

```
1  public class Solution {
2      public boolean hasCycle(ListNode head) {
3          if(head == null || head.next ==null){
4              return false;
5          }
6          ListNode slow = head;
7          ListNode fast =head.next;
8          while(fast != null && fast.next != null){
9              slow = slow.next;
10             fast = fast.next.next;
11             if (slow == fast){
12                 return true;
13             }
14         }
15         return false;
16     }
17 }
```

```
1  /**
2  * class ListNode {
3  *   public int value;
4  *   public ListNode next;
5  *   public ListNode(int value) {
6  *     this.value = value;
7  *     next = null;
8  *   }
9  * }
10 */
11 public class Solution {
12     public ListNode insert(ListNode head, int value) {
13         ListNode newNode = new ListNode(value);
14         //1.determine if the inserted node is before head.
15         if(head == null || head.value >= value){
16             newNode.next = head;
17             return newNode;
18         }
19         //2. insert the new node to the right postion.
20         //using the previous node to traverse the linked list
21         // the insert postion of the new node should be between prev and prev.next
22         ListNode prev = head;
23         while (prev.next != null && prev.next.value < value){
24             prev = prev.next;
25         }
26         newNode.next = prev.next;
27         prev.next = newNode;
28         return head;
29     }
30 }
31
```

```
1  /**
2   * class ListNode {
3   *   public int value;
4   *   public ListNode next;
5   *   public ListNode(int value) {
6   *     this.value = value;
7   *     next = null;
8   *   }
9   * }
10 */
11 public class Solution {
12     public ListNode merge(ListNode one, ListNode two) {
13         if(one == null) {
14             return two;
15         }
16         if (two == null) {
17             return one;
18         }
19         ListNode dummy = new ListNode(0);
20         ListNode curr = dummy;
21         while (one != null && two != null) {
22             if (one.value < two.value) {
23                 curr.next = one;
24                 one = one.next;
25                 curr = curr.next;
26             } else {
27                 curr.next = two;
28                 two = two.next;
29                 curr = curr.next;
30             }
31         }
32         if (one != null) {
33             curr.next = one;
34         } else {
35             curr.next = two;
36         }
37         return dummy.next;
38     }
39 }
40
```

```

1  /**
2  * class ListNode {
3  *     public int value;
4  *     public ListNode next;
5  *     public ListNode(int value) {
6  *         this.value = value;
7  *         next = null;
8  *     }
9  * }
10 */
11 public class Solution {
12     public ListNode partition(ListNode head, int target) {
13         if(head== null){
14             return null;
15         }
16         ListNode fakeHeadSmall = new ListNode(0);
17         ListNode fakeHeadLarge = new ListNode(0);
18         ListNode smallCurr = fakeHeadSmall;
19         ListNode largeCurr = fakeHeadLarge;
20         ListNode current = head;
21         while(current != null){
22             if(current.value < target){
23                 smallCurr.next = current;
24                 smallCurr= current;
25             } else {
26                 largeCurr.next = current;
27                 largeCurr = current;
28             }
29             current = current.next;
30         }
31         largeCurr.next = null;
32         smallCurr.next = fakeHeadLarge.next;
33         return fakeHeadSmall.next;
34     }
35 }
36

```

```
1 public class Solution {
2     public ListNode findMiddleNode(ListNode head) {
3         if (head == null || head.next == null){
4             return head;
5         }
6         ListNode fast = head;
7         ListNode slow = head;
8         while (fast != null && fast.next != null){
9             slow = slow.next;
10            fast = fast.next.next;
11        }
12        return slow; //针对奇数节点中间值的情况，slow 节点会落在中间点上
13    }
14 }
```

```
1  /**
2   * class ListNode {
3   *   public int value;
4   *   public ListNode next;
5   *   public ListNode(int value) {
6   *     this.value = value;
7   *     next = null;
8   *   }
9   * }
10 */
11 public class Solution {
12     public ListNode findMiddleNode(ListNode head) {
13         if(head ==null || head.next==null){
14             return head;
15         }
16         ListNode slow = head;
17         ListNode fast = head.next;
18         while(fast!= null && fast.next != null){
19             slow = slow.next;
20             fast = fast.next.next;
21         }
22         return slow;
23     }
24 }
25
```

```
1 public class Solution {
2     public ListNode findMiddleNode(ListNode head) {
3         if (head== null || head.next ==null){
4             return head;
5         }
6         ListNode slow = head;
7         ListNode fast = head.next;
8         while(fast!= null && fast.next != null){
9             slow = slow.next;
10            fast = fast.next.next;
11        }
12        return slow.next;
13    }
14 }
15
```

```
1  public class Solution {
2      public ListNode insertNode(ListNode head, int target) {
3          ListNode curr = head;
4          ListNode newHead = new ListNode (target);
5          while (curr.next != null){
6              if(target >= curr.value && target <= curr.next.value){
7                  ListNode temp = curr.next;
8                  curr.next = newHead;
9                  newHead.next = temp;
10                 return head;
11             }
12             curr = curr.next;
13         }
14         return head;
15     }
16 }
```



```
1  public class Solution {
2      public ListNode insertNode(ListNode head, int target) {
3          ListNode curr = head;
4          ListNode newNode = new ListNode(target);
5          if(curr == null){
6              return newNode;
7          }
8          while (curr.next != null){
9              curr = curr.next;
10         }
11         curr.next = newNode;
12         return head;
13     }
14 }
15 }
```

```
1
2
3
4
5 class Solution {
6     public ListNode middleNode(ListNode head) {
7         //边界条件不用忘记处理了
8         if(head==null || head.next==null) {
9             return head;
10        }
11        //定义慢指针，快指针
12        ListNode low = head;
13        ListNode fast = head.next;
14        while(fast!=null && fast.next!=null) {
15            //慢指针每次走一步，快指针每次走两步
16            low = low.next;
17            fast = fast.next.next;
18        }
19        //根据快指针是否为空判断边界条件
20        if(fast!=null) {
21            return low.next;
22        }
23        return low;
24    }
25 }
```