NAME: ______________________________ STUDENT ID: _______________

COMP 250 – Midterm #2 - 1
March 11th 2013

- This exam has 6 pages
- This is an open book and open notes exam. No electronic equipment is allowed.

1) Questions with short answers (28 points; 4 points each)

a) (No justification needed) If a dictionary containing *N* keys it implemented using a hash table with *K* buckets, where each bucket is implemented using a linked-list, what is the

i) best-case running time?

O(N/K)

ii) worst-case running time?

O(N)

b) Consider a sorted linked list containing *n* nodes, each storing an integer. On such a list, why is it not possible to do a binary search that runs in worst-case time O(log *n*)?

Because, unlike with an array, it is not possible to do random access within a linked list, i.e. to access in time O(1) element at a given index. With a linked list, this requires time O(n), which means that the running time of binarySearch is O(n).

c) A ternary tree is a tree where each node has up to three children. What is the maximum number of nodes in a ternary tree of height *h*? *Justify your answer.*

Maximum #nodes = $1 + 3 + 3^2 + \ldots + 3^h = (3^{h+1} - 1)/ (3 - 1) = (3^{h+1} - 1) / 2$

d) True or False? Justify your answer. A pre-order traversal executed on a heap will print the keys in increasing order.

False. If the heap is

```
    1
   / \
  3   2
```

then a pre-order traversal will output 1 3 2, which is not in increasing order.

e) When using a singly-linked list (not doubly-linked) to implement a queue ADT, is it better to implement the enqueue operation with the addLast() method and the dequeue operation with the removeFirst() method, or to implement enqueue with addFirst() and dequeue with removeLast()? Why?

It is better to use enqueue = addLast() and dequeue = removeFirst(), because these two operations run in time O(1), whereas removeLast() takes times O(n), where n is the number of nodes in the linked list.

f) (2 points each) What Abstract Data Type would be the most appropriate to represent each of the following situations? No justifications are needed.

1) When a student is busy taking several courses, he works on his assignments in the order he receives them, completing first the assignment that was assigned first.

Queue

2) When another (wiser) student is busy taking several courses, she works on her assignments in the order of their due dates, completing first on the assignment that is due first.

Priority queue

3) When you go to the doctor, he/she can access your medical record by simply typing in your Health Insurance number.

Dictionnary

4) Mathieu receives a lot of e-mails. He always replies to the most recently received un-answered e-mail, deletes it, and then repeats the process.

Stack

Question 2. (20 Points)

Consider the Java implementation of a linked list given below. Implement in Java the method removeElements(int *marker*), which remove all nodes with value equal to *marker* from the link list, leaving the rest intact. If no node with value *marker* exists in the list, then the list is left unchanged. For example:
calling removeElements(12) on the linked list: 5 → 12 → 6 → 3 → 12 → 12 → 9
results in : 5 → 6 → 3 → 9

```
class node {
        public int value;
        public node next;
};

class linkedList {
        public node head;
        public node tail;

        public void removeElements(int marker) {

                node current=head;
                while (current!=null) {
                        if (current.next!=null && current.next.value==marker) {
                                node newNext=current.next.next;
                                while (newNext!=null && newNext.value==marker) {
                                        newNext=newNext.next;
                                }
                                current.next=newNext;
                                if (newNext==null) tail=current;
                        }
                        current = current.next;
                }
        }
};
```

Question 3 (20 points)

Let *T* be a Binary Search Tree that contains a set of keys with *distinct* integers. Assume that you have a method subtreeSize(treeNode *n*) that returns the number of nodes in the subtree rooted at *n*, including *n* itself. Assume at any call to subtreeSize(treeNode *n*) takes time O(1).

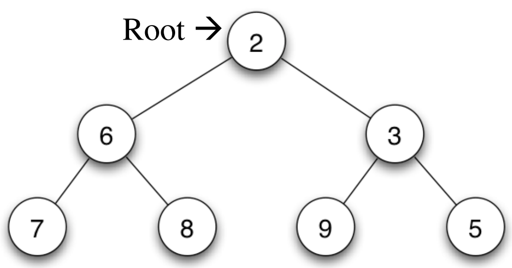
Problem: Write an algorithm that computes the number of nodes with a key greater or equal to a given integer *k*. Your algorithm should run in worst-case time O(*h*), where *h* is the height of the binary search tree (but you don't need to prove it).

Algorithm nbGreaterEqual(treeNode *n*, int *k*)
Input: A treeNode *n* and an integer *k*
Output: The number of nodes with key greater or equal to *k* in the subtree rooted at *n*.
/* WRITE YOUR PSEUDOCODE HERE */

```
if (n = null) return 0;
if (n.value < k ) return nbGreateEqual( n.leftChild , k) + subtreeSize( n. rightChild) + 1
if (n.value = k ) return subtreeSize( n. rightChild) + 1
if (n.value > k) return nbGreateEqual( n.rightChild, k)
```

4) Tree traversal algorithms (12 points)

Consider the following binary tree traversal algorithm.



```
Algorithm WeirdTraversal(treeNode n, int depth)
if (n!= null) then {
        if (depth is even) then {
                WeirdTraversal( n.getLeftChild() , depth+1 )
                WeirdTraversal( n.getRightChild() , depth+1 )
                Print n.getKey()
        }
        else {
                Print n.getKey()
                WeirdTraversal( n.getRightChild(), depth+1 )
                WeirdTraversal( n.getLeftChild(), depth+1 )
        }
}
```

What would be printed when executing WeirdTraversal(root, 0)? No justification is needed.

```
        Call stack                                          Printed
WT(2,0)
        WT(6,1)
                Print 6                                     6
                WT(8,2)
                        WT(null,3)
                        WT(null,3)
                        Print 8                             8
                WT(7,2)
                        WT(null,3)
                        WT(null,3)
                        Print 7                             7
        WT(3,1)
                Print 3                                     3
                WT(5,2)
                        WT(null,3)
                        WT(null,3)
                        Print 5                             5
                WT(9,2)
                        WT(null,3)
                        WT(null,3)
                        Print 9                             9
        Print 2                                             2
```

Question 5. Binary Search Trees and Heaps (20 points)

a) (10 points)
Consider the following Binary Search Tree.
Draw the Binary Search Tree after the remove(20)
operation has been performed, as seen in class.

b) (10 points)
Consider the following heap.
Draw the Heap after the insert(8) operation
has been performed, as seen in class.