Reduction

$X \rightarrow Y$?

algo_X {

algo_Y

}

$X \Rightarrow n C_r$ → $Y \Rightarrow n!$

$\Rightarrow n C_r \cdot (n, r)$ ⇐ algo_Y ✓

return $f(n) / (f r(r) \cdot f r(n-r))$

}

$X \Rightarrow$ given a n, print all the prime nos. $< n$

$Y \Rightarrow$ given a n, check if it is prime

algo_Y

printPrimes(n)

for(i=2; i<n; i++)

isPrime(i) ✓

S

1. We donot know how algo_Y works

2. Correctness of algo_X

sort

Recursion → it is a kind of reduction

Given a problem X, to write a recursive algorithm for X we follow two steps

1. If the given instance of the problem can be solved directly, solve it directly. [Base case]
2. Otherwise, reduce it into 1 or more smaller instances of the same problem. [Recursive case]

X → Y

(x)

X → x

```
algo_X {
    algo_x   // (recursion / assitant)
}
```

(1) $n \rightarrow n!$

$n = 5$, $5!$

$fact(5) = 5 \times \underbrace{4 \times 3 \times 2 \times 1}_{4!} = 120$

$fact(5) = 5 \times fact(4)$

$fact(n) = n \times fact(n-1)$

Recursive relation [Recursive case]

