# Recursion

Problem : do some work
+
solve a sub-problem
⟹ output

fact(5) = 5 × fact(4)

fact(n) : n * fact(n-1)

4 × fact(3)

we keep breaking into sub problems

we need a base case // stopping case

otherwise: infinite ~~loop~~ calls

Fibonacci Series : $f(n) = f(n-1) + f(n-2)$

increasing order : 1 to N

fact(5)

5!

fib(4)

4th term of Fibonacci series

N = 7

expected output

printSeries (1, 7) : 1, 2, 3, 4, 5, 6, 7

printSeries (2, 7) : 2, 3, 4, 5, 6, 7

printSeries (5, 7) : 5, 6, 7

2.) want to print values 1 to N in increasing order.

printSeries (a, b)

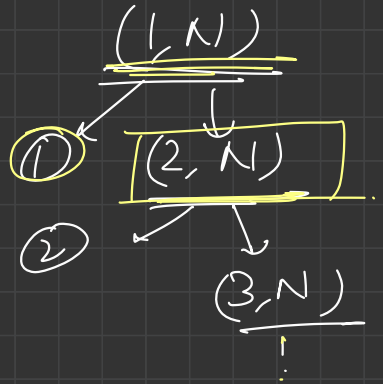
a → 1 b → N

printSeries(1, N) : 1, 2, 3, ..., N

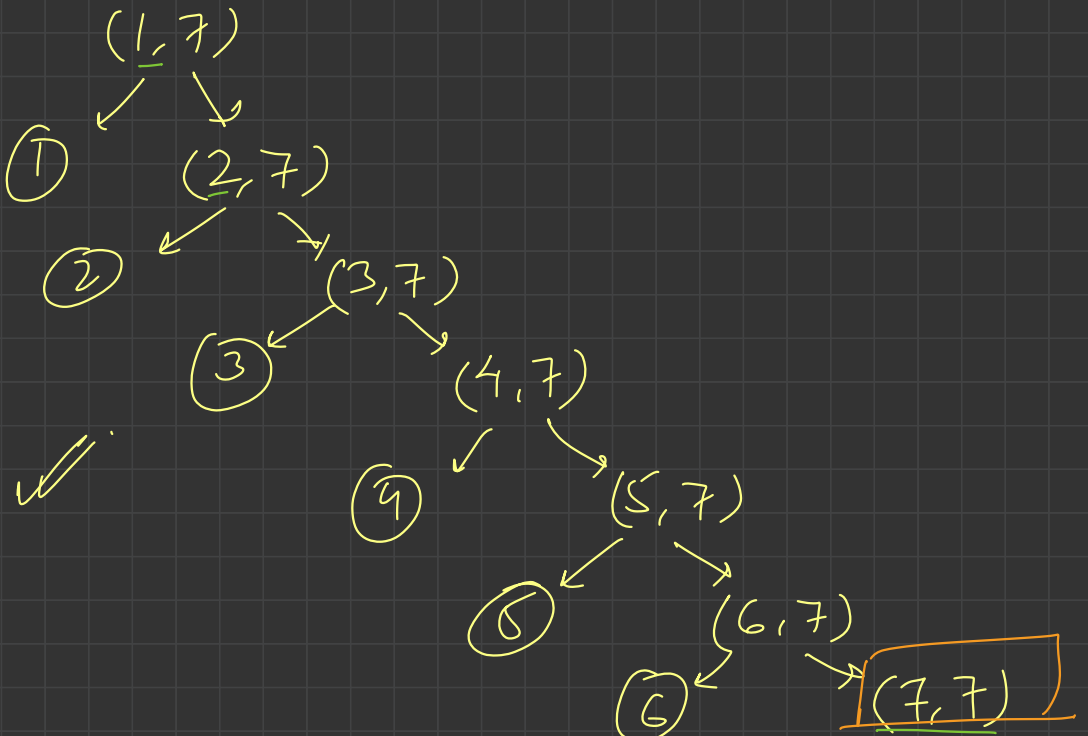
{ ① use a loop for i = 1 to N
→ print i }

