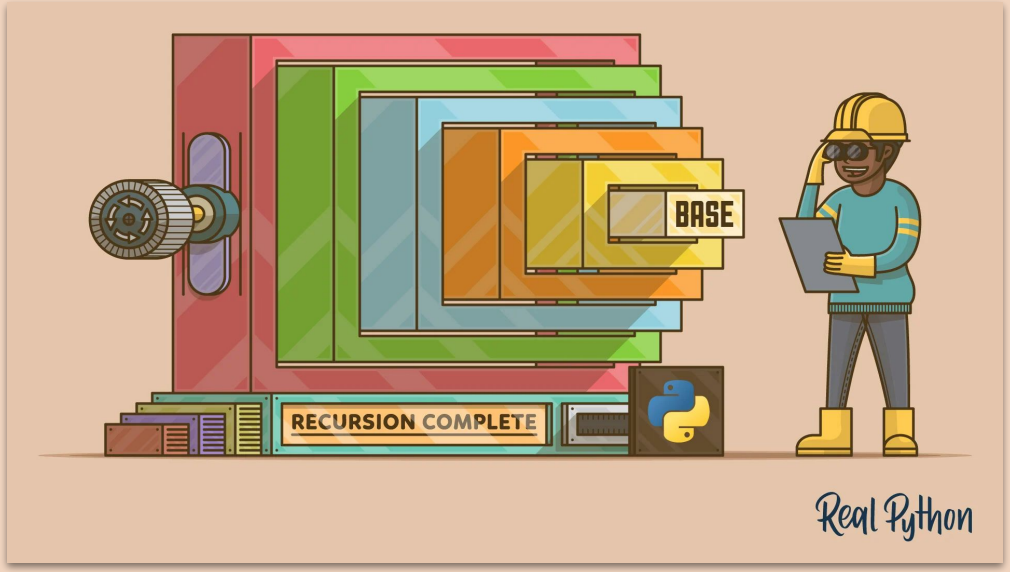
Week 04

Recursion
Binary Trees
Search Trees



Ivanovitch Silva
ivanovitch.silva@ufrn.br

Data Structure Review

Recursion

The Importance of Studying Recursion

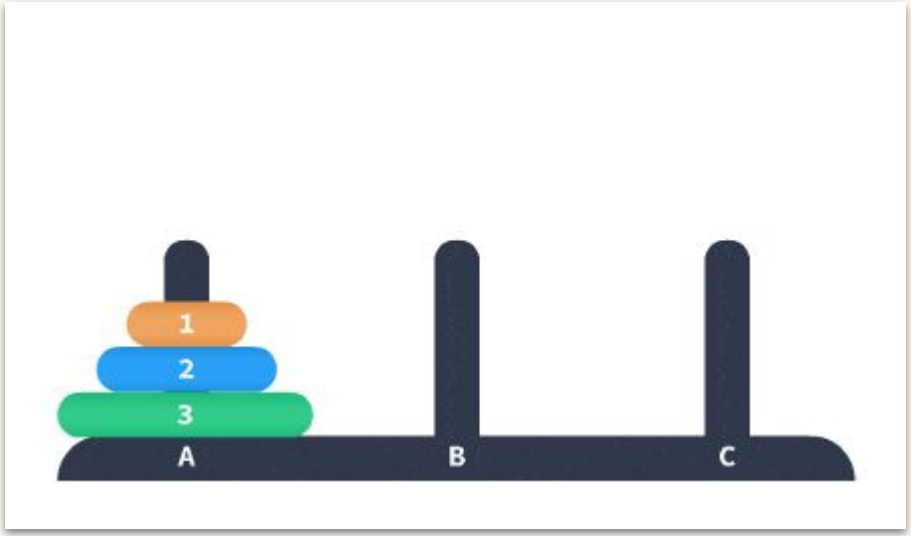
```
def iterative_sum(values):
    total = 0
    for value in values:
        total += value
    return total

result = iterative_sum([1, 2, 3, 4, 5,
                        6, 7, 8, 9, 10])
```

```
def recursive_sum(values):
    # Base case: the list is empty
    if not values:
        return 0
    # General case: the list is not empty
    return values[0] + recursive_sum(values[1:])

result = recursive_sum([1, 2, 3, 4, 5,
                        6, 7, 8, 9, 10])
```

