4CCS1DST – Data Structures

Exercises for Lecture 5

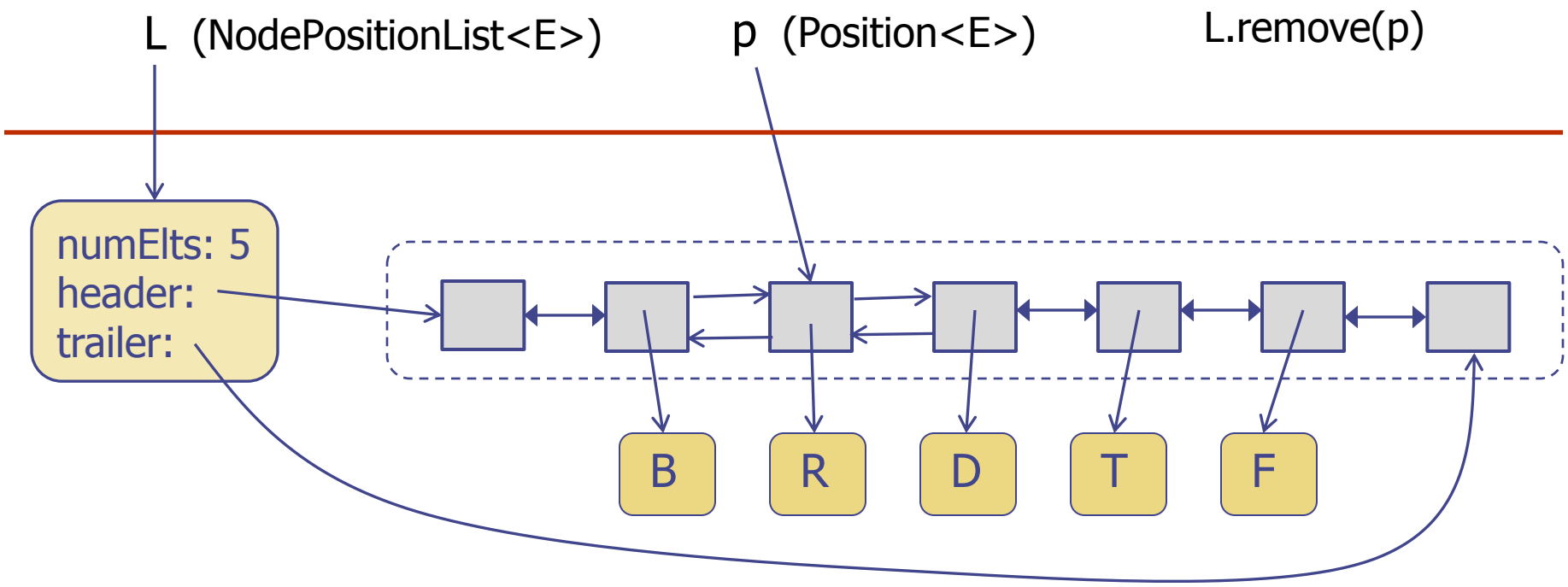
Exercise 1

Give the code for method remove(p) in class NodePositionList:

```
public E  remove(Position<E> p)  throws  InvalidPositionException {

    // give your code here




}
```

