1 Boxes and Pointers

Draw a box and pointer diagram to represent the IntLists L1, L2, and L3 after each statement.

1 \texttt{IntList L1 = IntList.list(1, 2, 3);} \\
2 \texttt{IntList L2 = \texttt{new} IntList(4, L1.rest);} \\
3 \texttt{L2.rest.first = 13;} \\
4 \texttt{L1.rest.rest.rest = L2;} \\
5 \texttt{IntList L3 = IntList.list(50);} \\
6 \texttt{L2.rest.rest = L3;} \\

\textbf{Solution:}

![Diagram showing the boxes and pointers for L1, L2, and L3 after each statement.](image)
2 Partition

Implement partition, which takes in an IntList lst and an integer k, and destructively partitions lst into k IntLists with the following properties:

- It is the same length as the other lists. You may assume the IntList is evenly divisible.
- Its ordering is consistent with the ordering of lst, i.e. items in earlier in lst must precede items that are later.

These lists should be put in an array of length k, and this array should be returned.

For instance, if lst contains the elements 6, 5, 4, 3, 2, 1, and k = 2, then a possible partition, is putting elements [6, 4, 2] at index 0, and elements [5, 3, 1] at index 1.

You may assume you have the access to the method reverse, which destructively reverses the ordering of a given IntList and returns a pointer to the reversed IntList. **Hint:** Think about how to build up the IntList backward at each index, starting with null.

You may not create any IntList instances.

```java
public static IntList[] partition(IntList lst, int k) {
    IntList[] array = new IntList[k];
    int index = 0;
    IntList L = ___________________________________________
    while (L != null) {
        ___________________________________________________
        ___________________________________________________
        ___________________________________________________
        ___________________________________________________
        ___________________________________________________
        ___________________________________________________
        ___________________________________________________
        ___________________________________________________
        ___________________________________________________
        ___________________________________________________
        ___________________________________________________
        ___________________________________________________
        ___________________________________________________
        ___________________________________________________
        ___________________________________________________
        index = ___________________ % __________________;
    }
    return array;
}
```
Solution: [Here] is a video walkthrough of the solution.

```java
public static IntList[] partition(IntList lst, int k) {
    IntList[] array = new IntList[k];
    int index = 0;
    IntList L = reverse(lst);
    while (L != null) {
        IntList prevAtIndex = array[index];
        IntList next = L.rest;
        array[index] = L;
        array[index].rest = prevAtIndex;
        L = next;
        index = (index + 1) % array.length;
    }
    return array;
}
```

Explanation: We reverse our IntList so that we can build up each element of the IntList[] array backwards—in general, it is much easier to build an IntList backward than forward.

The general idea is to initialize each element in the array to null, then put an element of L inside the correct index by assigning array[index] = L. Then, we get whatever we’ve built up so far (prevAtIndex) and add it to the end of our rest element so that we have the entire IntList again with one element at the front.

Afterwards, we advance L to the next element and increment the index.
3 Remove Duplicates

Using the simplified DLList class defined on the next page, implement the removeDuplicates method.

removeDuplicates should remove all duplicate items from the DLList. For example, if our initial list is [8, 4, 4, 6, 4, 10, 12, 12], our final list should be [8, 4, 6, 10, 12]. You may not assume that duplicate items are grouped together, or that the list is sorted!
public class DLList {
    Node sentinel;

    public DLList() {
        // ...
    }

    public class Node {
        int item;
        Node prev;
        Node next;
    }

    public void removeDuplicates() {
        Node ref = ________________________________;
        Node checker;

        while (______________________________) {
            checker = ________________________________;

            while (______________________________) {
                if (______________________________) {

                    Node checkerPrev = checker.prev;
                    Node checkerNext = checker.next;

                    ________________________________;
                    ________________________________;
                    ________________________________;
                    ________________________________;
                    ________________________________;

                    checker = ________________________________;
                } else {
                    ________________________________;
                }
            }

            ref = ________________________________;
        }
    }
Solution:

```java
public void removeDuplicates() {
    Node ref = sentinel.next;
    Node checker;
    while (ref != sentinel) {
        checker = ref.next;
        while (checker != sentinel) {
            if (ref.item == checker.item) {
                Node checkerPrev = checker.prev;
                Node checkerNext = checker.next;
                checkerPrev.next = checker.next;
                checkerNext.prev = checker.prev;
                checker = checkerNext;
            } else {
                checker = checker.next;
            }
        }
        ref = ref.next;
    }
}
```