Decomposition in Python

Lesson Objectives

At the end of this lesson, you should be able to:

- Describe Divide-and-Conquer and Recursion as a decomposition process
- Apply the method of Divide-and-Conquer and Recursion in Python coding

Topic Outline

NANYANG TECHNOLOGICAL UNIVERSITY SINGAPORE

- Divide and Conquer
- Recursion
- Binary Tree
- Complete Binary Tree (CBT) Representation and Operations

Decomposition

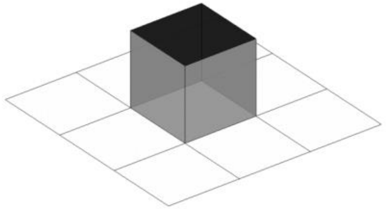
Divide-and-Conquer

- Decompose a problem into several sub-problems
- Solve each sub-problem
- Compose the solution to sub-problems

Recursion naturally supports divide-and-conquer.

Recursive Function

- A function that invokes itself
- Very useful and important in computer science



Example:

Factorial of n

$$n! = \begin{cases} 1\,, & n = 0 \\ n \times (n-1)!\,, & n > 0 \end{cases}$$

```python
def f(n):
    if n == 0:
        return 1
    else:
        return n * f(n - 1)
```

Recursion (Cont'd)

Recursive Function: General Form

```
def recursiveFunc(param1, param2, …):
    if exp:        # base case (conquer)
                   …
        return value

    else:          # recursive step (divide)
        recursiveFunc(subproblem1)
        recursiveFunc(subproblem2)
                   …
        return value
```

How to Write a Recursive Function

Determine the interface (signature) of the function

- How many **parameters**? What are they?
- What is the **return object**?
- What is the **functionality** of the function?

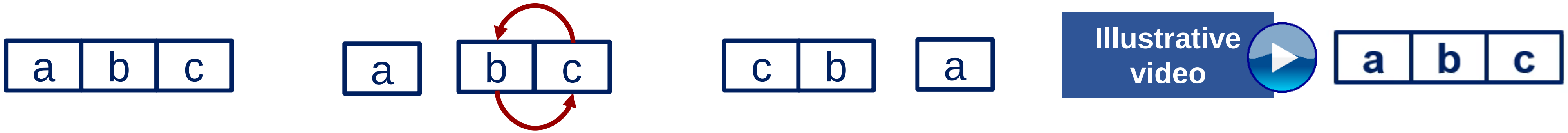
Assume you had finished the implementation of the function

Develop the function body

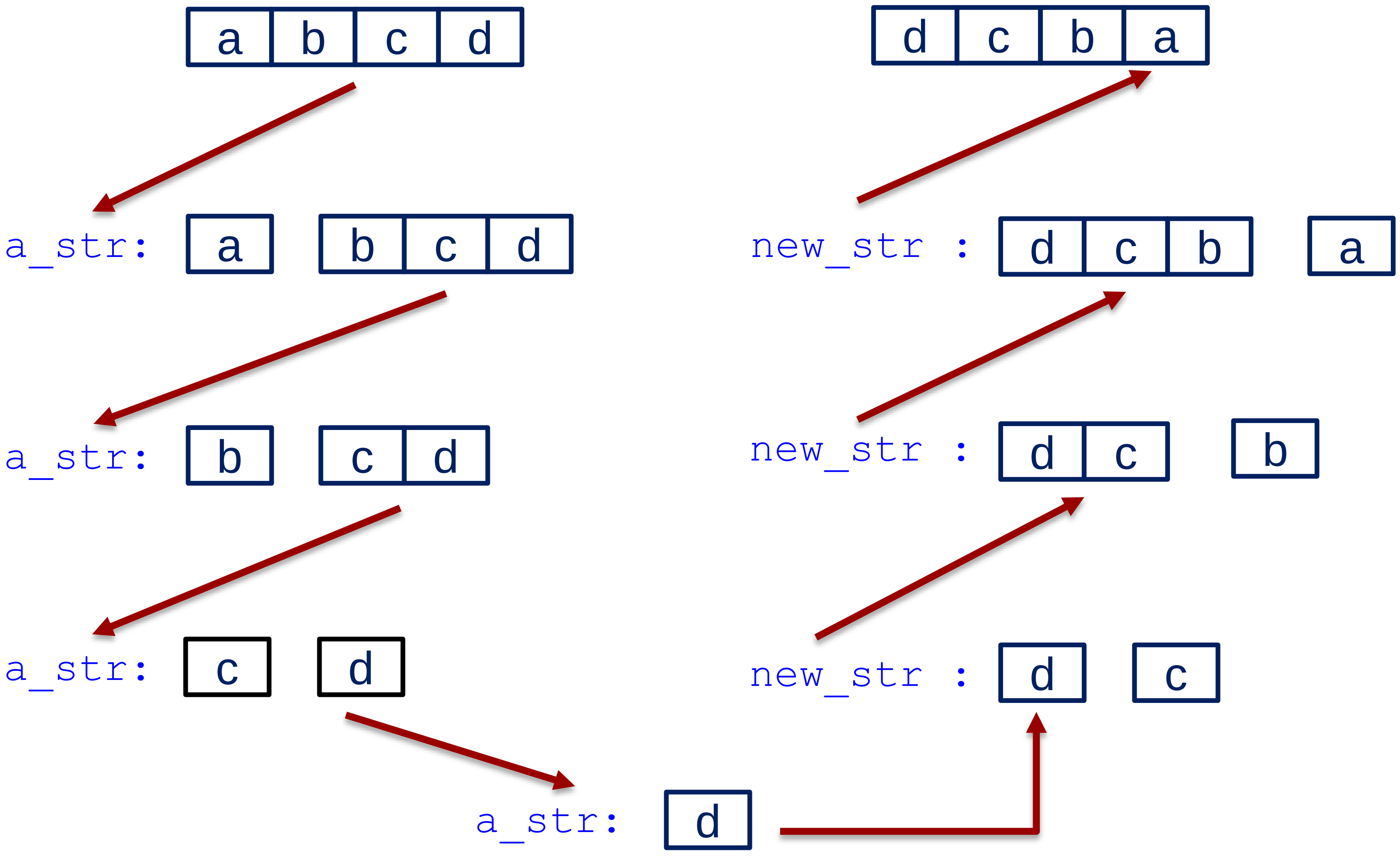
- **Base Case** (**Conquer**): Solve the primitive case, and then return the result
- **Recursive Step** (**Divide**)
  - Decompose the problem into subproblems (with the same structure)
  - Call the function to solve each subproblem
  - Compose the final result from subproblems, and then return it

Example: Reversing a String

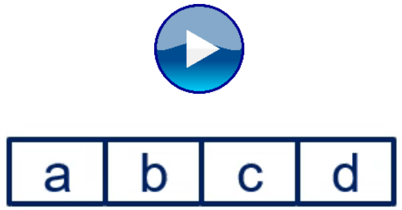
```python
def reverser(a_str):
   if len(a_str) == 1:    # base case
       return a_str

   else:                  # recursive step
       new_str = reverser(a_str[1:])+ a_str[0]
       return new_str
```



Example: Reversing a String (Cont’d)



Illustrative video



Performance of Recursion

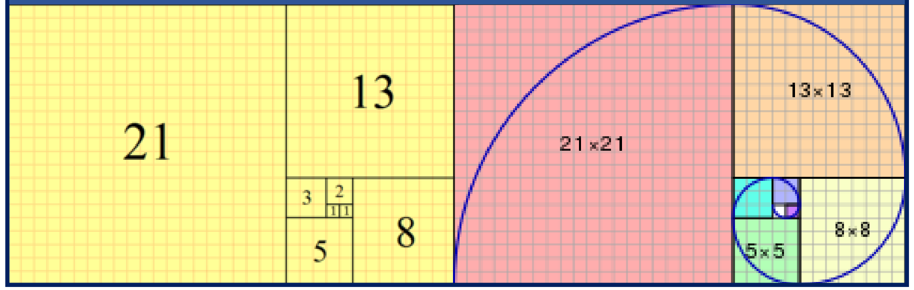
Recursive function may be inefficient!

