algorithms / programs

- iterative
  - for
  - while
  - do while
- recursive → A function calling itself

What is recursion?

Later

Both iterative and recursive are equally powerful.

i.e For every iterative program → there will be recursive program

Similarly For every recursive program → there is iterative program

```
A(n)
{
   A(n/2)
   A(n/2)
}
```

iterative:

```
for( int i=n^2, i>1, i=i/2)
{
    for(j=1; j<n^3; j++)
    {  num +=1;
    }
}
```

$O(n^3)$

$O(\log n)$ = ?

Yesterdays.

$O(n^3 \log n)$

```
for (i = n/2 ; i <= n ; i++) {          → O(n)
    for (j = 1 ; j <= n/2 ; j = j+1) {  ✓ O(n)
        for (k = 1; k <= n; k = k*2) { } → log n → Yesterday
            Print (RBR)
    }
}
```

$O(n^2 \log n)$

→ int (i=1), (S=1);

while (S <= n) {

✓ i++;

(S = S + i);

Print ("GBR");

}

itr	value of i	value of S
1	2	1+2
2	3	1+2+3
3	4	1+2+3+4
⋮	⋮	⋮
	K+1	1+2+3+.....+K+1

Why did loop fail (S <= n passing) K

$S > n$

$\frac{(K+1)(K+2)}{2}$ $\quad k^2$

$1+2+3+\dots K+1 > n$

$\frac{(K+1)(K+2)}{2} > n \Rightarrow k^2 + \dots > n \Rightarrow K = O(\sqrt{n})$

```
int i=1, s=1
{ while (s<=n)
  { i++;
    s = s+i^2
    printf(RBR);
  }
}
```

why loop failed?

iter	i	s
1	2	$1+2^2$
2	3	$1+2^2+3^2$
3	4	$1+2^2+3^2+4^2$
⋮		
K	K	$1+2^2+3^2+\cdots+k^2$

$s > n$

$1+2^2+3^2+\cdots k^2 > n$

$\frac{k(k+1)(2k+1)}{6} > n$ | $k^3 \cdots > n$ | $k = O(n^{1/3})$

$1 = 1 \wedge 2$

doesn't matter

$K \wedge k+1$

doesn't matter

Small changes in values doesn't impact answer in an asymptotic notation

```
p = 0
for (i=1 ; i<=n; i = i*2)
{
    p++;
}
for( j=1 ; j<p ; j = j*2)
{
    pf("BR");
}
```

$p++;$ → $p = O(\log n)$

$\log n$

$\left.\begin{matrix} \\ \\ \end{matrix}\right\} (\log_2 n)$

$\left.\begin{matrix} \\ \\ \end{matrix}\right\} \log_2 \log_2 n$

$TC = O(\log_2 n + \log_2 \log_2 n)$

$= O(\log_2 n)$

```
for(i=1; i<=n; i++)
{
    j=2
    while(j<=n)
    {
        j = j^2
    }
}
```

$\log\log n$

iteration	j
1	2
2	$(2)^2 = 4 = 2^{2^1}$
3	$(4)^2 = 16 = 2^{2^2}$
4	$(16)^2 = 256 = 2^{2^3}$
5	$(256)^2 = 65536 = 2^{2^4}$
⋮	⋮
K	$2^{2^{K-1}}$
fail ← K+1	2^{2^K}

why failed? $j > n$

$2^{2^K} > n$

apply log 2 times $\Rightarrow$ $K = O(\log\log n)$

```
int i = 1
while(i < n) {
    int j = 1
    while (j < n) {
        int k = 1
        while (k < n) {
            sum += 1;
            k = k * 2;
        }
        j = j * 2
    }
    i = i * 2
}
```

$(\log_2 n)^3$

~~$O(\log n)$~~

$O(n \log n)$

$\log n$

$\log n$

$$(\log_2 n) = O(\log_{100} n) = O(\log_{200} n)$$

```
for (i=1; i<=n; i++)
{
    for (j=1; j<=n; j=(j+i))
    {
        a = b+c;
    }
}
```

i	j
1	$1, 2, 3, \ldots n \rightarrow n$ times
2	$1, 3, 5, \ldots n \rightarrow n/2$ times
3	$1, 4, 7, \ldots n \rightarrow n/3$ times
$\vdots$	$\vdots$
n	1 time.

$$\text{Total} = n + \frac{n}{2} + \frac{n}{3} + \ldots\ldots 1 \quad (\log n)$$

$$= n\left(1 + \frac{1}{2} + \frac{1}{3} + \ldots \frac{1}{n}\right)$$

$$= \underline{n \log n}$$

```
i = 2;
while (i < n)
    i = i^2;
```

iteration	i
1	2
2	$2^2 = 4 = 2^{2^1}$
3	$4^2 = 16 = 2^{2^2}$
4	$16^2 = 256 = 2^{2^3}$
⋮	⋮
k	$2^{2^{k-1}}$
k+1	2^{2^k}

