Data Structures:

Linked list:

Sorted Merge:

Given two sorted linked lists merge them in place.

struct node\* result = NULL;

  /\* Base cases \*/

  if (a == NULL)

     return(b);

  else if (b==NULL)

     return(a);

  /\* Pick either a or b, and recur \*/

  if (a->data <= b->data)

  {

     a->next = SortedMerge(a->next, b);

return a;

  }

  else

  {

     b->next = SortedMerge(a, b->next);

return b;

  }

Floyd cycle detection proof:  
Let x be the distance from the start of the loop. And y be the distance from loop start to point where 2 pointers meet and z be the remaining measurement of cycle.

Suppose fast pointer has run over ‘m’ cycles before meeting the slow pointer. ‘i’ is the distance travelled by slow pointer and ‘2i’ is covered by fast pointer.

We can write the equations as:

i = x + y

2i = x + m(y+z) + y

This gives us:

2\*(x + y) = x + m(y + z) + y

x = (m-1)(y+z) + z

Hence, if we keep slow pointer at x start moving it, fast pointer would have cycled ‘m-1’ times with meeting slow pointer right at loop beginning. The same can be used to find the point of loop start so as to remove the loop from the linked list.

/\* If loop exists \*/

    if (slow == fast)

    {

        slow = head;

        while (slow != fast->next)

        {

            slow = slow->next;

            fast = fast->next;

        }

        /\* since fast->next is the looping point \*/

        fast->next = NULL; /\* remove loop \*/

    }

To reverse a list after every ‘k’ nodes. Use recursion over complete list and loop for reversal of k nodes.

**Delete nodes which have a greater value on right side**

To delete nodes in linked list which has its larger element on right, reverse the linked list. Take max = head->data and start looping. If cur\_node > max then max = cur\_node else delete the node.