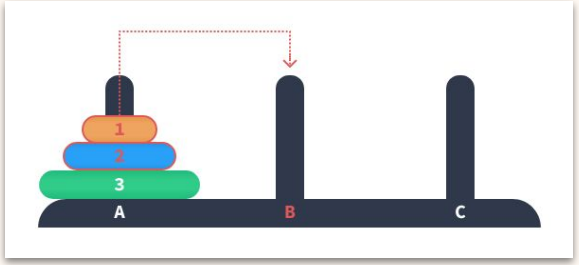
#stack_overflow

Some Problems are Naturally Recursive

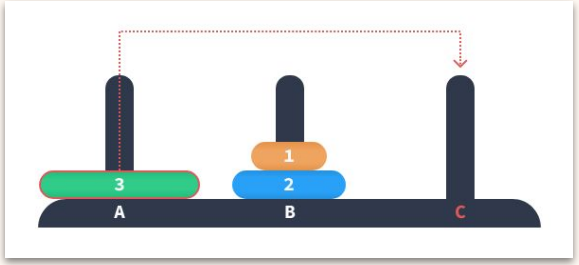
hanoi(N,origin, temp, dest)



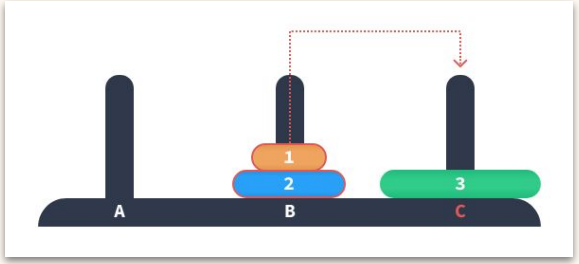
hanoi(N - 1, A, C, B)



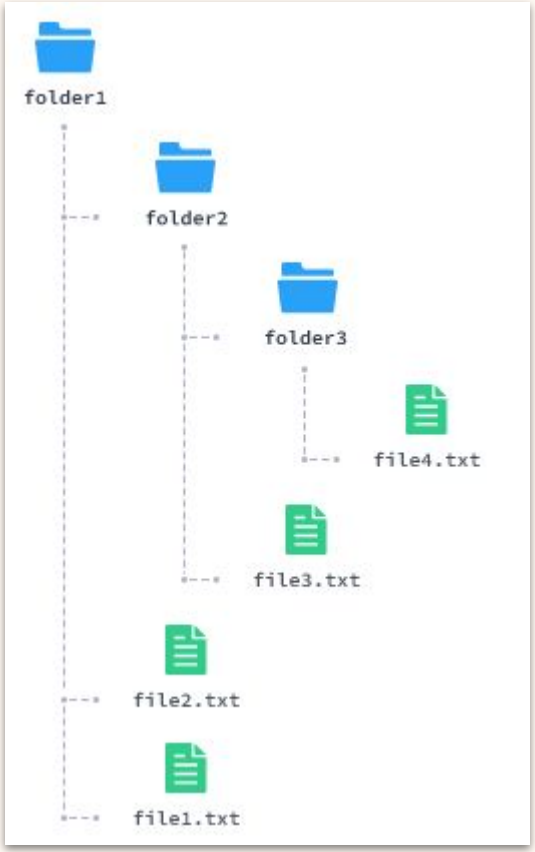
hanoi(1, A, - , C)



hanoi(N - 1, B, A, C)

