Prelim

CS410, Summer 1998

17 July 1998

Please note:

- This exam is closed book, closed note. Sit every-other seat.
- Put your answers in this exam.
- The order of the questions roughly follows the course presentation and not necessarily order of difficulty.
- There are 5 questions totalling 100 points and 1 extra credit question. Point values are indicated for each question.
- Feel free to separate pages; you can re-staple them at the end of the exam.
- You have 75 minutes to complete the exam. Do <u>not</u> continue writing after time is up.
- Partial credit is available for all questions. However, keep your answers concise – extraneous ramblings will take your time and may hurt you.
- Answers will be posted on the web.

1. Prove formally that if $f(n)$ is $O(g(n))$, then $g(n)$ is $\Omega(f(n))$. [10 points]

2. (a) What form of recurrence relations can be solved using the Master Theorem? (That is, state the part of the theorem which describes the relations for which the theorem applies.) [6 points]

   (b) Write a recurrence relation that cannot be solved by the Master Theorem. (Read the next part before answering, though.) [6 points]

   (c) Write a program (pseudo-code is fine) that has a running time accurately expressed by your answer to the previous question. Be sure to explain how the variable in your recurrence relation relates to your program's input. [6 points]

3. Below is a partial implementation for a data structure that is like a **sorted singly-linked list**. In addition, every node in the list has a pointer to the k^{th} next thing in the list.

   - Assume the implementation uses sentinels, i.e., *do not worry about following null pointers.* That is, if a next or kth pointer would be null, assume it points to a node whose key is "infinity". Equivalently, just assume you're never "near the end of the list".
   - Assume there are no duplicate keys.
   - You will not be graded on Java syntax. Pseudo-code is okay, but do **not** use it to avoid "low-level details".

```
class ListNode {
   int key;
   Object val;
   ListNode next;
   ListNode kth;
}

class List {
   ListNode head;
   Object lookup(int key) {...}
   void delete(int key) {...}
   ...
}
```

Write an efficient lookup method for this data structure. Assume a node with the key exists. Your method should examine less than $n/k + k$ nodes in the worst case. [15 points]

4. This problems concerns ways to print all the keys in a binary tree. Let the term "print a node" really mean "print the key at a node". Assume you have a method `print` which takes a node and prints its key. The following tree serves as an example below:

```
      A
    /   \
   B     C
  / \   /
 D   E  F
    / \
   G   H
```

(a) Suppose we wish to print nodes before their children and left subtrees before right subtrees. (So the result for the example would be "A B D E G H C F".) Write a non-recursive program to accomplish this task. Use an auxiliary ADT from class so that the program you write below is short. (Hints: Make sure you pick the right ADT. Your program is done when the ADT object is empty.) [12 points]

(b) Now suppose we with to print nodes in a top-to-bottom order with nodes at the same level printed left-to-right. (So the result for the example would be "A B C D E F G H".) Write a non-recursive program to accomplish this task. Use an auxiliary ADT from class so that the program you write below is short. (Same hints as in previous problem) [12 points]

(c) Your previous two answers should look very similar. Explain how could you have used object-oriented programming to avoid cut-and-paste programming. Hint: To make your explanation simpler, you may assume that the methods of your two auxiliary ADTs have the same names. [8 points]

This page intentionally left blank.

5. This problem concerns priority queues.

(a) Explain how a 2-3 tree can be used to implement a priority queue. Give the asymptotic running times of the operations. [5 points]

(b) Now consider a 2-3 tree *without the restriction that the leaves be ordered.* Explain how such a tree could be used to implement a priority queue. (Specifically, explain how the operations differ in implementation from those in the previous problem.) Give the asymptotic running times of the operations. [10 points]

(c) To "merge" priority queues P_1 and P_2 means to make P_1 a priority queue that has all the elements originally in P_1 or P_2. Explain how priority queues implemented as 2-3 trees with **unordered** leaves could be merged. Assume P_1 is bigger than P_2. Do not bother to write code for the merge method – just describe what it would do. The running time of your solution should be $O(\log n)$ where n is the size of P_1. (Hint: How would you insert a whole tree at once?) [10 points]

Extra Credit [10 points]

Write an efficient delete method for the data structure in problem 3. The method takes a key, finds the node with that key, and removes that node. Assume a node with the key exists. Your method should examine less than $n/k + 2k$ nodes in the worst case. (Hint: Notice the 2.)