/* Nishant Puri
 * np2577
* HW2 Written Questions
 */

Q1

```
	public static <E> void printLots(List<E> L, List<Integer> P) {
		Iterator<E> itr_l = L.iterator();
		Iterator<Integer> itr_p = P.iterator();
		int last = 0;
		//last will keep track of the last index printed
		while (itr_p.hasNext()) {
			int ind = itr_p.next();
			if (ind >= L.size()) {
				System.out.println("Index is out of bounds");
				return;
			}
			for (int i = last; i < ind - 1; i++) {
				itr_l.next();
			}
			System.out.println(itr_l.next());
			last = ind;
		}
	}
```

Q2

```
intersect(List L1, List L2) {
	List L1L2;
	\\i will keep track of which index we are at in List L1
	\\j will keep track of which index we are at in List L2
	int i = 0;
	int j = 0;
	while (i < L1.size() & j < L2.size()) {
		x = L1.get(i);
		y = L2.get(j);
		if (x == y) {
			L1L2.add(x);
			i++;
			j++;
		} else if (x < y) {
			i++;
		} else {
			j++;
		}
	}
	return L1L2;
}
```

Q3

```
public class OneArrayTwoStacks<AnyType> {
	private static int n = 5;
	private AnyType[] A;
	private int head_s1;
	// head_s1 is the index of top element of stack1. As we add more
	// elements it increases(i.e we are moving towards the right in the array)
	private int head_s2;
	// head_s2 is index of top of stack2. As we add more elements to stack2 it
	// decreases(i.e we are moving towards left in the array)

	public OneArrayTwoStacks() {
		A = (AnyType[]) new Object[n];
		head_s1 = -1;
		head_s2 = A.length;
	}

	public void push1(AnyType x) {
		if (head_s1 + 1 == head_s2) {
			System.out.println("Overflow");
			return;
		}
		A[++head_s1] = x;
	}

	public void push2(AnyType x) {
		if (head_s1 == head_s2 - 1) {
			System.out.println("Overflow");
			return;
		}
		A[--head_s2] = x;
	}

	public AnyType pop1() {
		if (isEmpty1()) {
			System.out.println("Stack is empty");
			return null;
		}
		return A[head_s1--];
	}

	public AnyType pop2() {
		if (isEmpty2()) {
			System.out.println("Stack is empty");
			return null;
		}

		return A[head_s2++];

	}

	public AnyType peek1() {
		return A[head_s1];
	}

	public AnyType peek2() {
		return A[head_s2];
	}

	public boolean isEmpty1() {
```

```
            return head_s1 == -1;
        }

        public boolean isEmpty2() {
            return head_s2 == A.length;
        }

        public int size1() {
            return head_s1 + 1;
        }

        public int size2() {
            return A.length - head_s2;
        }

}
```

Q4.

a) Sequence of Steps:

Let IP:Input Track

OP: Output Track

#	Step	Operation
1	4: IP – S1	S1.push(IP.dequeue)
2	3: !P – S1	S1.push(IP.dequeue)
3	1: IP – OP	OP.queue(IP.dequeue)
4	8: IP – S2	S2.push(IP.dequeue)
5	2: IP – OP	OP.queue(IP.dequeue)
6	3: S1 – OP	OP.queue(S1.pop)
7	4: S1 – OP	OP.queue(S1.pop)
8	7: IP – S2	S2.push(IP.dequeue)
9	6: IP – S2	S2.push(IP.dequeue)
10	9: IP – S1	S1.push(IP.dequeue)
11	5: IP – OP	OP.queue(IP.dequeue)
12	6: S2 – OP	OP.queue(S2.pop)
13	7: S2 – OP	OP.queue(S2.pop)
14	8: S2 – OP	OP.queue(S2.pop)
15	9: S1 – OP	OP.queue(S1.pop)

b) Example of train of length 9 that cannot be rearranged in increasing order using 3 holding tracks: 198765432.

Poof:

Lemma1: We can never put a number on a stack that is that is larger than any of the numbers below it on the stack.
Proof: Let if possible we do this. i.e. we put an y on top of a stack which already contains a x<y. Now eventually when we move the cars to output track we need to access x before y. However this is impossible since a stack is LIFO.

Lemma2: We can not put any car on output track before we put car 1.
Proof: We can never move something out of the Output track. Since car 1 needs to eventually be at the front of the output track we can not put any other car on it before car 1.

Thus in the above example we will need to put 2 on S1, 3 on S2 and 4 on S3. Now when we see car number 5 we cant put it on S1,S2 or S3 since it is larger than numbers on the stack. We also can't put it on output track since we haven't put car 1 on it yet. Thus this sequence can not be rearranged