Algorithm Analysis and Data Structures
CS 5343.001: Homework #3

Due on Wednesday September 21, 2016 at 11:59pm

Professor Greg Ozbirn

Lizhong Zhang **(lxz160730)**
lxz160730@utdallas.edu

Contents

Problem 1

Part (a)

Linked lists use a series of nodes each of which contains a pointer to the next node so that this allows a new node to be inserted or removed in $\mathcal{O}(1)$ time. But insertion or deletion in a array need move the list and these operation are $\mathcal{O}(N)$.

Part (b)

Arrays are systematic arrangement of similar objects which have their own signs so that accessing an item by its index can occur in $\mathcal{O}(1)$ time. But the only way to find an item in a linked list is to traverse the list and these operation are $\mathcal{O}(N)$.

Problem 2

Part (a)

If an Arraylist is passed for lst1 and lst2, the result is $\mathcal{O}(N^2)$. Because adding an item to the front in Arraylist requires moving the other elements down, which takes $\mathcal{O}(N)$ time, and there are N iterations.

Part (b)

If a LinkedList is passed for lst1 and lst2, the result is $\mathcal{O}(N)$. Because adding an item to the front in Linkedlist only requires $\mathcal{O}(1)$ time, and there are N iterations.

Problem 3

Part (a)

If a ArrayList is passed for lst1, it needs get its own iterator whose loop makes N iterations and use its remove method in which items must be shifted so the result is $\mathcal{O}(N^2)$.

Part (b)

If a LinkedList is passed for lst1, it needs get its own iterator whose loop makes N iterations and remove an item only takes $\mathcal{O}(1)$ time, so the result is $\mathcal{O}(N)$.

Problem 4

Part (a)

Arraylist is $\mathcal{O}(N^2)$ because lst1 and lst2 need get their own iterators whose loop makes N iterations.

Part (b)

Linkedlist is $\mathcal{O}(N^2)$ because lst1 and lst2 need get their own iterators whose loop makes N iterations.

Problem 5

Part (a)

Arraylist is $\mathcal{O}(N)$ because getting a value at an index position is $\mathcal{O}(1)$, with N iterations.

Part (b)

Linkedlist is $\mathcal{O}(N^2)$ since getting the ith value takes $\mathcal{O}(N)$ time, with N iterations.

Problem 6

Part (a)

Arraylist is $\mathcal{O}(N^2)$ because removing items from the front of the list and adding pushing it onto a Stack take $\mathcal{O}(N^2)$ time. And popping the items from the stack and inserting each item to the end of the list take $\mathcal{O}(N)$, $\mathcal{O}(N^2) + \mathcal{O}(N) = \mathcal{O}(N^2)$.

Part (b)

Consider it doesn't have a reference to the last node, so references to the end take $\mathcal{O}(N)$ time, with N iterations. Removing takes $\mathcal{O}(N)$ time and adding takes $\mathcal{O}(N^2)$ time, so $\mathcal{O}(N) + \mathcal{O}(N^2) = \mathcal{O}(N^2)$.
If it has a reference to the last node, references to the end take constant time, with N iterations. So $\mathcal{O}(N) + \mathcal{O}(N) = \mathcal{O}(N)$.

Problem 7

First, the symbol a is read, so it is passed through to the output. Then + is read and pushed onto the stack. Next b is read and passed through to the output. Next $a *$ is read. The top entry on the operator stack has lower precedence than $*$, so nothing is output and $*$ is put on the stack. Next, c is read and output. Thus far, we have the next symbol is $a +$. Checking the stack, we find that we will pop $a *$ and place it on the output; pop the other +, which is not of lower but equal priority, on the stack; and then push the +. The next symbol read is a (, which, being of highest precedence, is placed on the stack. Then d is read and output. We continue by reading $a -$. Since open parentheses do not get removed except when a closed parenthesis is being processed, there is no output. Next, e is read and output. Now we read a) , so the stack is emptied back to the (. We output $a -$. The input is now empty, so we pop and output symbols from the stack until it is empty. So the output is $abc * +de - +$.

Stack	Output
	a
+	a b
+ *	a b c
+	a b c * +
+ (	a b c * + d
+ (-	a b c * + d e
+	a b c * + d e -
	a b c * + d e - +