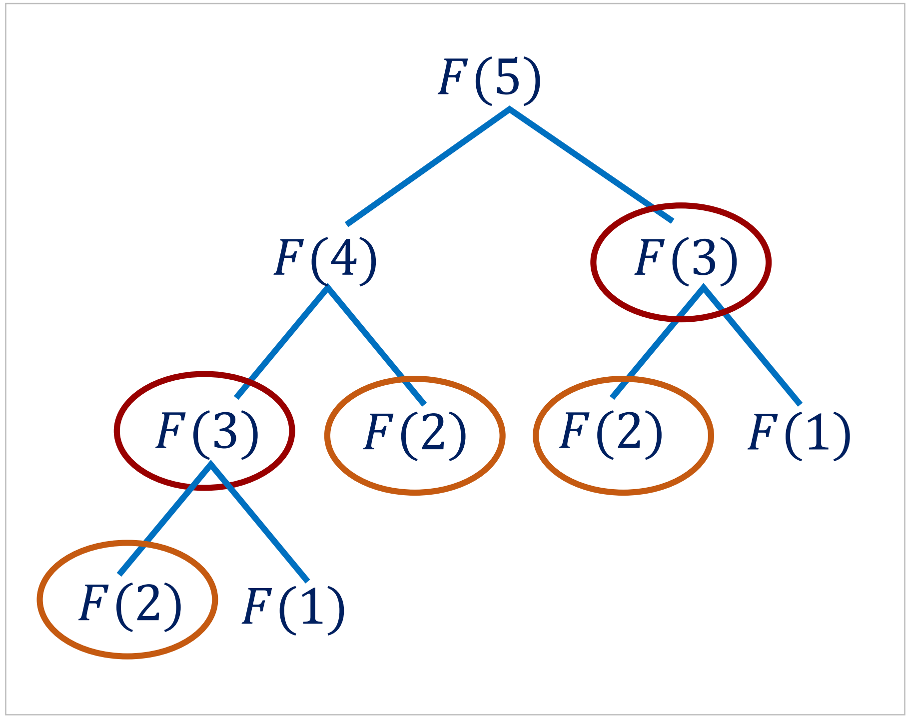
- Redundant computation!

Fibonacci Number:

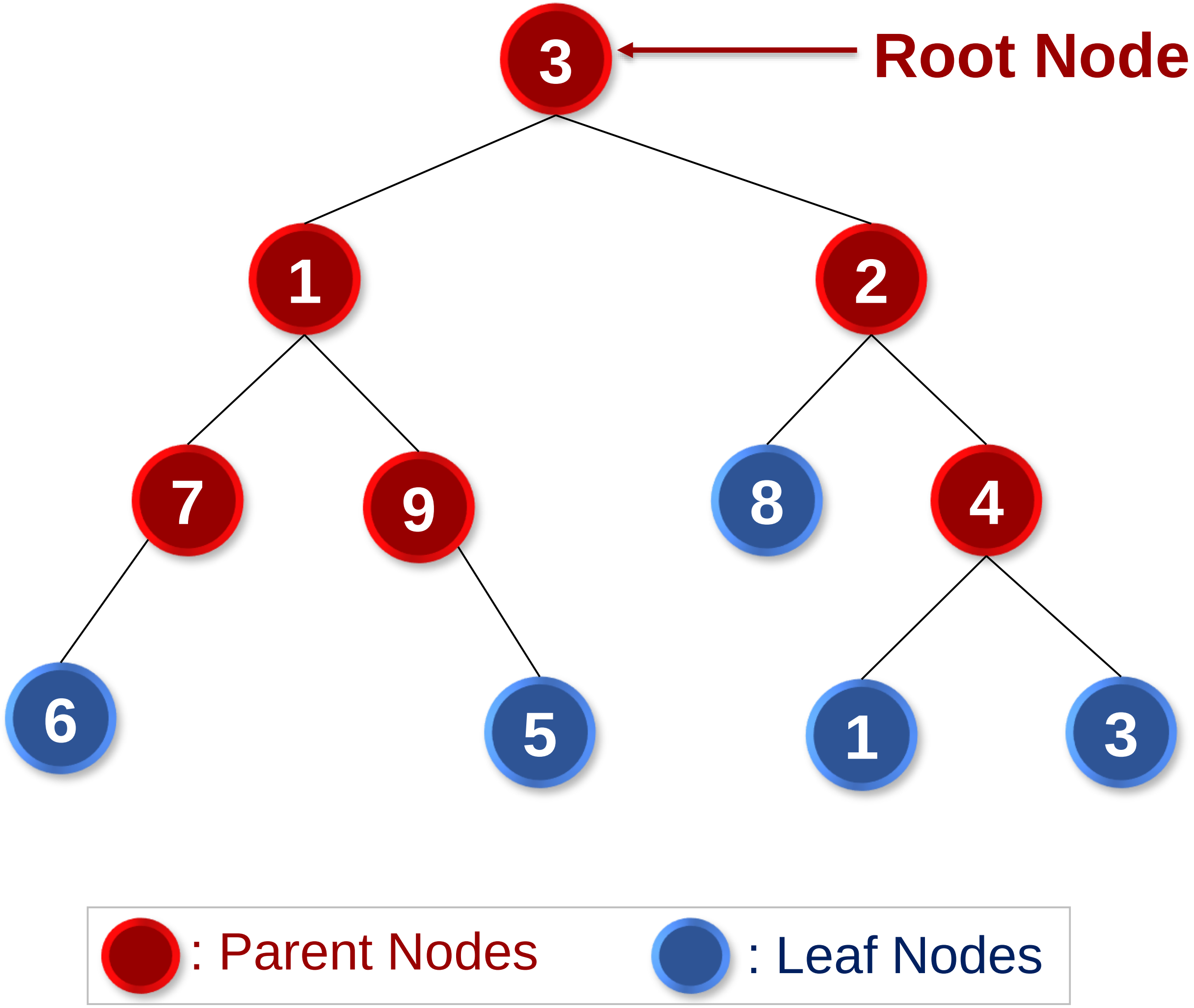
$F(1) = 1$ and $F(2) = 1$

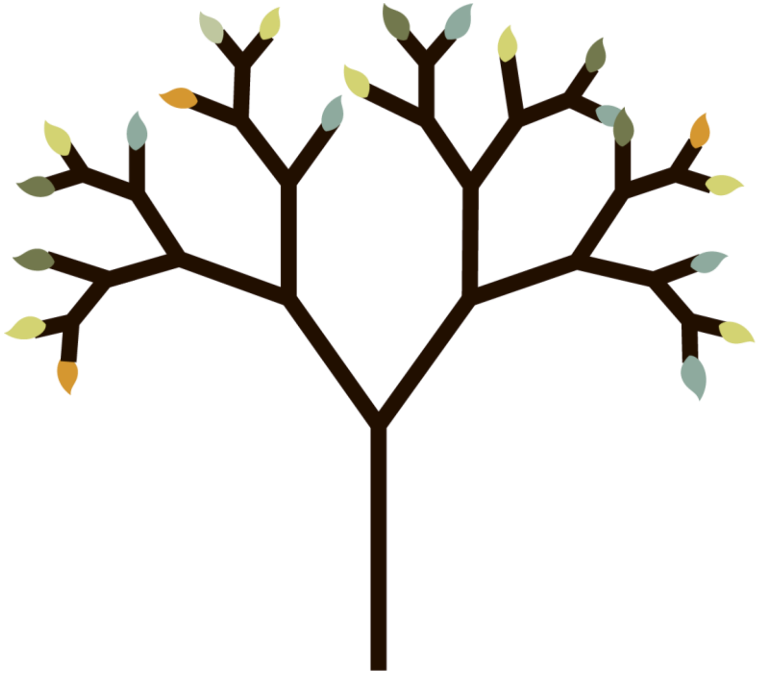
$F(n) = F(n-1) + F(n-2)\ ,\ n \geq 2$





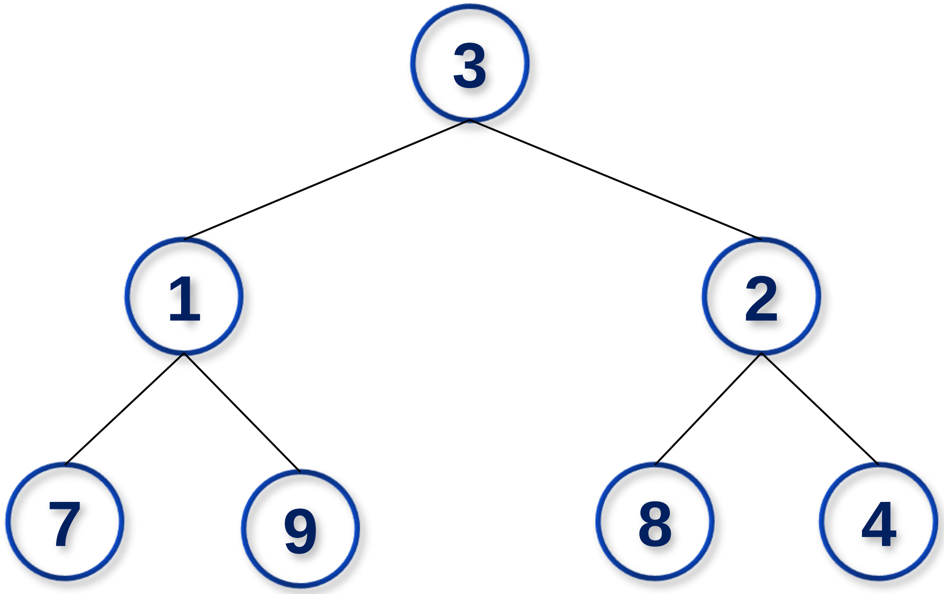
Binary Tree





Complete Binary Tree (CBT)

Every parent node in a ***complete binary tree*** (**CBT**) has exactly **two** child nodes.