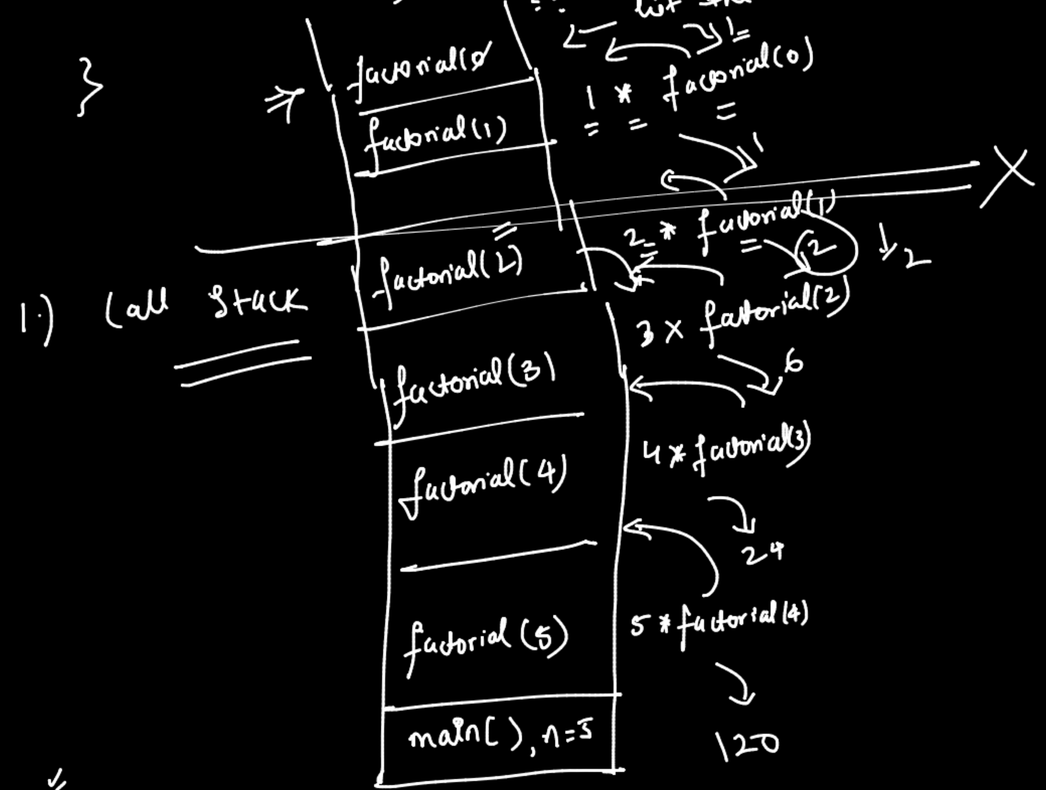
fact(n-1) → recursion / assitant

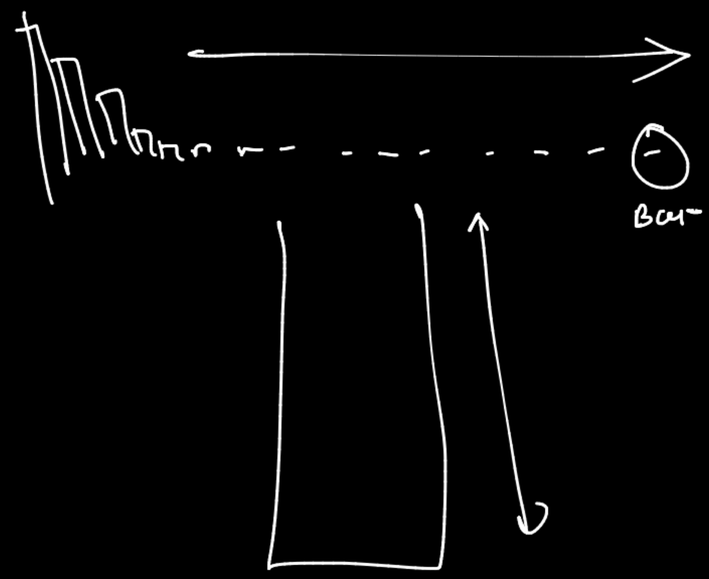
(0)! = 1

```
int factorial(int n) {
    if (n == 0) return 1        // [Base case]
    return n * factorial(n-1)   // [Recursive case]
}
```

done by my assitant / recursion

```
int factorial (int n) {
    if (n==0) return 1; // Base case
    return n * factorial(n-1); // Recursive case
}
```





constraint

$n >= 2$

```
if (n==2)          // Base
    return 2

return n * fact(n-1)
```

Rec ⇒

{ Recursion Tree = top-bottom

main() n=5
factorial(5) ×5 n=120
↓ ↑24
factorial(4) ×4
↓ ↑6
factorial(3) *3
↓ ↑2
factorial(2) *2
↓ ↑1
factorial(1) ×1
↓ ↑1
factorial(0)

bottom to top

fibonacci

0th 1st 2nd 3rd 4th 5th 6th & ...
0 1 1 2 3 5

Given a value n, find me nth fibonacci

n=5

0 1 1 2 3 5

fibo(5) = fibo(3) + fibo(4)

fibo(n) = fibo(n-2) + fibo(n-1)

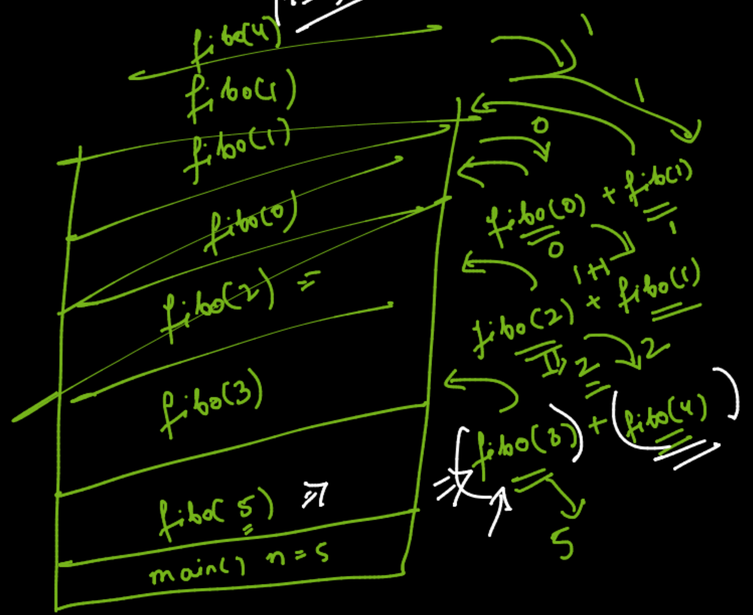
assistant/recursion

(-1)^n × 0th ✓ 1st 2nd

```
int fibo(int n)
{
    if(n==0 || n==1)        ] Base case
        return n;
    return [fibo(n-2) + fibo(n-1)]   [// Recursion]
}
```

