Arushi Sahai
as5976
COMS 3137
March 7, 2020

Homework 2: Written Assignment

1. **(7 pts): Weiss 3.8.**
   The following routine removes the first half of the list passed as a parameter:

```
public static void removeFirstHalf(List<?> lst)
{
   int theSize = lst.size()/2;
   for(int i = 0; i < theSize; i++)
      lst.remove(0);
}
```

   a. **Why is theSize saved prior to entering the for loop?**
      theSize is saved prior to entering the for loop because as items are removed from the list, the size of the list shrinks. However, we want to iterate through the list based on the original size of the list, so we save it first.
   b. **What is the running time of removeFirstHalf if lst is an ArrayList?**
      If lst is an ArrayList, it's an O(N) operation to remove each item from beginning of the array list because all subsequent entries must be shifted over one spot, and there are N/2 iterations, so it's an $O(N^2)$ runtime in total.
   c. **What is the running time of removeFirstHalf if lst is a LinkedList?**
      If lst is a LinkedList, it's an O(1) operation to remove each item from the beginning of the array because you just have to break and reassign a link. Because there are N/2 iterations of this, the total runtime is O(N).
   d. **Does using an iterator make removeFirstHalf faster for either type of List?**
      No, because we're looking at the first element of the list so using an iterator would not help.

2. **(7 pts): Weiss 3.24 - you only need to provide pseudocode for this problem.**
   Write routines to implement two stacks using only one array. Your stack routines should not declare an overflow unless every slot in the array is used.

```
array arr;
stack1.top = -1;
stack2.top = arr.length;

stack1.push():
    if (stack1.top + 1 != stack2.top) {
        stack1.top++;
        arr[stack1.top] = x;
```

```
    }
    else {
        throw stackOverflowError;
    }

stack1.pop():
    if (stack1.top - 1 > 0) {
        Object stackTop = arr[stack1.top]
        stack1.top--;
        return stackTop;
    }
    else {
        throw stackUnderflowError;
    }

stack2.push():
    if (stack2.top - 1 != stack1.top) {
        stack2.top--;
        arr[stack2.top] = x;
    }
    else {
        throw stackOverflowError;
    }

stack2.pop():
    if (stack2.top + 1 < arr.length - 1) {
        Object stackTop = arr[stack2.top]
        stack1.top++;
        return stackTop;
    }
    else {
        throw stackUnderflowError;
    }
```

3. **(8 pts): Adapted from Weiss 3.29.**
   a. **Write an iterative algorithm in Java-like pseudocode for printing a singly linked list in reverse in O(N) time. You can use as much extra space as you need. The original list pointers CAN NOT BE MODIFIED. State in big-O notation how much extra space is used by this algorithm.**
   O(N) extra space is used by this algorithm.

```
private static void printA(list) {
    ArrayList<Node> forwards = new ArrayList<Node>();
    for (int i = 0; i < list.size(); i++)
    {
       forwards.add(list.get(i));
    }

    for (int i = forwards.size() - 1; i >= 0; i--) {
          System.out.println(forwards.get(i));
    }
}
```

   b. **Write another iterative algorithm in Java-like pseudocode for printing a singly linked list in reverse using O(1) extra space. The original list pointers CAN NOT BE MODIFIED. This algorithm can have any runtime (it will be worse than the algorithm in part a). State the runtime of your algorithm in big-O notation.**
   This algorithm has $O(N^2)$ runtime.

