CSC 148

Week 9

Sadia ‘Rain’ Sharmin

Classes begin 10 minutes after the hour

This Week's Cool Programmer(s) Feature: *Isabel Sieh*

Week 9

Binary Search Trees

SUBTOPICS:

- Introduction to BSTs
- Deletion from BST
- Delete Root Helper

Introduction to BSTs

Binary Search Trees

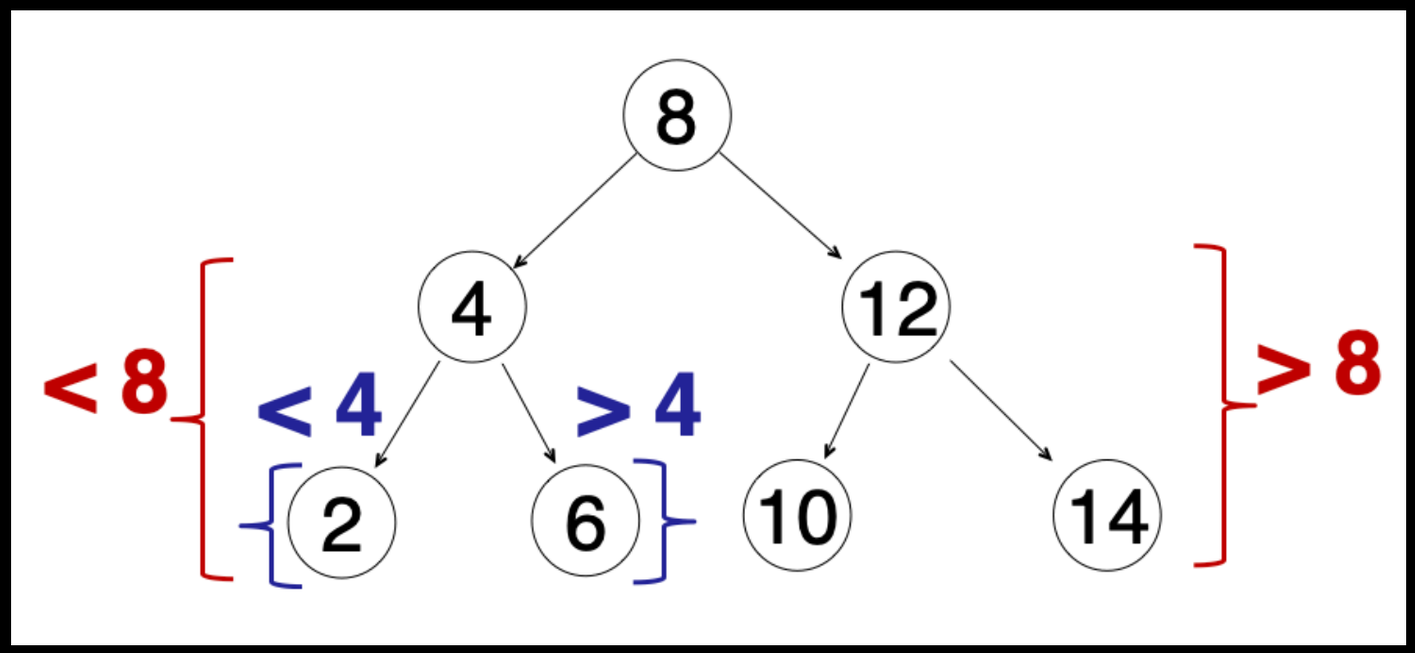
Definition

Add ordering conditions to a binary tree:

