Carleton UNIVERSITY

FINAL
EXAMINATION
FALL 2005

DURATION: 3 HOURS **No. Of Students: 19**

Department Name & Course Number: Systems and Computer Engineering
SYSC 2100

Course Instructor (s): Thomas Kunz

AUTHORIZED MEMORANDA
NONE

Students MUST count the number of pages in this examination question paper before beginning to write, and report any discrepancy to a proctor. This question paper has ____12____pages + cover page = _13_ pages in all.

This examination question paper may not be taken from the examination room.

In addition to this question paper, students require: an examination booklet no
Scantron Sheet no

Name: ____

Student Number: ____

Question 1: Big-O Notation (10 marks)

For each of the following functions f, where n = 0, 1, 2, 3, ..., estimate f using Big-O notation and plain English. Trivially, $O(2^n)$ is a valid upper bound for all of these functions, but I am looking for tight upper bounds (i.e., you won't get any marks for trivial upper bounds such as "*$O(2^n)$ or f(n) is exponential*", unless that is indeed a tight upper bound for the specific function).

1. $f(n) = (2 + n) * (3 + \log(n))$

2. $f(n) = 11 * \log(n) + n/2 - 3452$

3. $f(n) = 1 + 2 + 3 + ... + n$

4. $f(n) = n * (3 + n) - 7 * n$

5. $f(n) = 7 * n + (n - 1) * \log (n - 4)$

6. $f(n) = \log (n^2) + n$

7. $f(n) = \dfrac{(n + 1) * \log (n + 1) - (n + 1) + 1}{n}$

8. $f(n) = n + n/2 + n/4 + n/8 + n/16 + ...$

Question 2: Recursion (10 marks)

1. Given two positive integers *i* and *j*, the greatest common divisor of *I* and *j*, written *gcd (i, j)* is the largest integer *k* such that (i % k = 0) and (j % k = 0). For example, gcd (35, 21) = 7 and gcd (8, 15) = 1. Develop a recursive method that returns the greatest common divisor of *i* and *j*. Here is the method specification:

```
/**
* Finds the greatest common divisor of two given positive integers
*
* @param i – one of the given positive integers.
* @param j – the other given positive integer.
*
* @return the greatest common divisor of iand j.
*
* @throws IllegalArgumentException – if either i or j is not a positive integer.
*
*/
public static int gcd (int i, int j)
```

Hint: According to Euclid's algorithm, the greatest common divisor of *i* and *j* is *j* if *i % j* = 0. Otherwise, the greatest common divisor of *i* and *j* is the greatest common divisor of *j* and *(i % j)*.

2. A ***palindrome*** is a string that is the same from right-to-left as from left-to-right. For example, the following are palindromes: ABADABA, RADAR, OTTO, MADAMIMADAM, EVE
   For the time being, we restrict each string to upper-case letters only. Develop a method that uses recursion to test for palindromes. The only input is a string that is to be tested for palindromehood. The method specification is

```
/**
* Determines whether a given string of upper-case letters is a
* palindrome. A palindrome is a string that is the same from right-to-
* left as from left-to-right.
*
* @param s – the given string
*
* @return true – if the string s is a palindrome. Otherwise, return false.
*
*/
public static boolean isPalindrome (String s)
```

Question 3: Linked Lists (10 marks)

One of the possibilities for fields in the LinkedList class is to have head and tail fields (pointing to the first and last element in the linked list, respectively), both of type Entry, where the Entry class had element and next fields, but no previous field. Then we would have a *singly-linked* list.

