# Linked List lab handout

Evan Misshula

August 22, 2016

## 1 Impportant review material

### 1.1 Definition

1. *Data Structure* - containers that hold data.
2. *Linked List* - A linked list is a data structure that consists of a group of nodes which represent a sequence together.

### 1.2 Primary video tutorial

From Hacker Rank:

- Blondie Bytes (34 minutes)
- `https://youtu.be/n9p5WK7AeY0`

This tutorial provides a good overview and is directly linked to the quiz problem I will propose. Also has a Java implementation.

### 1.3 Secondary video tutorials

These tutorials are more bare bones. He concentrates on getting you the code you need to make it work very fast.

From the New Boston (Bucky):

- `https://www.youtube.com/watch?v=BRcY2vIr-EQ` (6 minutes)

This video has an implimentation of a linked list in Java.

- `https://www.youtube.com/watch?v=rW2OppsgJjQ` (7 minutes)

This adds the print, add node and delete methods.

### 1.4 How to use a linked list from an API perspective

I created a smaill github repository that demonstrates the use of the Java standard Linked List class. It is available from the "java.util.*" package. The code is based on code I took from:

- `http://www.tutorialspoint.com/java/java_linkedlist_class.htm`

The git repository is available at:

- git@github.com:cunyTP/linkedList.git

### 1.5 Important caveat

You need constant-time insertions/deletions from the list (such as in real-time computing where time predictability is absolutely critical):

- This is only true for the head of the list.

### 1.6 When should you use a linked list?

Linked lists are preferable over arrays when:

a) you need constant-time insertions/deletions from the list (such as in real-time computing where time predictability is absolutely critical)

b) you don't know how many items will be in the list. With arrays, you may need to re-declare and copy memory if the array grows too big

c) you don't need random access to any elements

d) you want to be able to insert items in the middle of the list (such as a priority queue)

Arrays are preferable when:

a) you need indexed/random access to elements

b) you know the number of elements in the array ahead of time so that you can allocate the correct amount of memory for the array

c) you need speed when iterating through all the elements in sequence. You can use pointer math on the array to access each element, whereas you need to lookup the node based on the pointer for each element in linked list, which may result in page faults which may result in performance hits.

d) memory is a concern. Filled arrays take up less memory than linked lists. Each element in the array is just the data. Each linked list node requires the data as well as one (or more) pointers to the other elements in the linked list.

Array Lists (like those in .Net) give you the benefits of arrays, but dynamically allocate resources for you so that you don't need to worry too much about list size and you can delete items at any index without any effort or re-shuffling elements around. Performance-wise, arraylists are slower than raw arrays.

- `http://bit.ly/2be9WSn`

Here are some additional professional resources on Linked Lists vs Arrays

- `http://www.geeksforgeeks.org/linked-list-vs-array/`

Another perspective is argued by Bjarne Stroutstrup. He is the inventor of the C++ language.

- `https://www.youtube.com/watch?v=YQs6IC-vgmo` (7:42 minutes)

# 2 Further reading

## 2.1 Academic resources

### 2.1.1 Presentation

- `http://bit.ly/2bWzFCv`

Slide 27 has the Big-O values of common operations with Linked Lists. You should mention these at interviews.

| OPERATION | RUNTIME (Big-Oh) |
|---|---|
| add to start of list | O(1) |
| add to end of list | O(n) |
| add at given index | O(n) |
| clear | O(1) |
| get | O(n) |
| find index of an object | O(n) |
| remove first element | O(1) |
| remove last element | O(n) |
| remove at given index | O(n) |
| set | O(n) |
| size | O(n) |
| toString | O(n) |

### 2.1.2 Textbooks

two open source textbooks that present Linked Lists in Java:

`http://bit.ly/2bPSAvx`

`http://opendatastructures.org/ods-java.pdf` (p. 63)

## 2.2 More sample interview questions

1. Given Linked List L of size n, determine if L contains a loop (cycle). Complete the function named check, whose parameter is a pointer to a LinkedListNode (i.e.: a list head), and return YES if it contains a loop, or NO otherwise.

   Note: A loop or cycle in a Linked List of size n is defined as the last element of the list, Node Ln, pointing to a previous element in the list, some Node Li (where $0 \leq i \leq n$).

   Constraints

   $1 \leq n \leq 1000$

   Output Format

   Return YES if L contains a loop, or NO otherwise.

   Comments: Do you understand what they are asking. What is O(n)? How much memory will the function take?

2. There is a linear linked list, L, whose tail node points to the head of a circular linked list, C.

   Complete the filter(LinkedListNode* L) function, which takes the head of L as a parameter, so that it returns a pointer to the head of C.

   Input Format

   Your filter(LinkedListNode* L) function should take a pointer to L (a linked list whose tail node points to the head of circular linked list C).

   Constraints:

   $$\begin{aligned} L &\geq 1 \\ C &\geq 0 \end{aligned}$$

   Output Format

   Your function should return a pointer to the head node of C, the circular linked list.