Problem1

```
import java.util.Iterator;
import java.util.List;

public class PrintLots {

	public void printLots(List<Integer> l, List <Integer> p) throws Exception{

		Iterator<Integer> l1 = l.iterator();  //create two iterators
		Iterator<Integer> p1 = p.iterator();
		int x = -1;                  //initial a variable used later

		int b = 0;
		while(p1.hasNext()){
			int a = p1.next();   //first element in p
			int diff = a - b;     //find the distance between position a and b
			if(a >= 0 && a < (l.size())){    //check if a falls in p's range
				for(int i = 0; i < diff; i++){
					x = l1.next();       //iterate to corresponding elements in p
				}
				System.out.println(x);
			}else{
				throw new Exception();
			}
		b = a;      / /set b to the old a
		}
	}
}
```

Problem2 Weiss 3.3:

```
find intersection of lists L1 and L2{

    myList =  new LinkedList;
    set index i1 to 0;
    set index i2 to 0;

    while (i1 < L1's size and i2 < L2's size){

       if (L1.get(i1) < L2.get(i2)){increment i1 by 1}

       if(L1.get(i1) > L2.get(i2)){increment i2 by 1}

       if (L1.get(i1) = L2.get(i2)){
          add L1.get(i1) to myList;
          and increment both i1 and i2 by 1;
       }

    }
     return myList;
}
```

Problem3 Weiss 3.24

```
import java.util.*;

public class ArrayTwoStack<AnyType> {
        int size = n; //set size of array to be n
        int i1 = -1;  //set the initial values for index i1 and i2
        int i2 = n;

        build AnyType Array arr of size n

        public boolean isEmpty1(){  //check if index i1 is at its inital posion
            if(i1 == -1){          //means have no element in the 1st stack
                return true;
            }else{
                return false;
            }
        }

        public boolean isEmpty2(){  //check if index i2 is at inital posion
            if(i2 == n){                //means have no element in the 2nd stack
                return true;
            }else{
                return false;
            }
        }

        public boolean isEmpty(){  //check if both stacks are empty
            if(isEmpty1() && isEmpty2()){  //which means the array is empty
                return true;
            }else{
                return false;
            }
        }

        public boolean isFull(){ //if the indices of two stacks add up to n
             if(i1+i2+1 == n){   //then the entire array is occupied
                 return true;
             }else{
                 return false;
```

```
        }
    }

    public void pop1(){
       if(isEmpty1() == false){
            remove the elemnet on position i1 in stack s1
       }else{
            System.out.println("Stack is empty");
       }
    }

     public void pop2(){
            if(isEmpty2()==false){
                    remove the elemnet on position i2 in stack s2
            }else{
                System.out.println("Stack is empty");
            }
    }

     public void push1(AnyType a){
             if(isFull()==false){      //push element a to top of the stack1
                     arr[++i1] = a;        //if the stack1 is not full
             }else{
                     System.out.println("Stack is full");
             }
     }

     public void push2(AnyType a)  {
             if(isFull()==false){                //push element a to top of the stack2
                     arr[--i1] = a;                      //if the stack1 is not full
             }else{
                     System.out.println("Stack is full");
             }

     }
}
```

Problem 4

(a). input train: back -> [5,9,6,7,2,8,1,3,4] -> front
wanted output train: back -> [9,8,7,6,5,4,3,2,1] -> front

Step 0: input track: [5,9,6,7,2,8,1,3,4] output track: [ ]
S1: bottom-> [ ] ->top S2: bottom-> [ ] ->top S2: bottom-> [ ] ->top

Step 1: move car 4 to holding track S1, then
input track: [5,9,6,7,2,8,1,3] output track: [ ]
S1: [4] S2: [ ] S3:[ ]

Step 2: move car 3 to holding track S1, then
input track: [5,9,6,7,2,8,1] output track: [ ]
S1: [4,3] S2: [ ] S3:[ ]

Step 3: move car 1 to output track, then
input track: [5,9,6,7,2,8] output track: [1]
S1: [4,3] S2: [ ] S3:[ ]

Step 4: move car 8 to holding track S2, then
input track: [5,9,6,7,2] output track: [1]
S1: [4,3] S2: [8] S3:[ ]

Step 5: move car 2 to output track, then
input track: [5,9,6,7] output track: [2,1]
S1: [4,3] S2: [8] S3:[ ]

Step 6: move car 7 to holding track S2, then
input track: [5,9,6] output track: [2,1]
S1: [4,3] S2: [8,7] S3:[ ]

Step 7: move car 6 to holding track S2, then
input track: [5,9] output track: [2,1]
S1: [4,3] S2: [8,7,6] S3:[ ]

Step 8: move car 9 to holding track S3, then
input track: [5] output track: [2,1]
S1: [4,3] S2: [8,7,6] S3:[9]

Step 9: move car 3 at top of holding track S1 to output track, then

input track: [5] output track: [3,2,1]
S1: [4] S2: [8,7,6] S3:[9]

Step 10: move car 4 at top of holding track S1 to output track, then
input track: [5] output track: [4,3,2,1]
S1: [ ] S2: [8,7,6] S3:[9]

Step 11: move car 5 to output track, then
input track: [ ] output track: [5,4,3,2,1]
S1: [ ] S2: [8,7,6] S3:[9]

Step 12: move car 6 at top of holding track S2 to output track, then
input track: [ ] output track: [6,5,4,3,2,1]
S1: [ ] S2: [8,7] S3:[9]

Step 13: move car 7 at top of holding track S1 to output track, then
input track: [ ] output track: [7,6,5,4,3,2,1]
S1: [ ] S2: [8] S3:[9]

Step 13: move car 8 at top of holding track S1 to output track, then
input track: [ ] output track: [8,7,6,5,4,3,2,1]
S1: [ ] S2: [ ] S3:[9]

Step 14: move car 9 at top of holding track S3 to output track, then
input track: [ ] output track: [9,8,7,6,5,4,3,2,1]
S1: [ ] S2: [ ] S3:[ ]

Finished!

(b). The input train back -> [1,9,8,7,6,5,4,3,2] -> front cannot be rearranged in increasing order using 3 holding tracks.

Then, we notice that 1 is the last car in the input train, which means we can only put it to the output train after arranging all other cars to the 3 stacks.

Suppose we have arranged all the other cars and moved car 1 to the output track, then we need car 2 as the next car, and then car 3, and so on. We currently have 8 cars in the 3 holding tracks, which means at least one of the three tracks has more than 2 trains.

Under the best condition, car 2 and car 3 is just at the top of two holding tracks and we can move them to the output track. However, it is impossible for us to get car 4, since if car 2, car 3 are on the top of two holding tracks, then car 4 can only be moved to the empty track and the rest of cars will be put on the top of car 4. Since we cannot move cars between holding tracks, we cannot get car 4 before moving other larger numbered cars. Therefore, we cannot rearrange [1,9,8,7,6,5,4,3,2] in increasing order using 3 holding tracks.