York University
EECS 2011Z Winter 2015 – Problem Set 2
Instructor: James Elder

This problem set will not be graded, but will help you consolidate the material from the first half of the course and prepare for the midterm. You are free to work together on these if you prefer.

1. Array Lists

   What is the asymptotic running time of the method makeList below as a function of N? Please justify your answer.

```
1  public static List<Integer> makeList( int N )
2  {
3    ArrayList<Integer> list = new ArrayList<Integer>();
4
5    for( int i = 0; i < N; i++ )
6    {
7      list.add( i );
8      list.trimToSize( );
9    }
10 }
11
12 /**
13 * Trims the capacity of this ArrayList instance to be the
14 * list's current size. An application can use this operation to minimize
15 * the storage of an ArrayList instance.
16 */
17 public void trimToSize() {
18     modCount++;
19     int oldCapacity = elementData.length;
20     if (size < oldCapacity) {
21         Object oldData[] = elementData;
22         elementData = (E[])new Object[size];
23         System.arraycopy(oldData, 0, elementData, 0, size);
24      }
25 }
```

2. Linked Lists

   You are to design an efficient iterative algorithm **merge**(A, B) that accepts two singly-linked lists, each containing a strictly increasing sequence of integers (i.e., with no repeating elements), and merge them (take their union) into a single strictly increasing list of integers (again, with no repeating elements). For example, the input $A = [1, 4, 8, 10, 11, 20]$, $B = [-5, 1, 8, 9, 20]$ should return a reference to a list containing $[-5, 1, 4, 8, 9, 10, 11, 20]$.

   Your algorithm should use only O(1) additional memory beyond the two input lists, and should run in $\mathcal{O}(\max(m, n))$ time, where m and n are the lengths of the two input lists.

   Your algorithm is given only references A and B to the first nodes in each of the two input lists. Each node is comprised only of *val* and *next* instance variables. You may assume that empty lists are represented as *null* nodes and the *next* field of the last node in each list is set to *null*. No other list variables are available to you.

   Input: Two singly linked lists A and B, each containing a strictly increasing sequence of integers.

   Output: The union of A and B as a singly linked list of strictly increasing integers.

(a) (20 marks) Your algorithm (in pseudocode or Java):
Algorithm merge(A,B):

(b) (5 marks) Your algorithm is efficient in both time and memory. What is the biggest sacrifice you have made in achieving this efficiency?

3. Choosing a data structure

State in one or two words the simplest ADT and implementation we have discussed that would meet each requirement.

(a) O(1) time removal of the most recently added element
ADT: Implementation:

(b) O(1) average time addition, removal, access and modification of (key, value) pairs with unique keys
ADT: Implementation:

(c) O(1) time insertion and removal when you are given the position
ADT: Implementation:

(d) O(1) time index-based access and modification and amortized O(1) addition of elements
ADT: Implementation:

(e) O(log n) time insertion of (key, value) entries and O(log n) removal of entry with smallest key
ADT: Implementation:

(f) O(1) time removal of the least recently added element
ADT: Implementation:

4. Binary Trees

You are to design a recursive algorithm **btDepths(u, d)**, where u is a node of a binary tree and d is the depth of u. Your algorithm will determine the minimum and maximum depths of the external nodes descending from u. Note that if u has no parent (i.e., is the root of the whole tree), then $d = 0$. You can assume that each node v of the tree supports the following four binary tree accessor methods: **left(v), right(v), hasLeft(v)** and **hasRight(v)**. You can also assume that u is not null. Your algorithm should run in $O(n)$ time, where n is the number of nodes descending from u.

Input: A non-null node **u** of a binary tree, and its depth d.

Output: An object **depths** consisting of the two integer fields **depths.min** and **depths.max**, containing the minimum and maximum depth over all external nodes descending from u.

(a) (20 marks) Your algorithm (in pseudocode or Java):
Algorithm btDepths(u, d):

(b) (5 marks) Provide a brief justification for why you think your algorithm is $O(n)$.

5. **Largest Imbalance**

We define the imbalance of a node x of a proper binary tree as the difference between the lengths of the shortest and longest paths from x to a descendent external node.

Provide the pseudocode for a recursive algorithm $Imbalance(x)$ that returns the largest imbalance i in the tree with root node x, as well as the lengths s and l of the shortest and longest paths from x to an external node. You may assume methods getLeft(x) and getRight(x) return the left and right child nodes of node x, and method isExternal(x) returns true if the node is external, false if internal.

Input: The root x of a proper binary tree.

Output: The largest imbalance i in the tree as well as the lengths s and l of the shortest and longest paths from x to an external node.

6. What are the asymptotic running times of the methods add and remove of the class SparseNumericVector that you modified for Programming Question 1?

7. Give an example of a Java code fragment that performs an array reference that is possibly out of bounds, and if it is out of bounds, the program catches that exception and prints the following error message: Dont try buffer overflow attacks in Java!

8. Suppose you have a stack S containing n elements and a queue Q that is initially empty. Describe (in pseudocode or English) how you can use Q to scan S to see if it contains a certain element x, with the additional constraint that your algorithm must return the elements back to S in their original order. You may not use an array or linked list only S and Q and a constant number of reference variables.

9. Describe the structure and pseudo-code for an array-based implementation of the array list ADT that achieves O(1) time for insertions and removals at index 0, as well as insertions and removals at the end of the array list.

10. Describe (in pseudocode or English) an algorithm for reversing a singly linked list L using only a constant amount of additional space and not using any recursion.

11. Describe how to implement an iterator for a circularly linked list. Since hasNext() will always return true in this case, describe how to perform hasNewNext(), which returns true if and only if the next node in the list has not previously had its element returned by this iterator.

12. Describe (in pseudocode or English) an O(n) recursive algorithm for reversing a singly linked list L, so that the ordering of the nodes becomes opposite of what it was before.

13. Describe (in pseudocode or English) an algorithm that will output all of the subsets of a set of n elements (without repeating any subsets). What is the asymptotic running time of your algorithm?

14. The balance factor of an internal node v of a proper binary tree is the difference between the heights of the right and left subtrees of v. Describe (in pseudocode or English) an efficient algorithm that specializes the Euler tour traversal of Section 7.3.7 to print the balance factors of all the internal nodes of a proper binary tree.

15. Let T be a tree with n nodes. Define the lowest common ancestor (LCA) between two nodes v and w as the lowest node in T that has both v and w as descendents (where, by definition, a node is a descendent of itself). Given two nodes v and w, describe (in pseudocode or English) an efficient algorithm for finding the LCA of v and w. What is the running time of your algorithm?

16. We can represent a path from the root to a given node of a binary tree by means of a binary string, where 0 means go to the left child and 1 means go to the right child. Use this to design an time algorithm for finding the last node of a complete binary tree with n nodes, assuming a linked structure implementation that does not keep a reference to the last node.

17. Given a heap T and a key k, give an algorithm to compute all of the entries in T with key less than or equal to k. The algorithm should run in time proportional to the number of entries returned.