Diagram for Exercise 1: Node List implementation

User's application



Node-List implementation

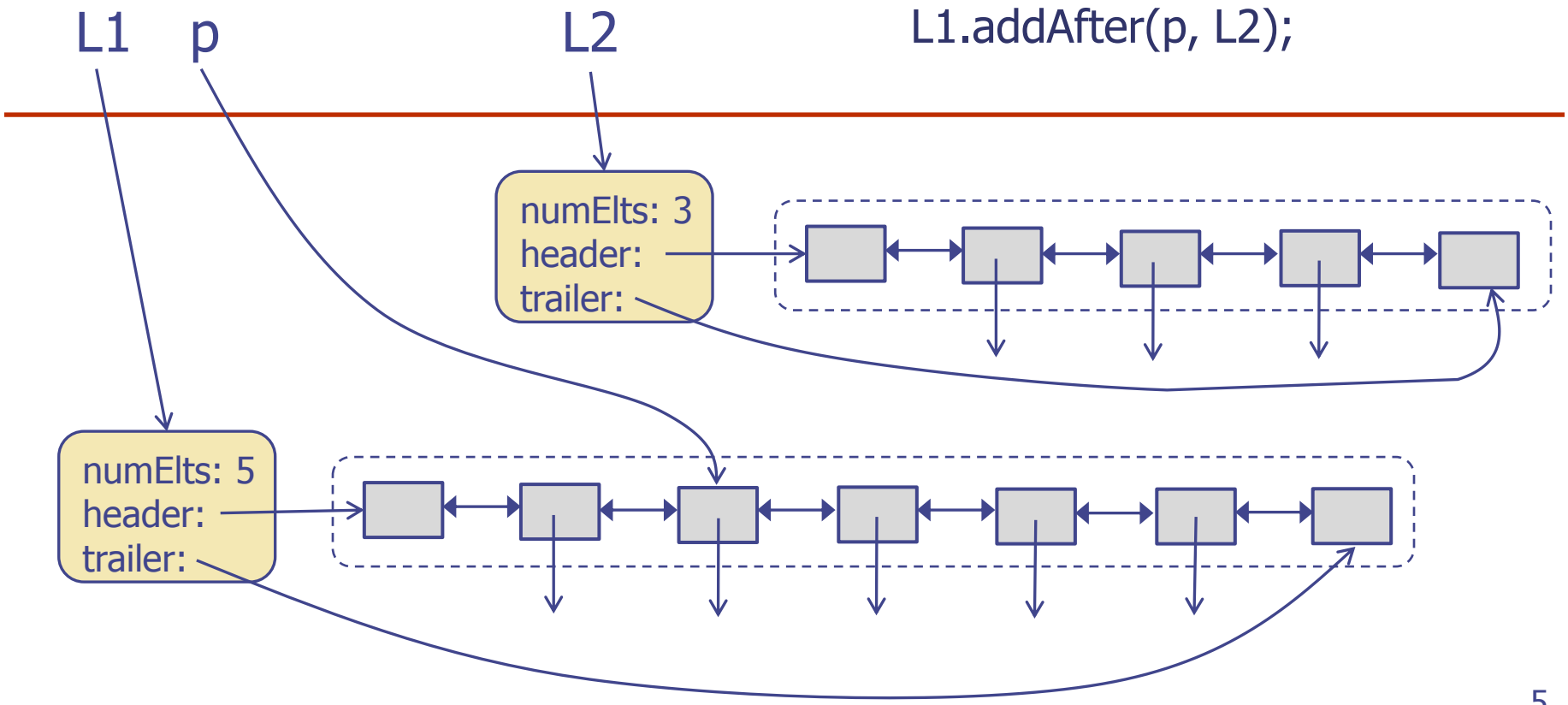
Exercise 2

Give the code for method addAfter(p, L) in the following extension of class NodePositionList. This method inserts the list L to "this" list after the position p. The nodes from L should be put directly into "this" list. (Don't create new nodes.)

```
public class NodePositionListPlus<E> extends NodePositionList<E> {
    public void addAfter(Position<E> p, NodePositionListPlus<E> L)
            throws InvalidPositionException {
        // give your code here
    }
    public static void main(String[] args) {
       NodePositionListPlus<Integer> L1 = new NodePositionListPlus<Integer>();
       NodePositionListPlus<Integer> L2 = new NodePositionListPlus<Integer>();
       L1.addLast(6);  L1.addLast(7);  L2.addLast(1);  L2.addLast(2);
       L1.addAfter(L1.first(), L2);
       System.out.println("L1 has " + L1.size() + " elements: ");
       System.out.println(L1);          // prints: "L1 has 4 elements: [6, 1, 2, 7]"
    }
}
```

Diagram for Exercise 2

```
public void addAfter(Position<E> p, NodePositionListPlus<E> L)
        throws InvalidPositionException { …. }
```



Exercise 2 (cont.)

```
public void addAfter(Position<E> p, NodePositionListPlus<E> L)
                                                    throws InvalidPositionException {
    // give your code here




}
```

Exercise 3

Consider the following class, which uses lists from Java Collections Framework.

```
import  java.util.*;  // we're using lists Java Collections Framework
public  class  ListTester {
    // count even numbers in a sequence of integers
    public  static  int  countEven(List<Integer>  seq) {
        int c = 0;
        for (int i = 0; i < seq.size(); i++) {
            if ( seq.get(i) % 2 == 0 ) {  c++; }
        }
        return c;
    }
    public  static  void  main(String[] args) {
        List<Integer> seqArr = new  ArrayList<Integer>();
        List<Integer> seqLL =  new  LinkedList<Integer>();
        // continues on the next slide
```

Exercise 3 (cont.)

```
for (int i = 0; i < 200000; i++) {
        int n = (int) (Math.random() * 100);
        seqArr.add(n);                  // appends n at the end of the list seqArr
        seqLL.add(n);                   // appends n at the end of the list seqLL
 }
long  startTime  =  System.currentTimeMillis();
int  c1 = countEven(seqArr);
long  elapseTime = System.currentTimeMillis() – startTime;
System.out.println( ...   c1  ...  elapseTime  ... );
... // similarly, run and time  countEven(seqLL)
```

When run on sequences of 200,000 numbers:

count even numbers in array: 99704/200000 [0.007s]
count even numbers in linked list: 99704/200000 [26.971s]

Explain the difference in the running time.
Predict the running times for sequences of 400,000 numbers.