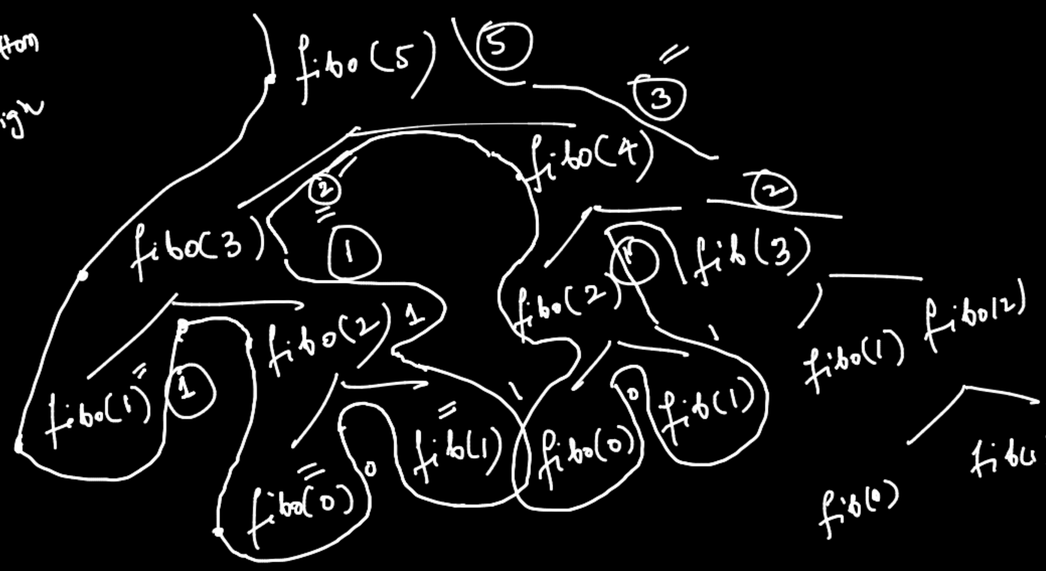
fibo(n) = fibo(n-2) + fibo(n-1)
or
fibo(n-1) + fibo(n-2)

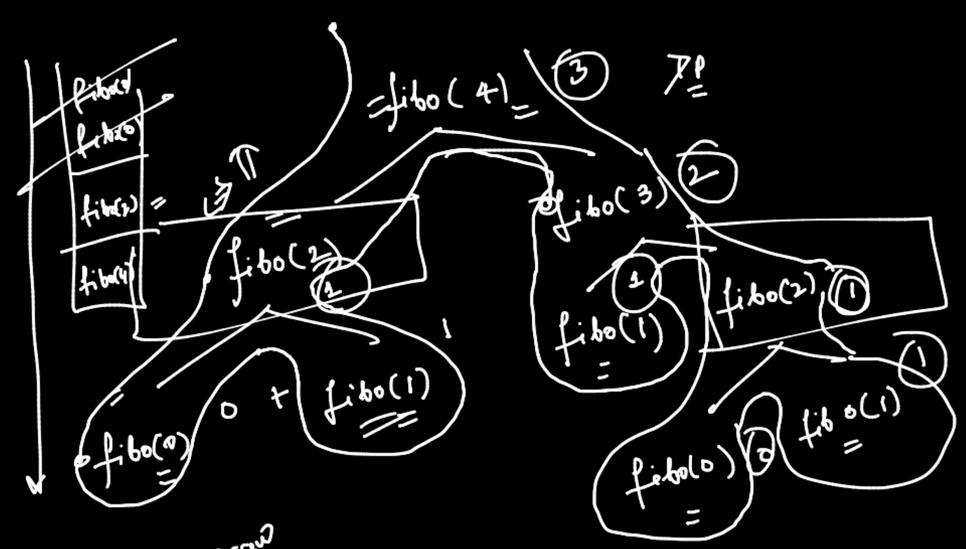
n = 5



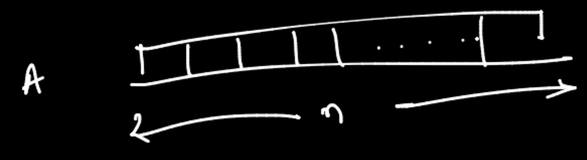
fibo(n) = fibo(n-2) + fibo(n-1)

