Linked List Efficiency & Extension Problems

- Efficiency/time complexities for the topic are considered. Note that these discussions are not exhaustive or representative of the types of questions that may be found on the exams.
- Practice problems are given to test your understanding of the topic. These problems are not for a grade, but promote a deeper level of understanding of the topic. Unless otherwise specified, assume you are using *n* data/objects/datas in the data structure or algorithm.
- We look at the worst/average/best case of Big O; however, do not expect all three cases to be explored extensively.
- Feel free to discuss these problems in recitation or during office hours at the TA Help Desk.

SINGLY Linked List time complexity is considered for the basic operations of add, remove, access, and search. The Linked List data structure is created for the purpose of operating at the head. It is not designed for moving around to arbitrary positions within the Linked List. Linked Lists are designed to have head references. Assume no tail reference for Singly Linked Lists.

Efficiency

Access and Search

The access and search operations do not modify the Singly Linked List. The purpose is to locate data in the Singly Linked List.

1) What is the time complexity of accessing/searching data at a known position in a Linked List that is not the head? _____

2) What is the time complexity of accessing/searching data at an unknown position in a Linked List that is not the head? _____

3) What is the time complexity of accessing/searching data at the head of a Linked List? _____

4) What is the time complexity of accessing/searching data at the tail of a Linked List without a tail reference? _____

5) What is the time complexity of accessing/searching data at the tail of a Linked List with a tail reference? _____

6) What is the time complexity of finding the size of a Linked List without a size variable? _____

7) What is the time complexity of finding the size of a Linked List with size information stored at the head ? _____

Add
The add operation does modify the Singly Linked List which has only a head reference.

1) What is the time complexity of adding data at the head in a Linked List? _____
2) What is the time complexity of adding data at the tail in a Linked List? _____
3) What is the time complexity of adding data at the tail in a Linked List that has a tail reference? _____
4) What is the time complexity of adding data at an arbitrary position in a Linked List that is not the head? _____

Remove
The remove operation does modify the Singly Linked List which has only a head reference.

1) What is the time complexity of removing data at the head in a Linked List? _____
2) What is the time complexity of removing data at the tail in a Linked List? _____
3) What is the time complexity of removing data at the tail in a Linked List that has a tail reference? _____
4) What is the time complexity of removing data at an arbitrary position in a Linked List that is not the head? _____

Efficiency Conceptual

Given an *allocuurrence()* method which removes all occurrences of a data value in the linked list. The method locates the data, then makes a call a *remove()* method. The *remove()* method begins at the head and searches for the data to then remove it.

Answer these questions for a DOUBLY Linked List which has both a head and tail reference.
Answer these questions for a CIRCULAR SINGLY Linked List which has a head reference.
Answer these questions for a CIRCULAR DOUBLY Linked List which has both a head and tail reference.

Extension

Coding

For these problems, create a copy of your completed LinkedList homework assignment, and then modify it so that it only stories integers. Then add the specified instance method for each problem. (So these methods are going inside your …. .java file.)

Unless otherwise specified, you may not create new nodes. Additionally, you may not modify the data within each node; you may only change the pointer reference of each node. You may also not use the size variable. Always keep in mind efficiency; using the

utility functions, such as add() and remove(), would make your code less efficient. These problems are in order of difficulty.

1) Write a method called *isSorted()* that returns if the values in the linked list are in sorted ascending order.

2) Write a method called numberOfDuplicates() that returns the number of adjacent duplicate nodes in the linked list. For example, the linked list: 2 -> 2 -> 5 -> 7 -> 7 -> 7 -> 2 would return 3 since there is one pair of adjacent 2's and two pairs of adjacent 7's.

3) Reverse the contents of a singly linked list.

4) Write a method called swapPairs() that swaps each pair of numbers. For example, the linked list: 1 -> 2 -> 3 -> 4 -> 5 -> 6 would make the linked list become: 2 -> 1 -> 4 -> 3 -> 6 -> 5. If the size is odd, leave the last node alone.

Diagramming

For these questions, draw a diagram that clearly shows each step being performed for the following actions. Assume that the linked list is a singly linked list with just a head pointer.

1) Original linked list: 1 -> 4 -> 2 -> 4. Find the index of the last occurrence of 4.

2) Original linked list: 1 -> 4 -> 2 -> 4. Remove the head.

Conceptual

1) Explain the advantages and disadvantages of using a linked list over an arraylist.