**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.

**To create a linked list in which children are 2\*i+1 and 2\*i+2**

Use queue and go for level order traversal while traversing through the list in parallel.

**Create BST from linked list**

Method 1:

* Get the middle of linked list, make it as root

Do

* root->left = createBST(0, mid-1); root->right = createBST(mid+1, n);

**Method 2:**

struct Node\* sortedListToBSTRecur(struct Node \*\*head\_ref, int n)

{

    /\* Base Case \*/

    if (n <= 0)

        return NULL;

    /\* Recursively construct the left subtree \*/

    struct Node \*left = sortedListToBSTRecur(head\_ref, n/2);

    /\* head\_ref now refers to middle node, make middle node as root of BST\*/

    struct Node \*root = \*head\_ref;

    // Set pointer to left subtree

    root->prev = left;

    /\* Change head pointer of Linked List for parent recursive calls \*/

    \*head\_ref = (\*head\_ref)->next;

    /\* Recursively construct the right subtree and link it with root

      The number of nodes in right subtree  is total nodes - nodes in

      left subtree - 1 (for root) \*/

    root->next = sortedListToBSTRecur(head\_ref, n-n/2-1);

    return root;

}

**To find a triplet in 2 list whose sum is equal to a given number:**

Let the lists be a, b and c. Sort ‘b’ in ascending order and ‘c’ in descending order . Start traversing a, while(a != NULL)

If(sum < requisite)

B=b->next

Else

C=c->next

**To flatten linked list having both right and down pointers. Recurse the merge by merging down lists of root and root->right**

// The main function that flattens a given linked list

Node\* flatten (Node\* root)

{

    // Base cases

    if (root == NULL || root->right == NULL)

        return root;

    // Merge this list with the list on right side

    return merge( root, flatten(root->right) );

}

**Flatten a multilevel linked list**

1) Take "cur" pointer, which will point to head of the fist level of the list

2) Take "tail" pointer, which will point to end of the first level of the list

3) Repeat the below procedure while "curr" is not NULL.

I) if current node has a child then

a) append this new child list to the "tail"

tail->next = cur->child

b) find the last node of new child list and update "tail"

tmp = cur->child;

while (tmp->next != NULL)

tmp = tmp->next;

tail = tmp;

II) move to the next node. i.e. cur = cur->next

**Sort list of 0’s, 1’s and 2’s**

**Approach 1:** Count number of 0, 1, and 2 and fill them**.**

**Approach 2 :** Segregate lists of 0, 1 and 2 and then merge them.

**Reverse alternate nodes and append to the end of list**

**Approach 1:** Separate the 2 lists. Reverse the 2nd and append to first.

**Approach 2:** Use a single loop to carry out the task. Below is the solution

while (odd && odd->next)

    {

       // Store the next node in odd list

       struct node \*temp = odd->next->next;

       // Link the next even node at the beginning of even list

       odd->next->next = even;

       even = odd->next;

       // Remove the even node from middle

       odd->next = temp;

       // Move odd to the next odd node

       if (temp != NULL)

         odd = temp;

    }

**To clone a linked list containing the random pointers**

In this, first store the next pointer of the original list in array and point them to the corresponsing element of the new linked list. Also, point the random pointer of new list to point to paralleled node of original list. After that do :

clone->arbit = clone->arbit->arbit->next;

And restore the original list after looking into the array.

Another approach could be to add the cloned list in interleaved original list. After that, do:

original->next->arbitrary = original->arbitrary->next;

and restore:

original->next = original->next->next;

copy->next = copy->next->next;

Another approach could be to maintain a hashmap of addresses of original and duplicated list. Traverse through the original list. Search its arbitrary pointer in O(1) time and point the arbitrary pointer of duplicated node to its adjacent one in map.

**Make arbitrary pointer in list to point to next highest node**

Copy the next pointer to arbitrary pointer and run mergesort based on arbitrary pointer.

**To sort a linked list that is sorted alternating ascending and descending orders**

1. Separate two lists.

2. Reverse the one with descending order

3. Merge both lists.