② using recursion

{ printSeries(s, e) : → cout << s;
printSeries(s+1, e). }



printSeries(1, N): cout << 1;
printSeries(2, N)

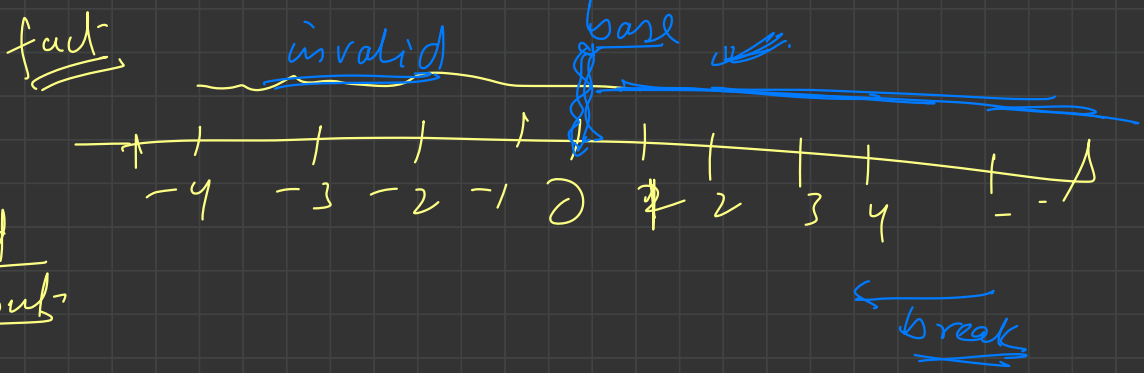


fact(4)
↓
fact(3)
↓
fact(2)
↓
fact(1)
↓
fact(0)
↓
fact(-1)

invalid input

7 → (&7) → use a base case to stop

this is an invalid input for this function




if(s <= e) → valid input

else(s > e) → invalid input

restrict in recursion

printSeries in descending order

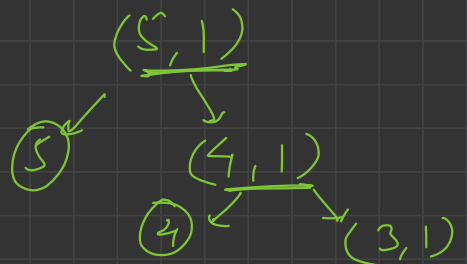
N to 1

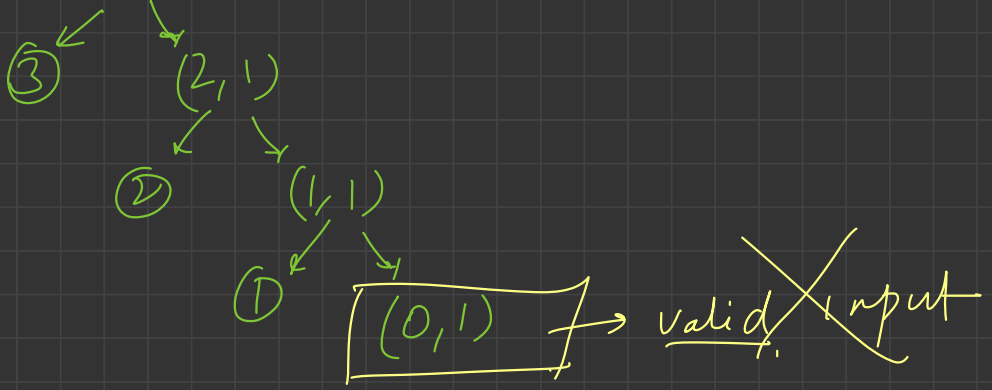
printSeries2(N, 1)

count<=N

printSeries(N-1, 1)

