**Problem set 3**

1. **Printing the odd values in a list of integers (10pts)**
   1. **Recursion**

**public static void printOddsRecursive(IntNode** iFirstNode**) {**

// base case -> No next node

if( iFirstNode.next == null)

return;

// if odd value, print it

if( iFirstNode.val%2 == 1)

System.out.print( iFirstNode.val )

// recursive call

printOddRecursive(iFirstNode.next)

**}**

* 1. **Iteration**

**public static void printOddsIterative(IntNode** iFirstNode**) {**

// init IntNode we will use to iterate

IntNode control = iFirstNode;

while(control != null){

if(control.val%2 == 1)

System.out.print(control.val )

control = control.next;

}

**}**

1. **Improving the efficiency of an algorithm (15pts)**
   1. **Worst case**

Illustrate the worst case:

L1 = {1,1,1,1,1,1,… } - (1 everywhere)

L2 = {……..,1} - (no 1 before the last)

Assuming:

- the first list has a length of n.

- the second list has a length of m.

**In red:** we will call n times (length of list 1) the second loop, which calls m times getItem() of list 2 (which is O(n)).

Therefore the worst case is **O(n3).**

* 1. **Improved method**

**Private static void intersect(LLList** iList1**, LLList** iList2**){**

LLList inters = new LLList();

ListIterator it1 = iList1.iterator();

ListIterator it2 = iList2.iterator():

while( it1.hasNext() ){

Object item1 = it1.next();

while( it2.hasNext() ){

Object item2 = it2.next();

if(item2.equals(item1)){

inters.addItem(item2, inters.length());

break;

}

}

}

**}**

* 1. **Improved worst case**

We iterate over the first list and compare to all of the elements in the second list. addItem() is only reached on time “per second” loop.

Therefore, the worst case is **O(n2)**

1. **Removing a value from a doubly linked list (10pts)**

**public static DNode removeAllOccurences( DNode** iFirstNode**, char** iChar**){**

// base case

if( iFirstNode == null)

return null;

DNode head = removeAllOcurrences(iFirstNode.next, iChar);

if(iFirstNode.value.equals(iChar)){

return head;

}

else{

// update pointers and return new head

if(head != null)

head.prev = iFirstNode;

iFirstNode.next = head;

return iFirstNode;

}

**}**

1. **Testing for palindromes using a stack (10 pts)**

**Public static bool isPalidrome(String** iWord**){**

// length 1 or empty strings

if(iWords.length() <= 1)

return true;

LLStack<Character> leftStack = new LLStack< Character >();

for(int i = 0; i <= iWord.length()/2; i++)

leftStack.push( iWord.charAt(i) );

for(int i = iWord.length()/2; i< iWord.length(); i++){

if( iWords.charAt(i) != leftStack.pop() )

return false;

}

return true;

**}**