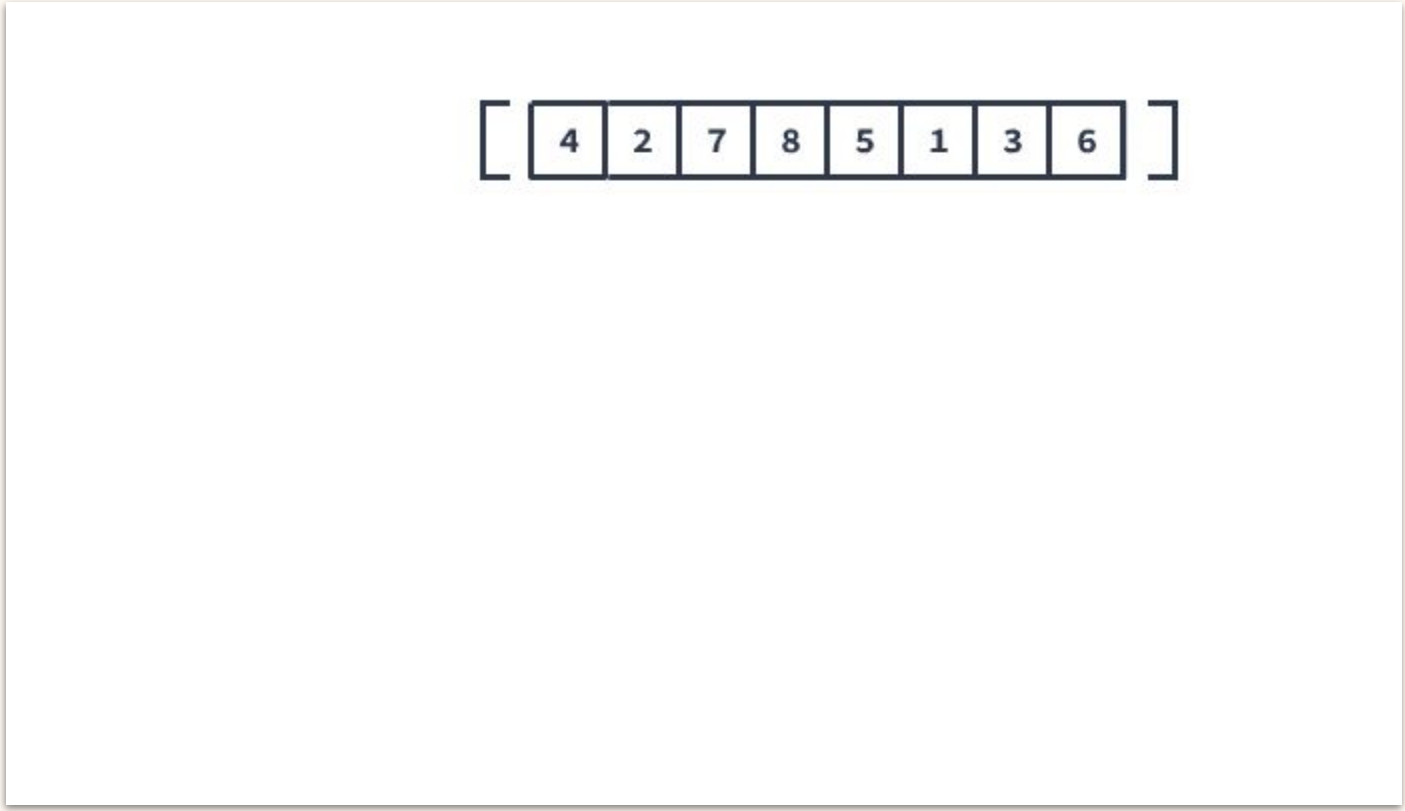


Some Problems are Naturally Recursive



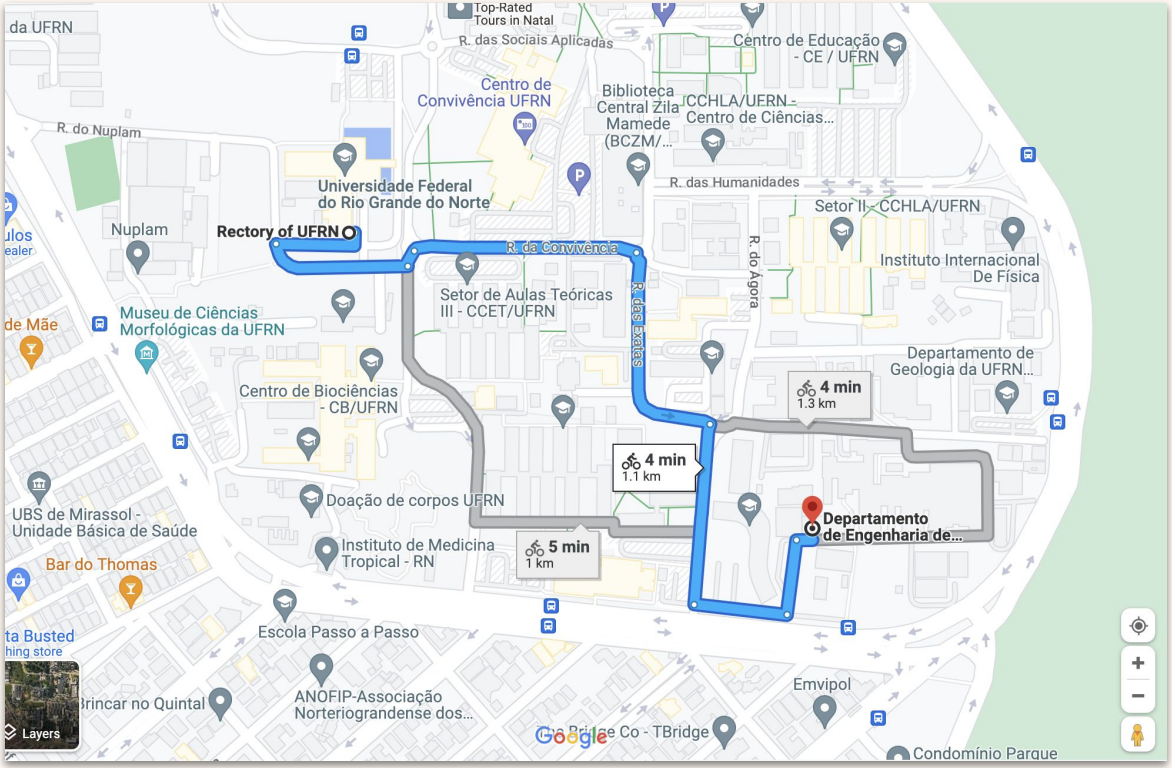

```python
import os
def list_files(current_path):
    #Base case
    if not os.path.isdir(current_path):
        print(current_path)
    else:
        # General case
        for name in os.listdir(current_path):
            file_path = os.path.join(current_path, name)
            list_files(file_path)

list_files("folder1")
```