values are comparable

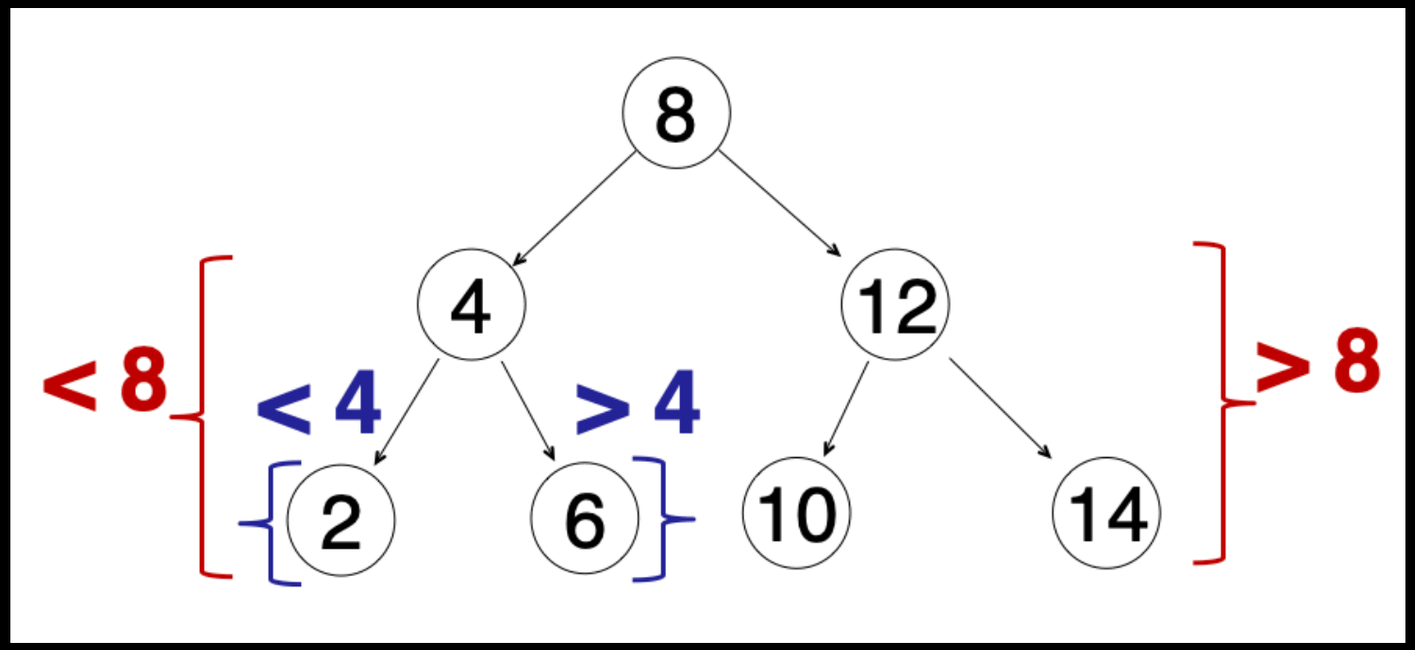
values in left subtree are less than the root's value

values in right subtree are more than the root's value



A "sorted" tree

Every item is
>= all items in its left subtree, and
<= all items in its right subtree.