1. Define the addLast method for this design. Here is the method specification:

```
/**
* Appends a specified element to (the back of) this LinkedList object.
*
* @param element – the element to be appended.
*
* @return true.
*
*/
public boolean addLast (Object element)
```

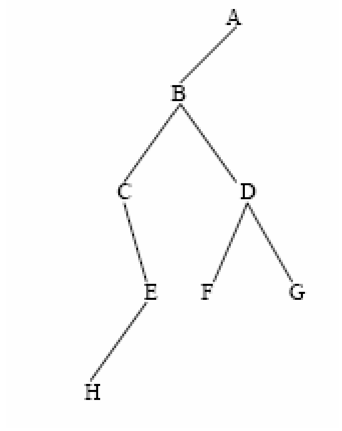
2. The definition of the removeLast() method would need to make **null** the next field in the Entry object before the Entry object tail. Could we avoid a loop in the definition of removeLast() if, in the LinkedList class, we added a beforeTail field that pointed to the Entry object before the Entry object tail? Explain.

3. Hypothesize the output from the following method segment:

```
LinkedList letters = new LinkedList();
ListIterator itr = letters.listIterator();
itr.add ('f'); itr.add ('t');
itr.previous();
itr.previous();
itr.add ('e');
itr.add ('r');
itr.next();
itr.add ('e');
itr.add ('c');
itr = letters.listIterator();
itr.add ('p');
System.out.println (letters);
```

Question 4: Binary Trees (10 marks)

Answer the questions below about the following binary tree:



1. What is the root element?

2. How many leaves are in the tree?

3. What is the height of the tree?

4. What is the height of the left subtree?

5. What are the descendants of B?

6. What are the ancestors of F?

7. What would the output be if the elements were written out during an inOrder traversal?

8. What would the output be if the elements were written out during a postOrder traversal?

9. What would the output be if the elements were written out during a preOrder traversal?

10. What would the output be if the elements were written out during a breadth-first traversal?

11. What is the maximum number of leaves possible in a binary tree with 10 elements? Construct such a tree.

12. What is the minimum number of leaves possible in a binary tree with 10 elements? Construct such a tree.

Question 5: Binary Search Trees/Huffman Trees (10 marks)

1. Show the effect of making the following insertions into an initially empty binary search tree: 30, 40, 20, 90, 10, 50, 70, 60, 80

2. Suppose we decide to implement the PurePriorityQueue interface with a TreeSet object:

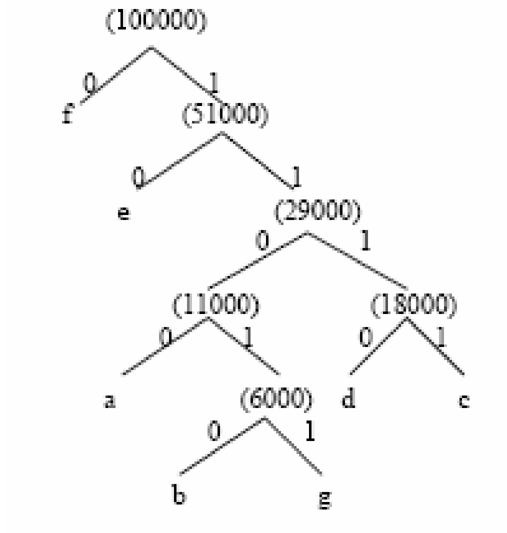
```
    public class TreeSetPriorityQueue
    implements PurePriorityQueue
    {
            TreeSet pq;
            …
```

   Then the definitions of the getMin, removeMin and add methods are oneliners. For example, here is the definition of the getMin method:

```
            public Object getMin()
            {
                    return pq.first();
            } // method getMin
```

   Estimate worstTime(*n*) for those three methods. What is the major drawback to implementing the PurePriorityQueue interface with a class that implements the Set interface? Explain how to overcome this drawback by implementing the PurePriorityQueue interface with the TreeMap class instead.

3. Use the following Huffman tree to translate the bit sequence 11101011111100111010 back into letters 'a' ... 'g':



Question 6: Hashing (10 marks)

1. Why does the HashMap class use singly-linked lists instead of the LinkedList class?

2. Suppose you have a HashMap object, and you want to insert an element unless it is already there. How could you accomplish this? **Hint:** The put method will insert the element even if it is already there (in which case, the new value will replace the old value).