Some Problems are Naturally Recursive



merge-sort

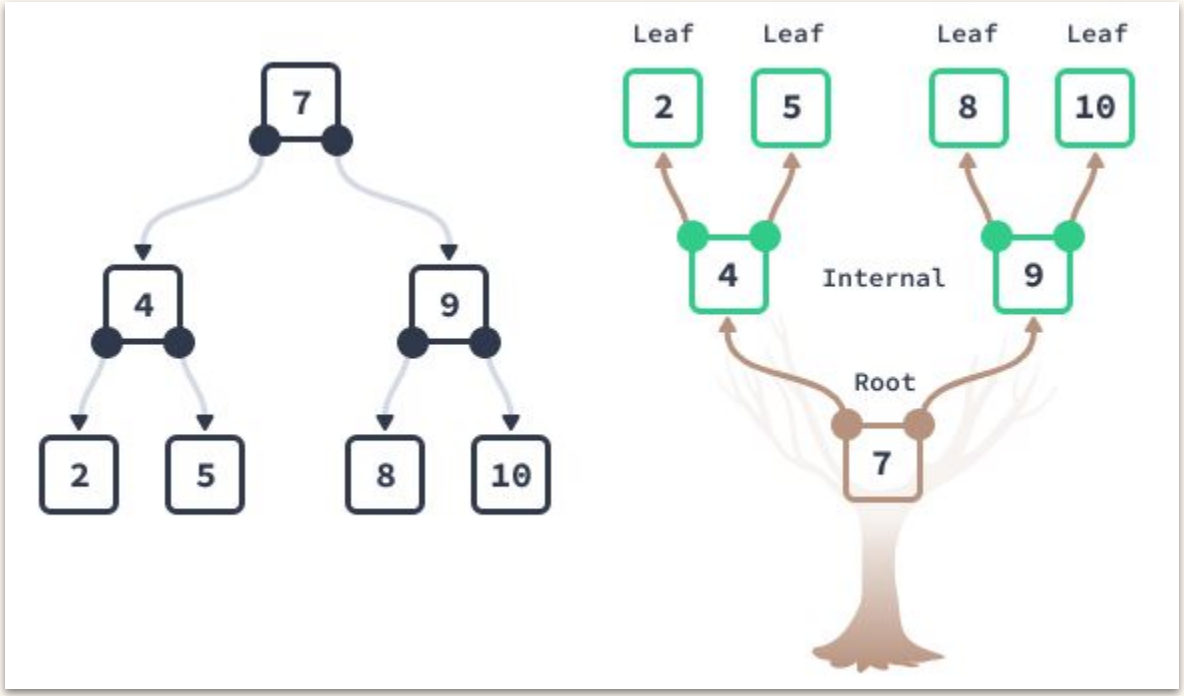
Some Problems are Naturally Recursive



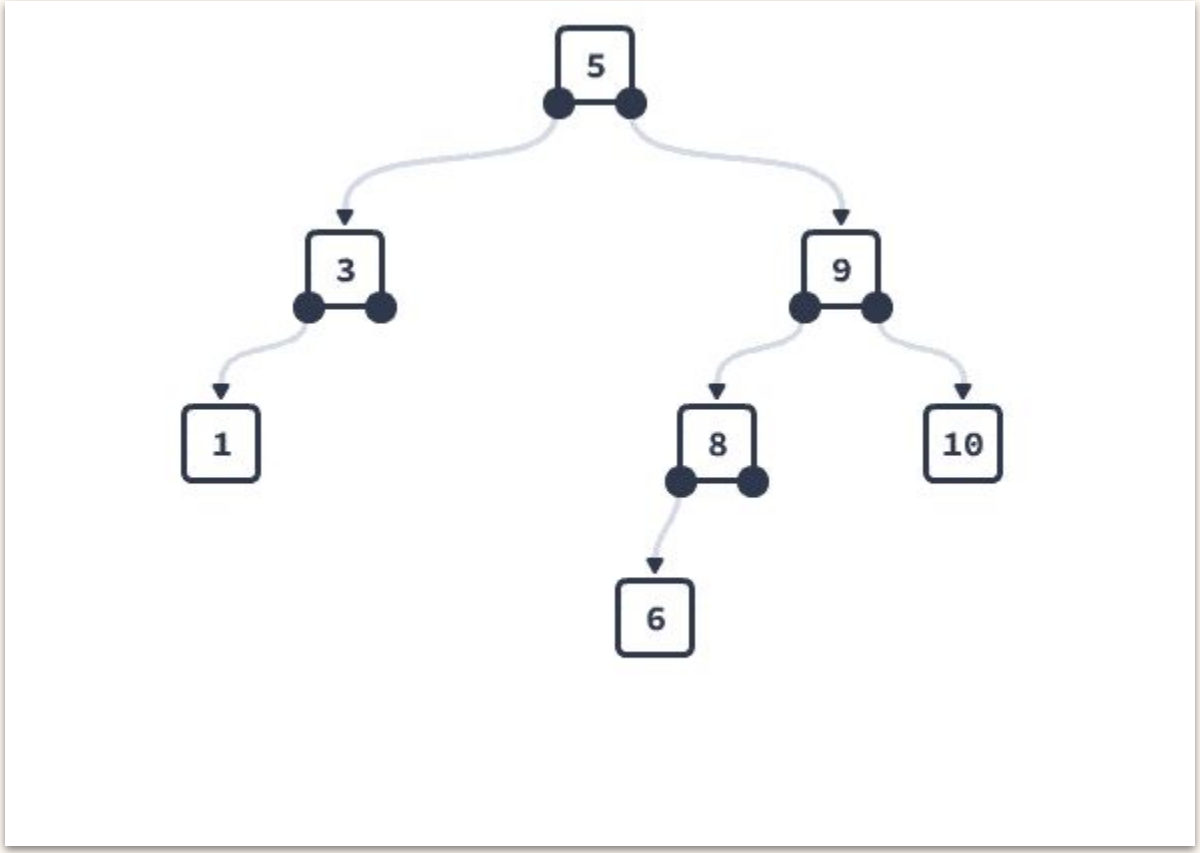
Data Structure Review

Binary Tree

Studying binary trees is important for several reasons

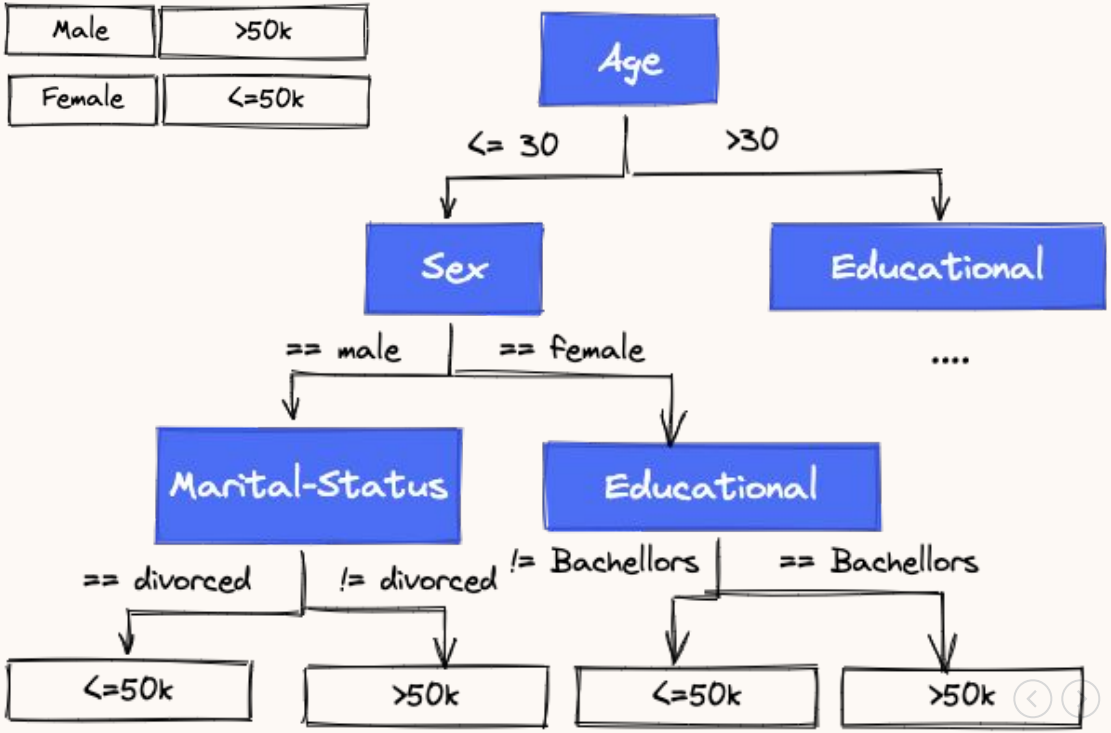


Efficient search, insertion, and deletion: Binary Search Tree (BST)



