Data Structures in Java - Homework 3

Problem 1

Answer:

```
public static <T> void printLots(List<T> L, List<Integer> P) {

    int targetIdx;

    Iterator<Integer> pIter = P.iterator();
    while (pIter.hasNext()) {

        targetIdx = pIter.next();
        int index = -1;

        Iterator<T> lIter = L.iterator();
        while (lIter.hasNext()) {

            index++;
            T currT = lIter.next();
            if (index != targetIdx) { continue; }
            System.out.println(currT);

        }
    }
}
```

Problem 2

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?
b. What is the running time of `removeFirstHalf` if `lst` is an `ArrayList`?
c. What is the running time of `removeFirstHalf` if `lst` is a `LinkedList`?
d. Does using an iterator make `removeHalf` faster for either type of `List`?

a) **Answer:** `lst.size()` changes during the loop, therefore `theSize` is initialized before the loop as the original `lst.size()/2`.

b) **Answer:** $O(N^2)$

Because `lst.size()` is $O(1)$, the `for` loop is $O(\text{theSize})$, and `lst.remove(0)` is $O(\text{theSize})$ since all elements shift to the front when removing the first.

c) **Answer:** $O(N)$

Because `lst.size()` is $O(N)$, the `for` loop is $O(\text{theSize})$, and `lst.remove(0)` is $O(1)$ since there is no need to shift elements when removing the first.

d) **Answer:** No, you still iterate through the list in $O(N)$ or even more.

Problem 3

Answer:

```
class TwoStacks<T> {
    private T [] myItems;
    private int mySize1;
    private int mySize2;
    private static final int DEFAULT_CAPACITY = 10;

    @SuppressWarnings("unchecked")
    public TwoStacks() {
        myItems = (T[]) new Object[DEFAULT_CAPACITY];
        mySize1 = 0;
        mySize2 = 0;
    }

    @SuppressWarnings("unchecked")
    public void ensureCapacity( int newCapacity )
    {
        if (newCapacity < size1() + size2()) {
            return;
        }

        T [] old = myItems;
        int oldCapacity = old.length;
        myItems = (T []) new Object[ newCapacity ];

        for (int i=0; i<size1(); i++) {
            myItems[i] = old[i];
        }
        for (int i=1; i<=size2(); i++) {
            myItems[newCapacity-i] = old[oldCapacity-i];
        }

    }
```

```
public void push1(T x) {

    if( myItems.length == size1() + size2()) {
        ensureCapacity( myItems.length * 2 + 1 );
    }
    myItems[size1()] = x;
    mySize1++;
}

public void push2(T x) {

    if( myItems.length == size1() + size2()) {
        ensureCapacity( myItems.length * 2 + 1 );
    }
    myItems[myItems.length-size2()-1] = x;
    mySize2++;
}

public T pop1() {
    mySize1--;
    return myItems[size1()];
}

public T pop2() {
    mySize2--;
    return myItems[myItems.length-size2()-1];
}

public T peek1() {
    if (isEmpty1()) { return null; }
    return myItems[size1()-1];
}

public T peek2() {
    if (isEmpty2()) { return null; }
    return myItems[myItems.length-size2()];
}

public boolean isEmpty1() {
    return size1() == 0;
}

public boolean isEmpty2() {
    return size2() == 0;
}
```

```
    public int size1() {
        return mySize1;
    }

    public int size2() {
        return mySize2;
    }
}
```

Problem 4

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.

Answer: $O(N)$ Extra space

```
public static <T> void printReverseNTime(Node<T> head) {
    String output = "";
    Node<T> currNode = head;

    if (head != null) {output += head.data.toString(); }
    while (currNode.next != null) {
        currNode = currNode.next;
        output = String.format("%s\n%s", currNode.data.toString(), output);
    } // Adds each item to the front of the string

    System.out.println(output.strip());
}
```

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.

Answer: $(O(\frac{N(N+1)}{2}) =)O(N^2)$ runtime

```
public static <T> void printReverse1Space(Node<T> head) {

    if (head == null) { return; }

    Node<T> endMarker = null;

    while (head != endMarker) {
        Node<T> currNode = head;
        while (currNode.next != null) {
            if (endMarker == null || currNode != endMarker) {
                currNode = currNode.next;
            } else {
                break;
            }
        }

        System.out.println(currNode.data.toString());
        endMarker = currNode;
    }
}
```