⇒ top - bottom
⇒ left - right





try to draw the call stack yourself



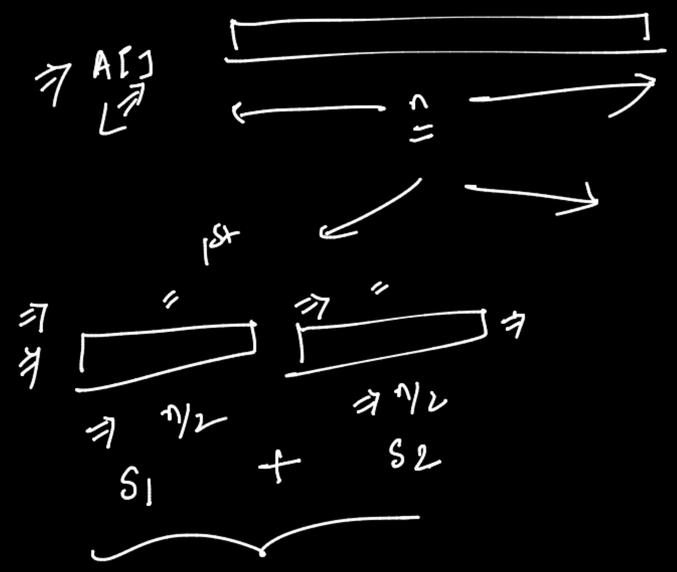
$fact(n) = n * fact(n-1)$

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

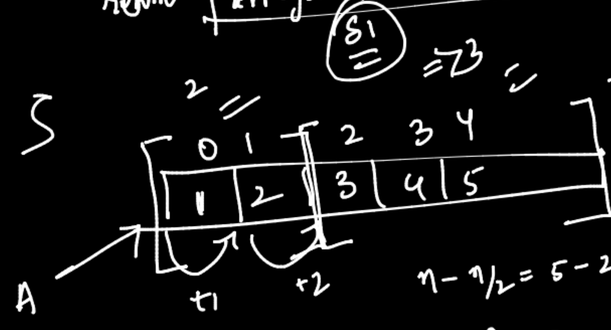
Base Case

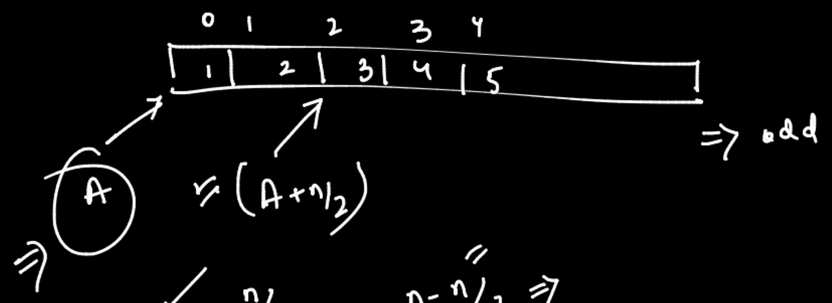
Recursive

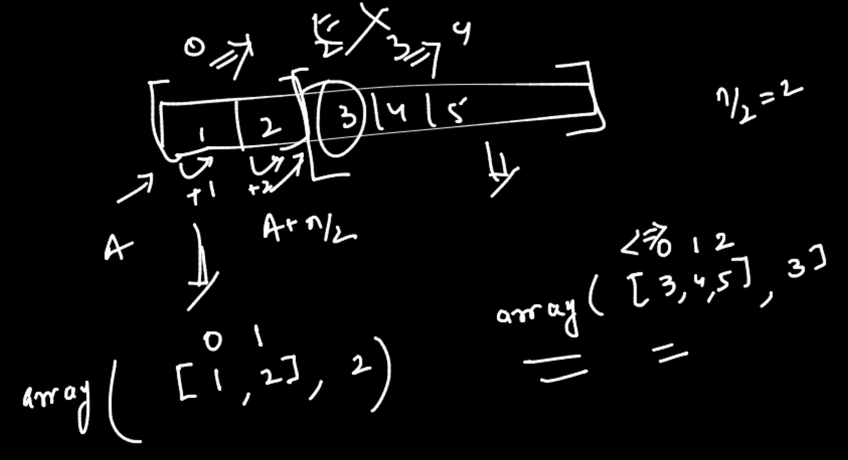
Self-work




```
int arraySum( A , n ) {
return arraySum( A , n/2) + array( A+n/2 , n - n/2)
```

