# Solve the below problems in C/C++ and give the time and space complexities

## Write Algorithm/Pseudocode for following operations on a Singly Linked Lists

* 1. Inserting a new node in a Linked List at the beginning
     + Start
     + InsertAtBeg(value)
     + Create a new node using CreateNode(value)
     + Set NewNode -> next = head
     + Set head = NewNode
     + End
  2. Inserting a new node in a Linked List at the end.
     + Start
     + InsertAtEnd(value)
     + Create a new node using CreateNode(value)
     + If head is NULL, set head = NewNode
     + Else set ptr = head
     + Iterate LL using while ptr -> next != NULL and set ptr = ptr -> next
     + Set ptr -> next = NewNode
     + End
  3. Inserting a new node in a Linked List after a given node.
     + InsertAfterNode(value,before)
     + Set ptr = head
     + Iterate LL using while ptr != NULL and ptr -> data != before, set ptr = ptr ->
     + If ptr = NULL, return
     + Create a new node using CreateNode(value)
     + Set NewNode -> next = ptr -> next
     + Set ptr -> next = NewNode
     + End
  4. Inserting a new node in a Linked List before a given node
     + Start
     + InsertBeforeNode(value,after)
     + Create a new node using CreateNode(value)
     + If head == null, set head = NewNode
     + Else set ptr = head
     + Iterate LL using while ptr -> next != NULL & ptr -> next -> data != before, set ptr = ptr -> next
     + Set NewNode -> next = ptr -> next
     + Set ptr -> next = NewNode
     + End
  5. Deleting a node from a Linked List from beginning
     + Start
     + DeleteFromBeg()
     + If head == NULL, return
     + Set ptr = head
     + Set head = head -> next
     + Free ptr
     + End
  6. Deleting a node from a Linked List from end
     + Start
     + DeleteFromEnd()
     + If head == NULL, return
     + Set ptr = head
     + Iterate LL using while ptr -> next -> next != NULL, set ptr = ptr -> next
     + Free ptr -> next
     + Set ptr -> next = NULL
     + End
  7. Deleting a node after a given node
     + Start
     + DeleteAfterNode(val)
     + Set ptr = head
     + Iterate LL using while ptr != NULL & ptr -> data != val, set ptr = ptr -> next
     + If ptr = NULL or ptr -> next = NULL, return
     + Set delete = ptr -> next
     + Set ptr -> next = delete -> next
     + Free delete
     + End
  8. Display the entire linked list (Traversal)
     + Start
     + Display()
     + Set ptr = head
     + Iterate LL using while ptr != NULL, print ptr -> data and set ptr = ptr -> next
     + End
  9. Search for an element in linked list.
     + Start
     + Search(value)
     + Set ptr = head, count = 1
     + Iterate LL using while ptr != NULL
       1. If ptr -> data == value, print pos and return
       2. Set ptr = ptr -> next, pos++
     + End
  10. Count the number of nodes in a Singly Linked List
      + Start
      + CountNodes()
      + Set count = 0, ptr = head
      + Iterate LL using while ptr != NULL, count++ and ptr = ptr -> next
      + Print count
      + End
  11. Reverse the singly linked list.
      + Start
      + ReverseLL()
      + Iterate LL using while current != NULL
        1. Set next = current -> next
        2. Set current -> next = previous
        3. Set previous = current
        4. Set current = next
      + Set head = previous
      + End

## Do an Apriori analysis on the Time and Space complexities of above algorithms on a Singly Linked List.

* 1. TC = O(1) SC = O(1)
  2. TC = O(n) SC = O(1)
  3. TC = O(n) SC = O(1)
  4. TC = O(n) SC = O(1)
  5. TC = O(1) SC = O(1)
  6. TC = O(n) SC = O(1)
  7. TC = O(n) SC = O(1)
  8. TC = O(n) SC = O(1)
  9. TC = O(n) SC = O(1)
  10. TC = O(n) SC = O(1)
  11. TC = O(n) SC = O(1)

## How can you optimise the insertion of an element at the end. Discuss.

We can optimize the insertion of an element at the end of a LL by using a tail pointer just like a head pointer

Then the time complexity of that operation will be reduced to O(1) as we no longer need to traverse through whole LL

## Discuss about why it’s not possible to delete a node before a given node

It is not possible to delete a node before a given node as we cannot move backwards in a singly LL and if we traverse through a LL and delete a particular node coming before a node then the node coming after deleted node will be lost.