Recursion

What is recursion?

A function that calls itself .

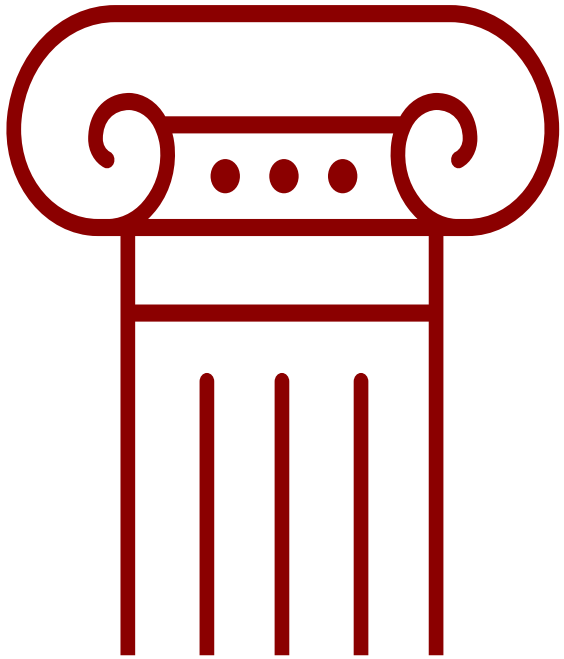
$$F(n) = F(n-1) + F(n\ -2)$$

It might be indirect!

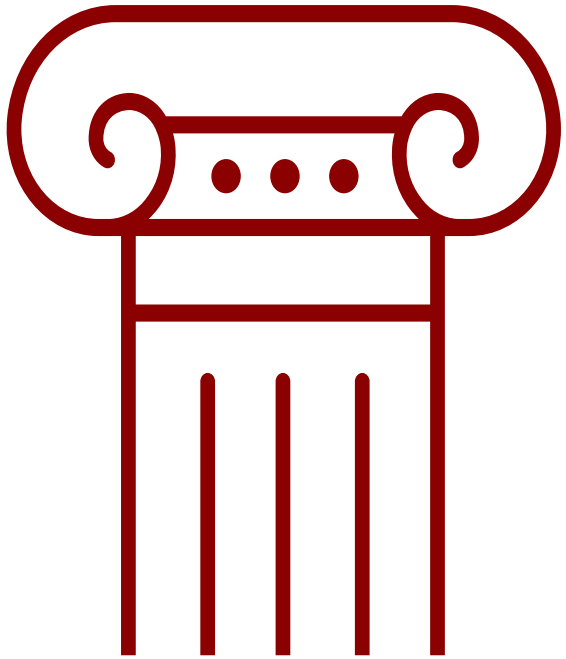
$$F \to G \to F$$

Two Pillars of Recursion

Base Case



Recursive Step



Base case: Every practical recursion should eventually stop calling itself
The condition under which it stops is called Base Case

Recursive step:
1. How the function calls itself
2. How it combines the results of the recursion calls to create the overall result

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

We can create loops with recursion

Print numbers from 1 to n

```
n = 5

for i in range(1, n + 1):
  print(i)
```

```
while i <= n:
  print(i)
```

```
def recursive_for(i, n):
  if i > n :
    return
  print(i)
  recursive_for(i+1, n)
```

Tail recursion

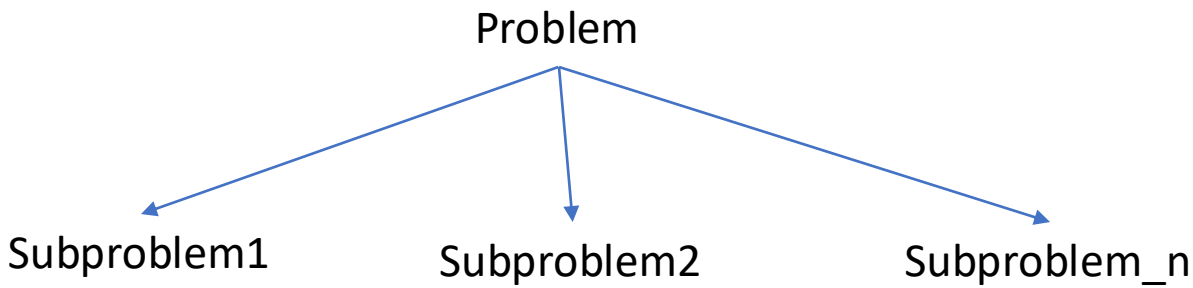
The last operation is recursive call

Breaking a problem into subproblems

Breaking a problem into subproblems

Sove each subproblem (recursively break them into subproblems of their own)

