1.

```
public static void printLots(ArrayList<Integer> l, ArrayList<Integer> p) {
        int i = 0; //index for l
        Iterator<Integer> j = p.iterator(); //iterator for p

        for (Integer x : l) {
                if (j.hasNext()){
                        //if the index for p matches element in l
                        if (i == (int) j.next()){
                                System.out.println(x);
                                j.remove();
                        } else {
                                //reset the iterator for l
                                j = p.iterator();
                        }
                }
                i++;
        }
}
```

2. Weiss 3.4 (From book solutions)

```
public static <AnyType extends Comparable<? super AnyType>> void
intersection(List<AnyType> L1, List<AnyType> L2, List<AnyType> Intersect)       {

        ListIterator<AnyType> iterL1 = L1.listIterator();
        ListIterator<AnyType> iterL2 = L2.listIterator();

        AnyType itemL1=null, itemL2=null;

        // get first item in each list
        if ( iterL1.hasNext() && iterL2.hasNext() ){

                itemL1 = iterL1.next();
                itemL2 = iterL2.next();
        }

        while ( itemL1 != null && itemL2 != null ){

                int compareResult = itemL1.compareTo(itemL2);

                if ( compareResult == 0 ) {
```

```
                    Intersect.add(itemL1);
                    itemL1 = iterL1.hasNext() ? iterL1.next() : null;
                    itemL2 = iterL2.hasNext() ? iterL2.next() : null;
                } else if ( compareResult < 0 ){
                    itemL1 = iterL1.hasNext() ? iterL1.next() : null;
                } else {
                    itemL2 = iterL2.hasNext() ? iterL2.next() : null;
                }
        }
}
```

3. TwoStacks.java

4.

a)

- s1.push(4) Output:
- s1.push(3) Output:
- Push 1 to output Output: [1]
- s2.push(8) Output:
- Push 2 to output Output: [2,1]
- Push s1.pop( ) // 3 Output: [3,2,1]
- Push s1.pop( ) // 4 Output: [4,3,2,1]
- s2.push(7)
- s2.push(6)
- s1.push(9)
- Push 5 to putput Output: [5,4,3,2,1]
- Push s2.pop( ) //6 Output: [6,5,4,3,2,1]
- Push s2.pop( ) //7 Output: [7,6,5,4,3,2,1]
- Push s2.pop( ) //8 Output: [8,7 6,5,4,3,2,1]
- Push s1.pop( ) //9 Output: [9,8,7,6,5,4,3,2,1]

b) 1,9,8,7,6,5,4,3,2

In this example, 1 must be inserted before all other elements into the output track, however, moving all other elements out of the way (into the holding tracks) to get to 1 will trap all of the smaller elements we would need to access before the larger elements underneath the larger elements, making them impossible to get to.