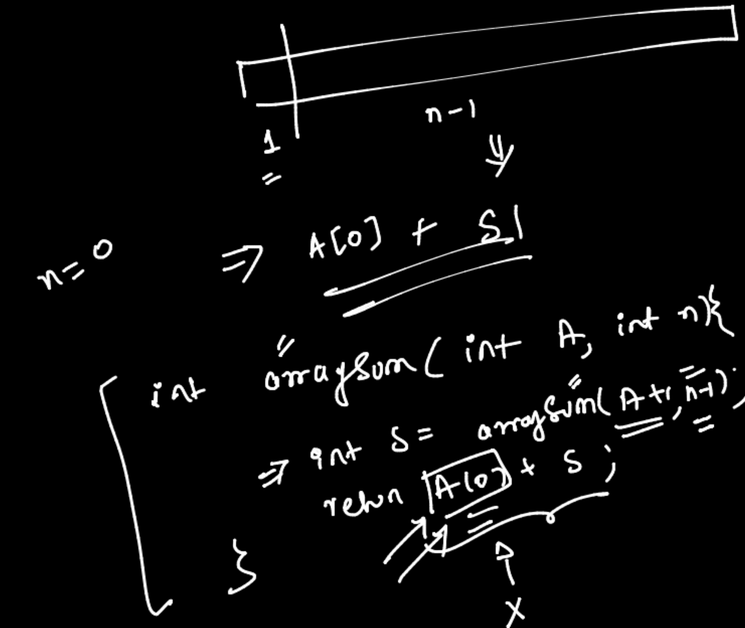




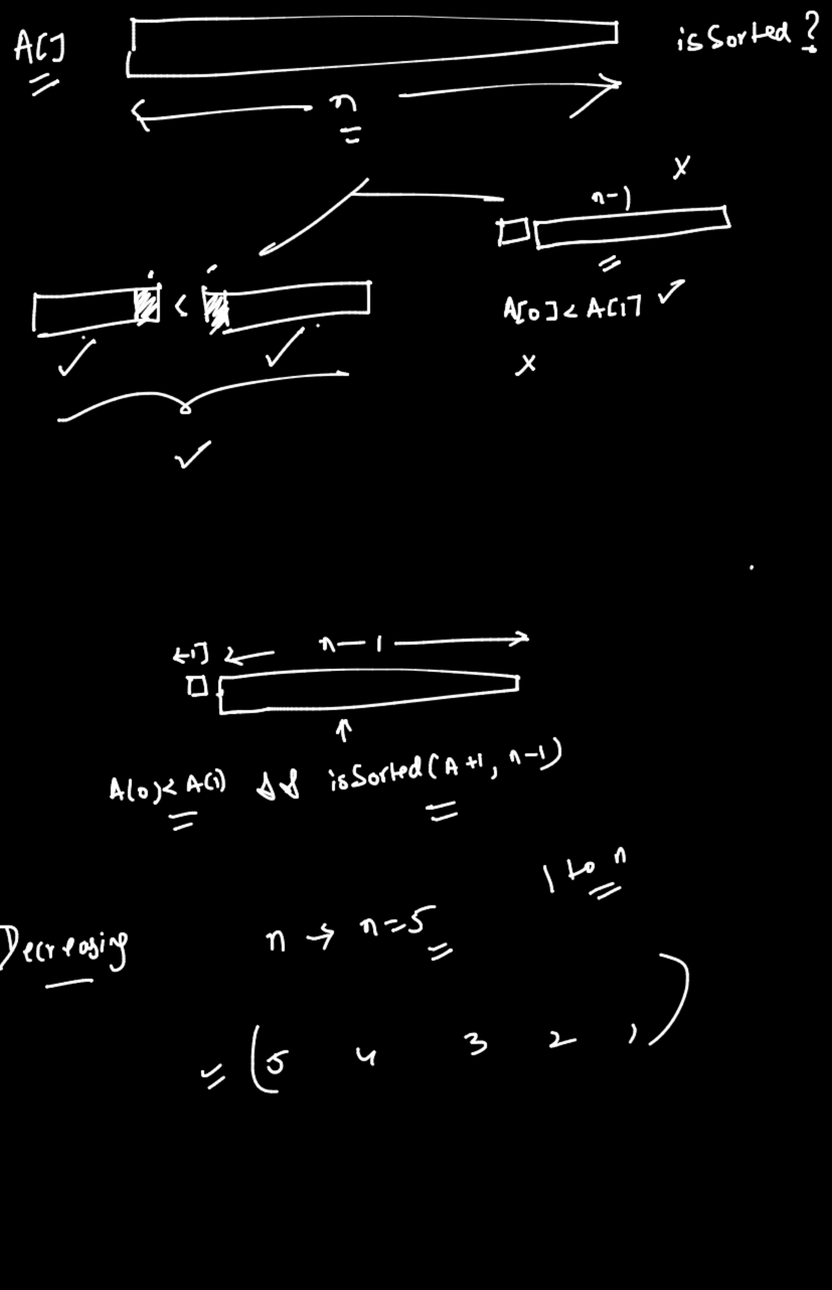


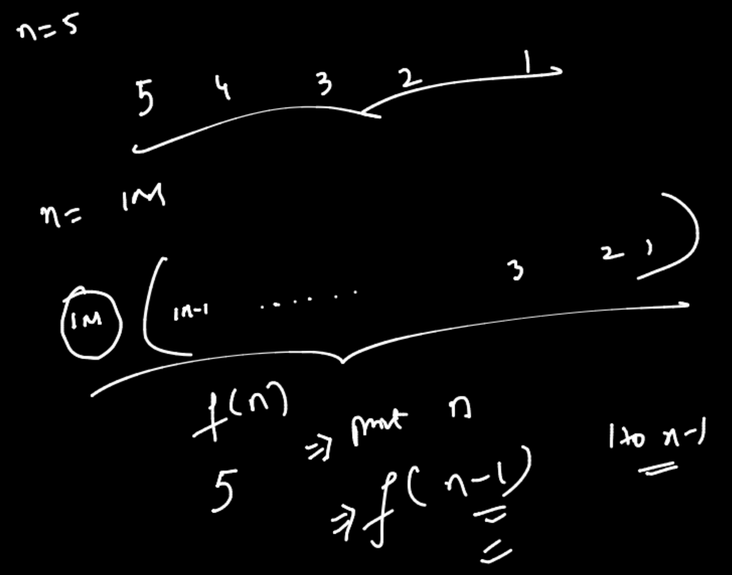

```
int arraySum( int A, int n) {
⇒ if (n==1) return A[0];