N=5 : 5, 4, 3, 2, 1






reversePrint(5) : 5, 4, 3, 2, 1

5

reversePrint(4) : 4, 3, 2, 1

4

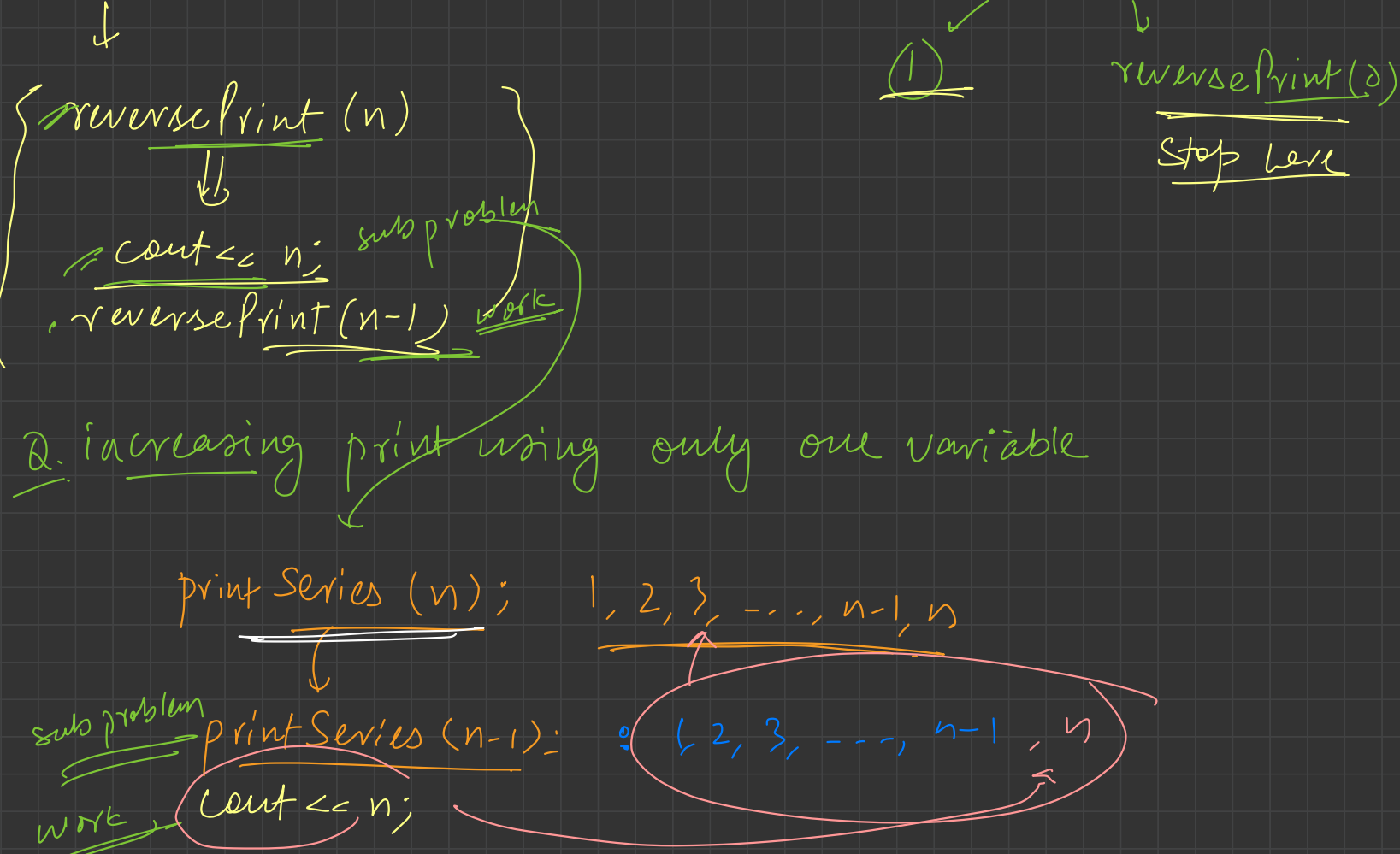
reversePrint(3) : 3, 2, 1

3

reversePrint(2)

