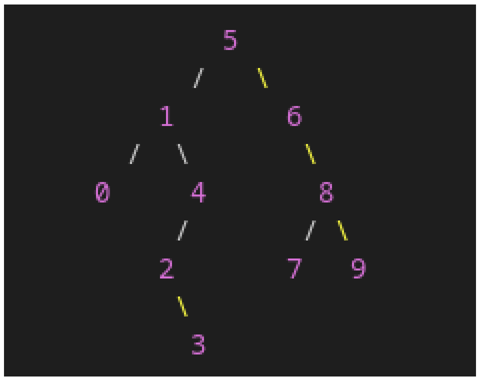
CMPSC 412 – Lab-6 (25 points)
Binary Search Tree - Basics

Due date: 10/11/2022

Lab Exercises:

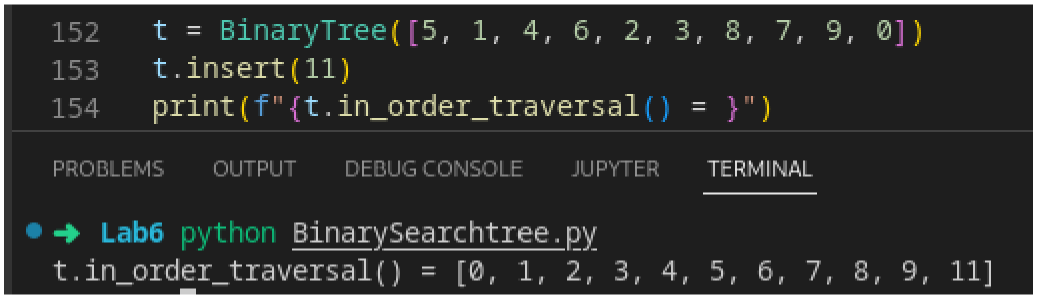
Exercise-1: (10 points)
visual representation of my data tree for reference that I build manually.



```
          5
        /   \
      1       6
    /  \       \
  0     4       8
       /       / \
      2       7   9
       \
        3
```

Develop a BinarySearchtree.py which can perform the following functions:

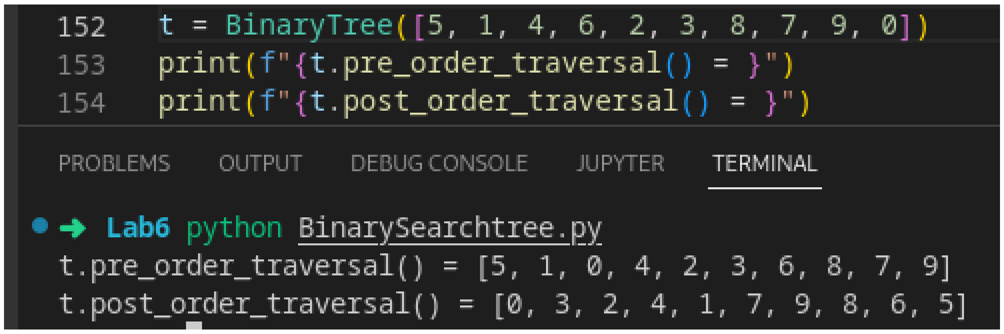
- Insert node to a tree
- Perform In-order traversal



```
t = BinaryTree([5, 1, 4, 6, 2, 3, 8, 7, 9, 0])
t.insert(11)
print(f"{t.in_order_traversal() = }")
```

```
➜ Lab6 python BinarySearchtree.py
t.in_order_traversal() = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 11]
```

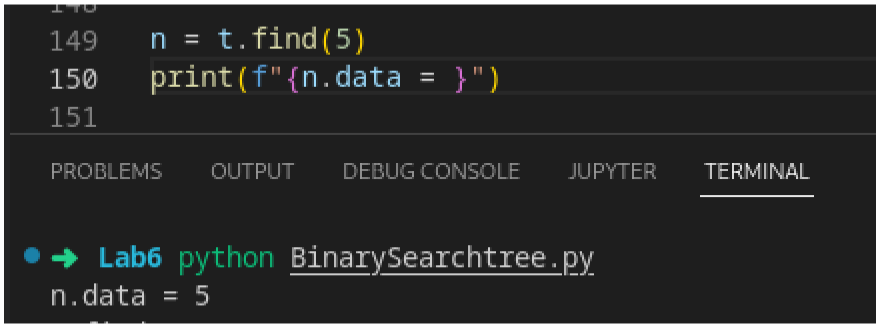
- Perform Pre-order traversal
- Perform Post-order traversal



```
t = BinaryTree([5, 1, 4, 6, 2, 3, 8, 7, 9, 0])
print(f"{t.pre_order_traversal() = }")
print(f"{t.post_order_traversal() = }")
```

```
➜ Lab6 python BinarySearchtree.py
t.pre_order_traversal() = [5, 1, 0, 4, 2, 3, 6, 8, 7, 9]
t.post_order_traversal() = [0, 3, 2, 4, 1, 7, 9, 8, 6, 5]
```