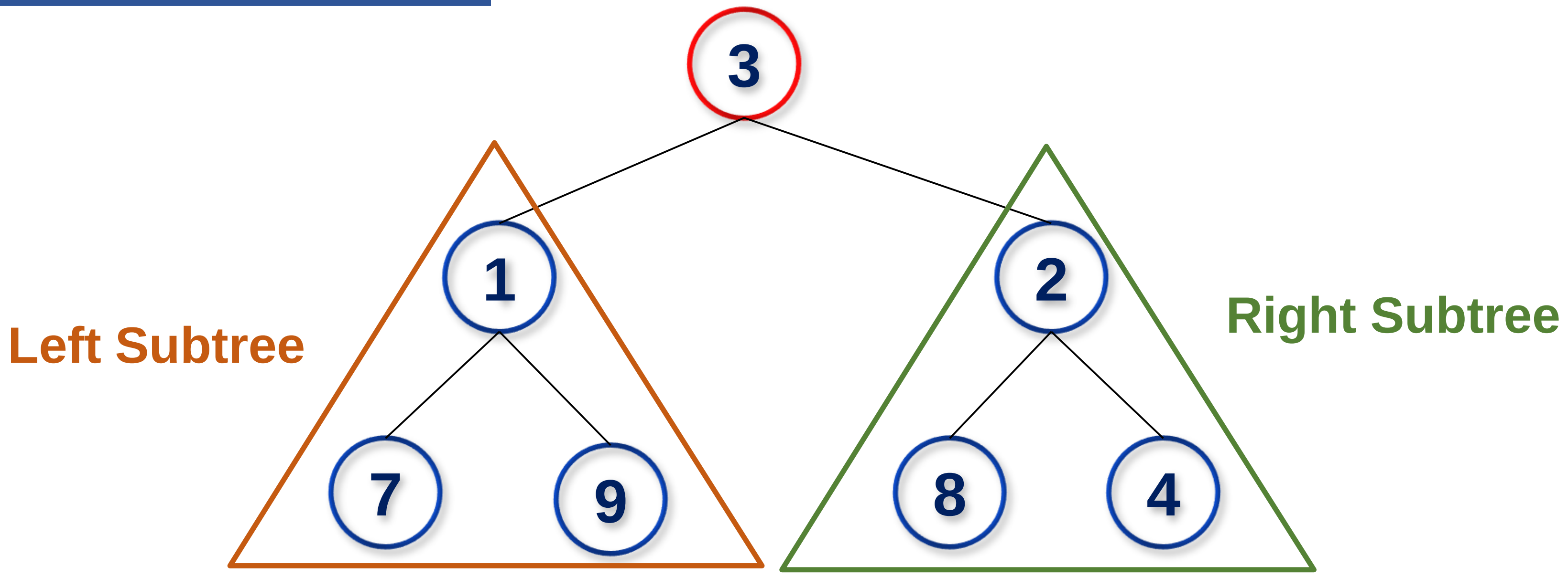
CBT is the focus in this session…

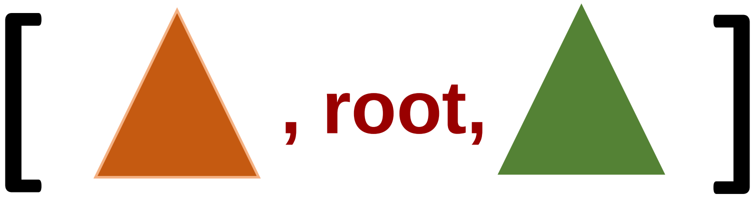
CBT Representation

- How do we represent a CBT?
- What data structures do we have now?
  - List
  - Tuple
  - Dictionary
- Which one is better?

CBT Representation (Cont'd)

Using list maybe a good idea

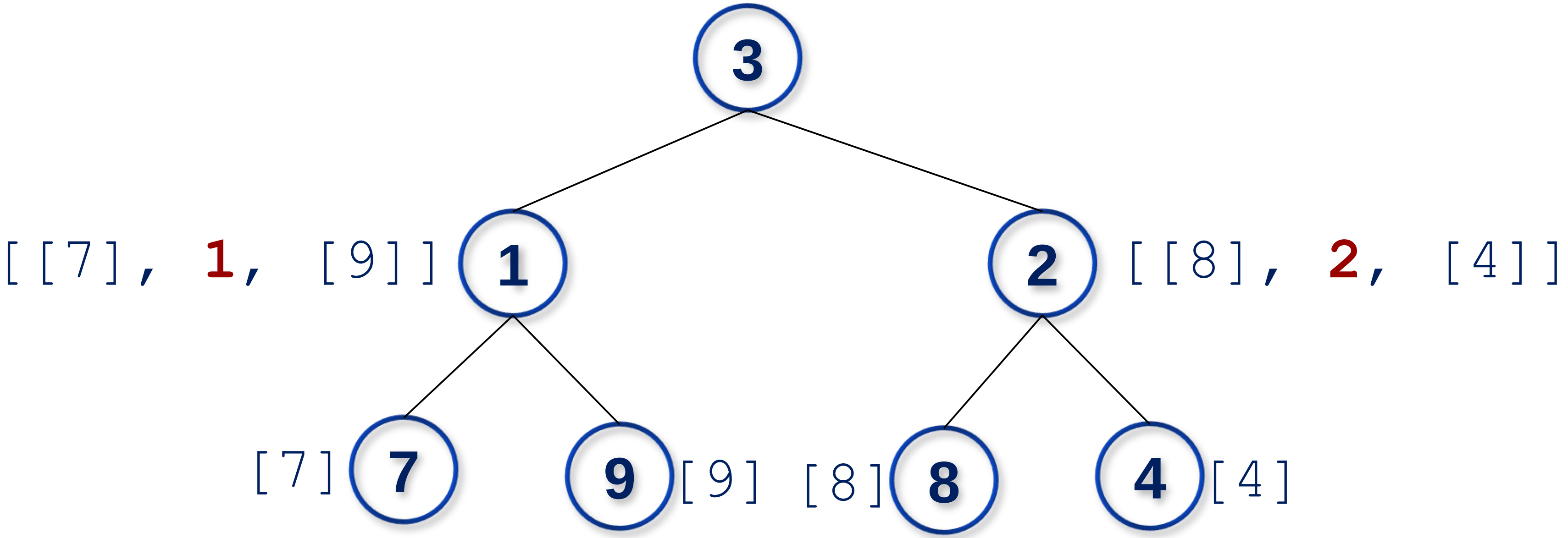




CBT Representation (Cont'd)

Using list maybe a good idea

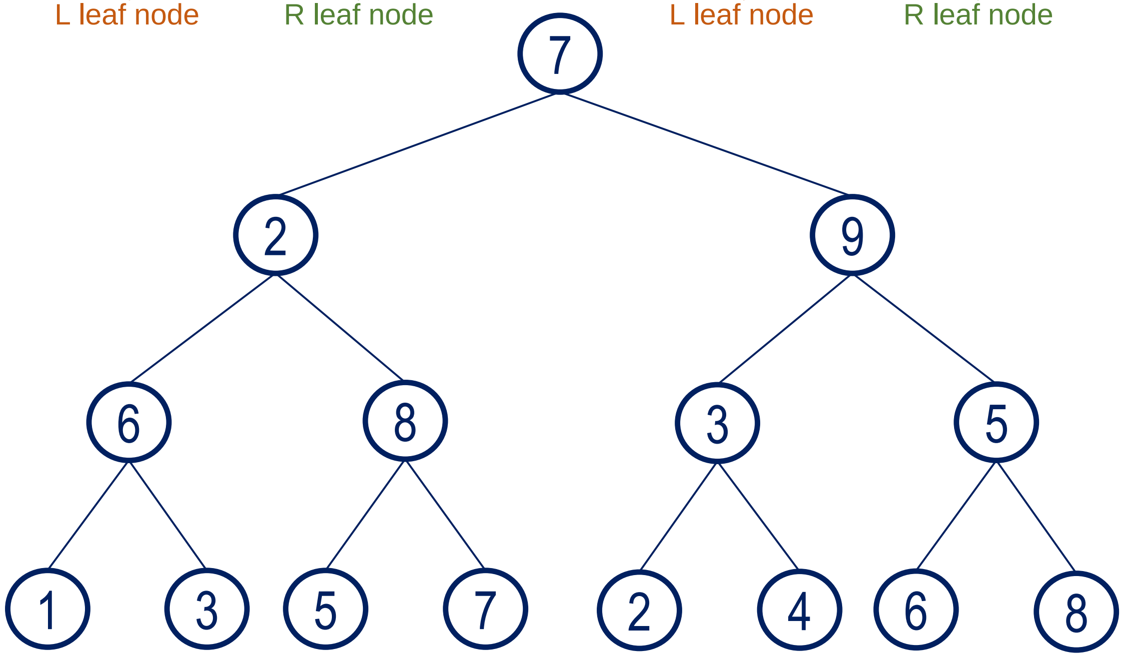
`[[[7], 1, [9]], 3, [[8], 2, [4]]]`



Creating CBT from the List: Example

What does the following CBT look like?

