Homework Week 2

Text Questions

Monica Quaintance

mjq2102@columbia.edu

1. Weiss Exercise 3.1

You are given a list, L, and another list, P, containing integers sorted in ascending order. The operation printLots(L,P) will print the elements in L that are in positions specified by P. For instance, if P = 1, 3, 4, 6, the elements in positions 1, 3, 4, and 6 in L are printed. Write the procedure printLots(L,P). You may use only the public Collections API container operations. What is the running time of your procedure?

```
import java.util.ArrayList;
import java.util.ListIterator;

public class collectionPrinter<T> {

  ArrayList<T> fullList;
  ArrayList<Integer> targetList;
  ArrayList<T> resultList = new ArrayList<T>();

  public collectionPrinter(){
  }

  public ArrayList<T> printLots(ArrayList<T> L, ArrayList<Integer> P) {
    fullList = L;
    targetList = P;

    ListIterator<Integer> iterTarget = targetList.listIterator();
    ListIterator<T> iterFull = fullList.listIterator();

    int i = 0;
    int targetIndex = iterTarget.next();
    while (iterFull.hasNext() && i<= targetIndex)
    {
      T element = iterFull.next();
      if (i == targetIndex) {
        resultList.add(element);
          targetIndex = iterTarget.next();
      }
```

```
      else if (iterTarget.hasNext() == false)
        return resultList;

      else {
      }
      i++;
    }
    return resultList;
    }
}
```

You must check at least j elements, where j is the value of the final element in P. Running time is O(N).

2. Weiss Exercise 3.2

Swap two adjacent elements by adjusting only the links (and not the data) using:

a. Singly linked lists.

List A: [w] -> [x] -> [y] -> [z]

becomes: [w] -> [y] -> [x] -> [z]

- w.next = y (x.next)
- x.next = z (y.next)
- y.next = x

b. Doubly linked lists.

List B: [w] <-> [x] <-> [y] <-> [z]

becomes: [w] <-> [y] <-> [x] <-> [z]

- x.next = z (y.next)
- w.next = y (x.next)
- y.next = x
- z.prev = x
- x.prev = y
- y.prev = w

3. Weiss, Exercise 3.24

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.

```java
import java.util.EmptyStackException;

public class doubleStack
{

//  default stack of size 16
    String[] stackArray = (String[]) new String[16];
    private int leftTop = -1;
    private int rightTop = stackArray.length;

    public doubleStack() {
    }

//  specified stack size
    public doubleStack(int arraySize) {
        this.stackArray = (String[]) new String[arraySize];
        this.leftTop = -1;
        this.rightTop = stackArray.length;
    }

//  prints left stack (for debugging)
    public String printLeft()
    {
        if (leftTop ==-1)
            throw new EmptyStackException();
        else {
            StringBuilder leftPrint = new StringBuilder();
            for (int i = 0; i < leftTop +1; i++)
                leftPrint.append(stackArray[i]);
            return leftPrint.toString();
        }
    }

//  prints right stack (for debugging)
    public String printRight()
    {
        if (rightTop == stackArray.length)
            throw new EmptyStackException();
        else {
            StringBuilder rightPrint = new StringBuilder();
            for (int i = stackArray.length-1; i > rightTop -1; i--)
                rightPrint.append(stackArray[i]);
            return rightPrint.toString();
        }
    }
```

```java
//  push onto left side of array
    public void leftPush(String newItem) {
        if (leftTop >= rightTop - 1)
            throw new IndexOutOfBoundsException("Both stacks are full");
        else {
            stackArray[leftTop+1] = newItem;
            leftTop++;
        }
    }

//  pop from left side of array
    public String leftPop() {
        if (leftTop == -1)
            throw new EmptyStackException();
        else {
            String popItem = stackArray[leftTop];
            leftTop--;
            return popItem;
        }
    }

//  push onto right side of array
    public void rightPush(String newItem) {
        if (leftTop >= rightTop-1)
            throw new IndexOutOfBoundsException("Both stacks are full");
        else {
            stackArray[rightTop-1] = newItem;
            rightTop--;
        }
    }

//  pop from right side of array
    public String rightPop() {
        if (rightTop == stackArray.length)
            throw new EmptyStackException();
        else {
            String popItem = stackArray[rightTop];
            rightTop++;
            return popItem;
        }
    }
}
```

Weiss, Exercise 4.5 (this is asking for a proof)

Show that the maximum number of nodes in a binary tree of height h is $2^{h+1}1$.

For max # of nodes, assume tree is always complete. Number of nodes at height $h = N(h), N(h) = 2^{h+1} - 1$.

Step One: Base Case: N(0) = 1

$2^{0+1} - 1 = 1$, Base Case is true.

Step Two: Assume for all values of h up to h, $2^{h+1} - 1 = N(h)$

Step Three: Using assertion in 2:

$N(h) = 2[N(h-1) - N(h-2)] + N(h-1)$

$N(h) = 2 * N(h-1) - 2 * N(h-2) + N(h-1)$

$N(h) = 2[2^{h-1+1} - 1] - 2[2^{h-2+1} - 1] + [2^{h-1+1} - 1]$

$N(h) = 2^{h+1} - 2 - 2^h + 2 + 2^h - 1$

$N(h) = 2^{h+1} - 1$

Weiss, Exercise 4.9 (using a full deletion)

a. Show the result of inserting 3, 1, 4, 6, 9, 2, 5, 7 into an initially empty binary search tree.

```
   3
 /   \
1     4
 \     \
  2     6
       / \
      5   9
         /
        7
```

b. Show the result of deleting the root.

```
   4
 /   \
1     6
 \   / \
  2 5   9
       /
      7
```