$\log_2 \log_2 n$

$2^{2^k} >= n$

$k = O(\log_2 \log_2 n$

```
i = 25
while (i < n)
    i = i^3;
```

situation	i
1	25^{3^0}
2	25^{3^1}
3	25^{3^2}
⋮	⋮
k	$25^{3^{k-1}}$
fail (k+1)	25^{3^k}

$= O(\log \log n)$

$= O(\log_{100} \log_{100} n)$

$= O(\log_2 \log_2 n)$

instead

why failed?

$i >= n$

$25^{3^k} = n$

$3^k = \log_{25} n$

$k = \log_3 \log_{25} n = O(\log_3 \log_{25} n)$

A(n)

{

1 if (n>0)

2 {

3 Print f("%d", n-1)

4 A(n-1) ✓

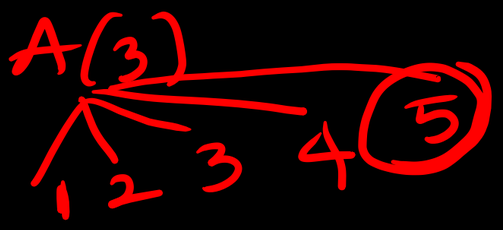
5 }

6 }

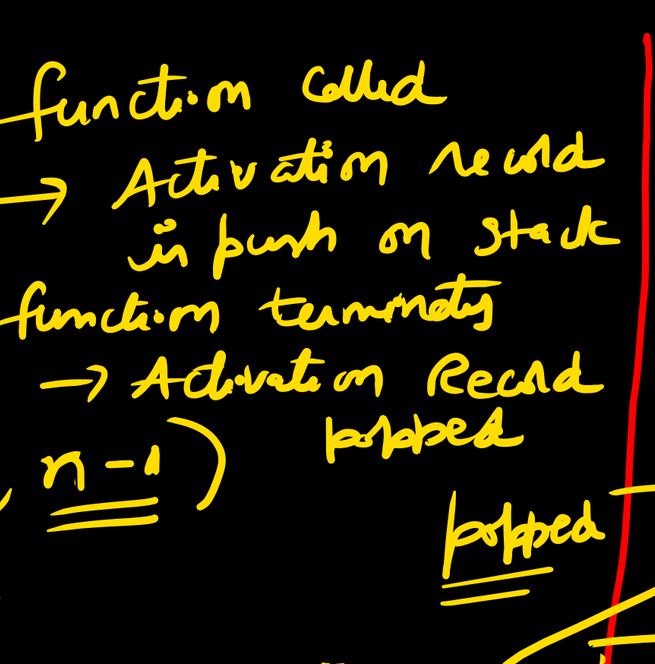
Height of the stack = O(n)

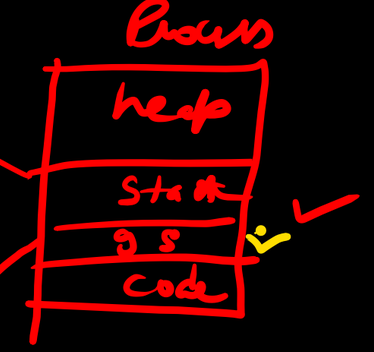
$T(n) = T(n-1) + k$

o/p: 2 1 0



function called
→ Activation record is push on stack
function terminates
→ Activation Record popped



