1. Mirror tree:
2. Start swapping from bottom, left and right subtrees.
3. Convert sorted array to BST.
   1. take middle element as root, and recurs for left and right subarray.
4. Convert sorted doubly list to BST

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

if (n <= 0)

return NULL;

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

struct Node \*root = \*head\_ref;

root->prev = left;

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

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

return root;

}

1. Convert sorted linked list to Balanced BST.
   1. same as 4.
2. Print all nodes at distance k from a given node
   1. Return distance of target to ancestors, and recurs for alternate subtrees. Return -1 to nodes if the target doesn’t lie in their either subtrees
3. Construct *complete binary tree* from its linked list representation.
   1. use queue method
4. Convert a binary tree to doubly linked list.
5. Print right view of binary tree

void rightViewUtil(struct Node \*root, int level, int \*max\_level)

{

    // Base Case

    if (root==NULL)  return;

    // If this is the last Node of its level

    if (\*max\_level < level)

    {

        printf("%d\t", root->data);

        \*max\_level = level;

    }

    // Recur for right subtree first, then left subtree

    rightViewUtil(root->right, level+1, max\_level);

    rightViewUtil(root->left, level+1, max\_level);

}

1. Largest sum path between two leaves.
2. Find whether a tree is symmetric tree.
   1. give arguments (root1, root2).

Array questions:

# Search an element in a sorted and rotated array.

# Fix array[mid] position and then make cases.

# Duplicates in an array in O(n) and by using O(1).

# Count Inversions of size three in a given array.

# Graphs:

# Longest path between any pair of vertices.