Combine the solutions of the subproblems to solve the problem



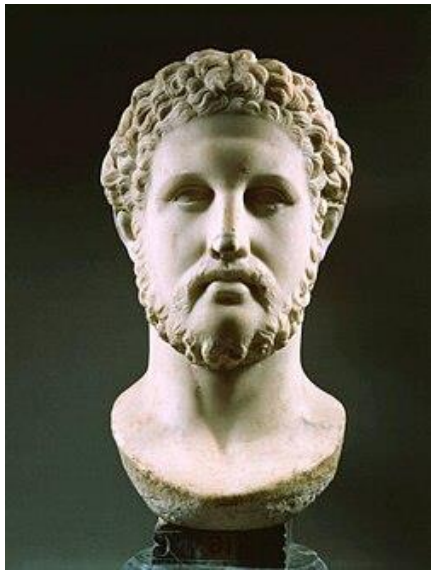
Divide and Conquer

Breaking a problem into subproblems

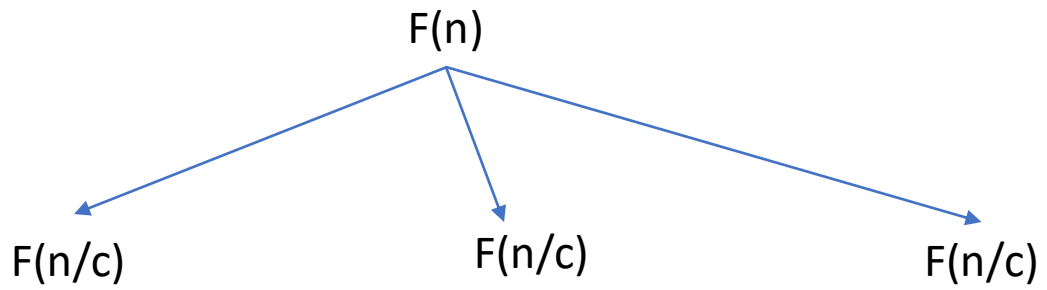
Divide by dividing the size of the problem by a constant c

Sove each subproblem (recursively break them into subproblems of their own)

Conquare Combine the solutions of the subproblems to solve the problem



Philip II of Macedon
382 BC – 336 BC



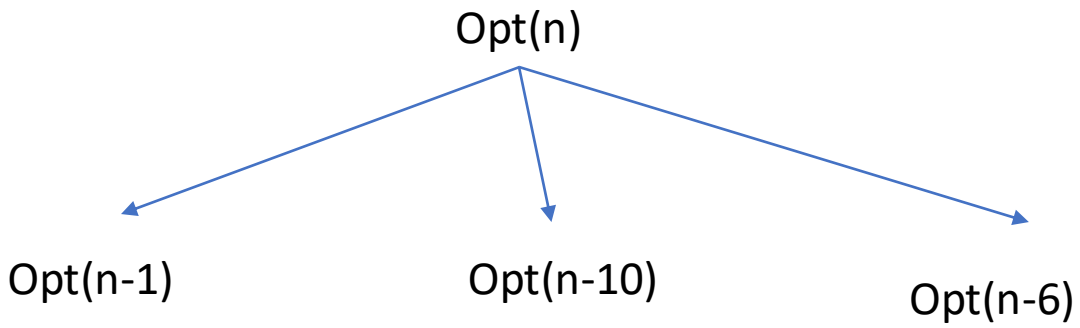
MergeSort

QuickSort

Optimal Substructure

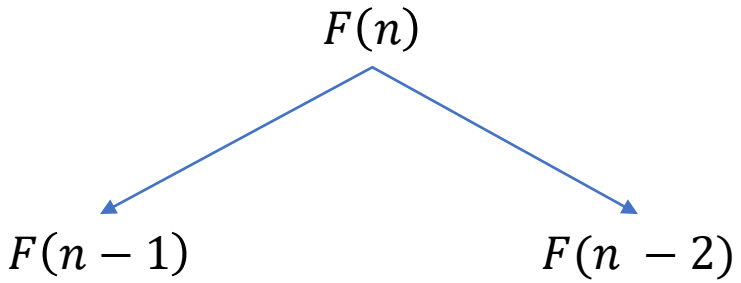
In Dynamic Programming

A problem has O.S if an Optimal solution can be constructed from optimal solution of its subproblems



