Remove duplicates in a sorted list

```
void remove_duplicates(struct node* source);
```

1->2->3->3->3->5->NULL becomes 1->2->3->5->NULL

Remember to free the node which you remove from the list.

REMOVE DUPLICATES IN A SORTED LIST

```c
void remove_duplicates(struct node* source) {
  if (source == NULL) {
    return;
  }

  struct node* curr = source->next;
  struct node* prev = source;

  while (curr != NULL) {
    if (curr != NULL && curr->data == prev->data) {
      prev->next = curr->next;
      curr->next = NULL;
      free(curr);

      curr = prev->next;
      continue;
    }
    prev = curr;
    curr = curr->next;
  }
}
```

TWO POINTERS ONE WALKING BEHIND THE OTHER, THESE ARE USED TO CHECK FOR DUPLICATES

THE ACTUAL DUPLICATE CHECK, THERE MIGHT BE MULTIPLE DUPLICATES SO KEEP MOVING CURR TILL YOU GET ALL THE DUPLICATES AND FREE THE NODES

WALK THE ENTIRE LIST TO FIND DUPLICATES

Move node from the head of one list and add to the front of another

```
void move_node(struct node** sourceRef, struct node** destRef);
```

For source

source: 0->2->4->6->NULL
dest: 1->2->3->5->NULL

the function should produce this output

source: 2->4->6->NULL
dest: 0->1->2->3->5->NULL

MOVE NODE

```
void move_node(struct node** sourceRef, struct node** destRef) {
  assert(sourceRef != NULL && destRef != NULL);
  if (*sourceRef == NULL) {
    return;
  }

  struct node* node_to_move = *sourceRef;
  *sourceRef = (*sourceRef)->next;
  node_to_move->next = *destRef;
  *destRef = node_to_move;
}
```

HANDLE THE CASE WHEN THE SOURCE LIST IS EMPTY

POINT SOURCE TO THE SECOND ELEMENT IN ITS LIST

THE FIRST ELEMENT FROM SOURCE BECOMES THE FIRST ELEMENT IN THE DEST LIST

Merge two sorted lists to get a new sorted list with all elements from both lists

```
struct node* sorted_merge(struct node* a, struct node* b);
```

Assume the lists are sorted in ascending order and the resultant list is also in ascending order

For input

a: 0->2->4->6->NULL
b: 1->2->3->5->NULL

return

0->1->2->2->3->4->5->6->NULL

NOTE: There are many special cases in this one: when one list is null, getting the very first element in the

SORTED MERGE

```
struct node* sorted_merge(struct node* a,
                          struct node* b) {
  if (a == NULL) {
    return b;
  } else if (b == NULL) {
    return a;
  }

  struct node* head;
  if (a->data < b->data) {
    head = a;
    a = a->next;
  } else {
    head = b;
    b = b->next;
  }
  head->next = NULL;

  struct node* curr = head;
  while(a != NULL && b != NULL) {
    if (a->data < b->data) {
      curr->next = a;
      a = a->next;
    } else {
      curr->next = b;
      b = b->next;
    }
    curr = curr->next;
  }

  if (a != NULL) {
    curr->next = a;
  } else {
    curr->next = b;
  }

  return head;
}
```

IF ONE LIST IS NULL THEN JUST RETURN THE OTHER ONE, SINCE THEY ARE BOTH SORTED IT JUST WORKS

THE CASE TO GET THE FIRST NODE IS SLIGHTLY DIFFERENT FROM THE BODY OF THE WHILE LOOP

COMPARE THE FIRST ELEMENT OF EACH LIST FOR EVERY ITERATION AND PICK THE SMALLER ELEMENT TO GET THE RESULT IN ASCENDING ORDER

WHEN ONE LIST RUNS OUT MAKE SURE THE REMAINING ELEMENTS OF THE OTHER ARE APPENDED TO THE END OF THE RESULT

Reverse a linked list

```
struct node* reverse(struct node* list);
```

This can be done iteratively rather than recursively, try it that way!

For input

list: 0->2->4->6->NULL

return

6->4->2->0->NULL

NOTE: Setting up the next pointers correctly in this one is tricky, watch out!

REVERSE

```
struct node* reverse(struct node* list) {
  if (list == NULL || list->next == NULL) {
    return list;
  }

  struct node* curr = list->next;
  struct node* prev = list;
  prev->next = NULL;

  while (curr != NULL) {
    struct node* next = curr->next;

    curr->next = prev;
    prev = curr;
    curr = next;
  }

  return prev;
}
```

EMPTY OR SINGLE ELEMENT LISTS ARE EASY - JUST DEAL WITH THEM UPFRONT!

SET UP TWO POINTERS WALKING THE LIST ONE BEHIND THE OTHER

STORE THE NEXT ELEMENT BEFORE DOING ANY POINTER SWAPPING SO WE CAN CONTINUE WALKING THE LIST

THIS IS TRICKY - AT EACH ITERATION GET THE CURRENT ELEMENT TO POINT TO THE PREVIOUS I.E. SECOND ELEMENT POINTS TO THE FIRST, THIRD TO THE SECOND ETC

THE LAST ELEMENT IS NOW THE NEW HEAD OF THE REVERSED LIST

Summing up

Pointers are tricky, practice and practice again to help you visualize these easily.

String functions and linked lists are favorites with C interviewers, these also form the foundation of more complex interview questions.

Just make sure that you can nail these before moving on to more difficult topics.

Happy interviewing!