Homework 9 by Timofei Podlorytov

9.1

The algorithms for a and b are implemented in the .cpp file attached.

Analysis of the algorithm is here:
For Stack:
Push (T x) =O(1)
Pop()=O(1)
isEmpty()=O(1)

All their asymptotic times are O(1) since we don't have any recursion as well as no loops meaning they are constant and thus O(1).

For Queue:
enQueue =O(1)
deQueue =O(n)
isEmpty =O(1)

For enQueue and isEmpty there are no loops and recursion making them constant and O(1). On the other hand taking an element can vary in time taken as in the best case when there are elements in stack2 we just call pop which is O(1), but in the worst case we have to move all the elements from stack 1 which is O(n), making the whole time complexity O(n) in the worst case when we first only put elements and then decide to take them.

9.2

a)Write down the pseudocode for an in-situ algorithm that reverses a linked list of n elements in Θ(n). Explain why it is an in-situ algorithm.

```
//First we need a structure of a node of a linked list
struct Node{
        T data;
        Node *next;
}
//as well as a linked list itself it ends with Nil
struct List{
        start* Node;
        int num; //-number of elements
}
//assume we are given a linked List
reverse(&List){
        Node* prev=Nil//pointer to the previous pointer
        Node* nextNode;//in order to switch nodes
        Node* current=List→start
        while current != Nil do:
                nextNode=current→next;
                current→next=prev;
                prev=current;
```

```
            current=NextNode;
        List→start=prev;// current node is null and the previous is actually the last.
}
```

Why in-situ?
In the algorithm we don't allocate any memory and only use pointers to traverse the list. Also, since we only traverse the list once using a while loop we have Θ(n) time complexity.

b)Implement an algorithm to convert a binary search tree to a sorted linked list and derive its asymptotic time complexity.

We need to traverse the tree going left – current -right in order to get a sorted list. It's implemented in the p2.cpp file.

For adding an element to a linked list the complexity is O(1). We have a recursion which leads us to pass through each element only once. Meaning we have a resulting time complexity of O(n) where n is the number of elements in a tree.

c)Implement an algorithm to convert a sorted linked list to a binary search tree and derive its asymptotic time complexity. The search time complexity of the resulting binary search tree should be lower than the one for the equivalent sorted linked list.

Here we first go to the middle and then call recursion from there and from the start:
The O(log(n)) is what it takes to add an element to a tree, but it's smaller than O(n).
$T(n)=T(n/2)+O(n/2)+O(\log(n))+T(n/2)=\underline{2T(n/2)+O(n)}$
here O(n/2) is the time it takes to get to the middle. We can use the master theorem. $\text{Log}_2 2=1$
$1==1$, $n^1==n^1$,meaning we have the 2nd case and the overall complexity is $\Theta(n*\log_2(n))$.