[[[[1],6,[3]],2,[[5],8,[7]]],7,[[[2],3,[4]],9,[[6],5,[8]]]]



Root node: 7
Parent nodes: Red nos.
Left leaf nodes: Orange nos.
Right leaf nodes: Green nos.

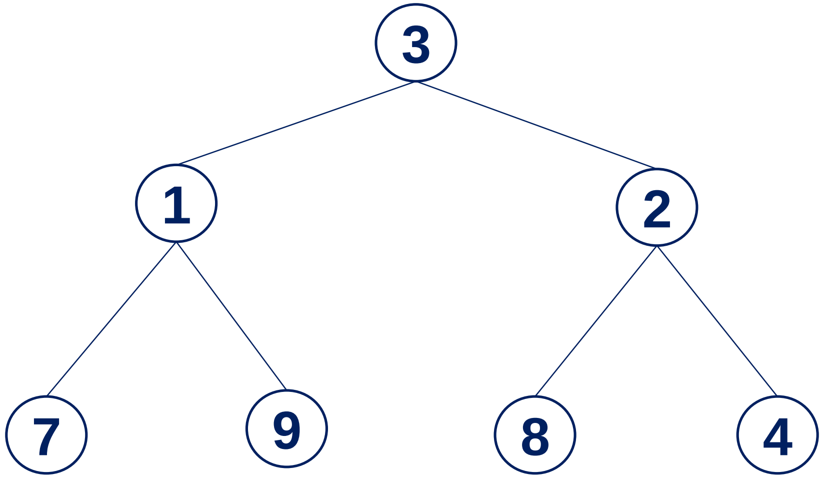
Operations in CBT

`numOfNodes(t)`	returns the total number of nodes in a CBT t
`sumNodes(t)`	returns the summation of all nodes in a CBT t
`maxNode(t)`	returns the maximum value of nodes in a CBT t
`minNode(t)`	returns the minimum value of nodes in a CBT t
`mirror(t)`	returns the mirrored CBT of a CBT t

Operations in CBT (Cont'd)

`numOfNodes(t)`

```
tree = [[[7], 1, [9]], 3, [[8], 2, [4]]]

print("# of Nodes: ", end='')
print( numOfNodes(tree) )
```



# of Nodes: 7

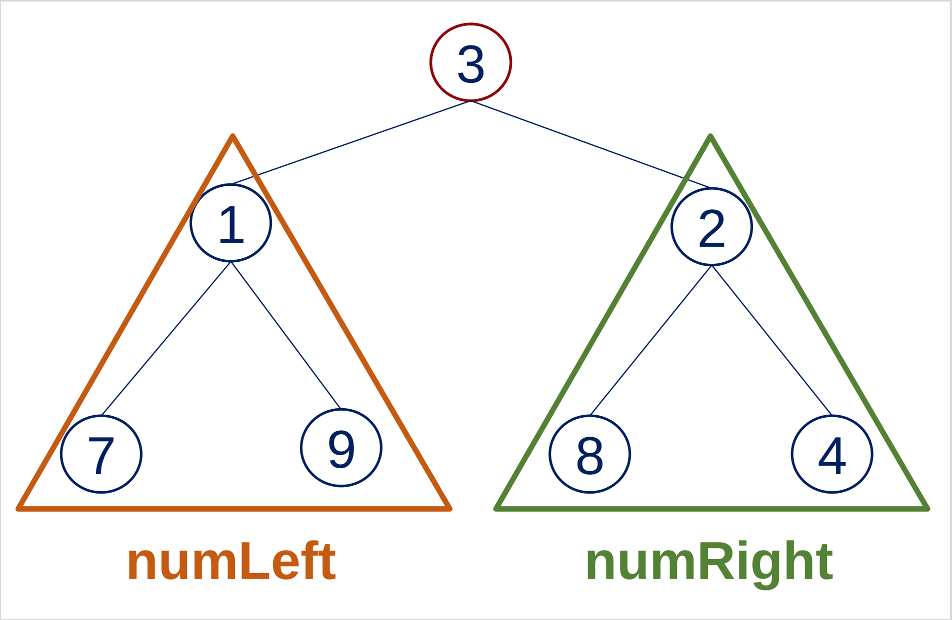
Operations in CBT (Cont'd)

`numOfNodes(t)`

Decompose the problem

- The **root node**
- The **left subtree**
- The **right subtree**

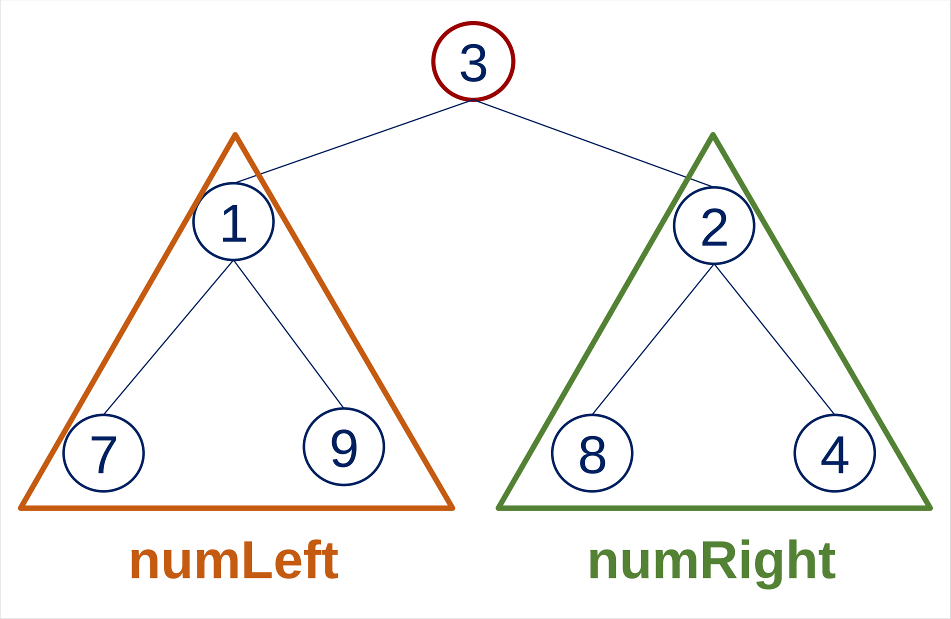
Result = **numLeft** + **1** + **numRight**



Operations in CBT (Cont'd)

numOfNodes(t)

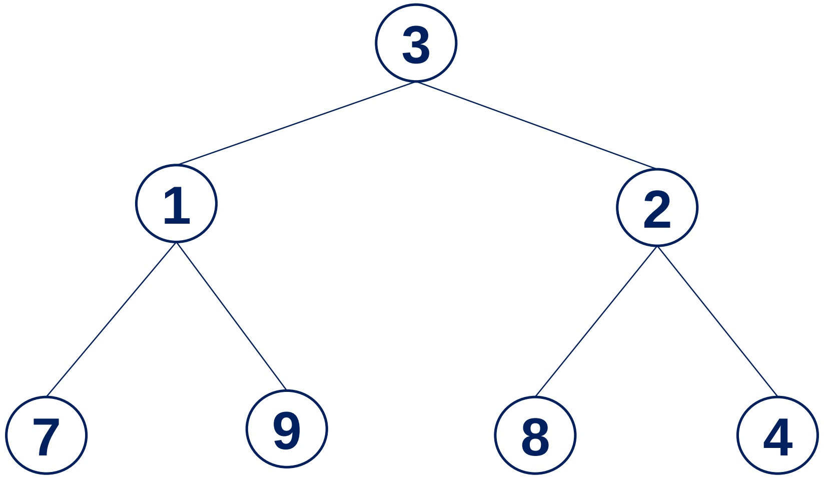
```
def numOfNodes(t):
    if len(t) == 1:
        return 1;

    else:
        numLeft = numOfNodes(t[0])

        numRight = numOfNodes(t[2])

        return ( numLeft + numRight + 1 )
```



