1. **Given two linked lists L1 and L2 (each of size n) sorted in an ascending order, describe an algorithm to create a new linked list L that contains all the nodes from L1 and L2 ensuring that L is still sorted. What is the running time of your algorithm?**
   Using two iterators to traverse two linkedlists, so that we can compare two literators, choosing the smaller one, and add it into the end of new linkedlist which we want to return.
   Running time: O(n).
   My algorithm is listed as follows:

```
ListNode itr, head;
if (L1 == null && L2 == null) return null;
else if (L1==null) head = L2;
else if (L2==null) head = L1;
else head = L1.val < L2.val ? L1 : L2;
itr = head;
while (L1 != null || L2 != null){
        if ( L1 == itr)
                L1 = L1.next;
        if ( L2 == itr)
                L2 = L2.next;
        if (L1==null) itr.next = L2;
        else if (L2==null) itr.next = L1;
        else itr.next = L1.val < L2.val ? L1 : L2;
        itr = itr.next;
}
return head;
```

2. **Suppose you are given two circular linked lists, S and T. Describe an algorithm for determining if S and T are really the same list of nodes, but with different starting points (head node). What is the running time of your algorithm?**
   We can use an iterator to traverse S first, find out the one is same with head of T, then use another iterator to traverse T at the same time traversing S. After a loop of traverse, we can find out if S is same with T.
   Running time: O(n).
   My algorithm is listed as follows:

```
ListNode itr1, itr2;
itr1 = sHead;
while(itr1 != tHead && itr1 != null){
        itr1 = itr1.next;
}
if(itr1 == null) return false;
itr2 = tHead;
itr2 = itr2.next;
itr1 = itr1.next;
while(itr2 != tHead){
```

```
        if(itr1 != itr2) return false;
        itr2 = itr2.next;
        itr1 = itr1.next;
}
return true;
```

3. **Postfix notation is an unambiguous way of writing an arithmetic expression without parentheses. It is defined so that if "(exp1) op (exp2)" is a normal fully parenthesized expression whose operation is op, then the postfix version of this is "pexp1 pexp2 op", where pexp1 is the postfix version of exp1 and pexp2 is the postfix version of exp2. The postfix version of a single number or variable is just that number or variable. So, for example, the postfix version of "((5 + 2) * (8 − 3))/4" is "5 2 + 8 3 − * 4 /". Describe an algorithm for evaluating an expression in postfix notation. What is the running time of your algorithm?**

   we can use stack. First we push the number separated by whitespace until we get an operator. when we get an operator, we pop out two elements in stack and using operator on this two elements, getting the result which we push into stack again. And so on. At the end, we pop out the final element in stack to get result.

   The running time of algorithm: O(n).

   My algorithm is listed as follows:

```
int isExpression(String str){
        if str is numeric valid expression return 1;
        if str is operator valid expression return 2;
        else return 0;
}
public static void main(String[] args){
        Stack<Integer> s = new Stack<Integer>();
        Map<String, Runnable> operator = new HashMap<String, Runnable>();
        operator.put("*", () -> { s.push(s.pop()*s.pop());});
        operator.put("+", () -> { s.push(s.pop()+s.pop());});
        operator.put("-", () -> { s.push(s.pop()-s.pop());});
        operator.put("/", () -> { s.push(1/(s.pop()/s.pop()));});
        String[] str = postfix.split(" ");
        for(int i=0;i<str.length;i++){
                if(isExpression(str[i])==1)
                        s.push(str[i]);
                if(isExpression(str[i])==2)
                        operator.get(str[i]).run();
        }
        return stack.pop();
}
```

4. **Consider the algorithm for traversing a maze from Section 15.6.4 Assume that we start at position A and push in the order West, South, East, and North. In which order will the lettered locations of the sample maze be visited?**

A->B->C->B->H->G->L->G->H->I->N->M->N->I->H->P->O->P->Q->J->K

5. **A linked list implementor, hoping to improve the speed of accessing elements, provides an array of Node references, pointing to every tenth node. Then the operation get(n) looks up the reference at position n – n % 10 and follows n % 10 links. a. With this implementation, what is the efficiency of the get operation? b. What is the disadvantage of this implementation?**
   Time complexity: O(1). two steps to access node. First, using O(1) to get access the position n-n%10, then O(1) to find the followed n%10 links.
   disadvantage : using Space complexity: O(n).

Theory:
+3 Q1 algorithm
+2 Q1 running time
+1 Q2 algorithm
- your algorithm will get stuck in an infinite loop if they are not the same circular linked list. The definition of a circular linked list is that it's a circle so there will never be an itr1.next == null. Also, if you do find that it1.next == tHead, you can just return true because if they are the same node then it is guaranteed that the rest of the nodes will be the same since each node can only point to one next node.
+2 Q2 running time
+5 Q3 algorithm
+3 Q3 using a stack
+2 Q3 running time
- very nice. slight bug in your code since you never got the string postfix from anywhere, it would have been better to write your code as a method that takes in a string argument perhaps, but overall it works.
+1 Q4 path A B H P Q R J K (O I N M G L C)
- followed incorrect pattern, you should be pushing in order West, South, East, North as it said in the algorithm, which means that when you are popping off the next node to explore you would see H and all of its following pushes before you get to C.
+1.5 Q5a running time
+1 Q5a explanation
+2.5 Q5b disadvantage
- true, but what else are you losing about expected Big-O of linked list operations if you change the implementation to get O(1) for the get(n) operation?