⇒ int S1 = arraySum(A, n/2);
⇒ int S2 = arraySum(A+n/2, n-n/2)
  return S1+S2;
}
```

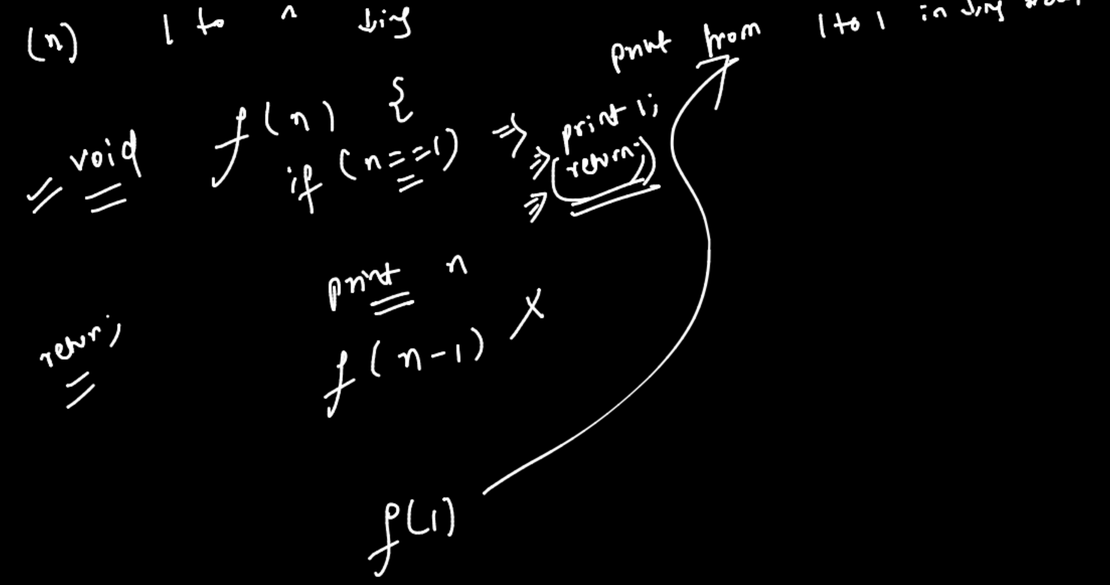
A [?] 0




```
if (n==1) // valid
  return A[0]
or
if (n==0) // valid
  return 0
```