Operations in CBT (Cont'd)

```
sumNodes(t)
```

```
tree = [[[7], 1, [9]], 3, [[8], 2, [4]]]

print("sum of Nodes: ”, end=‘’)
print( sumNodes(tree) )
```



sum of Nodes: 34

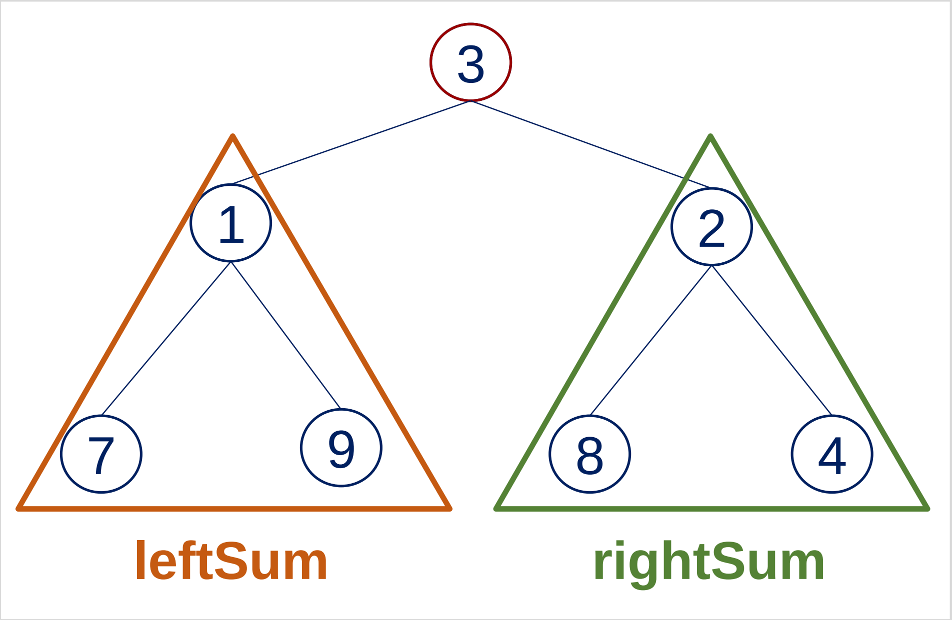
Operations in CBT (Cont'd)

`sumNodes(t)`

Decompose the problem

- The **root node**
- The **left subtree**
- The **right subtree**

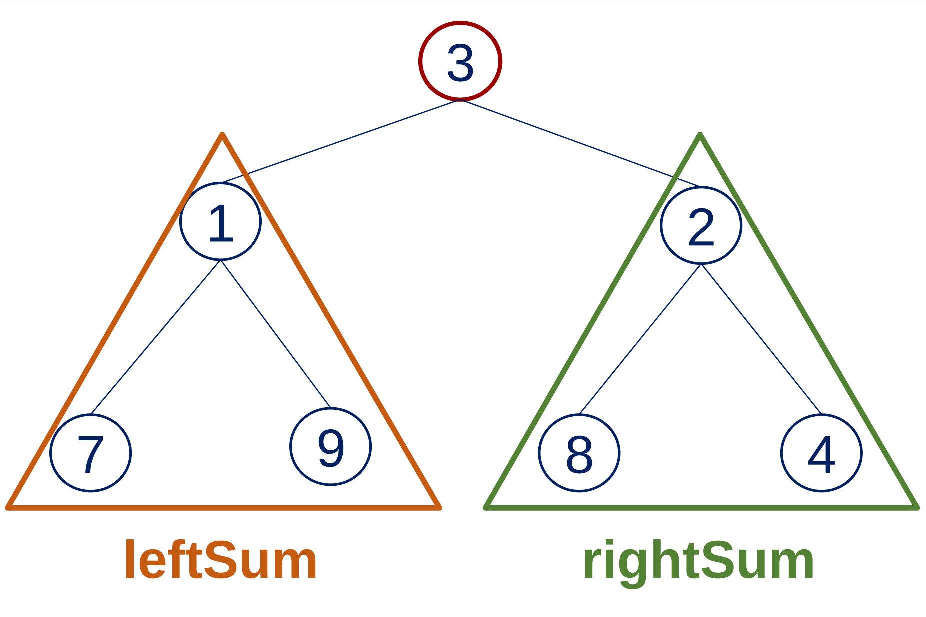
Result = leftSum + 3 + rightSum



Operations in CBT (Cont'd)

sumNodes(t)

```
def sumNodes(t):
    if len(t) == 1:
        return t[0];

    else:
        leftSum = sumNodes(t[0])

        rightSum = sumNodes(t[2])

        return ( t[1] + leftSum + rightSum)
```

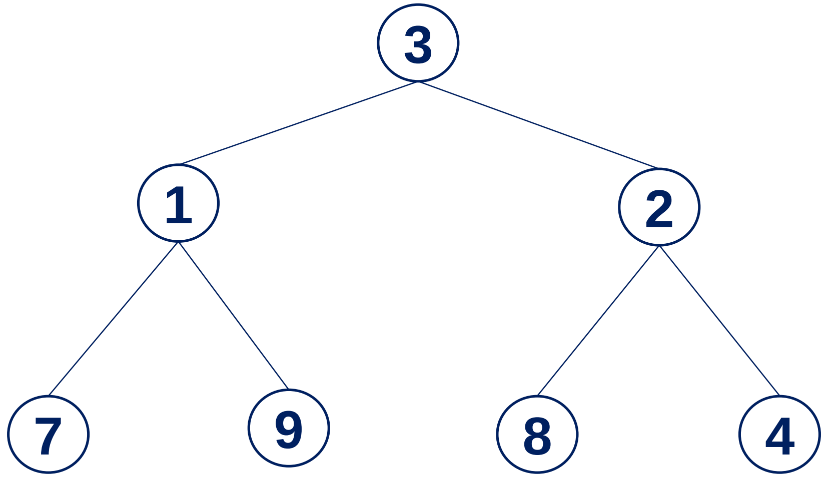


Operations in CBT (Cont’d)

NANYANG TECHNOLOGICAL UNIVERSITY SINGAPORE

maxNode(t)

```
tree = [[[7], 1, [9]], 3, [[8], 2, [4]]]

print("max of Nodes: ”, end=‘’)
print( maxNodes(tree) )
```



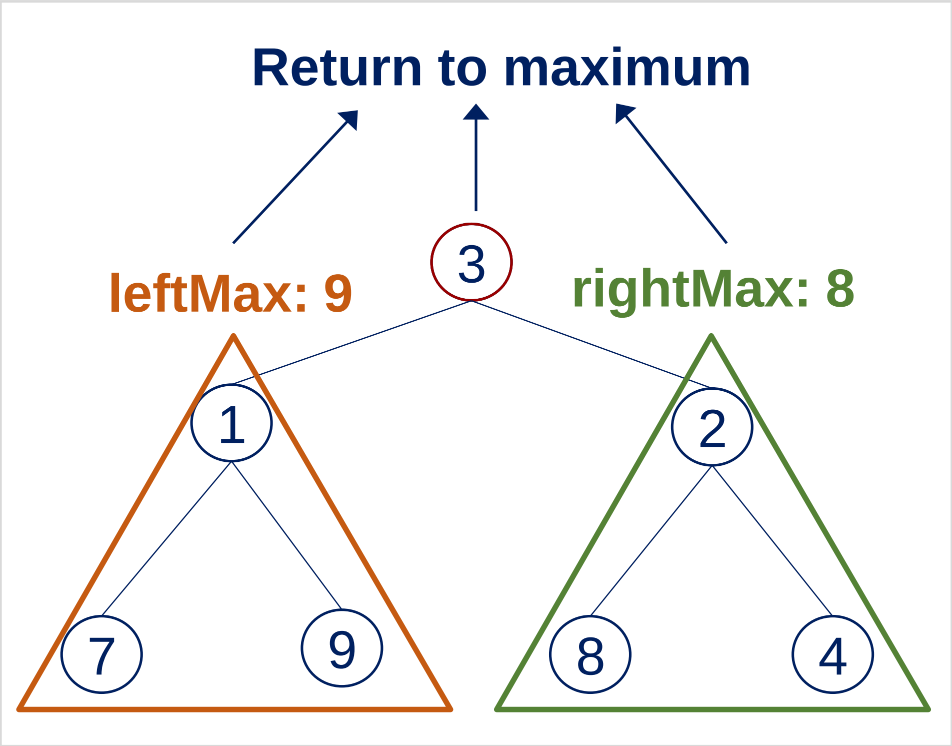
max of Nodes: 9

Operations in CBT (Cont'd)