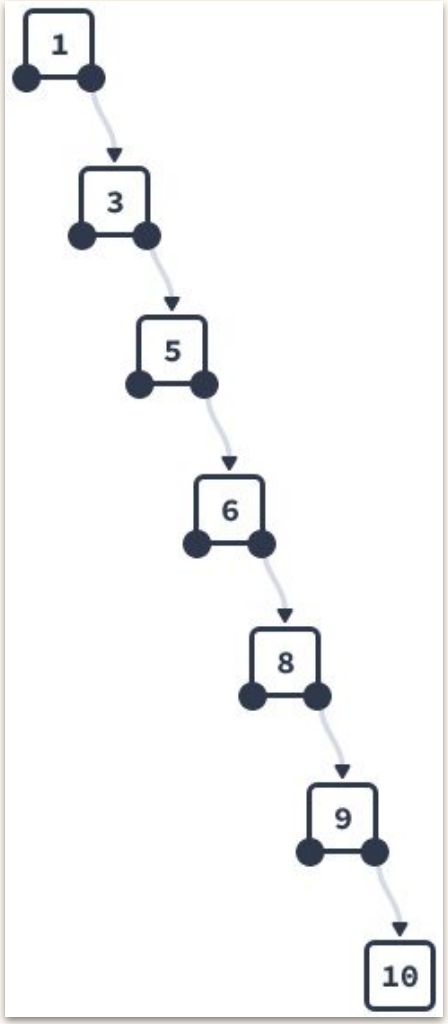
Age	Marital-Status	Educational	Sex	High-Income
28	Never-Married	Bachelors	Female	<=50k
46	Never-Married	Assoc-acdm	Female	<=50k
35	Married-civ-spouse	Some-college	Male	<=50k
27	Married-civ-spouse	Bachelors	Male	>50k
59	Divorced	Some-college	Female	<=50k

Decision Tree (classification)