2

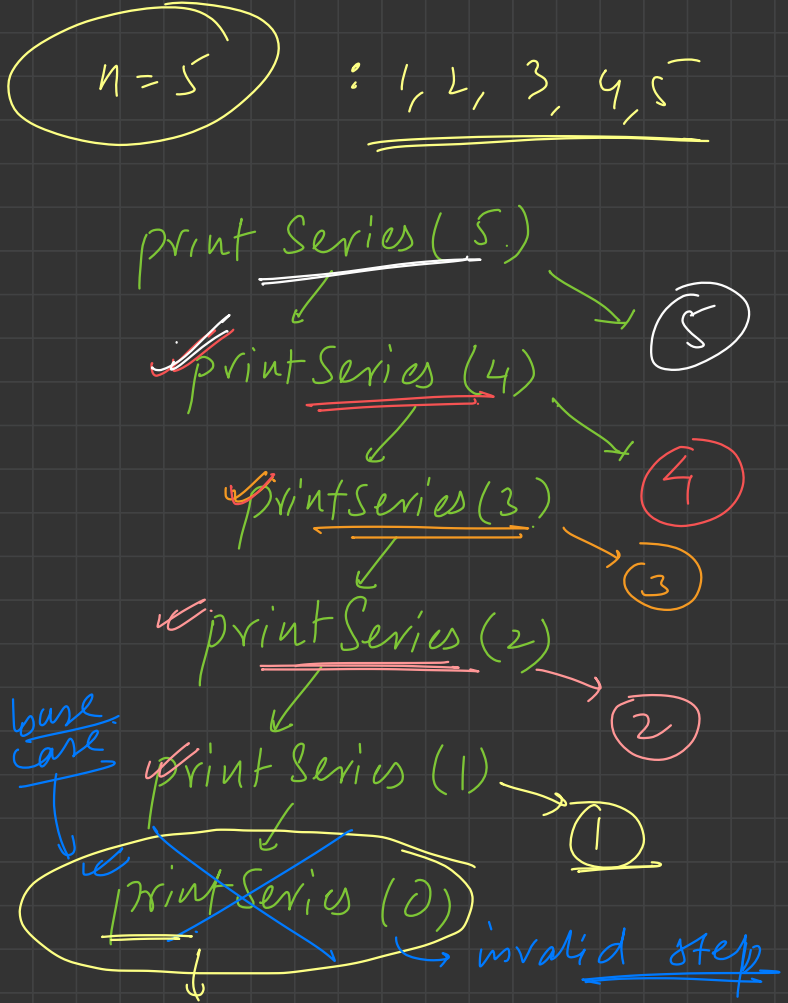
reversePrint(1) : 1


reversePrint(n)
cout << n;
reversePrint(n-1)
sub problem
work
1
reversePrint(0)
Stop here
Q. increasing print using only one variable
printSeries(n);
1, 2, 3, ..., n-1, n
sub problem
printSeries(n-1);
1, 2, 3, ..., n-1, n
work
cout << n;

n = 5 : 1, 2, 3, 4, 5

output:

1 2 3 4 5




: print values from 1 to 0 in increasing order

printSeries Inc(n): → print values from 1 to N in increasing order

first- printSeries Inc(n-1) :→ print from 1 to N-1

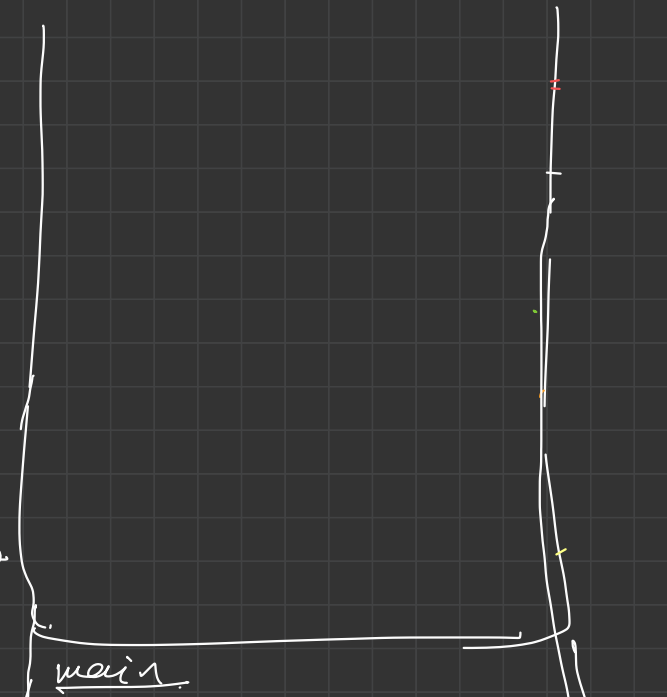
then print(n)

```
PS(int n){
    if(n==0)
        return;
    PS(n-1);
    cout<<n;
}
```

```
main(){
    PS(5);
}
```

n = 5

output- {1 2 3 4 5}



Q. Check if an array is sorted, recursively?

checkSorted
( checkSorted )

arr = {1, 2, 3, 4, 5}  n = 5

how to check if sorted ??

for i → 0 to n-2 {
    if (arr[i] > arr[i+1])
        → not sorted
}
→ sorted

to check if the array is sorted, I only need to compare consecutive elements

true
false checkSorted(arr, n) : {1, 2, 3, 4} → true

check {1, 2, 3, 4}

1<=2 ✓ check {2, 3, 4} ∴ overall sorted

check { arr, n } → compare first two elements ⇏ true

checkSorted(arr after first element)

checkSorted(arr, i, n)

if (arr[i] <= arr[i+1])

AND

if (checkSorted(arr, i+1, n))