`maxNode(t)`

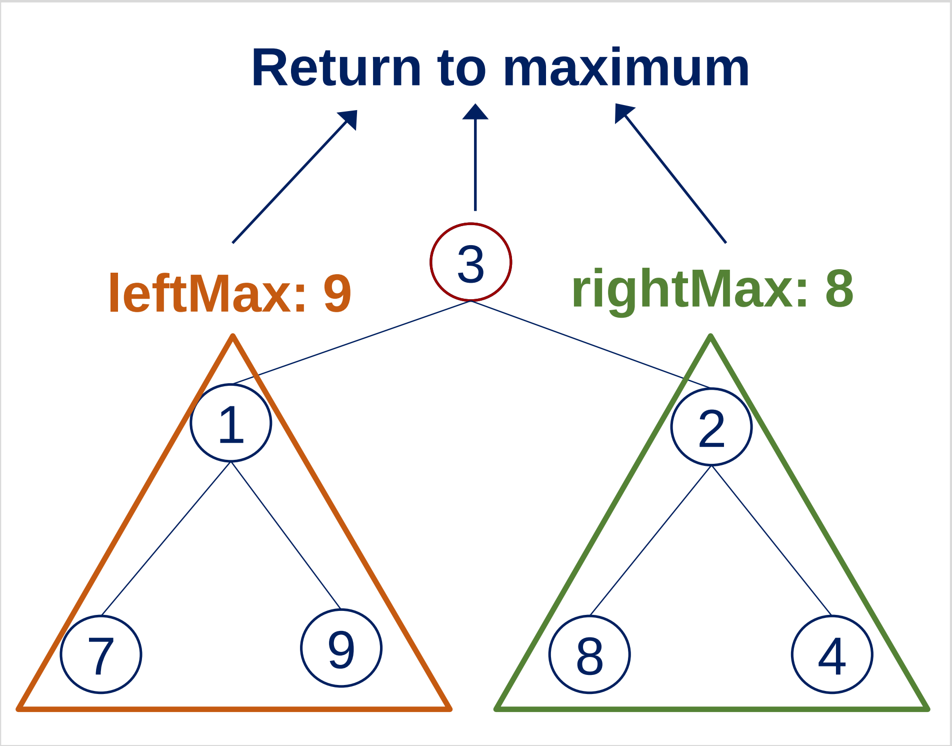
Decompose the problem

- The **root node**
- The **left subtree**
- The **right subtree**



Operations in CBT (Cont'd)

maxNode(t)

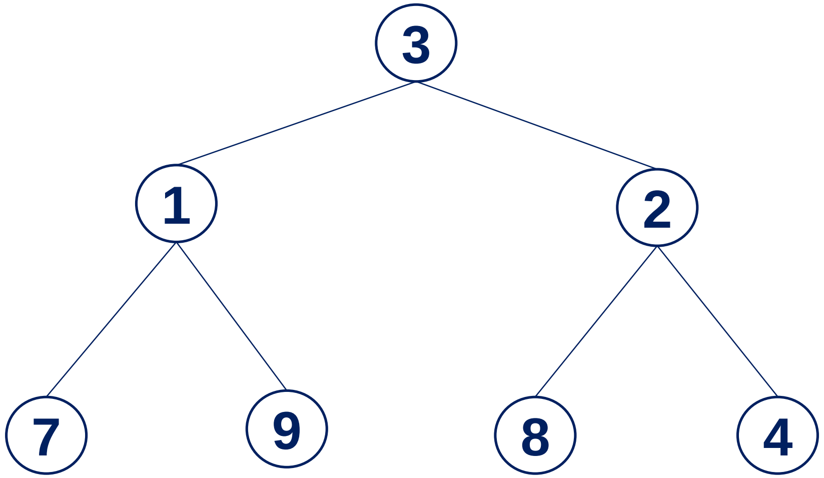



```python
def maxNode(t):
    if len(t) == 1:
        return t[0]
    else:
        leftMax = maxNode(t[0])
        rightMax = maxNode(t[2])

        maxValue = t[1]
        if leftMax > maxValue:
            maxValue = leftMax

        if rightMax > maxValue:
            maxValue = rightMax

        return maxValue
```

Operations in CBT (Cont'd)

minNode(t)

```
tree = [[[7], 1, [9]], 3, [[8], 2, [4]]]

print("min of Nodes: ", end='')
print( minNodes(tree) )
```



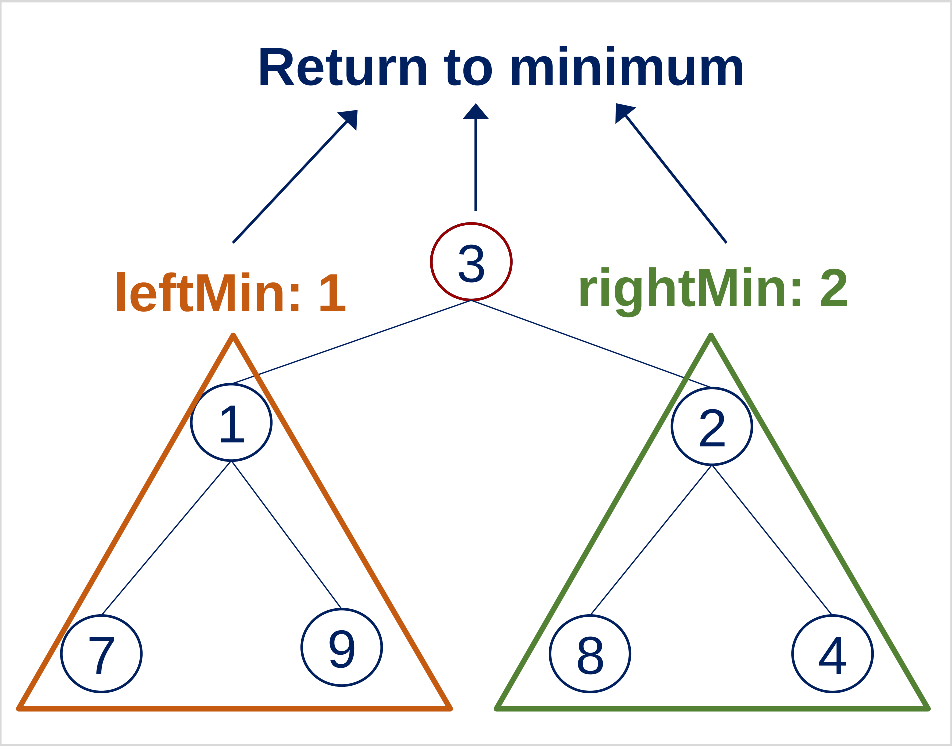
min of Nodes: 1

Operations in CBT (Cont'd)

`minNode(t)`

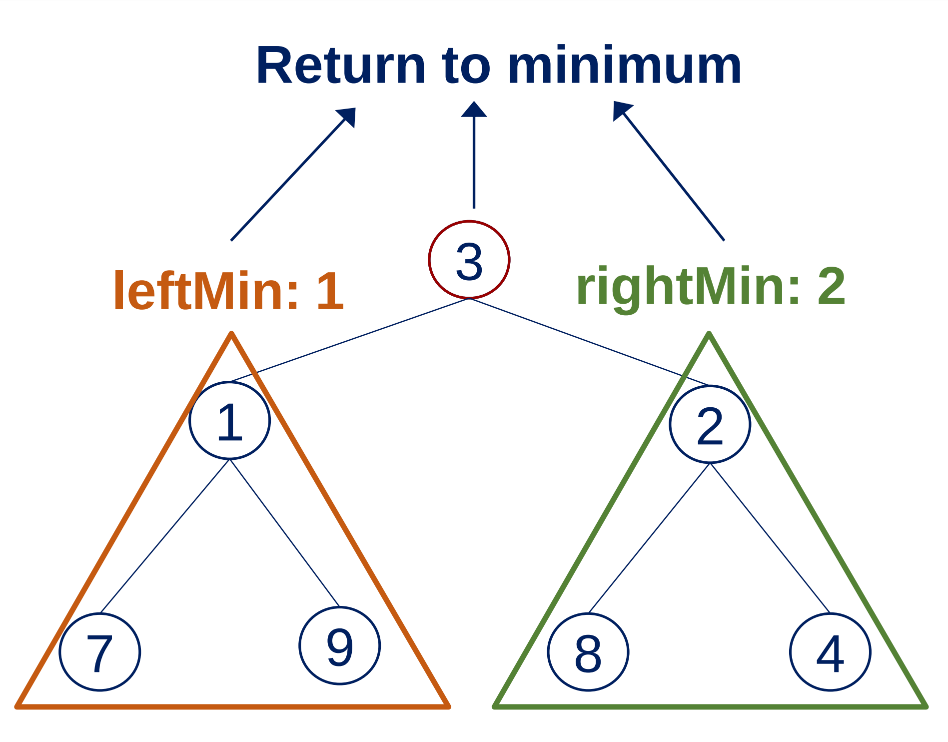
Decompose the problem

- The **root node**
- The **left subtree**
- The **right subtree**



Operations in CBT (Cont'd)