Some Famous Recursions

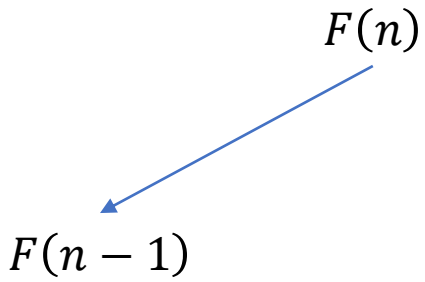
$$F(n) = F(n-1) + F(n-2)$$



Combining function? ADD

Factorial

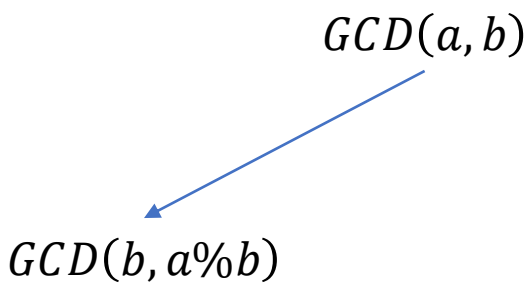
$$F(n) = F(n-1) \times n$$



Combining function? MultiplyByN

GCD

$$GCD(a, b) = GCD(b, a\%b)$$
$$GCD(a, 0) = a$$

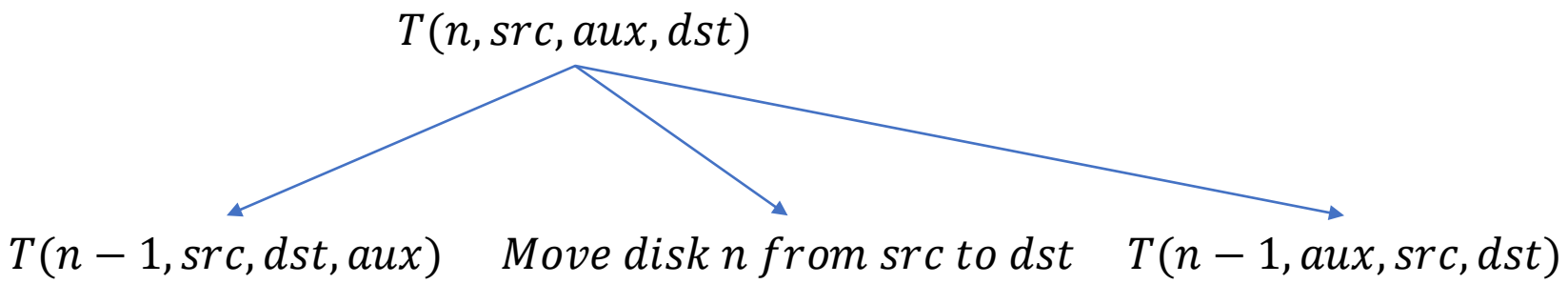