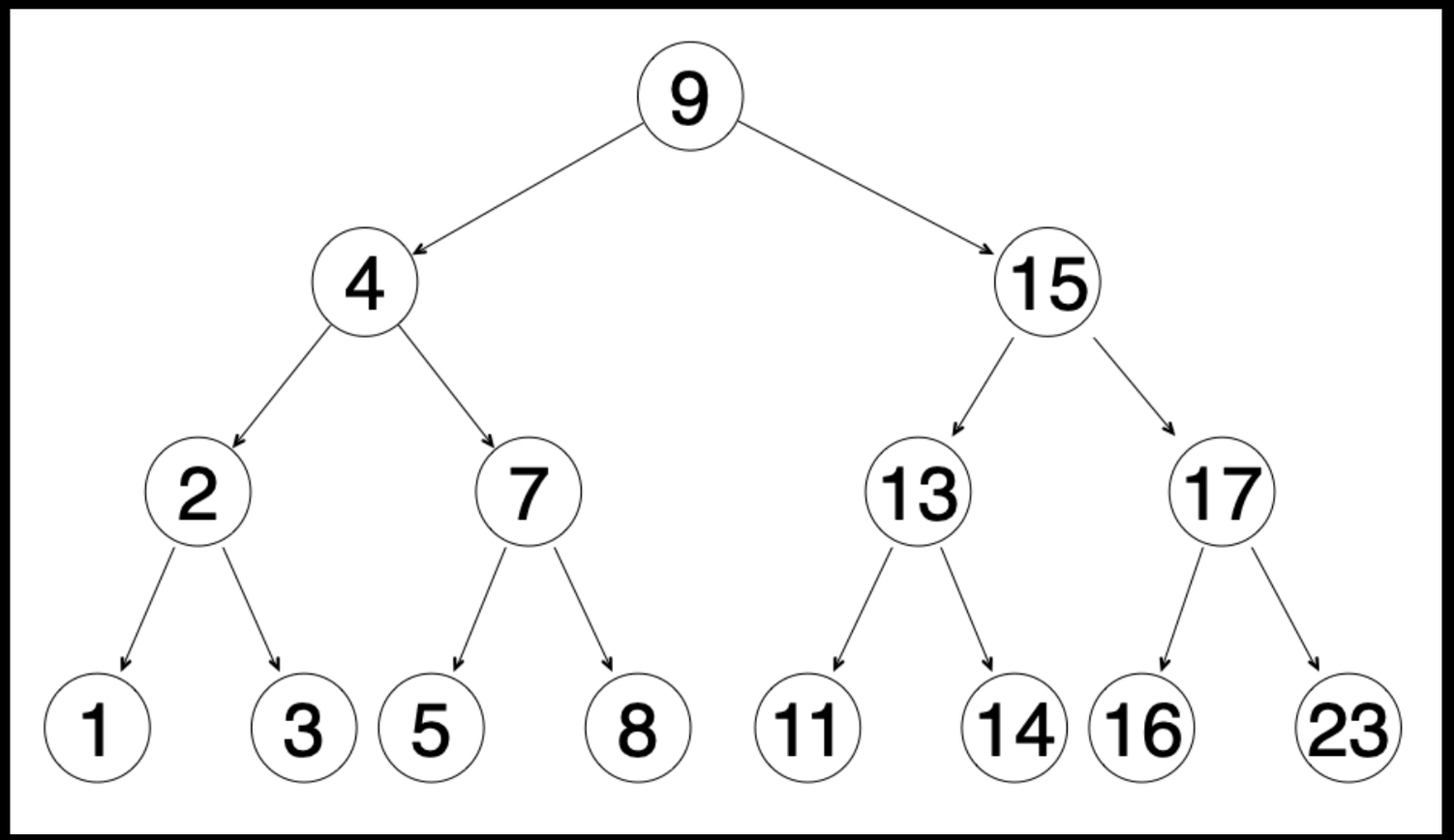
Searching in BST

Which nodes would we visit, to find out the following:

Find if value 5 is present...

Find if value 13 is present...

Find if value 12 is present...

