```java
public class Solution {
  public boolean hasCycle(ListNode head) {
    if(head == null || head.next ==null){
      return false;
    }
    ListNode slow = head;
    ListNode fast =head.next;
    while(fast != null && fast.next != null){
      slow = slow.next;
      fast = fast.next.next;
      if (slow == fast){
        return true;
      }
    }
    return false;
  }
}
```

```java
/**
 * class ListNode {
 *   public int value;
 *   public ListNode next;
 *   public ListNode(int value) {
 *     this.value = value;
 *     next = null;
 *   }
 * }
 */
public class Solution {
  public ListNode insert(ListNode head, int value) {
    ListNode newNode = new ListNode(value);
    //1.determine if the inserted node is before head.
    if(head == null || head.value >= value){
      newNode.next = head;
      return newNode;
    }
    //2. insert the new node to the right postion.
    //using the previous node to traverse the linked list
    // the insert postion of the new node should be between prev and prev.next
    ListNode prev = head;
    while (prev.next != null && prev.next.value < value){
      prev = prev.next;
    }
    newNode.next = prev.next;
    prev.next = newNode;
    return head;
  }
}
```

```java
/**
 * class ListNode {
 *   public int value;
 *   public ListNode next;
 *   public ListNode(int value) {
 *     this.value = value;
 *     next = null;
 *   }
 * }
 */
public class Solution {
  public ListNode merge(ListNode one, ListNode two) {
    if(one == null) {
      return two;
    }
    if (two == null) {
      return one;
    }
    ListNode dummy = new ListNode(0);
    ListNode curr = dummy;
    while (one != null && two != null) {
      if (one.value < two.value) {
        curr.next = one;
        one = one.next;
        curr = curr.next;
      } else {
        curr.next = two;
        two = two.next;
        curr = curr.next;
      }
    }
    if (one != null) {
      curr.next = one;
    } else {
      curr.next = two;
    }
    return dummy.next;
  }
}
```

```java
/**
 * class ListNode {
 *   public int value;
 *   public ListNode next;
 *   public ListNode(int value) {
 *     this.value = value;
 *     next = null;
 *   }
 * }
 */
public class Solution {
  public ListNode partition(ListNode head, int target) {
    if(head== null){
      return null;
    }
    ListNode fakeHeadSmall = new ListNode(0);
    ListNode fakeHeadLarge = new ListNode(0);
    ListNode smallCurr = fakeHeadSmall;
    ListNode largeCurr = fakeHeadLarge;
    ListNode current = head;
    while(current != null){
      if(current.value < target){
        smallCurr.next = current;
        smallCurr= current;
      } else {
        largeCurr.next = current;
        largeCurr = current;
      }
      current = current.next;
    }
    largeCurr.next = null;
    smallCurr.next = fakeHeadLarge.next;
    return fakeHeadSmall.next;
  }
}
```

```java
public class Solution {
  public ListNode findMiddleNode(ListNode head) {
    if (head == null || head.next ==null){
      return head;
    }
    ListNode fast = head;
    ListNode slow = head;
    while (fast != null && fast.next != null){
      slow = slow.next;
      fast = fast.next.next;
    }
    return slow; //针对奇数节点中间值的情况，slow 节点会落在中间点上
  }
}
```

```java
/**
 * class ListNode {
 *   public int value;
 *   public ListNode next;
 *   public ListNode(int value) {
 *     this.value = value;
 *     next = null;
 *   }
 * }
 */
public class Solution {
  public ListNode findMiddleNode(ListNode head) {
    if(head ==null || head.next==null){
      return head;
    }
    ListNode slow = head;
    ListNode fast = head.next;
    while(fast!= null && fast.next != null){
      slow = slow.next;
      fast = fast.next.next;
    }
    return slow;
  }
}
```

```java
public class Solution {
  public ListNode findMiddleNode(ListNode head) {
    if (head== null || head.next ==null){
      return head;
    }
    ListNode slow = head;
    ListNode fast = head.next;
    while(fast!= null && fast.next != null){
      slow = slow.next;
      fast = fast.next.next;
    }
    return slow.next;
  }
}

```

```java
public class Solution {
  public ListNode insertNode(ListNode head, int target) {
    ListNode curr = head;
    ListNode newHead = new ListNode (target);
    while (curr.next != null){
      if(target >= curr.value && target <= curr.next.value){
        ListNode temp = curr.next;
        curr.next = newHead;
        newHead.next = temp;
        return head;
      }
      curr = curr.next;
    }
    return head;
  }
}
```

```java
public class Solution {
  public ListNode insertNode(ListNode head, int target) {
    ListNode curr = head;
    ListNode newNode = new ListNode(target);
    if(curr == null){
      return newNode;
    }
    while (curr.next != null){
      curr = curr.next;
    }
    curr.next = newNode;
    return head;

  }
}
```

```java




class Solution {
    public ListNode middleNode(ListNode head) {
        //边界条件不用忘记处理了
        if(head==null || head.next==null) {
            return head;
        }
        //定义慢指针，快指针
        ListNode low = head;
        ListNode fast = head.next;
        while(fast!=null && fast.next!=null) {
            //慢指针每次走一步，快指针每次走两步
            low = low.next;
            fast = fast.next.next;
        }
        //根据快指针是否为空判断边界条件
        if(fast!=null) {
            return low.next;
        }
        return low;
    }
}
```