Problem 1

a) The implementation of Stack is in Stack.h and testStack.cpp.

Two main operations of the stack are Push and Pop method. Two constructors will link the new elements to the existing linked list. In Push method, new elements are just linked to the existing list in Last In First Out (LIFO) mode. In Pop method, each element will be popped out in LIFO mode. Hence, the order of elements, or the size of the stack is not required to consider for the time complexity of those two functions. Every element added to the stack will take a constant time. Hence, the time complexity for both operation is O(1).

b) The implementation of Queue algorithm is in testQueue.cpp, Stack2.h and Queue.h.

Problem 2

a) ReverseLinkedList(L,x)

```
List cursor, next, prev
Cursor = L
prev = NULL
while (cursor != NULL)
        next =  cursor->next
        cursor->next = prev
        prev = cursor
        cursor = next
L = prev
return L
```

In this pseudocode, we use three references : prev, next and cursor, to point and link elements back and forth. We do not require any additional inputs or counting modifications to reverse the list successfully. We just have to use three references to swap the pointer of the list successfully. Hence, the space complexity required for this algorithm is only $\Theta(1)$, which is constant. In-situ algorithms require only $\Theta(1)$ space complexity, so this pseudocode is also in-situ algorithm.

In the algorithm, the list will be reversed until the cursor pointer has reached to NULL in the while loop, and each iteration will take some constant time. So, the loop will iterate for n times, and the time complexity is $\Theta(n)$.

b) The implementation of this problem is in Binary.h, LinkedList.h and BinarytoList.cpp.

Asymptotic Time Complexity: Each element in the tree will be recursively traversed (in an inverse way: right to left) in "BtoLinkedList" function (in Binary.h file). Every element in the tree will be traversed to the list will only once recursively, and each element will take a constant time to be placed in the linked list. So the time complexity for n elements will become O(n).

c) The implementation of this problem is in ListToBinary.h and ListToBinary.cpp.

Asymptotic Time Complexity: Each element in the sorted linked list will be put into the binary search tree, starting from the left leaf nodes for the first half of elements, then to the right for the rest, and the root will be constructed afterwards in "ListToTree" function. So, we just place all sorted element in the right order, and not considering the height of tree for time complexity.

Each element will take a constant time to be placed in the binary search tree. Hence, the time complexity for n elements will be O(n).

References

A. (2018, December 25). Flatten a binary tree into linked list | Set-3. Retrieved April 3,2019, from https://www.geeksforgeeks.org/flatten-a-binary-tree-into-linked-list-set-3/
(I used this website to get ideas for Problem 2 b.)

R. (2019, March 11). Sorted Linked List to Balanced BST. Retrieved April 3, 2019, from https://www.geeksforgeeks.org/sorted-linked-list-to-balanced-bst/
(I used this website to get ideas for Problem 2 c.)

Queue using Stacks. (2019, February 25). Retrieved April 3, 2019, from https://www.geeksforgeeks.org/queue-using-stacks/
(I used this website to get ideas for Problem 1 b.)