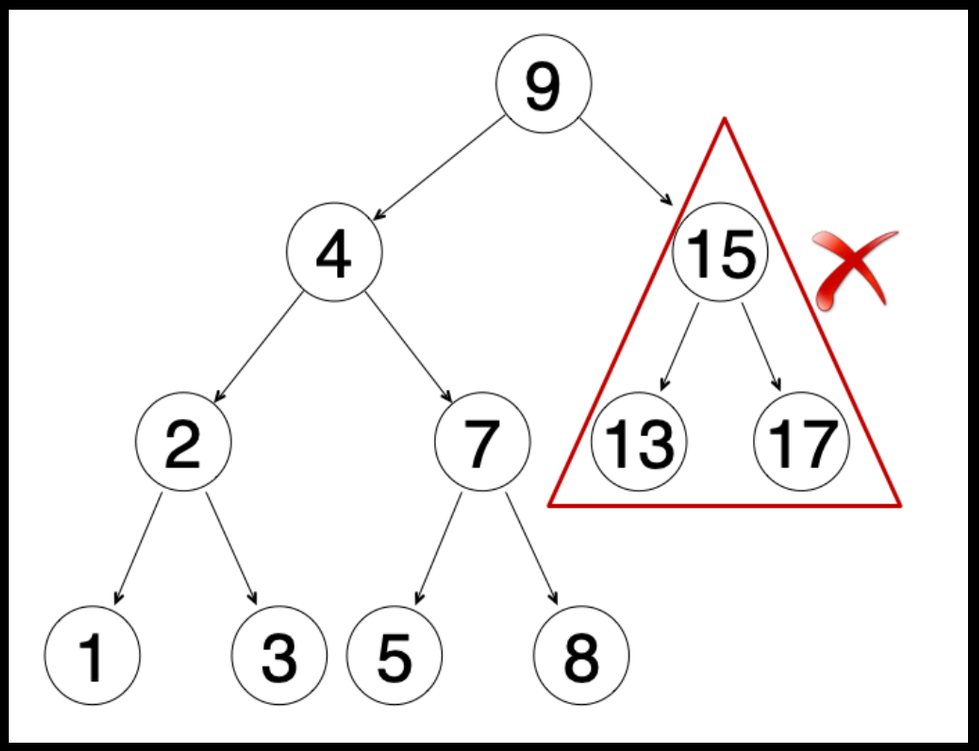


More structure => more efficiency

```
def __contains__(self, item: Any) -> bool:
  if self.is_empty():
    return False
  elif item == self._root:
    return True
  elif item < self._root:
    return self._left.__contains__(item)
  else:
    return self._right.__contains__(item)
```

Example:

Find if value 5 is present...



But not always!

```
def items(self) -> List:
    if self.is_empty():
        return []
    else:
        return (
            self._left.items() +
            [self.root] +
            self._right.items()
        )
```

Our Implementation

Instead of having a general list of subtrees like we did for our Tree implementation, we now have explicit attributes for left subtree and right subtree:

```
    # === Private Attributes ===
    # The item stored at the root of the tree, or None if the tree is empty.
    _root: Optional[Any]
    # The left subtree, or None if the tree is empty.
    _left: Optional[BinarySearchTree]
    # The right subtree, or None if the tree is empty.
    _right: Optional[BinarySearchTree]
```

Our Implementation

Similar to our Tree representation, we again do not allow the value None to be in the BinarySearchTree unless it is part of a valid empty tree representation.

We represent an empty BST with the following:

self._root = None

self._left = None

self._right = None

Note the representation invariants:

```
# === Representation Invariants ===
#  - If self._root is None, then so are self._left and self._right.
#    This represents an empty BST.
#  - If self._root is not None, then self._left and self._right
#    are BinarySearchTrees.
```

Our Implementation

So, how do we represent a leaf?

self._root will have its value, and

self._left and self._right will each be an empty BinarySearchTree

Representation invariants are key!

If self._root is not None, then self._left and self._right are BinarySearchTrees.

If you know that a BST is not empty, you **never** need to check if `self._left` or `self._right` are `None`.

You can call methods on them without an if-statement "guard".

Deletion from BST

BST Deletion

Delete operation

Hint: pull out the tree deletion worksheet from last week ...

Input:

An item that we wish to delete from the BST (if it exists!)

Outcomes:

if item found => remove the first occurrence of this item from the BST

This node gets “disconnected” / ”extracted” / “removed” from the BST

if item not found => do nothing

BST Deletion

Delete operation

```
def delete(self, item: Any) -> None:
    """Remove *one* occurrence of <item> from
    this BST.

    Do nothing if <item> is not in the BST.
    """
```

Notice the None -> we're not actually returning anything, but we will be mutating the tree as a result of the operation!

