1. cons(4,cons(6,cons(8,cons(5,cons(3,[])))))

2. i. tail(tail([4,6,8,5,3]))
   ii. value(tail(tail(tail([4,6,8,5,3]))))
   iii. tail(tail(tail(tail([4,6,8,5,3]))))
   iv. cons(value([4,6,8,5,3]), cons(value(tail([4,6,8,5,3])),tail(tail(tail([4,6,8,5,3])))))

3.

```
Algorithm: isSingle(list)
Requires: a nonempty list
Returns: True if list has single element; False otherwise
1:  if isEmpty(tail(list))
2:      return True
3:  else
4:      return False
5:  endif
```

4.

```
Algorithm: maxList(L)
Requires: a nonempty list L
Returns: largest element of the list
1:  return maxListHelper( value(L),tail(L) )
```

```
Algorithm: maxListHelper(ref_Val, list)
Requires: a number and a nonempty list
Returns: largest element of the list
1:  if isEmpty(list)
2:      return ref_Val
3:  elseif value(list)<ref_Val
4:      return maxListHelper(ref_Val, tail(list) )
5:  else
6:      return maxListHelper(value(list), tail(list) )
7:  endif
```

Q6, Q7, Q9: refer to Seminar 4. Q8: refer to Lecture 5. You should also trace these algorithms using either example given in the questions or your own test cases.

Q11: Idea of reversing a list:
Similar to the **insertionSort** algorithm in Lecture 5, initialize two lists in the helper function (so you have extra room for operating the list).

- Initially set leftList as original list, and rightList as an empty list. (This can be done by calling the helper function from main algorithm with suitable input arguments.)
- Then repeat the process: getting the front element from leftList and putting it into the front of rightList. (This should be done by a recursive call to your helper function.)
- When the leftList is empty, the rightList will be in reversed order.

Complete the pseudocode and come to your tutor if you need any help.