CSCI 23500 Exam 1

Name:

1 Linked Lists

This question involves a linked list using the header provided (Node.h).

1.1 length

Write a function `int length(Node *l)` that accepts a pointer to a linked list as a parameter. It will return the number of elements in the list

1.2 Middle

Write a function `Node *middle(Node *head)` that accepts as a parameter a pointer to the first node in the list (head). It will return a pointer to the middle node in the list. Examples:

List	Returns a pointer to
head $\rightarrow$ nullptr	nullptr
head $\rightarrow 5 \rightarrow 10 \rightarrow 3 \rightarrow 6$	the node with 3
head $\rightarrow 5 \rightarrow 10 \rightarrow 3 \rightarrow 6 \rightarrow 8$	also the node with 3

1.3 mergesort

Write a function `Node *msort(Node *head)` That performs a mergesort on the input list represented by the parameter `head`. You may use the **length** and **middle** routines defined above and can assume that they above work as specified regardless of what you wrote.

Assume the existence of a function `Node *merge(Node *a, Node *b)` that accepts two node pointers, each pointing to a sorted linked list and returns a node to a new sorted linked list that combines all the elements of **a** and **b**.

Note: In addition to knowing the mergesort algorithm and using the above routines, you will have to split the list sent in as a parameter. For full credit, do this without creating additional nodes.

1.4 List rotation

Write a function `Node *rotate(Node *head, int position)` that rotates a linked list such that the node in location **position** becomes the first node (whereas head originally pointed to the first node). Assume position is a number greater than 0 and less than or equal to the number of nodes in the list.

For example, given this list:

- head→6→-30→0→55→22→17→12

Rotating the linked list to position 5 (which currently has the value 22 - note that this is using 1 indexing not 0 indexing) yields:

- head→22→17→12→6→-30→0→55

Your routine should return the Node pointer pointing to the new head after you complete the rotation.

For full credit you must rearrange the nodes rather than creating new ones or a new list.

2 Run time (18 points - 3 each)

What is the Big-Oh worst case run time complexity for each of the following:

2.1 Removing an element from a vector with n elements

2.2 Finding the first non-vowel characer in a string with c characters

2.3 Inserting at the beginning of a singly linked list with n elements

2.4 Searching for an element in a sorted vector with n elements

2.5 Removing the middle eleemnt in a sorted doubly linked list with n elements

2.6 Searching for an element in a sorted doubly linked list with n elements

Code

Node

```
class Node{
 private:
  std::string data;
  Node *next;
 public:
  Node();
  Node(std::string data);
  Node(std::string data, Node* next);
  void setData(std::string data);

  void setNext(Node *next);
  std::string getData();
  std::string &getRef();
  Node* getNext();
};
```

Name:

1.1 Length

1.2 Middle

Name:

1.3 mergesort

1.4 rotation

Name:

2.

2.1

2.2

2.3

2.4

2.5

2.6