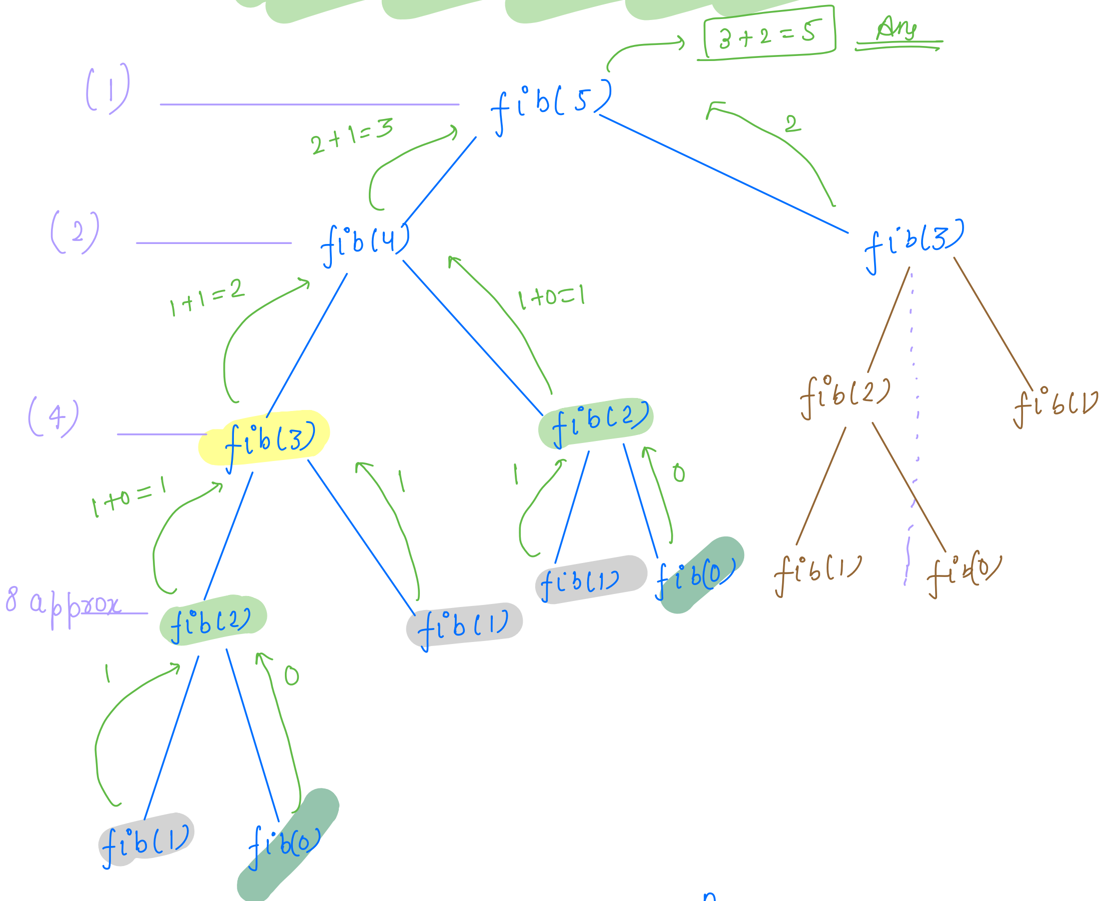
function call tracing of fibonacci number

```
int fib(int n)  {
  1 if(n==0 || n==1) {
        return n;
    }
  2 int sa1 = fib(n-1);
  3 int sa2 = fib(n-2);
  4 return sa1 + sa2;
}
```

fib(4)



Recursion tree of fibonacci number



$x = 1 + 2 + 4 + 8 + 16 - \cdots\cdots$ n times 2^n

$x = \frac{1 * (2^n - 1)}{1} \simeq O(2^n)$

$y = O(1)$

TC: $x * y = O(2^n) * O(1) = O(2^n)$ Ans

Step1: x = How many times you are generating this function?

Step2: y = Inside that fun in stack space. what is TC?

Qu: Given 2 integers a and n, find a^n using recursion.

Input: $a = 2$ $\quad 2 \,^\wedge\, 3 = 8$

$n = 3$

Brute force:

```
int pow(a, n) {
    if (n==0) {
        return 1;
    }
    sa = pow(a, n-1);
    return sa * a;
}
```

1) Assumption

Given a, n, find & return $a \,^\wedge\, n$.

