**Algorithm to insert node to end of singly linked list (SLL):**

**O(n)**

1. Create new node *n*, set *n*.next to null
2. If the SLL is empty:
   1. Set *n* as SLL head
   2. Return
3. Else, if elements are present in the SLL:
   1. Traverse till the last node in the SLL
   2. Change the next of the last node in SLL to *n*
   3. Return

**Algorithm to delete alternate nodes in a singly linked list (SLL):**

**O(n)**

1. If the SLL is empty, return
2. Else, initialise previous node and current node to the first and second nodes in the list
3. While neither the previous node or current node are empty, repeat:
   1. Change the next link of the previous node to the next link of the current node freeing the current node
   2. Empty the contents of current node
   3. Traverse updating the previous and current nodes to the next set of nodes in the list
4. Return

**Algorithm to merge three sorted singly linked lists (SLL):**

**O(n^2)**

1. Merge two SLL using helper function (mergeTwo):
   1. Create new empty SLL, *temp*
   2. Initialise first node of list *a* and list *b*
   3. While *a* and *b* are not empty, repeat:
      1. If *a* < *b* add *a* to end of *temp,* set *a* to *a*.next
      2. Else add *b* to end of *temp*, set *b* to *b*.next
   4. While *a* is not empty, repeat:
      1. Add *a* to end of *temp*, set a to *a*.next
   5. While *b* is not empty, repeat:
      1. Add *b* to end of *temp*, set a to *b*.next
   6. Return temp
2. Create new empty linked list, *L3*
3. Set *L3* to result mergeTwo (*L, L1*)
4. Set *L3* to result mergeTwo (*L2, L3*)
5. Remove any duplicates from *L3* using helper function (removeDups):
   1. Initialise new node *current* as *list*.first
   2. While current is not empty:
      1. Initialise new node *counter* as *current*
      2. While counter is not empty and *counter* equals *current*, set *counter* to *counter.*next
   3. Set *curren*t.next to *counter*
   4. Set *current* to *curren*t.next
   5. Return
6. Return *L3*