Week 9

9.2

a)

```
# doubly linked with head & tail
def reverse(self):
    prev = NULL
    current = self.head
    while current != NULL:
        next = current.next
        current.next = prev
        current.prev = next
        prev = current
        current = next
    self.head = prev
```

explanation

the reason this alg is in-situ is because it does not use any additional arrays & has a constant memory complexity & only modifies current data

it is O(n) time as it loops from the head to tail.next(NULL) once which n long.

b)

```
def tree_to_linked(root, node=None):
        if root is None:
            return
    tree_to_linked_list(root.left, node.next)
    node.data = root.data
    tree_to_linked_list(root.right, node.next.next)
```

this algorithm has two parts, traversing & adding to the list. The traversal is O(n) because we have to go to every node. Because we have a pointer to the next LL node, the time complexity of that is constant, meaning the time complexity of this algorithm O(n)

Begin with the tail

default is None for first time call

```
def link_to_tree(tree, node):
    if(node is None):
        return
    if(node is tail):
        root = TreeNode(node.data)
        return link_to_tree(root, node.prev)

  temp = TreeNode(node.data)

  if (temp.data > node.data):
      tree.right = temp
      return link_to_tree(temp, node.prev)
  else:
      tree.left = temp
      return link_to_tree(temp, node.prev)
```

time coplexity of this is still $O(n)$ because we need to traverse from tail to head.

However because this is a Bst, the search time is going to be $O(h) = O(\log n)$ which is less than the LL's $O(n)$.