BST Deletion

Deletion Algorithm

Locate the item to delete, by traversing the tree

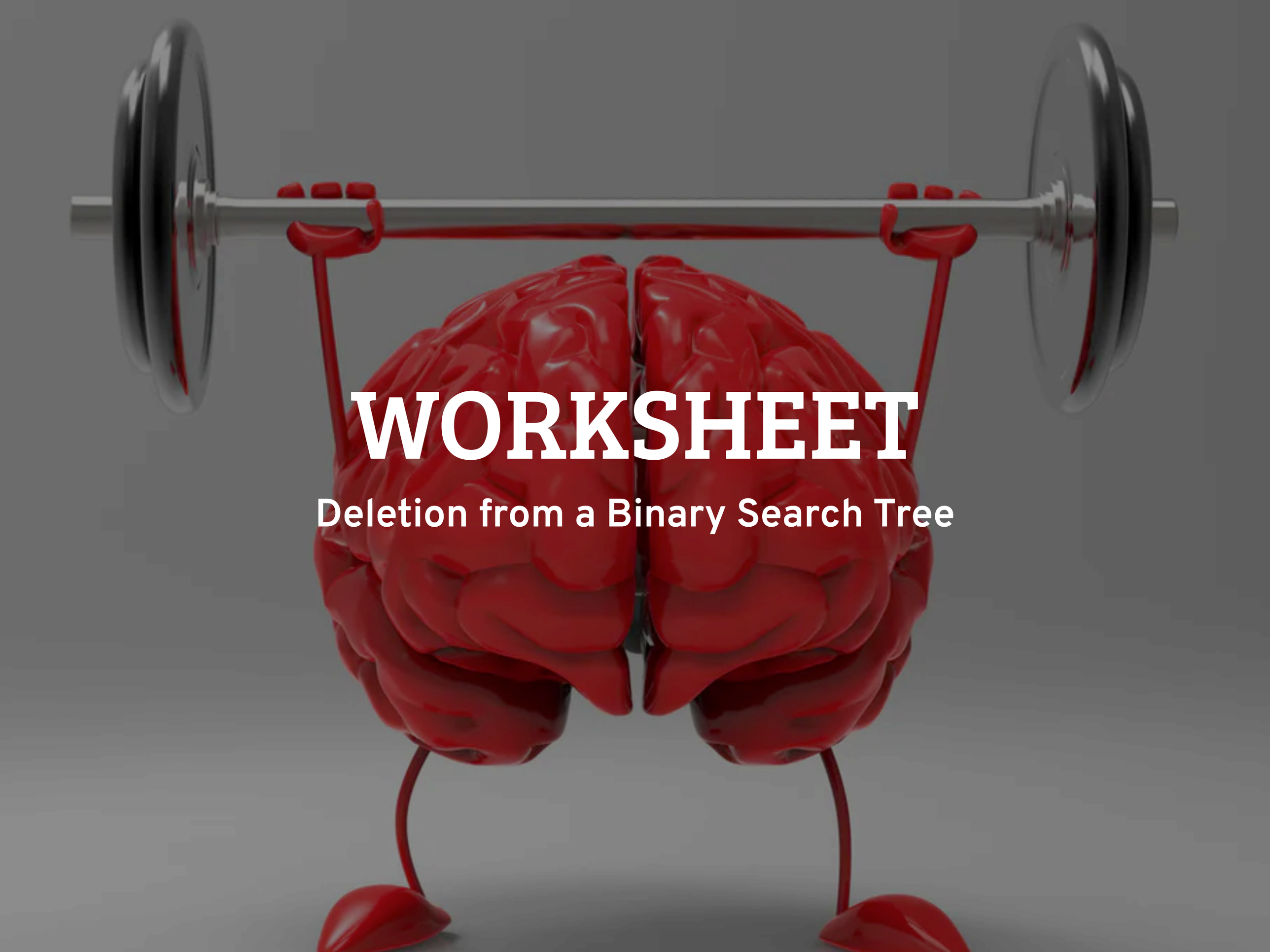
Say that self is the current subtree being inspected

What to do if the BST is empty (self._root is None)?

What if item to delete is less than the value self._root?

What if item to delete is more than the value self._root?

What if item to delete equals the value self._root?



WORKSHEET

Deletion from a Binary Search Tree

delete_root Helper

BST Deletion

Different Cases

Item found => time to remove the node

Three cases:

1. It's a leaf (Has no children)
2. Has only one child
3. Has two children

WORKSHEET

delete_root Helper for BST Deletion

Quizizz time!

Binary Search Trees

https://quizizz.com/admin/quiz/62301a0ad96470001d5e4358/binary-search-trees

BST Deletion

Delete operation steps - recap

Traverse the tree to locate the node with the intended value

If we reach a leaf and no match => not found, done!

If value to delete is smaller => inspect left subtree

If value to delete is larger => inspect right subtree

If value to delete found (equal)

Case a) No children => easy, just remove the node

Case b) One child => easy, just connect the child to current subtree's parent

Case c) Two children => clear the value, pick a replacement value from a descendant under it, and remove that descendant node

Max from left subtree, or min from right subtree

LET'S CODE

delete_root helper