`minNode(t)`

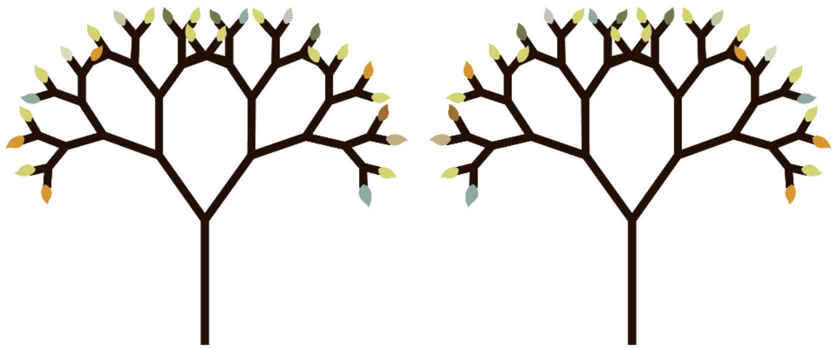


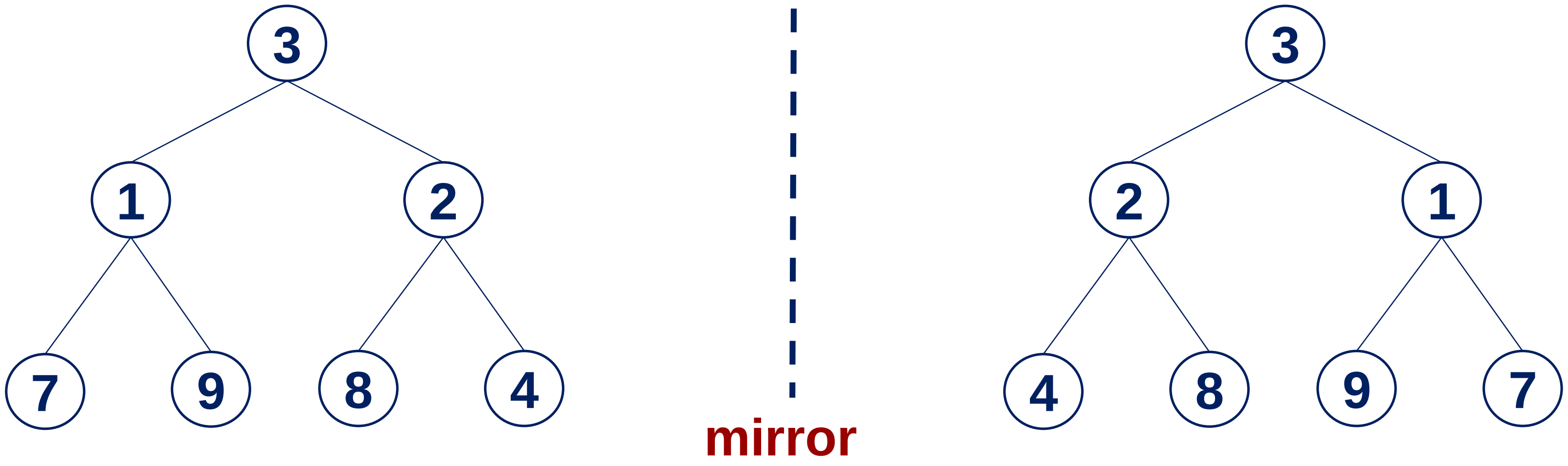

```
def maxNode(t):
    if len(t) == 1:
        return t[0]
    else:
        minValue = t[1]
        leftMin = minNode(t[0])
        rightMin = minNode(t[2])

        if leftMin < minValue:
            minValue = leftMin

        if rightMin < minValue:
            minValue = rightMin

        return minValue
```

Operations in CBT (Cont'd)

```
mirror(t)
```

```
tree = [[[7], 1, [9]], 3, [[8], 2, [4]]]

mirrortree = mirror(tree)
```



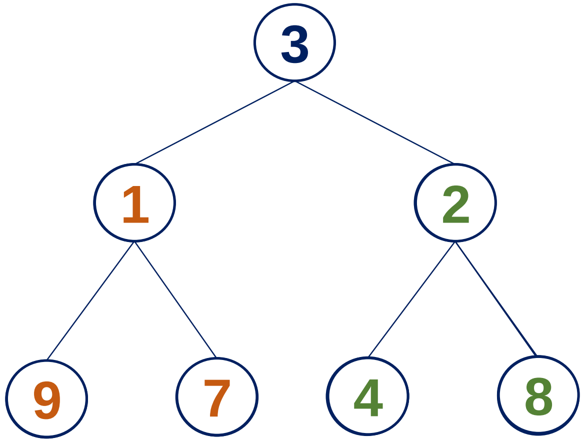


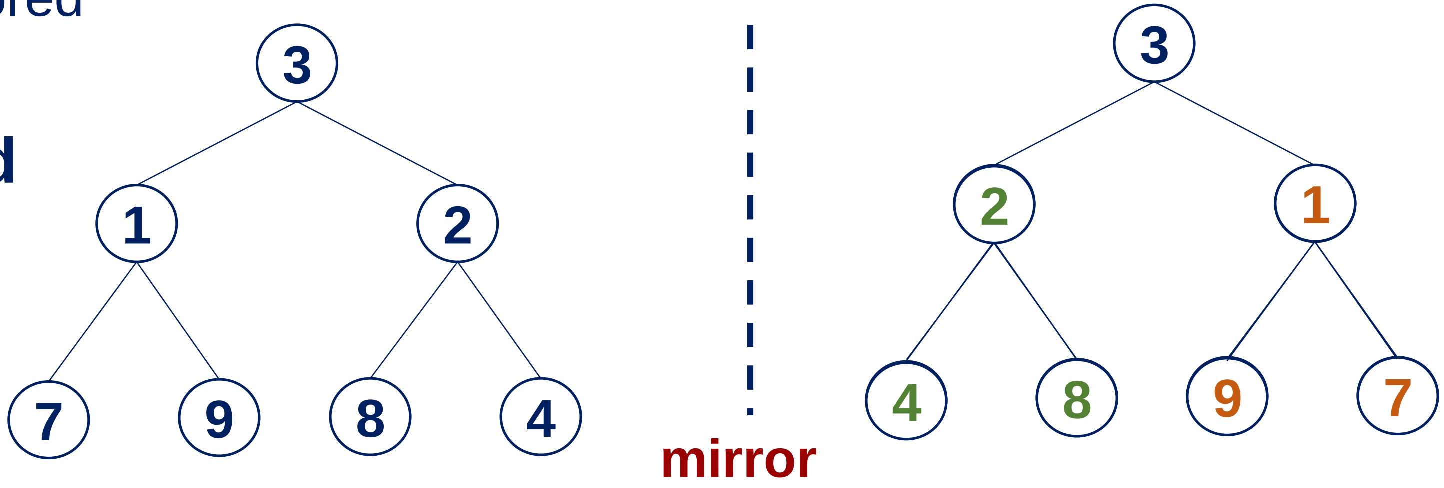
Operations in CBT (Cont'd)

`mirror(t)`

- **Decompose the problem:**
  - The **left subtree**
    - Make the left subtree mirrored
  - The **right subtree**
    - Make the right subtree mirrored

- **Switch the mirrored left and right subtree**

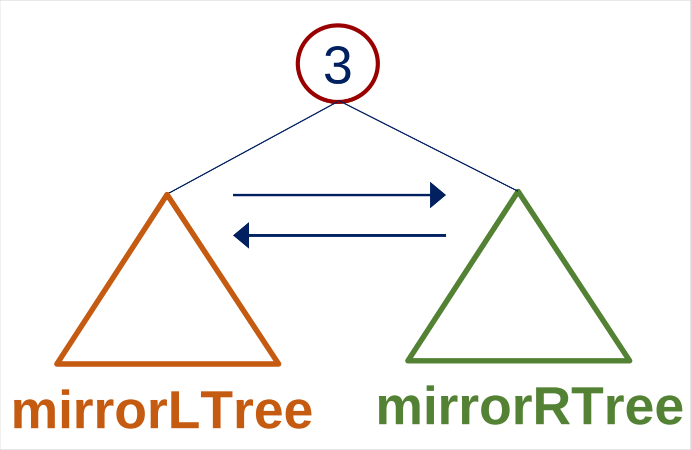




Operations in CBT (Cont'd)

