Christian Santana

Prof: Chandra N Sekharan

COMP 272-001

18 September 2021

Homework 2

Source Code for question 1 and 2 with test method main:

```
import java.util.*;
public class ExtLinkedList<E> extends LinkedList<E>{

    public ExtLinkedList <E> secondHalfList(){
        ExtLinkedList <E> secondhalf = new ExtLinkedList<E>();
        int size = this.size();
        int midpoint;
        if (size%2 == 0)
            midpoint = size / 2;
        else
            midpoint = (size / 2)+1;
        // for (int i = midpoint; i < size; i++){
        //     secondhalf.add(this.get(i));
        // }
        ListIterator <E> li = this.listIterator(midpoint);
        while(li.hasNext())
            secondhalf.add(li.next());
        return secondhalf;
    }

        public ExtLinkedList <E> oddList(){

        ExtLinkedList <E> oddindexs = new ExtLinkedList<E>();
        ListIterator <E> li = this.listIterator();
        int index = 0;
        while(li.hasNext()){
            E value = li.next();
            if (index %2 != 0)
                oddindexs.add(value);
            index += 1;
```

```
        }
        return oddindexs;
    }

    public ExtLinkedList <E> evenList(){
        ExtLinkedList <E> evenindexs = new ExtLinkedList<E>();
        ListIterator <E> li = this.listIterator();
        int index = 0;
        while(li.hasNext()){
            E value = li.next();
            if (index %2 == 0)
                evenindexs.add(value);
            index += 1;
        }
        return evenindexs;
    }
    public static void main(String[] args) {
        ExtLinkedList<Integer> test = new ExtLinkedList<Integer>();
        ExtLinkedList<String> empty = new ExtLinkedList<String>();
        test.add(1);
        test.add(2);
        test.add(3);
        test.add(4);
        test.add(5);
        //test.add(6); //comment in and out for even and odd number list
        ExtLinkedList second = test.secondHalfList();
        ExtLinkedList secondempty = empty.secondHalfList();
        System.out.println(test);
        System.out.println(second);
        System.out.println(secondempty);
        ExtLinkedList evenindexes = test.evenList();
        ExtLinkedList evenempty = empty.evenList();
        System.out.println(evenindexes);
        System.out.println(evenempty);
        ExtLinkedList oddindexes = test.oddList();
        empty.add("testing");
        ExtLinkedList oddempty = empty.oddList();
        System.out.println(oddindexes);
        System.out.println(oddempty);

    }
}
```

1. I believe the runtime complexity of my solution secondHalfList() to be O(n). I could make the argument that the complexity is O(n/2), but I don't think this is the case as the LinkedList.size() method likely iterates through the entire linkedlist to find the size. Meaning my method has to run through the n elements of the linkedlist before iterating through the second half of the linkedlist. So the complexity is O(n).
2. Runtime complexity of oddList() is O(n) as it must iterate through the whole linkedlist of elements n once to find all the odd index elements to return. The same is true for evenList() where the function has to loop through all the n elements of the linkedlist to find all even index values to return.
3. Source Code with test method:

```
public class replaceChar {

    public String replaceChar(String p, int k, char c){
        try{
            return p.substring(0,k)+ c + p.substring(k+1);
        }
        catch(Exception e) {
            e.printStackTrace();
            return null;
        }
    }

    public static void main(String[] args) {
        replaceChar replacer = new replaceChar();
        String test = "Hi testing that this works";
        String empty = "";
        String newstr = replacer.replaceChar(test,3,'+');
        String outofbounds = replacer.replaceChar(empty, 0, 'I');
        String testbounds = replacer.replaceChar(test, 50, 'T');
        System.out.println(test);
        System.out.println(newstr);
        System.out.println(outofbounds);
        System.out.println(testbounds);
    }
}
```

As requested it returns a null value when k is out of bounds along with the error code/message.

4. i. The time complexity is O(n) as the function needs to loop through the whole of n at least once these are 2 non-nested for loops(No indentation) so the loops would be n+n or 2n which can then be simplified to O(n).

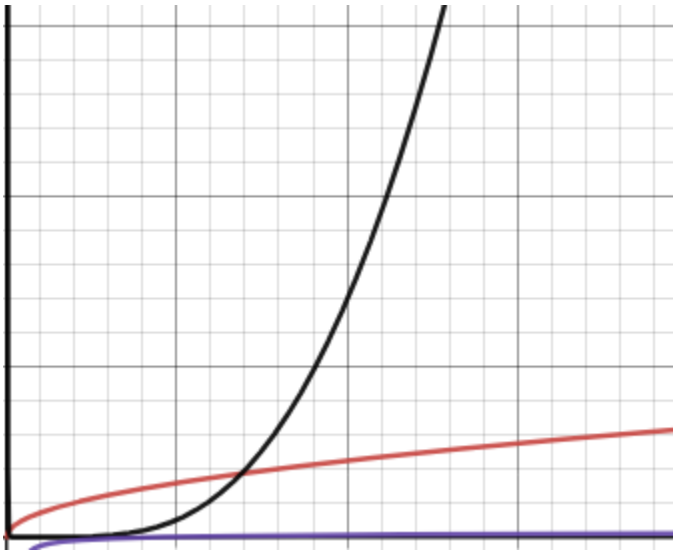
   ii. i. F(n)=(10+2n)(n2 +nlog3 n)

   Expanding algebra will get you a long equation. Looking for the dominant term we can see we will get $2n*n\text{^}2 = 2n^3$. Which you simplify to remove coefficient.

   Solution = $O(n^3)$

   ii. F(n)=n0.5+log10n+log log n

   All are separate terms so we need to find the dominant term that grows at the largest rate. This can be done with a graphing calc or with testing some values or seeing power of 10 as the highest exponent in the equation term. To double check I also graphed the possible answers.