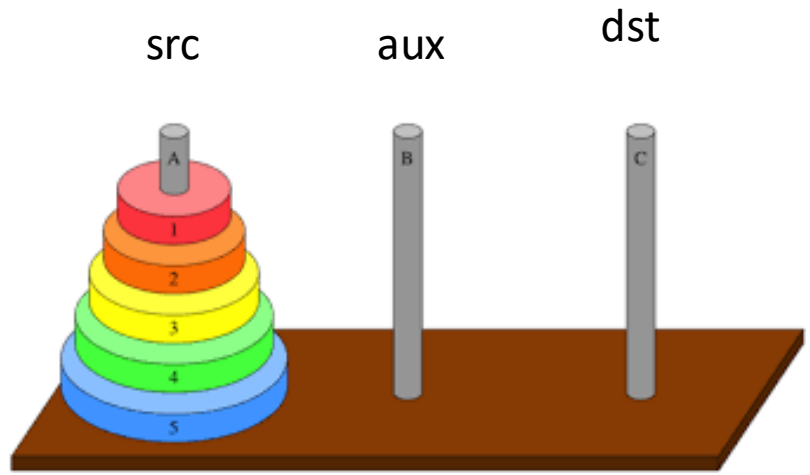
Combining function? Replacement

Tower of Hanoi

$$T(n, src, aux, dst)$$

$$T(n, A, B, C)$$

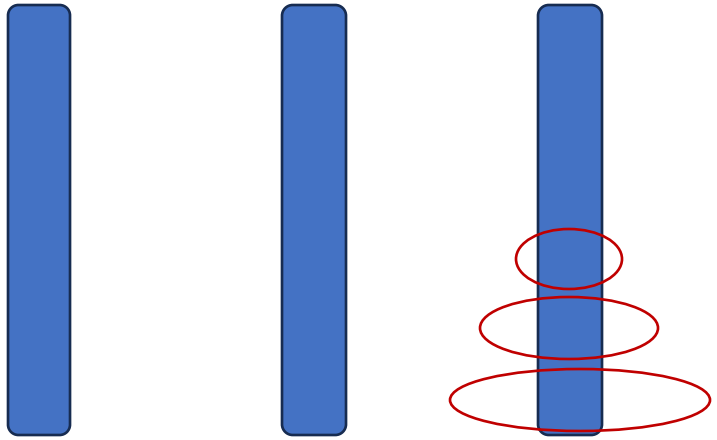




Combining function? Sequence

What should I do when I get a recursive problem

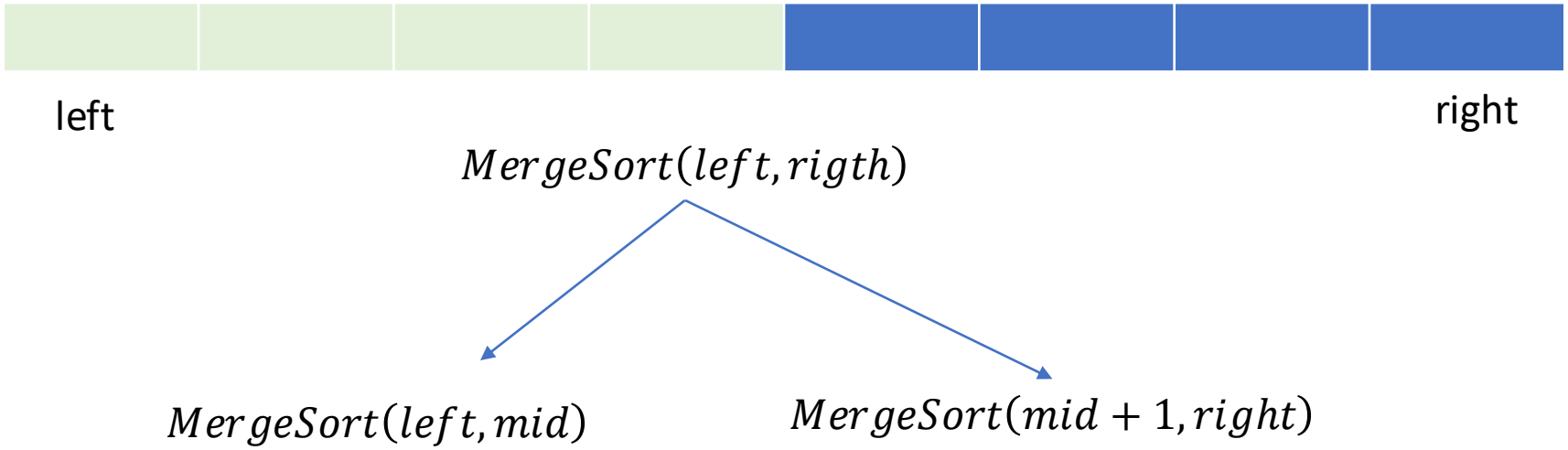
Find a special case where it's super easy to solve the problem