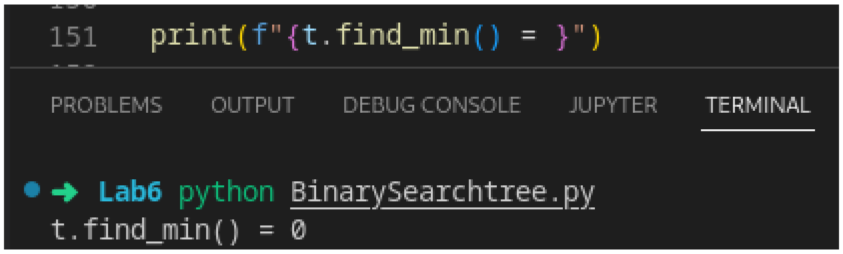
- Find a node



```
n = t.find(5)
print(f"{n.data = }")
```

```
➜ Lab6 python BinarySearchtree.py
n.data = 5
```

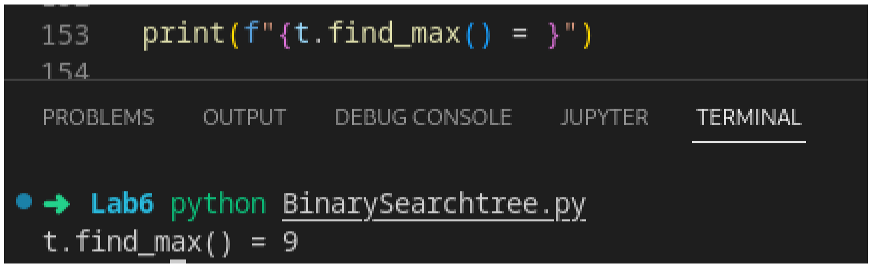
- Minimum value in the tree



```
print(f"{t.find_min() = }")
```

```
➜ Lab6 python BinarySearchtree.py
t.find_min() = 0
```

- Maximum value in the tree

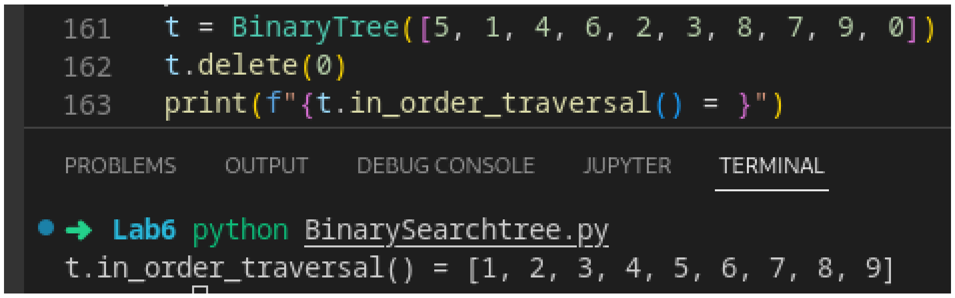


```
print(f"{t.find_max() = }")
```

```
➜ Lab6 python BinarySearchtree.py
t.find_max() = 9
```

Exercise-2: (7.5 points)
Write a function to remove a node from a tree data structure? This function should consider all the three cases: case-1: remove a leaf node, case-2: remove a node with one child and case-3: remove a node with two children.
Perform the time complexity for this function. Briefly explain?

Case-1:

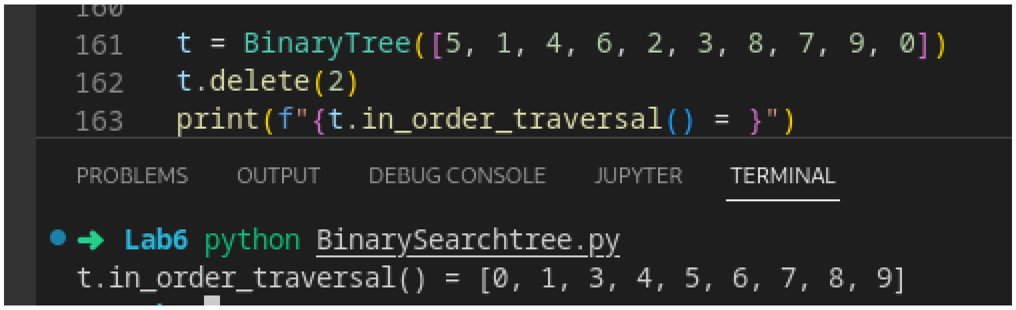


```
t = BinaryTree([5, 1, 4, 6, 2, 3, 8, 7, 9, 0])
t.delete(0)
print(f"{t.in_order_traversal() = }")
```