A[]
isSorted ?
n
n-1
X
A[0] < A[1] ✓
X
✓
✓
✓
<i
n-1
A[0] < A[1] && isSorted(A+1, n-1)
Decreasing
n → n=5
1 to n
= (5 4 3 2 1)


n=5
5 4 3 2 1
n= 1M
1M
1M-1 ..... 3 2 1
f(n)
5
⇒ print n
⇒ f(n-1)
1 to n-1


(n) 1 to n Dig
void f(n) {
if (n==1) ⇒
print i;
return;
print from 1 to 1 in Dig order
return;
print n
f(n-1) X
f(1)

```
printDec(int n) {
  if(n==1) print 1
  prints n
  printDec(n-1) = //optional
  return;
}
```

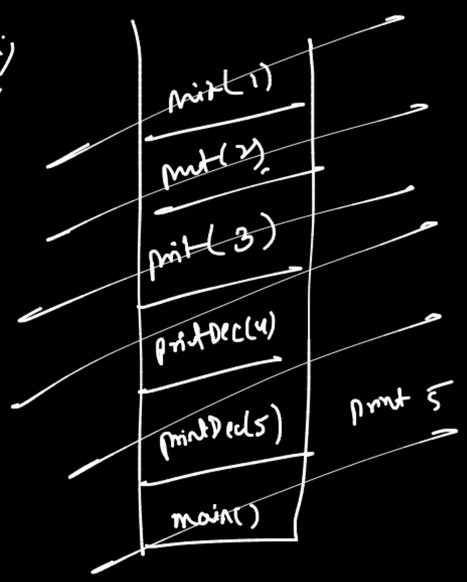
void
↓
once
↓
return;

print(1) → 1

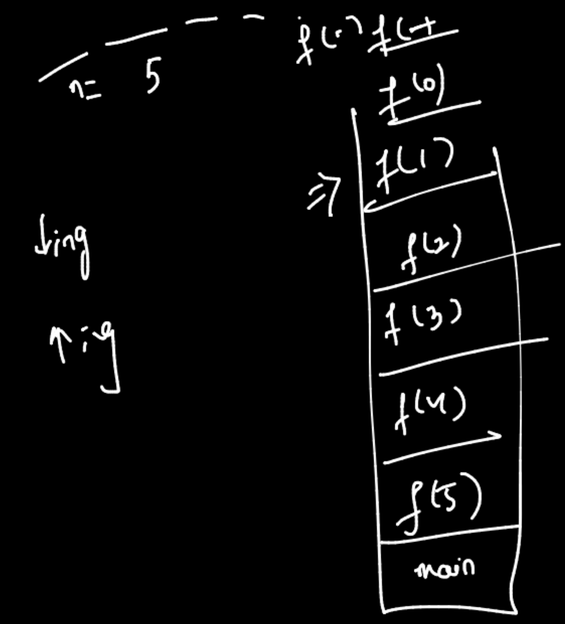
print(0) → 0

print(-1) : .

n=5 ⇒ 5 4 3 2 1



o/p: 5 4 3 2 1



5 4 3 2 1 0 -1

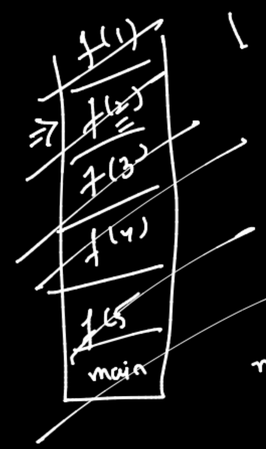
sing

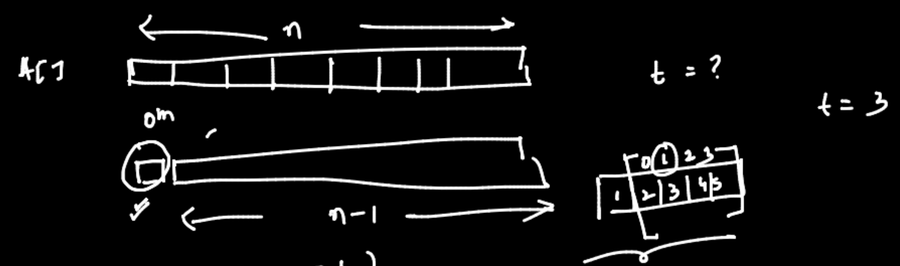
↑ing

return;

```
void printOne(int n)
  if(n==1) print 1
  return;
  print(n-1);
  print n;
```