I can solve tower of hanoi for n=0, 1, 2,3

MergeSort

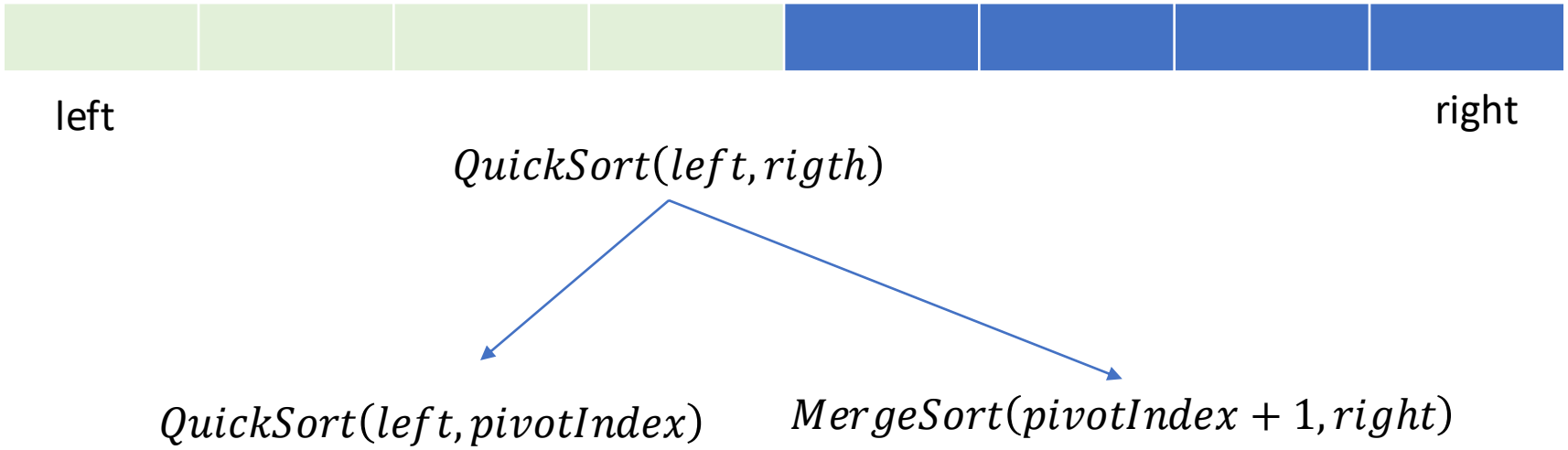
$MergeSort(left, rigth)$



Combining function? MERGE

QuickSort

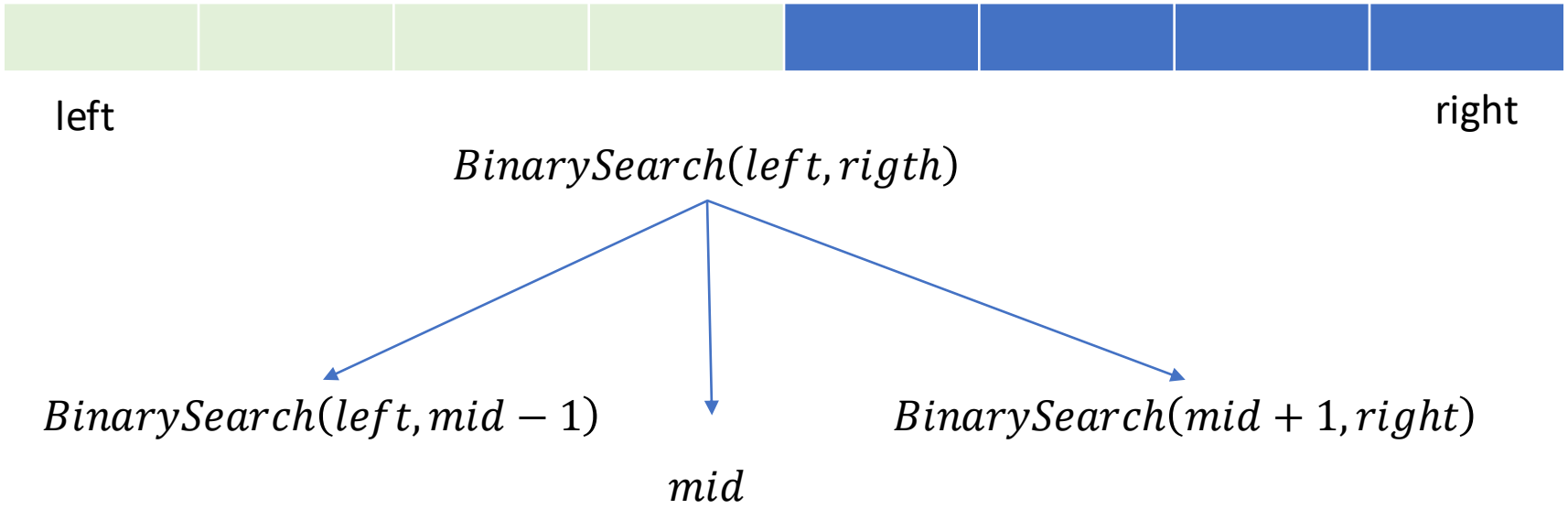
$QuickSort(left, rigth)$



Combining function? Concat

BinarySearch

$BinarySearch(left, rigth)$

