Note: Code snippets shown below describe some of the many ways the problems can be solved. As long as your code does what is asked and is efficient, your answer is also correct.

1)
```
public E remove(int pos) {
            if (pos<0 || pos>numItems-1) throw new IndexOutOfBoundsException();

            if (pos == 0){
                  if (head == null) throw new IndexOutOfBoundsException();
                  E obj = head.getData();
                  head = head.getNext();
                  if (numItems == 1) tail = head;f
                  numItems--;
                  return obj;
            }
            ListNode<E> toRemove,prev;
            prev = null; toRemove = head;
            for (int i = 0; i < pos; i++){
                  prev = toRemove;
                  toRemove = toRemove.getNext();
            }
            if (pos == numItems - 1) tail = prev;
            prev.setNext(toRemove.getNext());
            numItems--;
            return toRemove.getData();
      }
```
Runtime: O(n)

2)
```
public ListNode<E> append(ListNode<E> first, ListNode<E> second){
        ListNode<E> tmp1 = first;
        while(tmp1.getNext() != null){
                tmp1 = tmp1.getNext();
        }
        //tmp1 points to last node
        tmp1.setNext(second.getNext());
        return first;
}
```

This method always goes through every node in the first list O(n)

3)
```
public E secondLast () throws NoSuchElementException{
        // Handle base cases : empty list and single - element list
        if ( head == null || head.getNext () == null ) {
                throw new NoSuchElementException() ;
```

```
    }
        ListNode <E > curr = head ;
        while(curr.getNext().getNext() != null){
        curr = curr.getNext () ;
    }
    return curr.getData () ;
}
```

4)

```
public boolean isPalindrome () {
    // Handle trivial cases : empty and single - element lists
    if ( numItems == 0 || numItems == 1) {
        return true ;
    }
    // Traverse to end , then loop over half of elements , comparing the
    // two on either side of the middle each time .
    ListNode<E> elemR = head;
    while(elemR.getNext() != null){
        elemR = elemR.getNext();
    }
    ListNode <E > elemL = head . getNext () ;
    boolean foundMismatch = false ;
    for ( int i = 0; i < numItems /2; i ++) {
        if ( elemR . getData () != elemL . getData () ) {
            foundMismatch = true ;
            break ;
        }
        // Advance toward the middle .
        elemR = elemR . getPrev () ;
        elemL = elemL . getNext () ;
    }
    return !foundMismatch ; // If there was a mismatch , it ’s not a palindrome .
}
```

Complexity: O(n)