ADS Homework 8

Sanjay Timilsena

March 2020

1 Problem 8.1

1.1

Please see the attached file stack.cpp for the implementation of stack.

The time complexity of different operations of stack are as follows:

- Push:

  The time complexity for Push is $O(1)$ because adding a new element to the stack requires constant amount of time regardless of the position of stack and data type provided.

- Pop:

  The time complexity for Pop is $O(1)$ because removing an element from top of the stack requires constant amount of time regardless of the position of stack and data type of the element.

- isEmpty:

  The time complexity for isEmpty is $O(1)$ because isEmpty checks if the pointer of the top of the stack is pointing to NULL which requires a constant amount of time.

1.2

The required pseudo-code is as follows provided that the Stack class and its methods are already implemented as in above program:

```
1
2  Stack<Type> stack1(size), stack2(size);
3
4  enqueue(value){
5      stack1.push(value);
6  }
7
8  dequeue(){
9      if(stack1 and stack2 is NULL)
10         return "underflow message";
11     //reversely store data of stack1 in stack2
```

```
    while(stack1 is not empty){
        stack2.push(stack1.pop());
    }
    stack2.pop();
}

checkEmpty(){
    if(stack1 && stack2 = NULL){
        return true;
    }
    else{
        return false;
    }
}
```

2

2.1 Problem 8.2

Please check the attached file reverse_stack.cpp.

Explanation: The algorithm uses constant storage to reverse a stack of any size by using three helping nodes to switch pointers. In the while loop, no extra storage is being reserved to copy elements. As the loop transverses the stack, the pointers are being swapped in order to reverse the stack. Therefore, this algorithm is in-situ as it only needs 3 helping nodes to reverse a stack of any size. The asymptotic run time is O(n) as the stack is only traversed once.

(Also included in comments)

2.2

Please see the attached file bst_to_ll.cpp for the implementation of the algorithm. The asymptotic time complexity for this algorithm is $O(n)$ as the recursive function **ConvertToLinkedList()** needs to pass through every node and change the pointers.

2.3

Please see the attached file ll_to_bst.cpp for the implementation of the algorithm. The asymptotic time complexity for this algorithm is $O(n \log n)$