Problem 9.2 Linked Lists & Rooted Trees

a)

```
Reverse_Linked_List (node head)
    if head == NIL
        return
    end if

    node previous = NIL
    node next = NIL

    while head ≠ NIL
        next = head->link
        head->link = previous
        previous = head
        head = next
    end while

    head = previous
    return head
end Reverse_Linked_List
```

This algorithm is an in-situ algorithm because it only needs a constant space O(1) to create the two pointers *previous* and *next* to point to the nodes. It does not need an extra space to produce the output which is the reversed linked list. This is done in the same memory that contains the list, we just change the direction of the nodes.

b)
```
struct list* BSTtoLinkedList (BST* root, struct list* head) {
        if (root == NULL) {
            return head;
        }
        BST* node1 = Minimum (root);
        BST* newnode = node1;
        while (newnode != NULL) {
            head = AddAtEnd (head, newnode->data);
            newnode = Successor (root, newnode);
        }
        return head;
  }
```

In my implementation I have converted the binary search tree to linked list using Minimum and Successor function and then I have passed the elements of BTS to the linked list by inserting the elements at the end of the linked list. Inserting an element at the end takes O(n) time because of iterating till the end. But we perform the insertion for every element of the BTS, so the time complexity is **$O(n^2)$.** Minimum function takes **O(h)** time and Successor function also takes O(h) time. If the tree is balanced then, O(h) = O(log n). But we perform the Successor function for every element of the BTS so, if there are n elements the time complexity is **O(n logn).**

So, the overall time complexity is:

$$T(n) = O(\log n) + O(n^2) + O(n \log n)$$
$$= \mathbf{O(n^2)}$$

c) In my implementation I have built the tree from the leaves to the root. Firstly, we count the number of nodes in the sorted linked list. Half of the nodes are used to build the left subtree. The middle node is used as the root and the remaining half of the nodes are used to build the right subtree. Searching an element in binary search tree takes O(h) time, while to search for an element in the linked list, we must traverse the list until we find that element, hence searching in the linked list takes O(n) time. The code is implemented in this way to make possible that the time complexity for searching in the BST is smaller than that of the linked list. We can understand that we reach to this point since the height of the BST would be half of the number of elements(nodes) in the linked list since we chose the root to be the middle node.