**Add First in LL**

public void addFirst(int val) {

Node newnode=new Node();

newnode.data=val;

if(size==0){

head=newnode;

tail=newnode;

}

else{newnode.next=head;

head=newnode;

}

size++;

}

}

**Add Last in LL**

void addLast(int val) {

Node newnode=new Node(vale);

tail.next newnode;

tail=newnode;

size++;

}

}

**Add at Index in LL**

public void addAt(int idx, int val){

if(idx<0 || idx>size){

System.out.println("Invalid arguments");

}

else if(idx==0){

addFirst(val);

}

else if(idx==size){

addLast(val);

}

else{

Node newnode= new Node();

newnode.data=val;

Node prev=head;

for(int i=0;i<idx-1;i++){

prev=prev.next;

}

newnode.next=prev.next;

prev.next=newnode;

size++;

}

}

}

**Remove Last in LL**

public void removeLast(){

Node s=head;

Node f=head;

f=f.next;

while(f!=tail){

s=s.next;

f=f.next;

}

s.next=null;

tail=s;

size--;

}

}

**Reverse a LL**

public void reversePI(){

Node p=null;

Node c=head;

while(c!=null){

Node n=c.next;

c.next=p;

p=c;

c=n;

}

Node temp=head;

head=tail;

tail=temp;

}

}

**Kth Element from the end in LL**

public int kthFromLast(int k){

Node s=head;

Node f=head;

for(int i=0;i<k;i++){

f=f.next;

}

while(f!=tail){

s=s.next;

f=f.next;

}

return s.data;

}

}

**Middle of LL**

public int mid(){

Node s=head;

Node f=head;

while(f.next!=null && f.next.next!=null){

s=s.next;

f=f.next.next;

}

return s.data;

}

}

**Merge two Sorted LL**

public static LinkedList mergeTwoSortedLists(LinkedList l1, LinkedList l2) {

LinkedList res=new LinkedList();

Node t1=l1.head;

Node t2=l2.head;

while(t1!=null && t2!=null){

if(t1.data < t2.data){

res.addLast(t1.data);

t1=t1.next;

}

else{

res.addLast(t2.data);

t2=t2.next;

}

}

while(t1!=null){

res.addLast(t1.data);

t1=t1.next;

}

while(t2!=null){

res.addLast(t2.data);

t2=t2.next;

}

return res;

}

}

**Merge Sort a LL**

public static LinkedList mergeSort(Node head, Node tail) {

if (head == tail) {

LinkedList br = new LinkedList();

br.addLast(head.data);

return br;

}

Node mid = midNode(head, tail);

LinkedList fsh = mergeSort(head, mid);

LinkedList ssh = mergeSort(mid.next, tail);

LinkedList sl = mergeTwoSortedLists(fsh, ssh);

return sl;

}

}

**K Reverse a LL**

Public static Node reverse(Node head, int k) {

if(head == null)

return null;

Node current = head;

Node next = null;

Node prev = null;

int count = 0;

while (count < k && current != null) {

next = current.next;

current.next = prev;

prev = current;

current = next;

count++;

}

if (next != null){

head.next = reverse(next, k);

}

return prev;

}

**Is a LL Palindrome**

public static boolean isPalindrome(Node head){

Node s = head;

boolean ispalin = true;

Stack<Integer> stack = new Stack<Integer>();

while (s != null) {

stack.push(s.data);

s = s.next;

}

while (head != null) {

int i = stack.pop();

if (head.data == i) {

ispalin = true;

}

else {

ispalin = false;

break;

}

head = head.next;

}

return ispalin;

}

}

**Fold a ll**

void rearrange(Node head)

{

Node slow= midpoint(head)

Node node1 = node;

Node node2 = slow.next;

slow.next = null;

node2 = reverselist(node2);

head = new Node(0);

Node curr = head;

while (node1 != null || node2 != null) {

if (node1 != null) {

curr.next = node1;

curr = curr.next;

node1 = node1.next;

}

if (node2 != null) {

curr.next = node2;

curr = curr.next;

node2 = node2.next;

}

}

head = head.next;

}

**Add two LL**

private static int addhelper(Node one, int pv1, Node two, int pv2, LinkedList res) {

if (one == null && two == null) {

return 0;

}

int sum = 0;

if (pv1 > pv2) {

int oc = addhelper(one.next, pv1 - 1, two, pv2, res);

sum = one.data + oc;

}

else if (pv2 > pv1) {

int oc = addhelper(one, pv1, two.next, pv2 - 1, res);

sum = two.data + oc;

}

else {

int oc = addhelper(one.next, pv1 - 1, two.next, pv2 - 1, res);

sum = one.data + two.data + oc;

}

int c = sum / 10;

int d = sum % 10;

res.addFirst(d);

return c;

}

**Intersection Point of LL**

public Node getIntersectionNode(Node head1, Node head2)

{

while (head2 != null) {

Node temp = head1;

while (temp != null) {

if (temp == head2) {

return head2;

}

temp = temp.next;

}

head2 = head2.next;

}

return null;

}