1 2 3 4 5 ⇐ 1 to n-1 in asc order





t = 3

if (A(0) == t)
⇒ return 0; ←

int i = f(A+1, n-1, t)

i+1

index of 1st occ of t in the array A+1

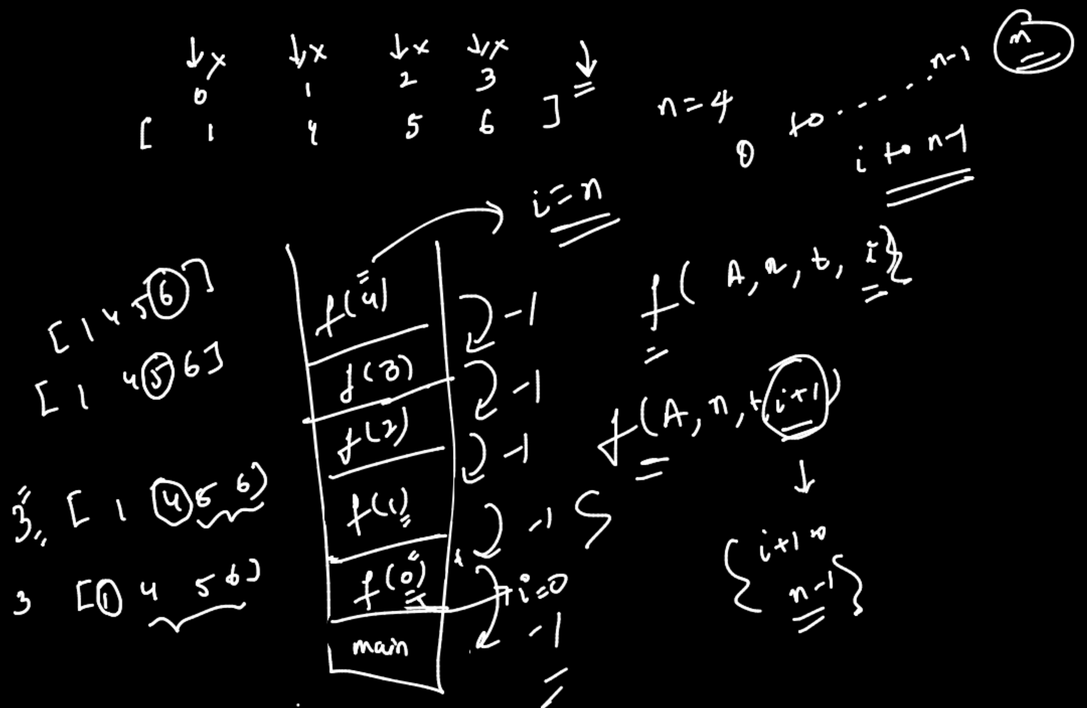
⇒ firstOcc (A, n, t) {
if (n==0) return -1;
X = if (A(0) == t) return 0;
X = int i = firstOcc(A+1, n-1, t)
return i+1;
}

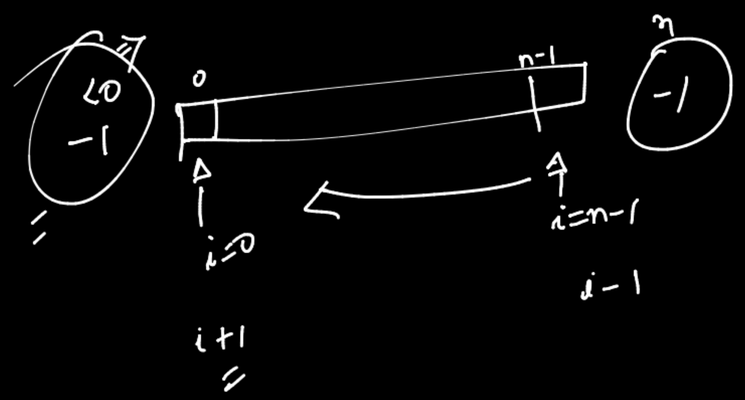
n=1 n=0

if (n == 1) {
⇒ if (A(0) == t)
return 0
else
ret -1
} Base

⇒ [if (n==0) {
ret -1;

⇒ firstOcc (A, n, t, 0) ← initial call


0 1 2 3
[1 4 5 6]
n=4
0 to n-1
i to n-1
i=n
f(4)
f(3)
f(2)
f(1)
f(0)
main
-1
i=0
f(A, a, b, i)
f(A, n, i+1)
{ i+1 to n-1 }
[1 4 5 6]
[1 4 5 6]
[1 4 5 6]
3 [0 4 5 6]


0
n-1
n
-1
i=0
i=n-1
i+1
i-1