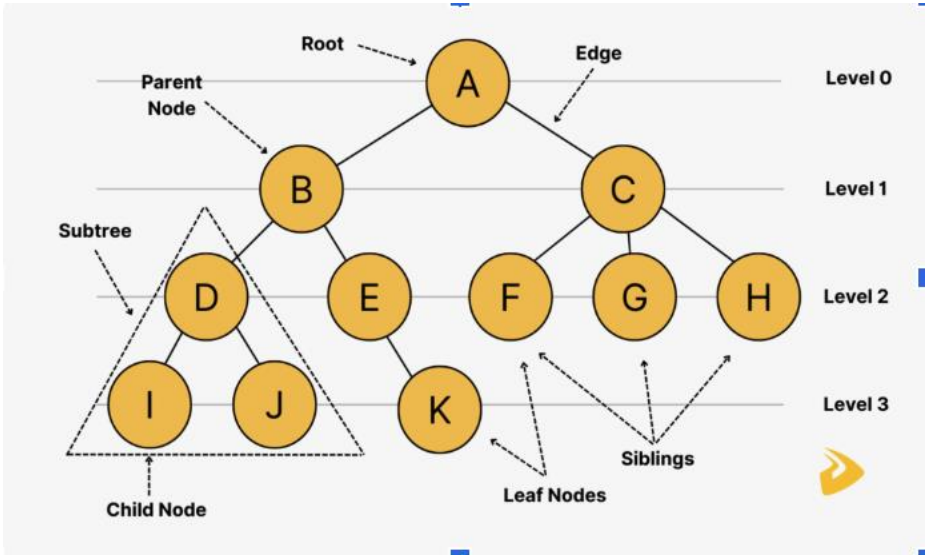


Combining function? OR

Indicators For Recursion

Is there any recursive data structure involved?

A subtree is itself a tree -> recursion!



Graphs

Lists

Indicators For Recursion

Is there any recursive data structure involved?