   Doing so we find that the dominant term to be $\log^{10}n$.

   Solution = $O(\log^{10}n)$

5. Source Code with test method:

```
public class squarematrix {

    public void rowSums(int[][] arr){
        String output = "";
        for (int i = 0; i < arr.length; i++){
            int rowsum = 0;
            //Here I use the length of the column, but doesn't matter since it's
a square
            //so it's the same length/size
            for (int j = 0; j < arr[i].length; j++){
                rowsum += arr[i][j];
            }
            output += rowsum + ", ";
        }
        System.out.println(output.substring(0,output.length()-2));
    }

    public void columnMins(int[][] arr){
        String output = "";
        for (int i = 0; i < arr.length; i++){
            int columnmin = arr[0][i];
            for (int j = 0; j < arr.length; j++){
                //for some reason (arr[j][i]<columnmin) wasn't working
                //when columnmin was 1 higher than arr[j][i] used this roundabout
 way
                //not sure of the java syntax issue or why it didn't work
                if (columnmin-arr[j][i]>0)
                    columnmin = arr[j][i];
            }
            output += columnmin + ", ";
        }
        System.out.println(output.substring(0,output.length()-2));
    }
    public static void main(String[] args) {
        squarematrix driver = new squarematrix();
        int square[][] = {{3,2,5},{1,0,4},{5,6,7}};
        driver.rowSums(square);
        driver.columnMins(square);
    }
}
```

Source Code for questions 6 and 7:

```
import java.util.*;

public class prefixSums{

    public void prefixSums(LinkedList<Integer> input){
        int prefixsum = 0;
        if(input.isEmpty()){
            System.out.println(prefixsum);
            return;
        }
        ListIterator <Integer> li = input.listIterator();
        String output = "";
        while(li.hasNext()){
            prefixsum += (int) li.next();
            output += prefixsum+ ", ";
        }
        output =output.substring(0,output.length()-2);
        System.out.println(output);
    }

    public void reversePrefixSums(LinkedList<Integer> input){
        int prefixsum = 0;
        if(input.isEmpty()){
            System.out.println(prefixsum);
            return;
        }
        Iterator li = input.descendingIterator();
        String output = "";
        while(li.hasNext()){
            prefixsum += (int) li.next();
            output += prefixsum+ ", ";
        }
        output =output.substring(0,output.length()-2);
        System.out.println(output);
    }

    public static void main(String[] args) {
        prefixSums driver = new prefixSums();
        LinkedList<Integer> test = new LinkedList<Integer>();
        LinkedList<Integer> empty = new LinkedList<Integer>();
        test.add(5);
        test.add(3);
```

```
        test.add(2);
        test.add(9);
        test.add(3);
        test.add(15);
        test.add(22);
        driver.prefixSums(test);
        driver.reversePrefixSums(test);
        driver.prefixSums(empty);
        driver.reversePrefixSums(empty);
    }
}
```

8. Source Code for question 8 and test method.

```
import java.util.*;

public class linkedListmerger{

    public LinkedList<String> Sortedmerge(LinkedList<String> first, LinkedList<St
ring> second){
        if (first.isEmpty())
            return second;
        else if (second.isEmpty())
            return first;
        LinkedList <String> merged = new LinkedList<String>();
        ListIterator firstlist = first.listIterator();
        ListIterator secondlist = second.listIterator();
        while(firstlist.hasNext() && secondlist.hasNext()){
            if(((String) firstlist.next()).compareTo((String) secondlist.next())>
0){
                merged.add((String) secondlist.previous());
                secondlist.next();
                firstlist.previous();
            } else{
                merged.add((String) firstlist.previous());
                firstlist.next();
                secondlist.previous();
            }
        }
        if(!firstlist.hasNext())
            merged.add((String) secondlist.next());
        else
            merged.add((String) firstlist.next());
        return merged;
    }
```

```
    public static void main(String[] args) {
        LinkedList<String> firstlist = new LinkedList<String>();
        LinkedList<String> secondlist = new LinkedList<String>();
        firstlist.add("Apple");
        firstlist.add("Banana");
        firstlist.add("Cabbage");
        firstlist.add("Grapes");
        firstlist.add("Peaches");
        firstlist.add("Watermelon");
        secondlist.add("Alabama");
        secondlist.add("Apple");
        secondlist.add("Delaware");
        secondlist.add("Illinois");
        secondlist.add("New York");
        secondlist.add("Wiscousin");
        linkedListmerger driver = new linkedListmerger();
        LinkedList<String> mergedlist = driver.Sortedmerge(firstlist, secondlist)
;
        System.out.println(firstlist);
        System.out.println(secondlist);
        System.out.println(mergedlist);
        LinkedList<String> test1 = new LinkedList<String>();
        LinkedList<String> test2 = new LinkedList<String>();
        LinkedList<String> test3 = driver.Sortedmerge(test1, test2);
        System.out.println(test1);
        System.out.println(test2);
        System.out.println(test3);
    }
}
```

9. Source Code for question 9 and test method.

```
public class arrayPairs {
    public void uniquePairs(int[] arr, int k){
        String output = "";
        for(int i = 0; i<arr.length;i++){
            for(int j = i; j<arr.length;j++){
                int target = Math.abs(arr[i] - arr[j]);
                if (target == k)
                    output += "(" + arr[i] + "," + arr[j] + "), ";
            }
        }
        output = output.substring(0, output.length()-2);
        System.out.println(output);
    }
```

```
    public static void main(String[] args) {
        int[] intArray = new int[]{ 1,4,9,12, 6, 15, 5, 13,17 };
        arrayPairs driver = new arrayPairs();
        driver.uniquePairs(intArray, 3);
        driver.uniquePairs(intArray, 4);
    }
}
```