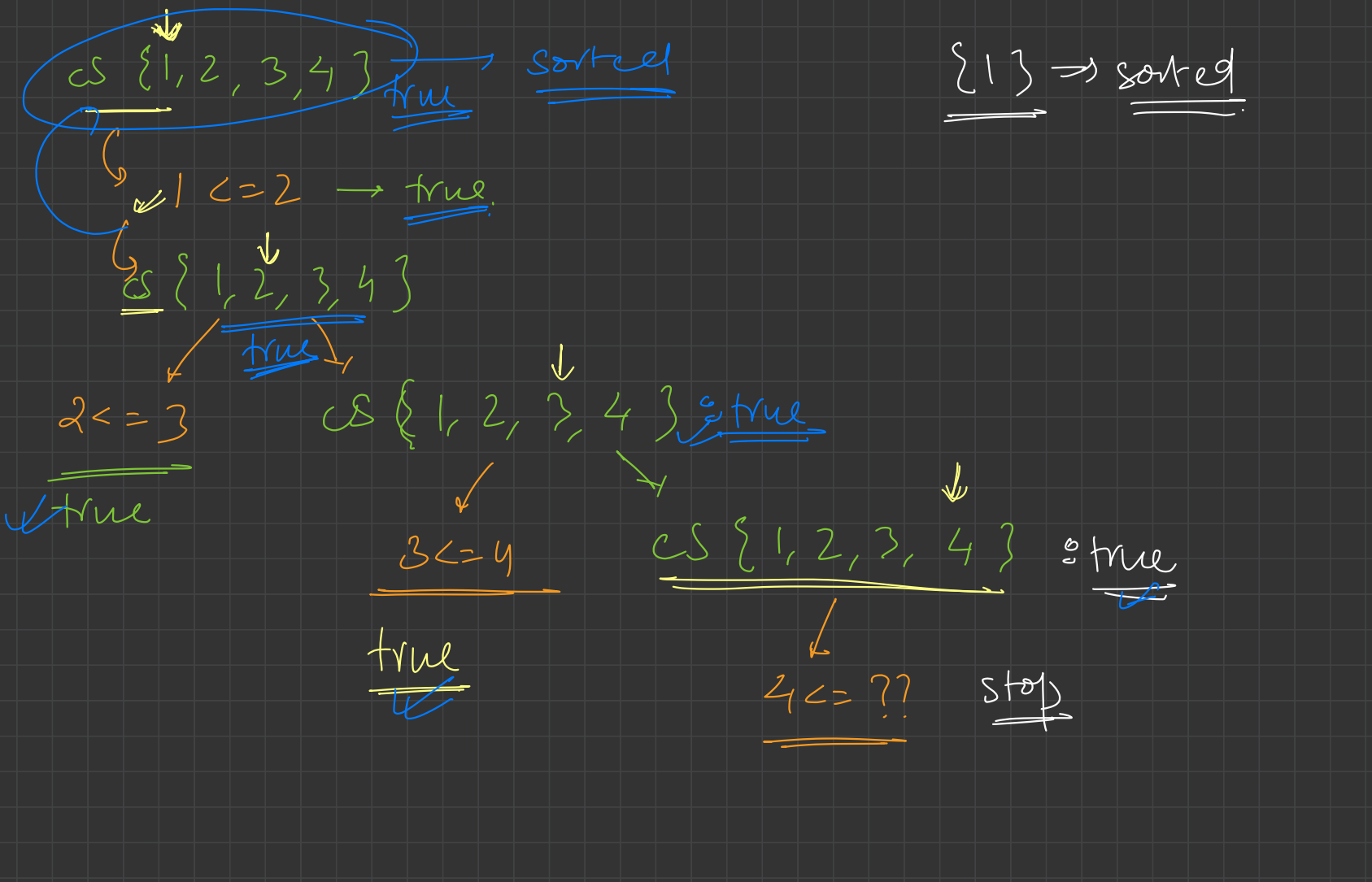
} ⟹ true

else ⟹ false

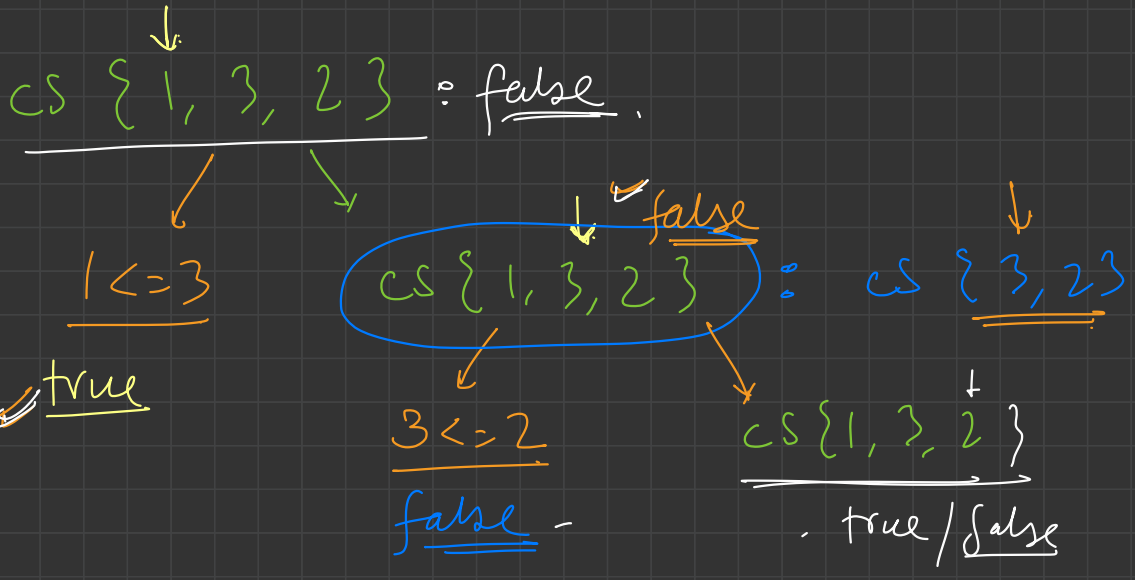
arr = {1, 2, 3, 4}

n = 4

i = 0


cs {1, 2, 3, 4}
sorted
true
{1} → sorted
1 <= 2 → true
cs {1, 2, 3, 4}
true
2 <= 3
true
cs {1, 2, 3, 4} : true
3 <= 4
true
cs {1, 2, 3, 4} : true
4 <= ??
stop

arr = {1, 3, 2} : not-sorted



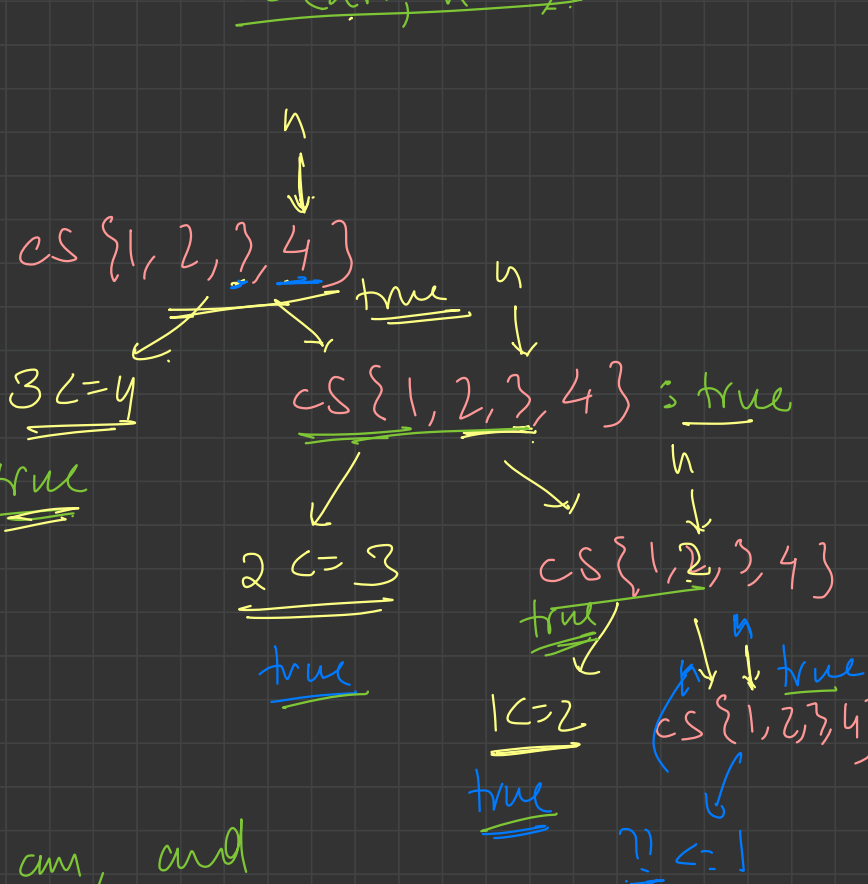
```
bool checkSorted(int arr[], int n){
    //base case
    //stop and return true when only one element left
    if(n == 0){
        //we are at first index
        return true;
    }

    bool currentElements = (arr[n-1] <= arr[n]);//work
    bool subAhead = checkSorted(arr, n - 1);//recursive call

    if(currentElements and subAhead){
        //sorted..
        return true;
    }
    //unsorted..
    return false;
}
```

index 0 1 - - - - n-1

→ i need to know where i am, and where to stop? → (n)



i know where I am ← right-to-left

to stop index → 0

Q. Calculate sum of array using recursion

Q. Convert number to words.

163: one six three : using recursion

Q. Search an element in array using recursion

Q. multiply two numbers using recursion

Q. calculate power using recursion (try efficiently)