THE UNIVERSITY OF CALGARY

FACULTY OF SCIENCE

FINAL EXAMINATION

COMPUTER SCIENCE 331

WINTER SESSION: LECTURE 02

April 20, 2007 **Time: 2 hrs.**

NAME: ______________________________

Please DO NOT write your ID number on this page.

Instructions:

Answer all questions in the space provided.

Use the last two pages to continue answers if you need more space.

No aids are allowed.

There are 80 marks available on this test.

Name:____________________ **ID:**__________________

Question	Score	Out Of
1. (Short Answer)		15
2. (Correctness of Algorithms)		10
3. (Java Programming)		15
4. (Dictionaries and Binary Trees)		12
5. (Generalization of Merge Sort)		7
6. (Quicksort)		11
7. (Graphs)		10
Total:		80

Name: ____________________ **ID:** ________________

1. Short answer questions

   You do *not* need to provide any justification for your answers. Just fill in your answer in the space provided.

(1 mark) (a) True or false: if we do not know the postcondition of a function, then we cannot test whether the method works correctly with black-box testing.

Answer: T

(1 mark) (b) True or false: using an array-based implementation, all operations of the queue ADT have worst-case cost $O(1)$.

Answer: F

(1 mark) (c) Which of the following data structures is most appropriate as an implementation of the `stack` ADT: singly-linked list or doubly-linked list?

Answer: SLL

(1 mark) (d) True or false: the worst-case running time of heap sort on an array with n elements is in $O(n^2)$

Answer: T (time is in $\Theta(n \lg n)$, therefore also in $O(n^2)$)

(1 mark) (e) True or false: the adjacency matrix representation of a graph is more space-efficient than the adjacency list representation for sparse graphs.

Answer: F

Name: ____________________ **ID:** ____________________

(4 marks) (f) In the following table, fill in *true* or *false* in the appropriate box.

$f(n)$	$g(n)$	$f(n) \in O(g(n))$	$g(n) \in o(f(n))$
$15n^2 - 7$	$30 + 20n$	F	T
$5n\log(n) + 3n$	$17 + 9n\log(n)$	T	F

(6 marks) (g) Consider the `search` function for the `dictionary` abstract data type. **Using big-Oh notation**, fill in the following table to indicate the asymptotic running time as a function of n, where n is the number of entries in the dictionary, assuming that the search is unsuccessful (i.e., the key is *not* in the dictionary).

Data Structure	**worst-case**	**average-case**
ordered array	$\theta(\log n)$	$\theta(\log n)$
binary search tree	$\theta(n)$	$\theta(\log n)$
hash table with chaining	$\theta(n)$	$\theta(1)$

Assume that the load factor $\alpha < 1$ for the hash table.

Name: ____________________ **ID:** ____________________

2. Correctness of algorithms

(4 marks) (a) Define a *loop invariant*. Be sure to include the three properties a loop invariant must satisfy.

See notes (lecture 2-4)

(3 marks) (b) Define a *loop variant*. Be sure to include the three properties it must satisfy.

See notes (lecture 2-4)

Name:____________________ ID:____________________

(3 marks) (c) Describe how a loop invariant and loop variant can be used together to prove that an algorithm consisting of a single loop is correct.

See notes (lecture 2-4)

3. Java Programming

   Recall that a *bounded queue* is an abstract data type that supports the following operations.

   - `isEmpty()`: Report whether the queue is empty.
   - `peek()`: Report the element at the front (head) of the queue without changing it if the queue is not empty. Throw an `EmptyQueue` exception otherwise.
   - `dequeue()`: Remove the element at the front (head) of the queue if the queue is not empty. Throw an `EmptyQueue` exception otherwise.
   - `enqueue(x)`: Add a new element to the back (tail) of the queue unless the queue size is already `maxQueueSize`. Throw a `QueueFull` exception otherwise.

   Note: You may assume that `maxQueueSize` is a global constant whose value is a positive integer, and that the exceptions `QueueFull` and `QueueEmpty` have already been provided.

(5 marks) (a) Write an interface in Java for this abstract data type. You do *not* need to include any comments or other documentation.

See notes (lecture 14) for a unbounded Queue interface. This also suffices for the bounded queue.

Name: ____________________ **ID:** ________________

(10 marks) (b) Write a class in Java that uses an **array implementation** of a bounded queue and that implements the interface you gave as your answer for Part (a) of this question. No comments nor documentation are required. Assume that the elements to be stored in the queue are of type `Object`. Your class should have one constructor that initializes an empty queue with maximum capacity `maxQueueSize`.

See notes (lecture 14) for an unbounded queue implementation. Modify add to throw an IllegalStateException when full instead of resizing the array.

Name: ____________________ ID: ________________

(Question 3b continued)

Name: ____________________ **ID:** ________________

4. Dictionaries and Binary Trees

(4 marks) (a) Define the dictionary abstract data type.

See notes, or the descriptions of search, insert, delete from SimpleSortedMap (A3)

Name: ____________________ **ID:** ____________________

(6 marks) (b) Give pseudocode for a **recursive** algorithm that **deletes** an element with a given key (if it exists) from a dictionary that is implemented using a **binary search tree.** You may assume the existence of a function `findMin(T)` that returns the smallest data value stored in the binary search tree `T`.

See notes (lecture 18 - 19)

Name: __________________ ID: __________________

(2 marks) (c) Explain why, when applying this deletion algorithm to a red-black tree, that no further adjustments to the tree are required if the deleted node was red.

1) parent and child of a red node must be black (Property #4) $\Rightarrow$ deleting a red node will not cause adjacent red nodes.