2) Main logic

$a=2,\ n=4$

2^4

$\downarrow \quad \uparrow * 2$

2^3

$2^4 = 2^3 * 2$

pow(a, n)

$\downarrow$

sa = pow(a, n-1)

return sa * a;

3) Base case

n == 1, return a.

n == 0, return 1.

function call tracing

```
int pow(a, n) {
 1  if(n==0) {
        return 1;
    }
 2  sa = pow(a,n-1);
 3  return sa * a;
}
```

Step1: x = How many times you are generating this function?

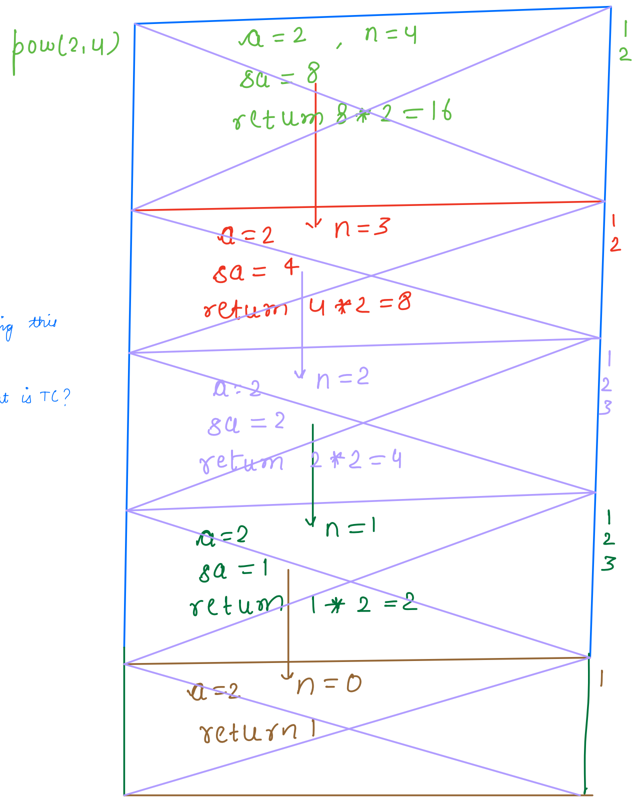
Step2: y = Inside that fun in stack space. what is TC?

pow(a,n)

$x = n+1 \simeq n.$

$y = O(1)$

TC: $O(n) * O(1) = O(n)$



a = 2, n = 5 , stack height = 6

2 13 " = 14

5 2 " = 3

Optimised approach :

$2^{12} = 2^{11} * 2.$

$pow(a,n) = pow(a, n-1) * a.$

$2^{13} = 2^{12} * 2.$

$2^{12} = 2^{6} * 2^{6}$

$pow(a,n) = pow(a, \frac{n}{2}) * pow(a, \frac{n}{2})$

$2^{13} = 2^{6} * 2^{6} * 2$

$a^{n} = a^{n/2} * a^{n/2} * a.$

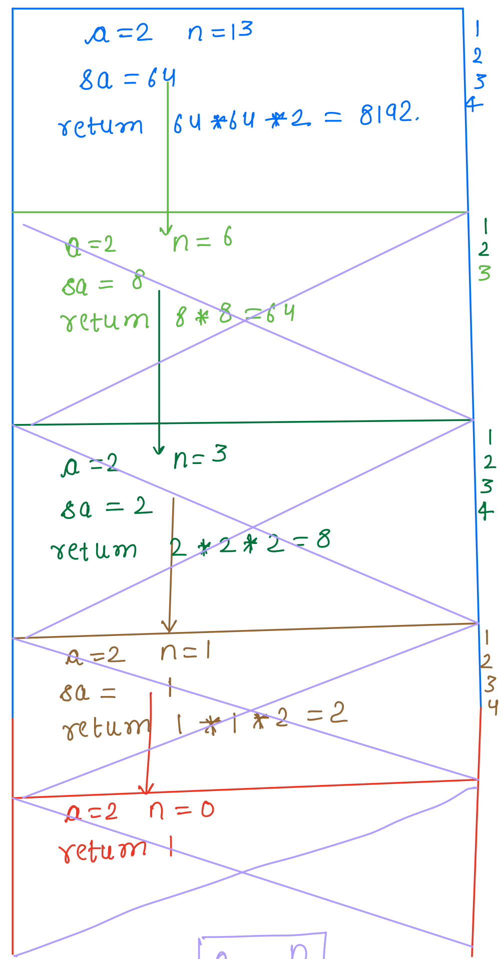
if n is even — $a^{n} = a^{n/2} * a^{n/2}$
" " " odd — $a^{n} = a^{n/2} * a^{n/2} * a$

```
int pow( a, n) {
    if(n==0) {
        return 1;
    }
    sa = pow( a, n/2 );
    if(n % 2==0) {
        return sa * sa;
    }
    return sa * sa * a;
}
```

function call tracing

```
int pow( a, n) {
1   if(n==0) {
        return 1;
    }
2   sa = pow( a, n/2 );
3   if(n % 2 == 0) {
        return sa * sa;
    }
4   return sa * sa * a;
}
```

pow(2, 13)



Step1: x = How many times you are generating this function?