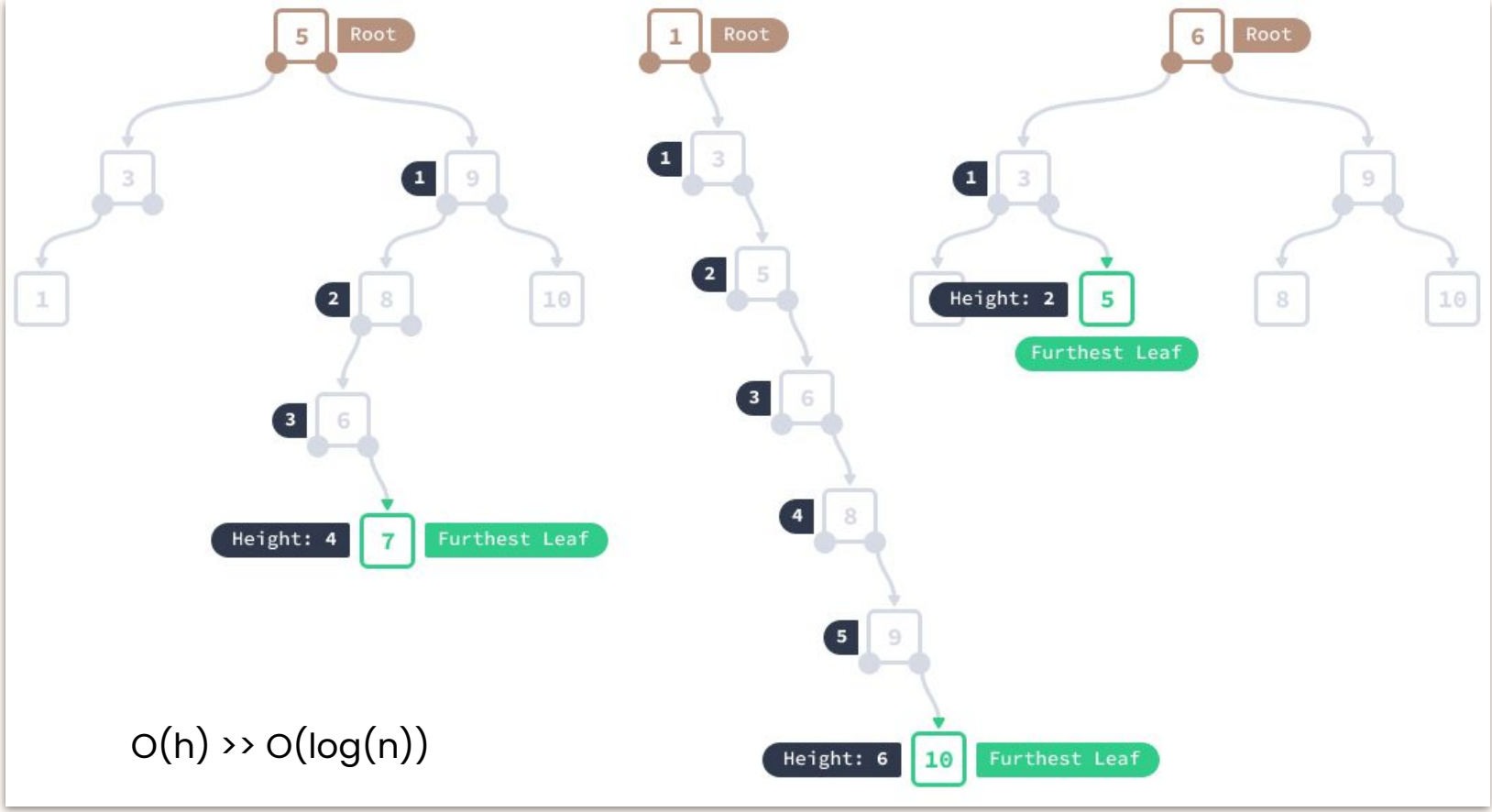
ps axj

```
(anaconda3) ❯ ps axj
                                                                  ~
USER               PID  PPID  PGID    SESS JOBC STAT   TT       TIME COMMAND
root                 1     0     1       0    0 Ss     ??   11:12.90 /sbin/launchd
root                99     1    99       0    0 Ss     ??    4:50.32 /usr/libexec/logd
root               101     1   101       0    0 Ss     ??    0:12.63 /usr/libexec/UserEventAgent (System)
root               103     1   103       0    0 Ss     ??    0:03.10 /System/Library/PrivateFrameworks/Uninstall.framewo
root               104     1   104       0    0 Ss     ??    3:10.30 /System/Library/Frameworks/CoreServices.framework/V
fseventsd
root               105     1   105       0    0 Ss     ??    0:20.92 /System/Library/PrivateFrameworks/MediaRemote.frame
root               108     1   108       0    0 Ss     ??    2:42.44 /usr/sbin/systemstats --daemon
root               110     1   110       0    0 Ss     ??    1:04.08 /usr/libexec/configd
root               112     1   112       0    0 Ss     ??    1:13.68 /System/Library/CoreServices/powerd.bundle/powerd
root               113     1   113       0    0 Ss     ??    0:00.02 /usr/libexec/IOMFB_bics_daemon
root               118     1   118       0    0 Ss     ??    0:00.29 /usr/libexec/remoted
root               123     1   123       0    0 Ss     ??    0:03.43 /usr/libexec/watchdogd
root               127     1   127       0    0 Ss     ??    4:51.08 /System/Library/Frameworks/CoreServices.framework/F
root               129     1   129       0    0 Ss     ??    0:02.16 /usr/libexec/kernelmanagerd
root               130     1   130       0    0 Ss     ??    0:05.39 /usr/libexec/diskarbitrationd
root               134     1   134       0    0 Ss     ??    0:06.78 /usr/sbin/syslogd
root               137     1   137       0    0 Ss     ??    0:37.52 /usr/libexec/thermalmonitord
root               138     1   138       0    0 Ss     ??    3:44.73 /usr/libexec/opendirectoryd
```



