## Homework

1. develop an implementation of the equals method in the context of the SinglyLinkedList class.

public boolean equals(Object o) {

if (this == o) {

return true;

}

if (o == null || getClass() != o.getClass()) {

return false;

}

SinglyLinkedList<?> that = (SinglyLinkedList<?>) o;

if (size() != that.size()) {

return false;

}

Node<T> currentThis = head;

Node<?> currentThat = that.head;

while (currentThis != null) {

if (!currentThis.data.equals(currentThat.data)) {

return false;

}

currentThis = currentThis.next;

currentThat = currentThat.next;

}

return true;

}

}

1. Give an algorithm for finding the second-to-last node in a singly linked list in which the last node is indicated by a null next reference.

public Node<T> findSecondToLastNode() {

if (head == null || head.next == null) {

return null; // List is empty or has only one node

}

Node<T> current = head;

while (current.next.next != null) {

current = current.next;

}

return current;

}

}

1. Give an implementation of the size( ) method for the SingularlyLinkedList class, assuming that we did not maintain size as an instance variable.

public int size() {

int count = 0;

Node<T> current = head;

while (current != null) {

count++;

current = current.next;

}

return count;

} }

1. Implement a rotate( ) method in the SinglyLinkedList class, which has semantics equal to addLast(removeFirst( )), yet without creating any new node.

public void rotate() {

if (head != null && head.next != null) {

Node<T> newHead = head.next;

Node<T> current = head;

while (current.next != null) {

current = current.next;

}

current.next = head;

head.next = null;

head = newHead;

}

}

}

1. Describe an algorithm for concatenating two singly linked lists L and M, into a single list L′ that contains all the nodes of L followed by all the nodes of M.

- Iterate through the nodes of L until you reach the last node.

- Set the next reference of the last node of L to the head of M.

1. Describe in detail an algorithm for reversing a singly linked list L using only a constant amount of additional space.

- Initialize three pointers: prev, current, and next.

- Set prev to null and current to the head of the list.

- Iterate through the list, at each step:

- Set next to the next node after current.

- Set the next reference of current to prev.

- Move prev and current one step forward in the list.

- After the iteration, set the head of the list to prev, which will be the new head of the reversed list