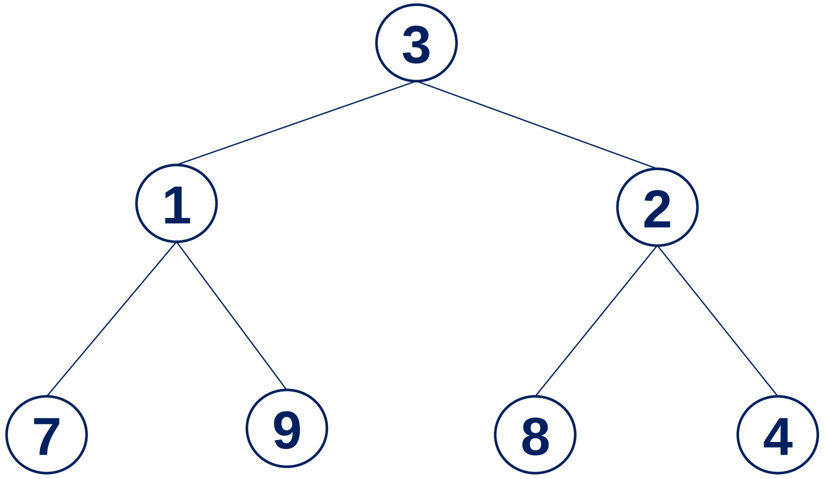
mirror(t)

```
def mirror(t):
    if len(t) == 1:
        return t

    else:
        parent = t[1]
        mirrorLTree = mirror(t[0])
        mirrorRTree = mirror(t[2])

        return [ mirrorRTree, parent, mirrorLTree ]
```



Print Out a CBT

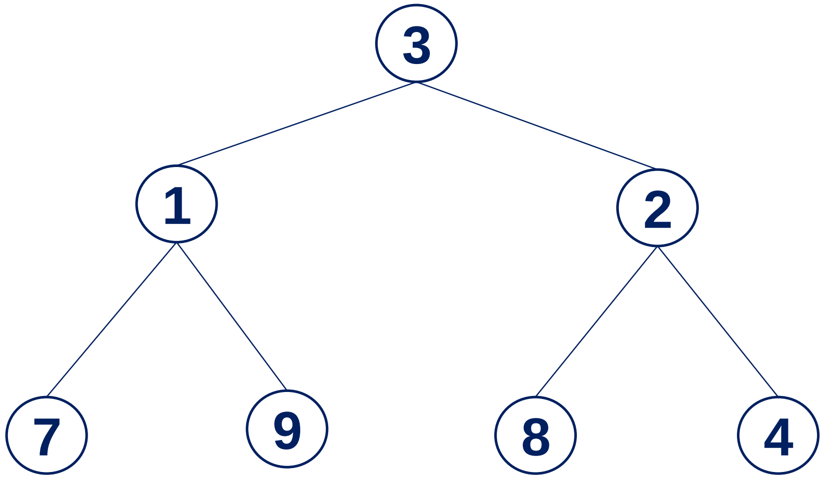
```
tree = [[[7], 1, [9]], 3, [[8], 2, [4]]]

printTree(tree, 0)
```



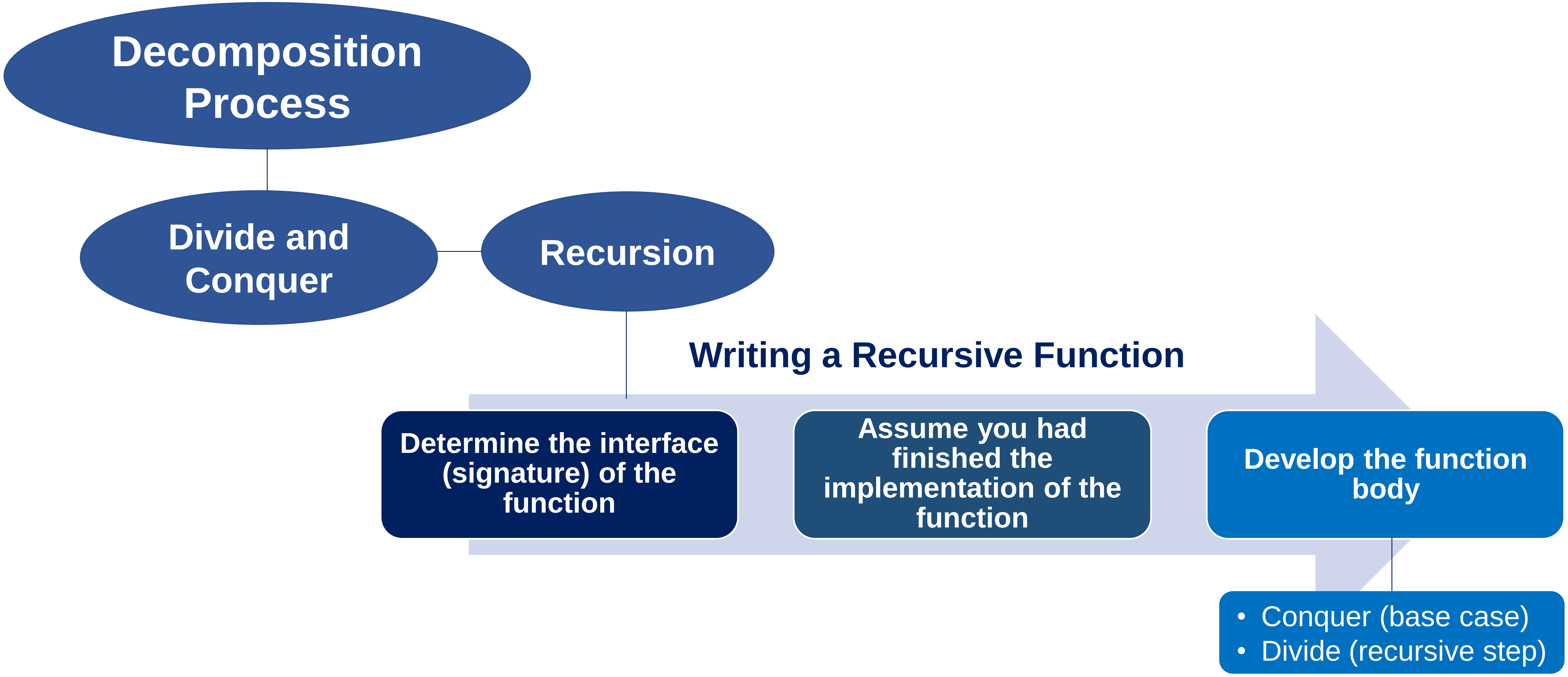

```
        4
    2
        8
3
        9
    1
        7
```

Print Out a CBT

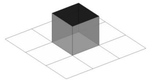
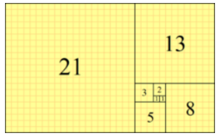
```
def printTree(t, level):
    if len(t) == 1:
        print("  " * level, end="")
        print(t[0])

    else:
        printTree(t[2], level + 1)

        print("  " * level, end="")
        print(t[1])

        printTree(t[0], level + 1)
```



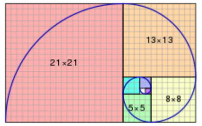
Summary


Decomposition Process
Divide and Conquer
Recursion
Writing a Recursive Function
Determine the interface (signature) of the function
Assume you had finished the implementation of the function
Develop the function body
• Conquer (base case)
• Divide (recursive step)

References for Image

No.	Slide No.	Image	Reference
1	5		Online Image. Retrieved April 24, 2018 from https://www.flickr.com/photos/epublicist/8718123610.
2	6		By Guillaume Jacquenot - Own work, CC BY-SA 3.0,retrieved April 24, 2018 from https://commons.wikimedia.org/w/index.php?curid=11678451.
3	7, 9, 21, 24, 27, 30, 33, 35		Python Logo [Online Image]. Retrieved April 24, 2018 from https://pixabay.com/en/language-logo-python-2024210/.
4	9, 10		Play Button [Online Image]. Retrieved April 24, 2018 from https://pixabay.com/en/play-button-round-blue-glossy-151523/.
5	11		By 克勞棣 - Own work, CC BY-SA 4.0,retrieved April 24, 2018 from https://commons.wikimedia.org/w/index.php?curid=38708516.

References for Images

No.	Slide No.	Image	Reference
6	11		By Jahobr - Own work, CC0,retrieved April 24, 2018 from https://commons.wikimedia.org/w/index.php?curid=58460223,
7	14		Question problem [Online Image]. Retrieved April 24, 2018 from https://pixabay.com/en/question-problem-think-thinking-622164/,