Step2: y = Inside that fun in stack space. what is TC?

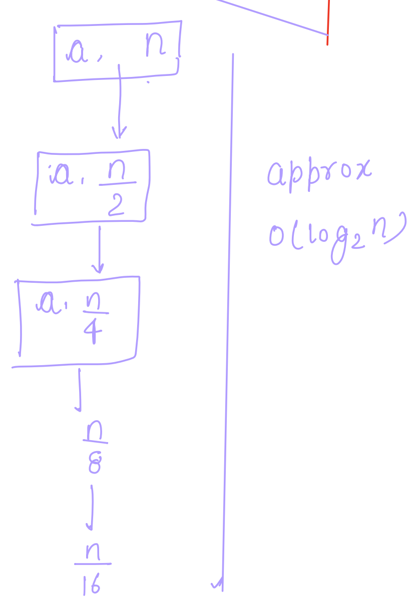
pow(a, n)

$x = \log_2 n.$

$y = O(1)$

TC: $O(\log_2 n) * O(1) = O(\log_2 n)$

a=2, n=13 stack height = 5



Time complexity of recursive code

Recursive codes: multiple instances of same function.

Step1: x = How many times you are generating this function?

Step2 y = Inside that fun in stack space. what is TC?

$\boxed{\text{Total T.C. of recursive code} \div x * y}$

Space complexity of Recursion

Recursive codes: multiple instances of same function.

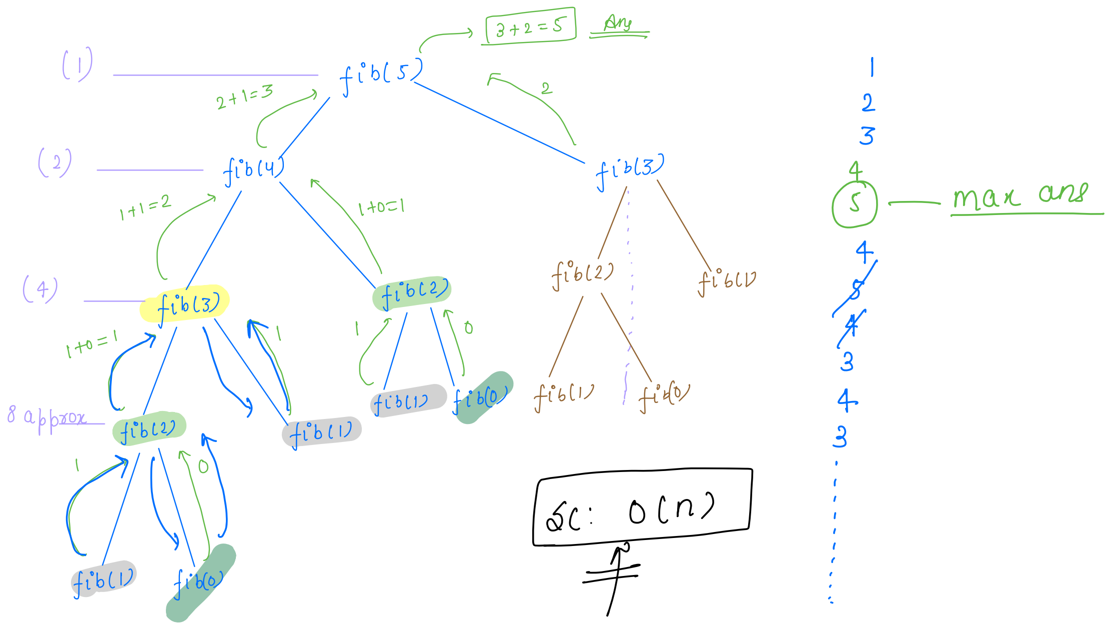
Step 1: x = stack height

Step 2 y = Inside that fun in stack space. what is SC?

$$\boxed{SC: \; x * y}$$

H/W: SC: fibonacci
power ?

Thank you



fib(5)
3+2=5
Ans
(1)
(2)
(4)
2+1=3
2
fib(4)
fib(3)
1+1=2
1+0=1
fib(3)
fib(2)
fib(2)
fib(1)
1+0=1
8 approx
fib(2)
fib(1)
fib(1)
fib(0)
fib(1)
fib(0)
fib(1)
fib(0)
SC: O(n)
1
2
3
4
5
max ans
4
3
4
3