A list consists of:
1. Two half lists

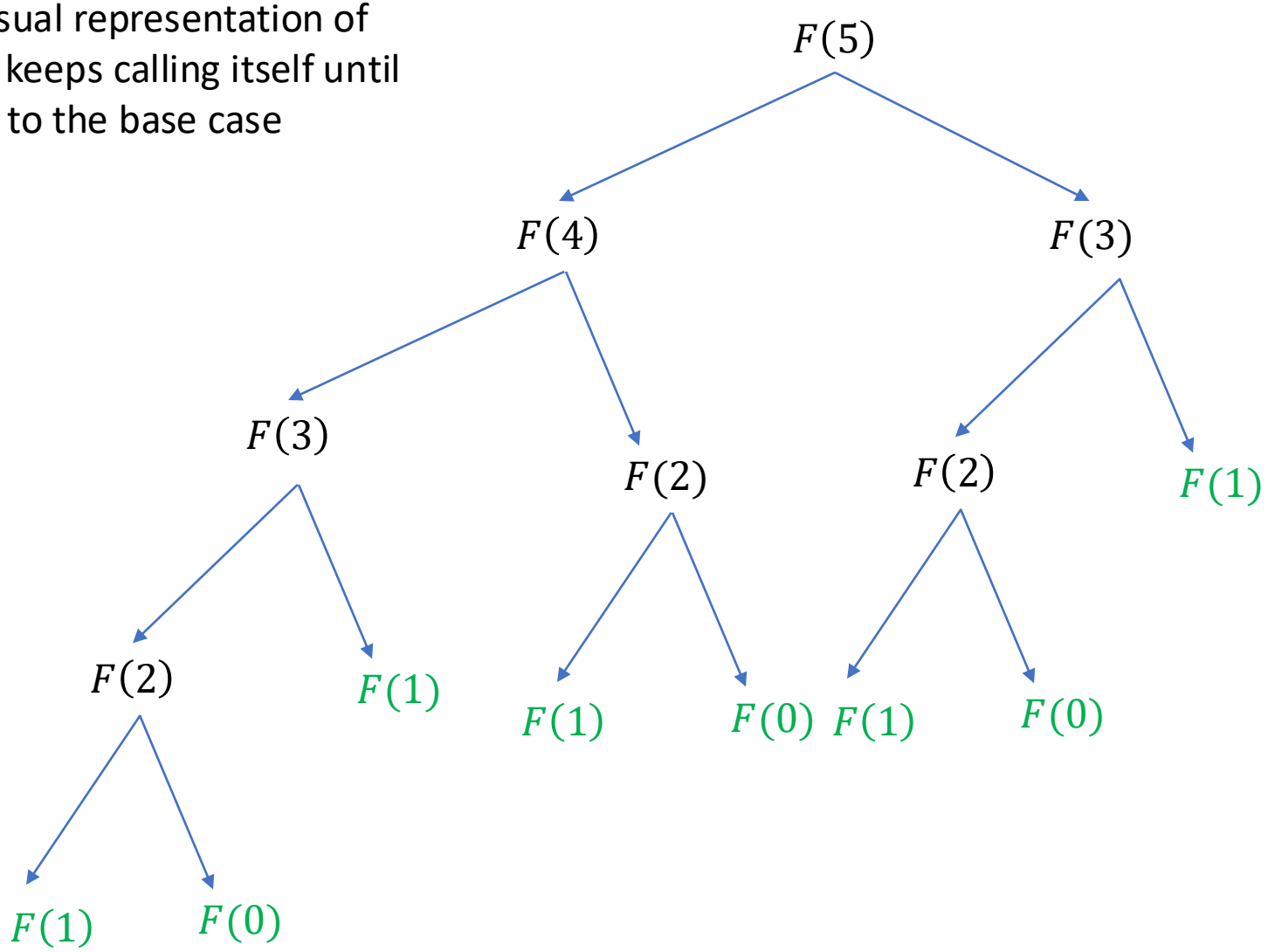
Indicators For Recursion

Is there any recursive data structure involved?

A list consists of:
1. Two half lists
2. The head + the rest of the list (which is a list itself!)

What is the Recursion Tree?

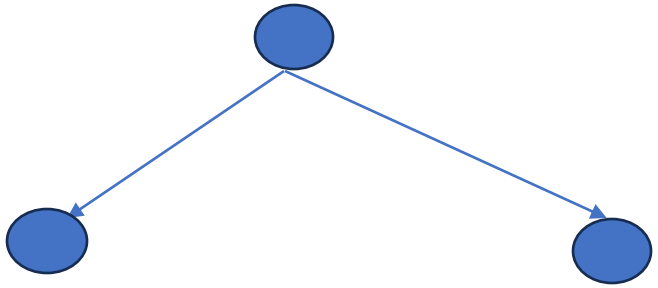
The visual representation of how F keeps calling itself until it gets to the base case

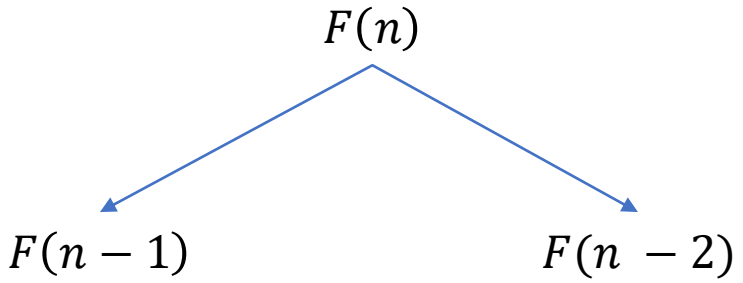


The Most Important Algorithm in CS?

