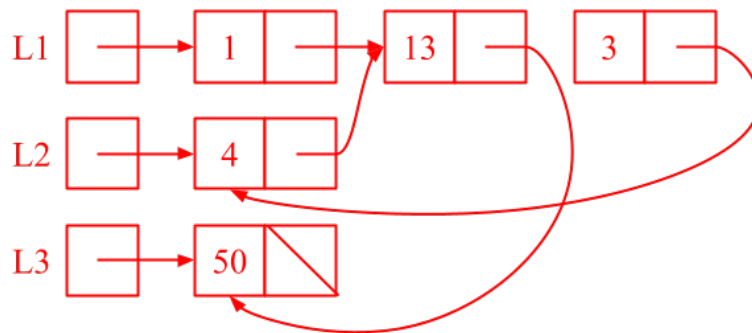# 1   Boxes and Pointers

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

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

Solution:

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.

```
1   public static IntList[] partition(IntList lst, int k) {
2       IntList[] array = new IntList[k];
3       int index = 0;
4       IntList L = _____
5       while (L != null) {
6
7           _____
8
9           _____
10
11          _____
12
13          _____
14
15          _____
16
17          _____
18
19          index = _____ % _____;
20      }
21      return array;
22  }
```

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!

```
1   public class DLList {
2       Node sentinel;
3
4       public DLList() {
5           // ...
6       }
7
8       public class Node {
9           int item;
10          Node prev;
11          Node next;
12      }
13
14      public void removeDuplicates() {
15
16          Node ref = _____;
17          Node checker;
18
19          while (_____) {
20
21              checker = _____;
22
23              while (_____) {
24
25                  if (_____) {
26
27                      Node checkerPrev = checker.prev;
28                      Node checkerNext = checker.next;
29
30                      _____;
31
32                      _____;
33
34                      _____;
35
36                      checker = _____;
37                  } else {
38
39                      checker = _____;
40                  }
41              }
42
43              ref = _____;
44          }
45      }
46  }
```

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;
    }
}
```