CS 61B, Summer 2001
Midterm Exam II
Professor Michael Olan

Problem 1 (8 points - 1 pt each parts a.-e., 2 pts part f.)

a. Consider the following proof: Suppose *f(n)* is *O(g(n))*. From this we know that *g(n)* is *bigOmega(f(n))*. Thus we can always conclude that when *f(n)* is *O(g(n))* it must also be *bigOmega(g(n))*. If you agree justify your answer. If not, give a counterexample.

b. A certain sorting algorithm has estimated running time *O(N^2)*. Does this mean that it always takes longer to sort than an algorithm that is *O(N log N)*? Explain your answer.

c. What is the overall running time for each of the following algorithms:

Algorithm 1

1. Read N data items into a Vector
2. Sort the Vector with tree sort
3. Search for a key k with binary search

Algorithm 2

1. Read N data items into a Vector
2. Search for a key k with linear search

d. How would the estimates you calculated in part c. change if the search step is repeated N times?

e. In lecture we saw a heapify method with run time bigTheta(N log N). However, it is possible to accomplish this in bigTheta(N) using a "bottom up" heapify method which builds subheaps from the leaves up to the root. Complete the method below so that it is as efficient as possible:

```
protected void heapify( ) {
        int i;
        for (_____________ ; _________________ ; _________________ ) {
                reHeapifyDown( i );
        }
}
```

f. Dr. I.M. Daprof has a policy of assigning "A" grades to the top sqrt(N) students in a class of N students. Always pressed for time, he uses the most efficient algorithm he knows:

- Sort the gradebook in descending order by numeric score using the fastest sorting algorithm we have seen so far in CS 61B

- Select the first sqrt(N) elements in the resulting sorted list

Dr. Daprof is pleased that his algorithm is O(N log N), but with the knowledge you've gained in CS61B, you know that there is a more efficient method. Impress Dr. Daprof by describing an algorithm that is O(N).

Problem 2(4 points - 1 pt each part)

a. When deleting a node from a binary search tree, if we decide to replace it with a node in its *left subtree* (assuming it has one), which node should we choose?

b. True or False: There is exactly one binary search tree associated with any set of keys { k1, k2, ... , kn }. Explain your answer.

c. Consider a binary tree with m leaf nodes. What is the minimum number of nodes in the tree? What is the maximum number of nodes in the tree?

d. Show the contents of the heap H after the operations:

```
H.removeFirst( );
H.insert(25);
```

Before:	35	29	14	22	18	11	12	15	8	10
After:										

Problem 3 (8 points)

You are writing an application to monitor the number of defects produced by a collection of machines. You are given the following (partial) class definitions:

```
class Defect {
        . . .
        // Effect: Returns the name of the machine that
        //                       produced this defect
        public String getMachine( ) { . . . }
        . . .
}

class DefectMonitor {
        //Modifies: this
        //Effect : Records the occurrence of d
        public void recordDefect( Defect d )  { . . . }
        //Effect : Returns the name of the machine that currently
        //                       has produced the most defects
        public String getWorst( ) { . . . }
        //Effect : Returns a Vector of machines arranged in descending
        //                       order by number of defects produced
        Vector currentStatus( ) { . . . }
}
```

Efficiency is critical to this application. Quickly identifying the machine most in need of attention is the most important task, thus you must make getWorst() as efficient as possible. You also want recordDefect(..) to be very fast. currentStatus() will only be used periodically, and so its performance is not as critical as the others. No other assumptions can be made about the number of machines, frequency of defects, etc.

Describe in words how you would design the DefectMonitor class. Specifically, indicate

- What data structure you would use to implement the class. You can use any of the data structures we've studied, and can enhance them by adding extra state variables (private) if it would help to improve the efficiency of the methods for this application.
  Note: if you do include additional state variables with a data structure, be specific in describing what they are and what their purpose is.
- Describe in words how each of the 3 methods would be implemented.
- Indicate the running time estimates for each of the 3 methods, using your implementation.

Problem 4 (6 points)

Write the implementation of the isComplete() method as part of the binary tree class shown below. You may optionally write helper methods if it will improve the implementation.

```
class BinTree {

private Object item;
private BinTree left, right; // left and right children

// No other methods are known to you

// Effects: returns true iff this is a complete binary tree
public boolean isComplete( ) {
```

Problem 5 (10 points)

Write an implementation for the priority queue interface that uses two Stacks as the data structures for representing the priority queue. The Stack class provides the methods: void push(int X), int pop(), int top(), and boolean isEmpty() with the usual meanings. You may not use any other data structures, but may include instance variables having primitive types. **The methods to be implemented are a constructor, insert, and removeFirst.**

```
interface PriorityQueue {
        /* Modifies: this
         * Effect: Inserts X into this queue */
        public void insert(int X);  /* IMPLEMENT THIS METHOD */

        /* Requires: !isEmpty()
         * Modifies: this
         * Effect: Removes and returns an instance */
        public int removeFirst();       /* IMPLEMENT THIS METHOD */

        /* Requires: !isEmpty()
         * Effect: Returns the largest element in this queue */
        public int largest();           /* DO NOT IMPLEMENT */

        /* Effects: Returns true iff this queue is empty */
        public boolean isEmpty();       /* DO NOT IMPLEMENT */
}
```