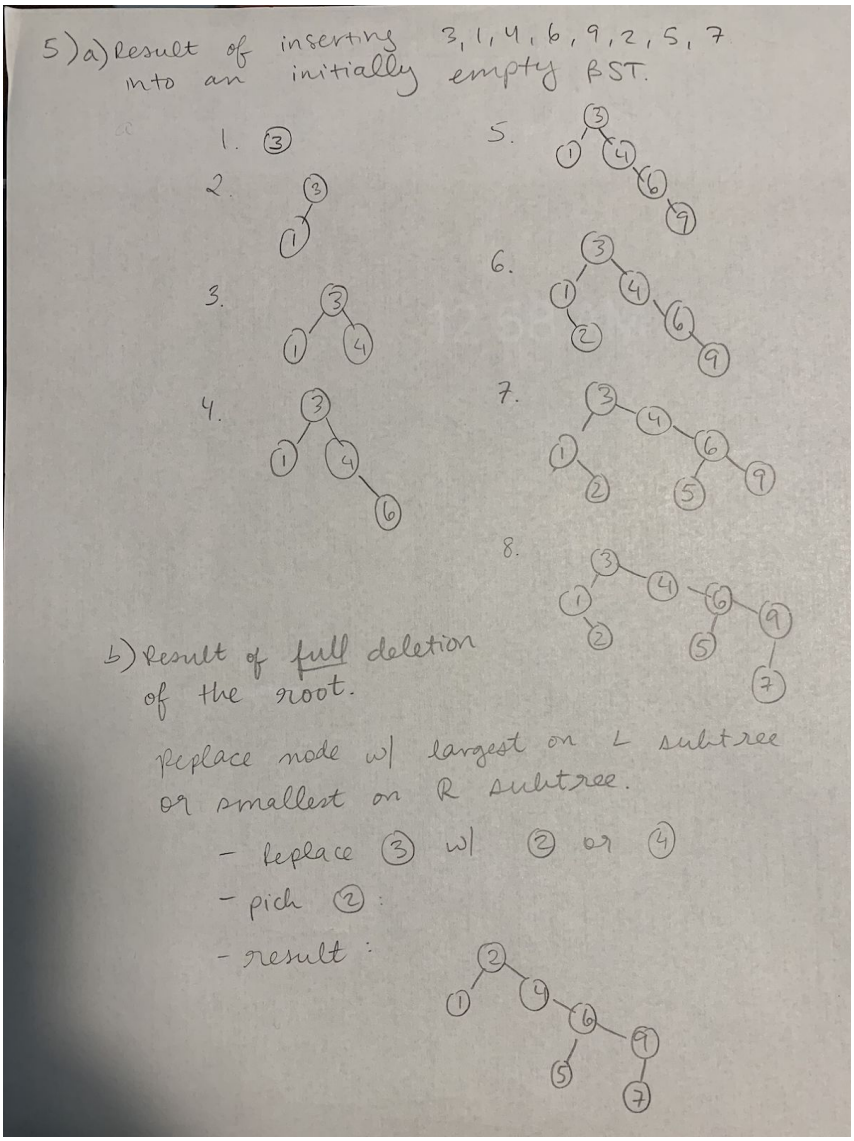
```
private static void printB(list)
    for (int i = list.size() - 1; i >= 0; i--) {
          System.out.println(list.get(i));
    }
}
```

4. **(7 pts): Weiss 4.6 - We're looking for a full induction proof**
   A full node is a node with two children. Prove that the number of full nodes plus one is equal to the number of leaves in a nonempty binary tree.
   Base case: 0 full nodes + 1 = 1 leaf (true).
   Inductive hypothesis: assume n+1 = number of leaves holds true for a general tree with n full nodes (true).
   Show that (n+1)+1 = number of leaves for a tree with n+1 full nodes
   There are 2 ways to add one more node to a general tree:
      1) To a node with 1 child, add 1 node (adds 1 leaf)

The number of full nodes increases by 1, and the number of leaves does too: (n+1) + 1 = number of leaves for n + 1 full nodes

2) To a node with 0 children, add 1 node (adds 1 leaf)
This does not increase the number of full nodes nor leaf nodes because one leaf has been replaced with another so the hypothesis still holds.

5. **(7 pts): Weiss 4.9 - In part a show the tree after each insert - a total of 8 trees. In part b, we're doing full deletion, not lazy deletion.**
   a. **Show result of inserting 3,1,4,6,9,2,5,7 into an initially empty binary search tree.**
   b. **Show the result of deleting the root.**



6. **(7 pts): Based on Weiss 4.16 - Describe any modifications that you would need to make to the BinaryNode itself, and then show the implementation for findMin. You don't actually have to give us a full working class.**
   Redo the binary search tree class to implement lazy deletion. Note carefully that this affects all of the routines. Especially challenging are findMin and findMax, which must now be done recursively.
   The main modification to make to the BinaryNode is adding in a boolean field "valid" so that when a node is removed, the valid field is set to false. It is initialized to true.

```
private BinaryNode<AnyType> findMin(BinaryNode<AnyType> t) {
    if (t == null) {
        return null;
    }
    BinaryNode<AnyType> leftMin = findMin(t.left);
    if (leftMin == null) {
        if (t.valid == false) {
            return findMin(t.right);
        }
        else {
            return t;
        }
    }
    else {
        return leftMin;
    }
}
```