3. For each of the following methods, estimate averageTime(n) and worstTime(n):
   - making a successful call – that is, the element was found – to the *contains* method in the LinkedList class;
   - making a successful call to the *contains* method in the ArrayList class;
   - making a successful call to the generic algorithm *binarySearch* in the Arrays class; assume the elements in the array are in order;
   - making a successful call to the *contains* method in the BinarySearchTree class;
   - making a successful call to the *contains* method in the TreeSet class;
   - making a successful call to the *contains* method in the HashSet class. You should make the Uniform Hashing Assumption. Estimate averageTime(n, m) and worstTime(n, m).

Question 7: Graphs, Trees, and Networks (20 marks)

1. Draw an undirected graph that has four vertices and as many edges as possible. How many edges does the graph have?

2. Draw an undirected graph that has five vertices and as many edges as possible. How many edges does the graph have?

3. What is the maximum number of edges for an undirected graph with *V* vertices, where *V* is any non-negative integer?

4. What is the maximum number of edges for a directed graph with V vertices?

5. Prim’s algorithm (getMinimumSpanningTree), Dijkstra's algorithm (getShortestPath), and Huffman codes are ***greedy***: the locally optimal choice has the highest priority. In these cases, greed succeeds in the sense that the locally optimal choice led to the globally optimal solution. Do all greedy algorithms succeed for all inputs? In this question we look at coin-changing algorithms. In one situation, the greedy algorithm succeeds for all inputs. In the other situation, the greedy algorithm succeeds for some inputs and fails for some inputs. Suppose you want to provide change for any amount under a dollar using as few coins as possible. Since “fewest” is best, the greedy (that is, locally optimal) choice at each step is the coin with largest value whose addition will not surpass the original amount. Here is a method to solve this problem:

```
/**
* Prints the change for a given amount, with as few coins (quarters,
* dimes, nickels and pennies) as possible.
*
* @param amount – the amount to be given in change.
*
*/
public static void printFewest (int amount)
{
        int coin[ ] = {25, 10, 5, 1};
        final String RESULT = “With as few coins as possible, here is the change for “;
        System.out.println (RESULT + amount + “:”);
        for (int i = 0; i< 4; i++)
                while (coin [i] <= amount)
                {
                        System.out.println (coin [i]);
                        amount – = coin [i];
                } // while
} // printFewest
```

For example, suppose that amount has the value 62. Then the output will be 25, 25, 10, 1, 1. Five is the minimum number of coins needed to make 62 cents from quarters, nickels, dimes and pennies. Give an example to show that a greedy algorithm is not optimal for all inputs if nickels are not available. That is, if we have

```
int coins[ ] = {25, 10, 1};
…
for (int i = 0; i < 3; i++)
...
```

then the algorithm will not be optimal for some inputs.

6. In the Network class, develop a method to produce a topological order. Here is the specification:

```
/**
* Sorts this acyclic Network object into topological order.
* The averageTime(V, E) is linear in V + E.
*
* @return an ArrayList object of the vertices in topological order. Note: if
* the size of the ArrayList object is less than the size of this
* Network object, the Network object must contain a cycle, and
* the ArrayList will not be in topological order.
*
*/
public ArrayList sort()
```

Hint: First, construct a HashMap object, inCount, which maps each vertex w to the number of vertices to which w is adjacent. For example, if the Network object has three edges of the form <?, w>, then inCount will map w to 3. After inCount has been filled in, push onto a stack (or enqueue onto a queue) each vertex that inCount maps to 0. Then loop until the stack is empty. During each loop iteration,

1. pop the stack;
2. append the popped vertex v to an ArrayList object, orderedVertices;
3. decrease the value, in inCount's mapping, of any key w such that <v, w> forms an edge;
4. if inCount now maps w to 0, push w onto the stack.

After the execution of the loop, the ArrayList object, containing the vertices in a topological order, is returned.