What if we add the values in increasing order [1, 3, 5, 6, 8, 9, 10]?

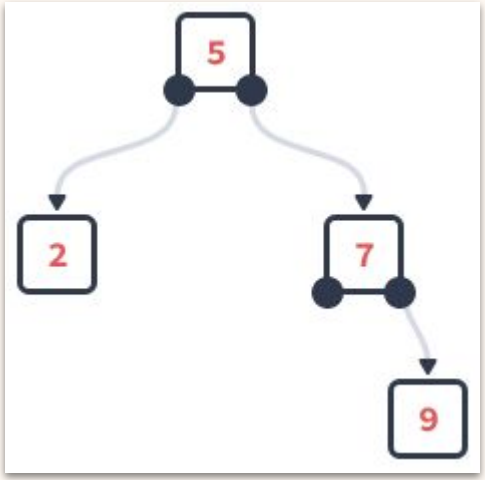
BST Complexity



O(h) >> O(log(n))

BST Implementation

```python
class Node:
    """
    A class representing a node in a binary search tree.

    Attributes:
    - value: the value of the node
    - left_child: the left child of the node
    - right_child: the right child of the node
    """

    def __init__(self, value):
        """
        Initializes a new instance of the Node class.

        Args:
        - value: the value of the node
        """
        self.value = value
        self.left_child = None
        self.right_child = None
```



```python
class BST:

    def __init__(self):
        """
        Initializes a new instance of the BST class.
        """
        self.root = None

    def add(self, value):
        """
        Adds a new node with the given value
Args:
        - value: the value of the node to add
        """
        if self.root is None:
            # The root does exist yet, create it
            self.root = Node(value)
        else:
            # Find the right place and insert value
            self._add_recursive(self.root, value)
```

```python
def _add_recursive(self, current_node, value):
 """
 A helper method to recursively traverse the tree and find the
correct position to add the new node.

 Args:
 - current_node: the current node to traverse
 - value: the value of the node to add
 """
 if value <= current_node.value:
     # Go to the left
     if current_node.left_child is None:
         current_node.left_child = Node(value)
     else:
         self._add_recursive(current_node.left_child, value)
 else:
     # Go to the right
     if current_node.right_child is None:
         current_node.right_child = Node(value)
     else:
         self._add_recursive(current_node.right_child, value)
```

```
def contains(self, value):
        """
        Checks whether a node with the given
value is present in the tree.

        Args:
        - value: the value to search for

        Returns:
        - True if a node with the given value is
found, False otherwise
        """
        return self._contains(self.root, value)
```

```
    def _contains(self, current_node, value):
        """
        A helper method to recursively traverse the tree and find
the node with the given value.

        Args:
        - current_node: the current node to traverse
        - value: the value to search for

        Returns:
        - True if a node with the given value is found, False
otherwise
        """
        if current_node is None:
            return False
        if current_node.value == value:
            return True
        if value < current_node.value:
            return self._contains(current_node.left_child, value)
        return self._contains(current_node.right_child, value)
```

DATAQUEST Catalog Resources Plans For Teams

Recursion and Trees for Data Engineering [8 lessons]

Overview of Recursion · 2h
Lesson Objectives

Introduction to Binary Trees · 1h
Lesson Objectives

Working with Binary Search Trees · 1h
Lesson Objectives

Implementing a Binary Heap · 1h
Lesson Objectives

Performance Boosts of Using a B-Tree · 1h
Lesson Objectives

Implement a B-Tree data structure · 1h
Lesson Objectives

Implementing a CSV Index with a B-Tree · 1h
Lesson Objectives

Guided Project: Implementing a Key-Value Database · 1h
Lesson Objectives

Challenge for Programming Interviews