PROBLEMS   OUTPUT   DEBUG CONSOLE   JUPYTER   TERMINAL

```
● ➜ Lab6 python BinarySearchtree.py
t.in_order_traversal() = [1, 2, 3, 4, 5, 6, 7, 8, 9]
```

Case-2:

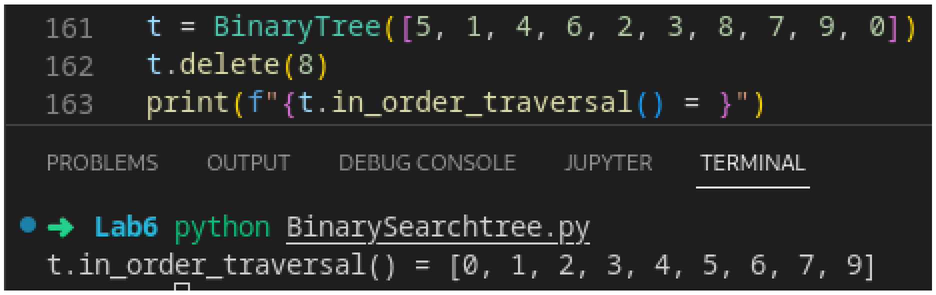


```
t = BinaryTree([5, 1, 4, 6, 2, 3, 8, 7, 9, 0])
t.delete(2)
print(f"{t.in_order_traversal() = }")
```

PROBLEMS   OUTPUT   DEBUG CONSOLE   JUPYTER   TERMINAL

```
● ➜ Lab6 python BinarySearchtree.py
t.in_order_traversal() = [0, 1, 3, 4, 5, 6, 7, 8, 9]
```

Case-3:



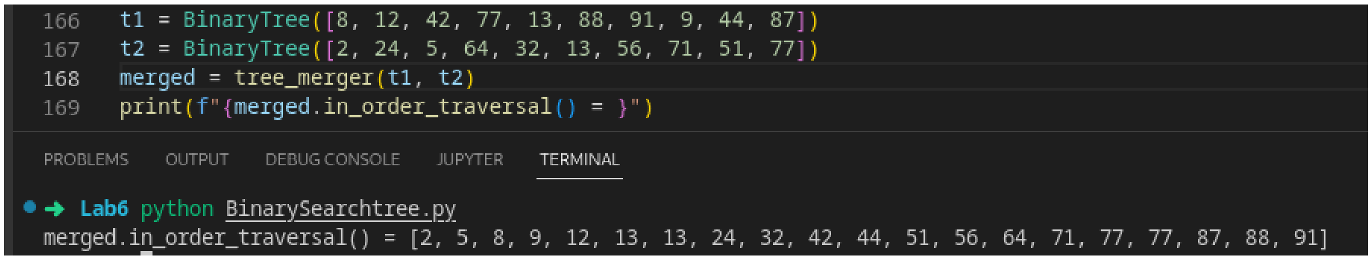
```
t = BinaryTree([5, 1, 4, 6, 2, 3, 8, 7, 9, 0])
t.delete(8)
print(f"{t.in_order_traversal() = }")
```

PROBLEMS   OUTPUT   DEBUG CONSOLE   JUPYTER   TERMINAL

```
● ➜ Lab6 python BinarySearchtree.py
t.in_order_traversal() = [0, 1, 2, 3, 4, 5, 6, 7, 9]
```

Time complexity:
This function works by recursively calls to left node or right node depending on the values of the currently node, at any case there is only one single recursive call therefore it only take N time, with N being the height of the tree, therefore it have O(n) time complexity.

Exercise-3: (7.5 points)
Write a function which takes two trees and merges the two trees. The function should return the merged tree. Perform the time complexity for this function. Briefly explain?



```
t1 = BinaryTree([8, 12, 42, 77, 13, 88, 91, 9, 44, 87])
t2 = BinaryTree([2, 24, 5, 64, 32, 13, 56, 71, 51, 77])
merged = tree_merger(t1, t2)
print(f"{merged.in_order_traversal() = }")
```

```
➜ Lab6 python BinarySearchtree.py
merged.in_order_traversal() = [2, 5, 8, 9, 12, 13, 13, 24, 32, 42, 44, 51, 56, 64, 71, 77, 77, 87, 88, 91]
```

This function make use of the tree pre_order_traversal() method twice, tree1 and tree2, which traverse the tree twice therefore giving it n + n time, which is 2n, however when converted to big O notation coefficient get drop so the final big O notation of the function is O(n).

Note: create a tree with minimum 10 values and demonstrate all the functions. Attach the screenshots of the results. Balancing the tree is not necessary for these exercises.