* Linked List Extension Sheet Coding

1. public boolean isSorted() {

Node<T> curr =head;

boolean sort = true;

if(curr == null) {

throw Exception;

}

if (curr.next == null) {

sort = true;

} else {

While (curr.next != null) {

If (curr.getData > curr.getNext.getData) {

Sort = false;

Break;

}

curr = curr.getNext;

}

}

Return sort;

}

**OR**

Public Boolean isSorted(Node curr) {

If (curr.next == null) {

Return true;

}

if (curr.data > curr.next.data) {

Return false;

} else {

Return isSorted(curr.next);

}

}

1. public int numberOfDuplicates() {

node<T> curr = head;

int duplicates=0;

if (curr == null) {

throw Exception;

}

if (curr.next == null) {

duplicates = 0;

} else {

While (curr.next != null) {

If (curr.getData == curr.getNext.getData) {

Duplicates++;

}

Curr = curr.getNext;

}

}

Return duplicates;

}

1. public Node reverseOrder(Node n) {

Node<T> curr = head;

Node<T> prev = null;

Node<T> next = null;

While (curr != null) {

Next = curr.getNext;

Curr.setNext(prev);

Prev = curr;

Curr = next;

}

n = prev;

return n;

}

1. public void swapPairs () {

Node<T> curr = head;

If (size%2 == 0) {

While (curr!= null) {

T temp = curr.getData();

Curr.setData(curr.getNext.getData);

Curr.getNext.setData(temp);

Curr = curr.getNext.getNext;

}

} else {

While (curr.next!= null) {

T temp = curr.getData();

Curr.setData(curr.getNext.getData);

Curr.getNext.setData(temp);

Curr = curr.getNext.getNext;

}

}

}

1. Reversal of a Linked List

Public Node reverse (Node curr) {

If(curr == null || curr.next == null) {

Head = curr;

Return curr;

}

Node temp = reverse (curr.next);

Curr.next.next = curr;

Curr.next = null;

Return temp;

}

**OR**

Public void reverse (Node curr) {

If(curr == null || curr.next == null) {

Head = curr;

Return;

}

revese (curr.next);

Curr.next.next = curr;

Curr.next = null;

}