```
import java.util.*;
public void printLots(List<Integer> l, List <Integer> p) throws Exception{
        Iterator<Integer> l1=l.iterator();
        Iterator<Integer> p1=p.iterator();
        int output;
        int a=0;

        while(p1.hasNext()){
                int pvalue=p1.next();
                int d=pvalue-a;//find the difference between pvalue and a
                if(pvalue>=0 && a<l.size()){
                        for(int i=0;i<d;i++){
                                output=l1.next();//# of d's next
                        }
                        System.out.println(outpu);
                }else{
                        throw new Exception();
                }
                a=pvalue;//pass a to the old pvalue
        }
}
```

3.4/

```
list  intersect ( list  L1, list  L2)
list result;
position L1pos= first(L1);
position L2pos= first (L2);
position resultPos= first (result);

while( L1pos != null  && L2pos != null){
        if ( L1pos. data > L2pos. data ){
                L2pos= L2pos. next;
        }else if (L1pos.data < L2pos.data) {
                L1pos= L1pos. next;
        }else{
            add ( result pos, L1pos.data, result)
            L1pos = L1pos. next;
            L2pos = L2pos. next;
            resultPos= resultPos. next;
        }
}.
    return  result;
}.
```

3.24.

```
    int  maxElement = 50;
    stack<AnyType> S= new  stack<AnyType>();
     S.size = maxElement;
     S.top1 = -1                    //initialize  empty  stack
     S.top2 = maxElement            //initialize  empty  stack

  boolean  isEmpty (Stack S,  int  stackNum){
        if (stackNum == 1 ){                   //check  stack 1
              return  S.top1 == -1
           }
      else if (stack Num== 2){              //check  stack 2
            return   S.top2 == S.size
          }

  boolean  isFull (Stack  s){
         return   S.top1+1 = S.top2
     }
```

```
void push (AnyType x, Stack S, int StackNum)
     if (isFull(S)) {
         System.out.println("stack is full")
     }
     if (stacknum == 1){              //push on stack 1
          S.Array[++S.top1] = x;
     }else{
          if (stacknum == 2){            //push on stack 2
               S.Array[--S.top2] = x;
          }
}

void pop (Stack S, int StackNum) {
     if (isEmpty(S, stackNum)) {
          System.out.println ("stack is empty");
     }
     if (stacknum == 1) {
               S.top1 --;
     }else {
          if (stacknum == 2) {
               S.top2 ++;
          }
     }
}
```

back front
4/a). input : [5, 9, 6, 7, 2, 8, 1, 3, 4]

desired output: [9, 8, 7, 6, 5, 4, 3, 2, 1]

step 1: move 4 $^{\text{from input}}$ to the holding track 1

input : [5, 9, 6, 7, 2, 8, 1, 3]

output : [null].

holding track: S_1[4] S_2[null] S_3[null].

step 2: move 3 $^{\text{from input}}$ to the holiding track 2

input : [5, 9, 6, 7, 2, 8, 1]

output : [null]

holding track : S_1[3, 4], S_2[null] S_3[null]

step 3: move 1 $^{\text{from input}}$ to the ouptut track

input : [5, 9, 6, 7, 2, 8]

output : [1]

holding track : S_1 [3, 4], S_2 [null] S_3[null]

step 4. Move 8 $^{\text{from input}}$ to the holding track 2

input : [5, 9, 6, 7, 2]

output : [1]

holding track : S_1[3, 4], S_2[8] S_3[null].

step 5: Move 2 from input to the output

input: [5, 9, 6, 7]

output: [2, 1]

holding track: S_1[3, 4] S_2[8] S_3[null].

step 6. Move 3 from holding track 1 to output

input: [5, 9, 6, 7]

output: [3, 2, 1]

holding track: S_1[4] S_2[8] S_3[null]

step 7. Move 4 from holding track 1 to output

input: [5, 9, 6, 7]

output: [4, 3, 2, 1]

holding track: S_1[null] S_2[8] S_3[null].

step 8. Move 7 from input to holding track 2.

input: [5, 9, 6,].

output: [4, 3, 2, 1]

holding track: S_1[null] S_2[7, 8] S_3[null]

step 9. Move 6 from input to holding track 2.

input: [5, 9]

output: [, 4, 3, 2, 1].

holding track: S_1[null] S_2[6, 7, 8] S_3[null]

step 10. Move 9 from the input track to holding track 1
input: [5]   output [4, 3, 2, 1]
holding track: S1: [9] S2: [6, 7, 8] S3: [null]

step 11. Move 5 from the input track to output
input: [null]   output [5, 4, 3, 2, 1]

step 12. Move 6 from the holding track 2 to output
input: [null]   output: [6, 5, 4, 3, 2, 1]
holding track: S1: [9] S2: [7, 8] S3: [null]

step 13. Move 7 from the holding track 2 to output
input: [null]   output: [7, 6, 5, 4, 3, 2, 1]
holding track: S1: [9] S2 [8] S3 [null]

step 14. Move 8 from the holding track 2 to output
input: [null]. output: [8, 7, 6, 5, 4, 3, 2, 1]
holding track: S1 [9] S2 [null] S3 [null]

step 15. Move 9 from the holding track 1 to output
input: [null]   output: [9, 8, 7, 6, 5, 4, 3, 2, 1]
holding track: S1 [null] S2 [null] S3 [null]

Then, We got out desired output

b). An example for a train of length 9 that cannot be rearrange in increasing order ~~using~~ using 3 holding tracks

[1, 6, 5, 7, 9, 4, 2, 8, 3]

In this case, 1 is the last train
there're no consecutive number in this train.
There's no way to use three holding track to rearrange the train.