DFS

The Most Important Algorithm in CS?






```
# For binary tree
def DFS(node):
  if is_leave(node):
    return

  DFS(node.left)
  DFS(node.right)
```

```
def fib(n):
  if n == 0 or n == 1:
    return 1

  left = fib(n - 1)
  right = fib(n - 2)

  return left + right
```

Every Recursion is really DFS on the **recursion** tree

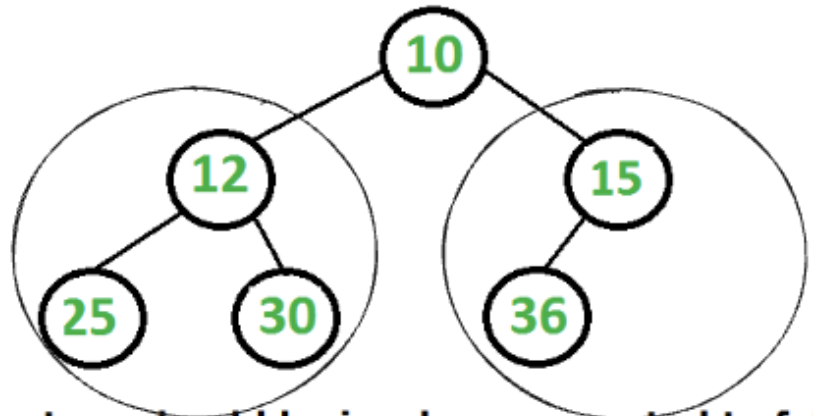
Every Recursion is really DFS on the **recursion** tree

Runtime complexity of DFS in a tree of n nodes: $O(|V| + |E|) = O(n + n - 1) = O(n)$
Memory complexity of DFS: $O(h)$, where h is the height of the tree

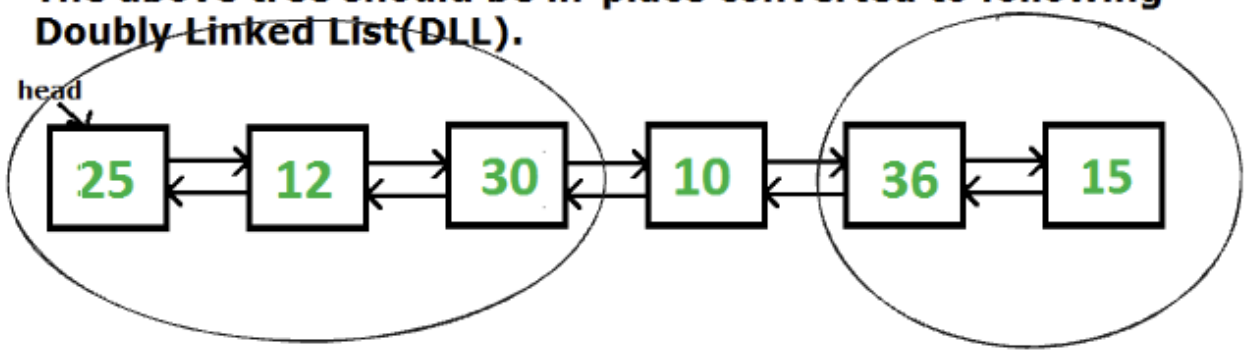
For a recursive function

Runtime complexity: $O(n)$, where n is the number of nodes in the recursion tree
Memory complexity: $O(h + |output|)$, where h is the height of the recursion tree

* This assumes that the combine operation is done in O(1). If that's not the case, use a famous theorem called the Master Theorem



The above tree should be in-place converted to following Doubly Linked List(DLL).



```
def combine(left, right, node):

if left is None and right is None:
  node.left = node
  node.right = node
  return node
elif left is None:
  node.right = right
  node.left = right.left
  right.left.right = node
  right.left = node
  return node
elif right is None:
  left.left.right = node
  node.left = left.left
  left.left = node
  node.right = left
return left
else:
 left.left.right = node
 node.left = left.left
 right.left.right = left
 left.left = right.left
 right.left = node
 node.right = right
return left


def binaryTreeToDLL(root):
 if not root:
  return

  dll_left = binaryTreeToDLL(root.left)
  dll_right = binaryTreeToDLL(root.right)

  combine(dll_left, dll_right, root)
```