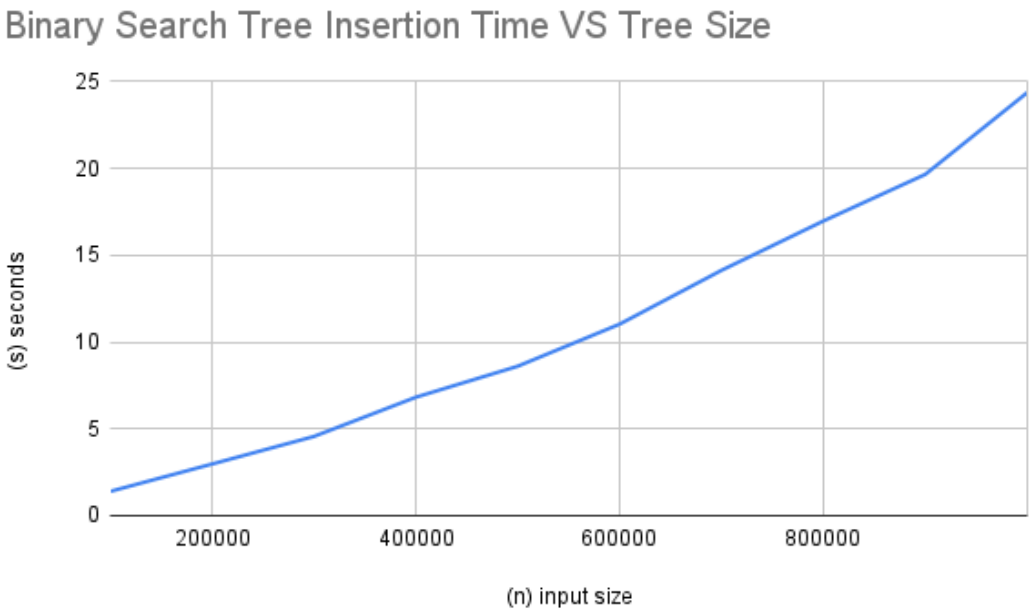
Lab 4 Run Time Analysis:

(Insert vs. look_up) for Binary Search Trees

Insert Function:

```python
def insert(tree: BinTree, value: Any, comes_before: Callable[[Any, Any], bool]) -> BinTree:
    match tree:
        case None:
            return Node(value, None, None)
        case Node(v, left, right):
            if comes_before(value, v):
                return Node(v, insert(left, value, comes_before), right)
            else:
                return Node(v, left, insert(right, value, comes_before))
```

- This function will take much longer than look_up, since it has the potential to keep calling insert (n times), and each call may require rebuilding part of the tree.

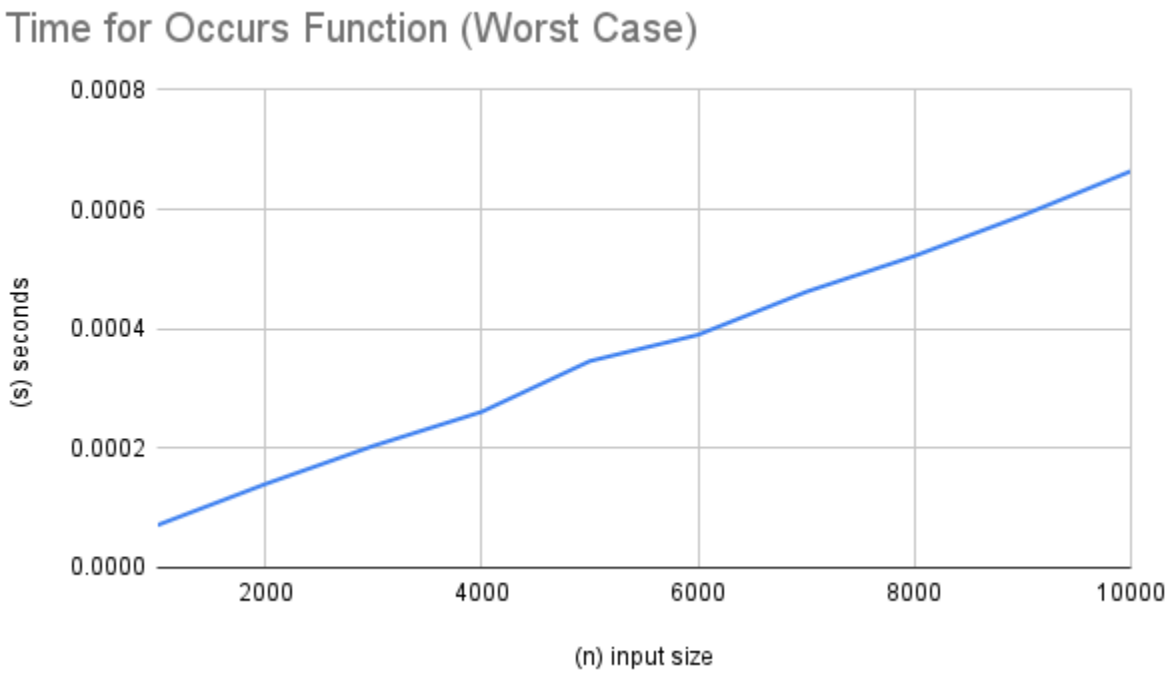


- Graph has a linear trend and resembles O(n) with the output time proportional to input (n) size of the tree. Although it takes more time than look_up, it is still linear. Insert still resembles O(n) in the worst case.

look_up Function:

```python
def lookup(tree: BinTree, value: Any, comes_before: Callable[[Any, Any], bool]) -> bool:
    match tree:
        case None:
            return False
        case Node(v, left, right):
            if not comes_before(value, v) and not comes_before(v, value):
                return True
            elif comes_before(value, v):
                return lookup(left, value, comes_before)
            else:
                return lookup(right, value, comes_before)
```

- look_up requires much less time since in the worst case it has to go through the full binary search tree, basically one long linked list of length (n) the input.



- The graph is linear so look_up also resembles O(n). look_up is significantly faster, and only resembles O(n) in the worst case.