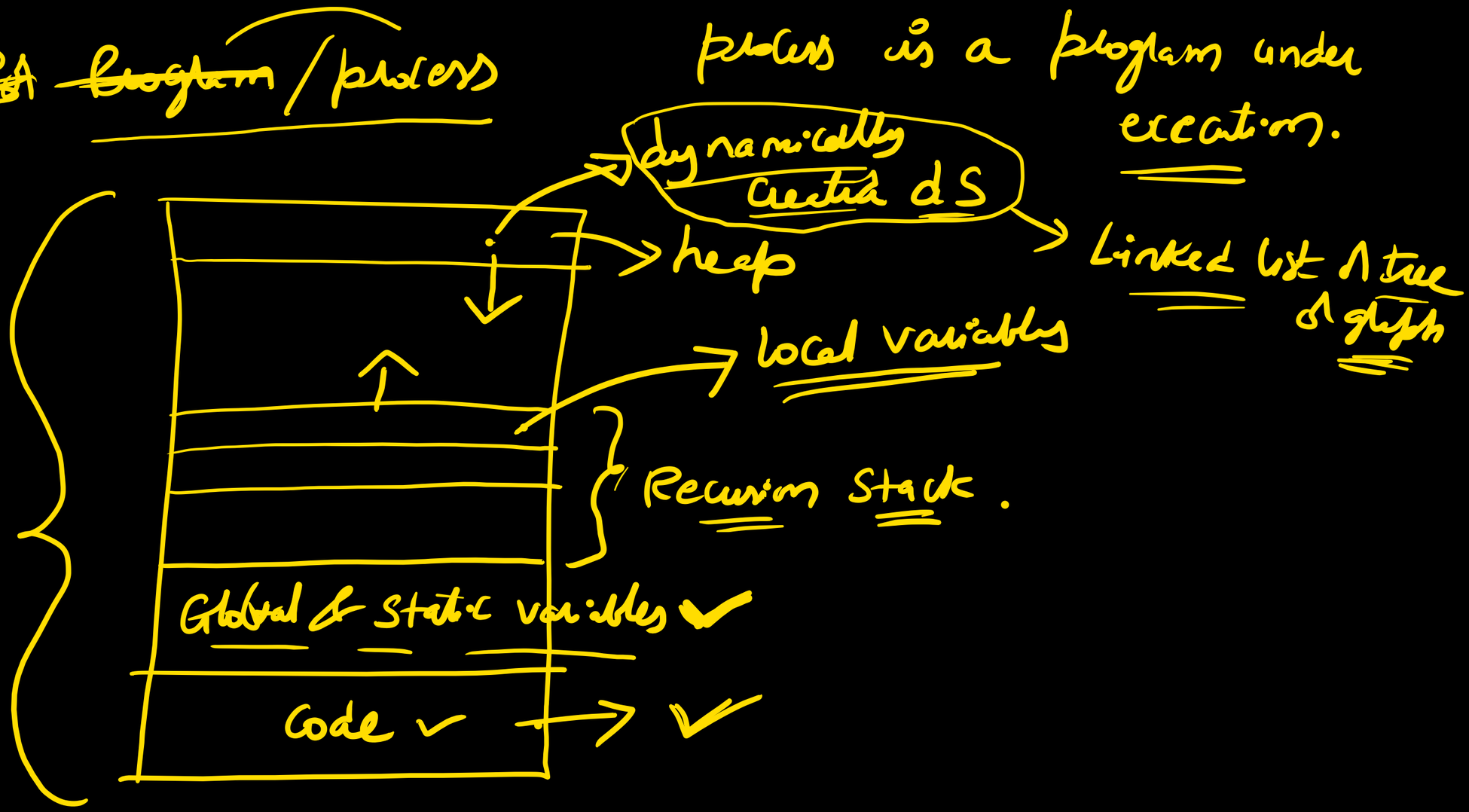


address of the next instruction to be executed

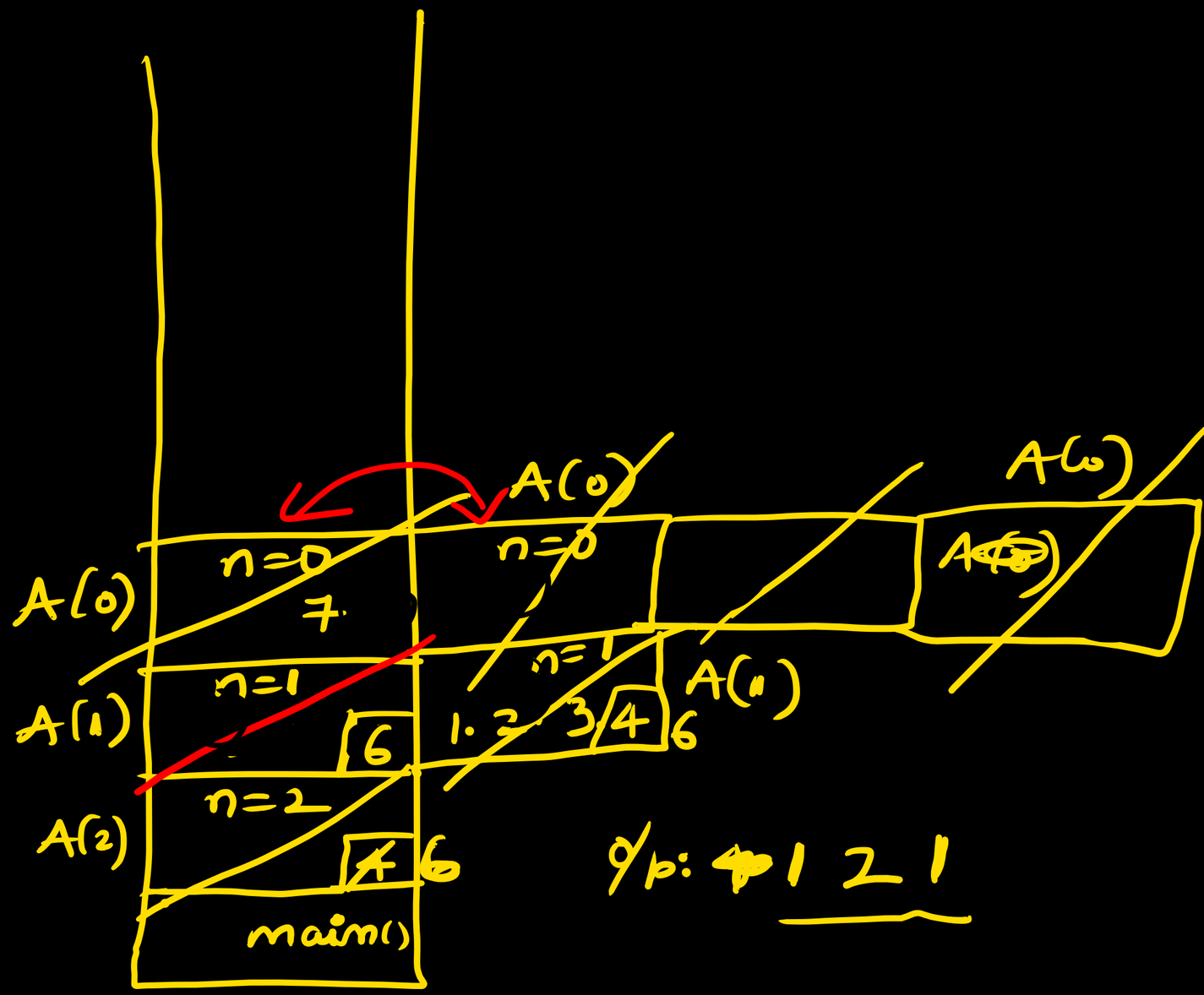
instruction pointer

PA ~~Program~~ / process

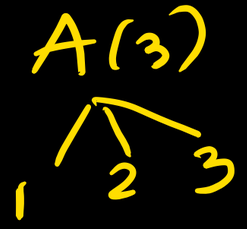
process is a program under execution.



```
A(n)
{
1 if (n>0)
2 {
    3 A(n-1)
    4 Pf(n)
    5 A(n-1)
  }
}
7
```



o/p: 1 2 1



A(n) => T(n)

1 { if (n>0)

2 {

3 A(n-1) T(n-1)

4 Pf(n)

5 A(n-1) T(n-1)

6 }

7 }

$T(n) = 2T(n-1)+C$

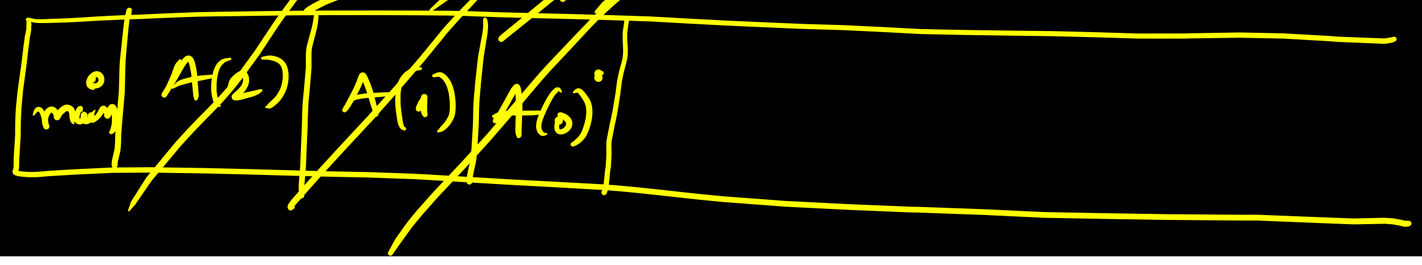
Recurrence: 3 methods

A(2)

A(1) P(2) A(1)

A(0) P(1) A(0) A(0) P(1) A(0)

A(1) A(0) A(0)



Recurrence relations

- substitution
- Tree method
- master theorem.

$A(n) \Rightarrow T(n)$

Let us assume $T(n)$ is the time complexity of $A(n)$

```
{
  if (n > 0)
  {
    A(n-1) ⇒ T(n-1)
    P(n) ⇒ Constant
    A(n-1) ⇒ T(n-1)
  }
}
```

$\therefore$

$$T(n) = 2T(n-1) + C$$

Recurrence relation

Recurrence relation

S T M

```
A(n)
{
  if (n >0)
    {
      Print(n)
      A(n-1)
      Print(n)
      A(n-1)
    }
}
```