2) deleting a red node does not effect black heights, So Property #5 will still hold,

Name:____________________ **ID:**________________

5. Generalization of Merge Sort

Consider the following generalization of Merge Sort that sorts the input array A and writes the result to a second array B :

kmergeSort(A, B)
  if *length*(A) $< k$ **then**
    sort A using insertion sort, write sorted array to B
  else
    Split A into k arrays $A_1, \ldots, A_k$, each of size at most $\lceil length(A)/k \rceil$
    kmergeSort(A_1, B_1)
    kmergeSort(A_2, B_2)
    ...
    kmergeSort(A_k, B_k)
    kmerge($B_1, B_2, \ldots, B_k, B$)
  end if

The function **kmerge** is a generalization of the merge algorithm described in class that merges the k sorted arrays $B_1, \ldots, B_k$ into the array B.

(3 marks) (a) Give a recurrence relation as a function of n and k that describes the worst-case running time of **kmergeSort**. Assume that the function **kmerge** runs in time $O(n \log k)$ where the k arrays $B_1, \ldots, B_k$ have a total of n elements.

$$T(n) \leq \begin{cases} c_1 & \text{if } n < k \\ kT(\lceil \frac{n}{k} \rceil) + c_2 n \log k & n \geq k \end{cases}$$

Name: ____________________ **ID:** ________________

(4 marks) (b) Describe (using English or pseudocode) an algorithm that implements the **kmerge** function using a min-priority queue that runs in worst-case time $O(n \log k)$. You may assume that the min-priority queue is implemented with a binary heap but do *not* have to give the implementation of the priority queue nor the heap — just use the priority queue ADT functions as required.

Note: Describing an algorithm that runs in time $O(nk)$ that does not use a priority queue is also acceptable, but will earn a maximum of **2** marks.

```
kmerge(B1, B2, ..., Bk, B) {
  Init empty priority queue Q.
  j = i1 = i2 = ... = ik = 0
  Q.insert(B1[0], 1)
  Q.insert(B2[0], 2)
     :
  Q.insert(Bk[0], k)
  while !Q.is-empty()) {
      (e, l) = Q.extract_min();
      B[j] = e; ++j; ~~++il~~
      ++il; ~~Q.insert(Bl[il], l)~~;
      if il < Bl.length
          Q.insert(Bl[il], l);
  }
}
```

Name: ____________________ **ID:** ________________

6. Quicksort

Consider the following array.

0	1	2	3	4	5
18	3	24	2	1	6

(3 marks) (a) Suppose that the **deterministic version of Quicksort**, discussed in class, is used with the above array as input. Show the array that is obtained after the first **partition** operation is performed.

1	3	2	6	18	24

(2 marks) (b) Show the arrays that are recursively sorted using Quicksort after the above partitioning operation has been applied.

1	3	2

and

18	24

(2 marks) (c) What is the number of operations that is used by deterministic Quicksort to sort an array of size n in the worst case? Give a description of input arrays that result in the worst case running time.

$\Theta(n)$, sorted array.

Name:____________________ ID:________________

(2 marks) (d) What is the *expected* number of operations used by this Quicksort algorithm, assuming that the entries of the input array are distinct and each relative ordering of the entries is equally likely?

$\Theta(n \log n)$

(2 marks) (e) Briefly describe one modification to the deterministic Quicksort algorithm described in class that significantly improves its running time in practice and explain why it works.

- Select pivot as median of this 1st, middle, and last elements of the array

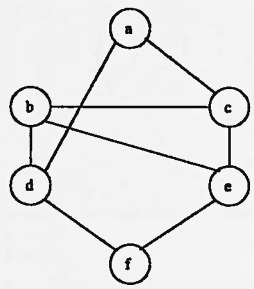
or

- stop recursion early by sorting small subarrays with insertion sort

Name:____________________ **ID:**____________________

7. Graph Algorithms

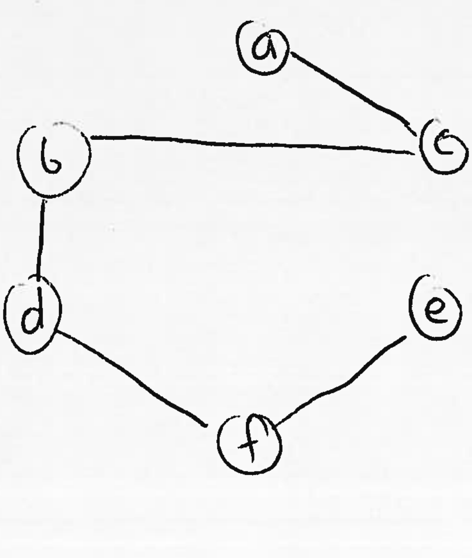
Consider the following graph:



(2 marks) (a) Draw an **adjacency-list** representation for this graph. Order all vertices alphabetically by label in arrays and lists when you do this.

a → c → d
b → c → d → e
c → a → b → e
d → a → b → f
e → b → c → f
f → d → e

(4 marks) (b) Draw the **breadth first search tree** (tree consisting of shortest paths from the source vertex to every other reachable vertex in the graph) that would be obtained from this graph using the breadth first search algorithm described in class, assuming that vertex "a" is the source vertex.



Name: ____________________ **ID:** ____________________

(2 marks) (c) What is the running time of this algorithm as a function of $|V|$ and $|E|$? Give your answer using big-O notation.

$\Theta(|V| + |E|)$

(2 marks) (d) Does this algorithm also work for directed graphs? Justify your answer.

Yes. Just make sure edges are followed in the correct direction.