Computer Science 136

Bruce - Spring '04

2nd Midterm Examination

April 28, 2004

Question	Points	Score
1	12	____
2	11	____
3	10	____
4	14	____
5	8	____
6	8	____
TOTAL	63	____

Your name (Please print)

_____

I have neither given nor received aid on this examination.

_____

(sign here)

1. Short answers.

a. Many text editors provide the possibility for the user to undo an arbitrary number of commands. What data structure should the implementors use to save past commands in order to provide this capability? Circle the correct answer:

Stack Queue Tree PriorityQueue

b. Some word processors collect embedded footnotes in text and print them all (in the order they appeared) at the end of the document. Which data structure should the implementors of the word-processor use to save the footnotes as they are encountered? Circle the correct answer:

Stack Queue Tree PriorityQueue

c. When we rewrite a recursive algorithm to be iterative, we generally must introduce which kind of data structure to aid in simulating the recursion? Circle the answer:

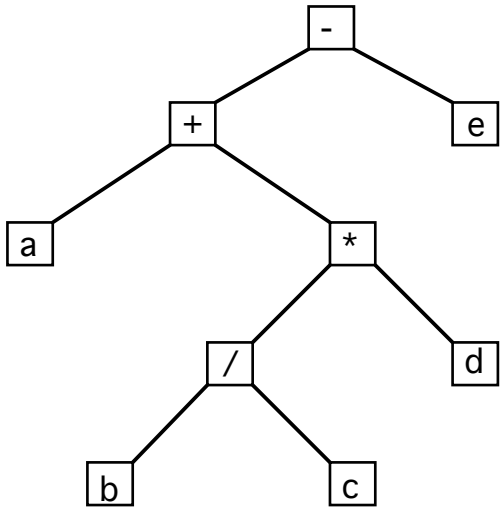
Stack Queue Tree PriorityQueue

d. Your wish to design a device that can record? Circle the answer:

Stack Queue Tree PriorityQueue

e. List a circumstance in which it might be more desirable to use heapsort rather than quicksort. Give a brief justification for your answer.

e. Please list the elements of the following expression tree in the orders determined by each of the four traversals *(do not insert parentheses in your answers!)* :



Preorder:

Inorder:

Postorder:

Levelorder:

2. Concurrency and threads:

a. What is a thread and what does it have to do with concurrency?

b. Please explain what the impact of sending the following messages to a thread would be:

   i. start()

   ii. sleep(50);

   iii. wait();

c. Explain under what circumstances a thread will resume execution if it has executed:

   i. sleep(50);

   ii. wait();

c. Suppose an object, `testObj`, with `int` field `value`, has methods,

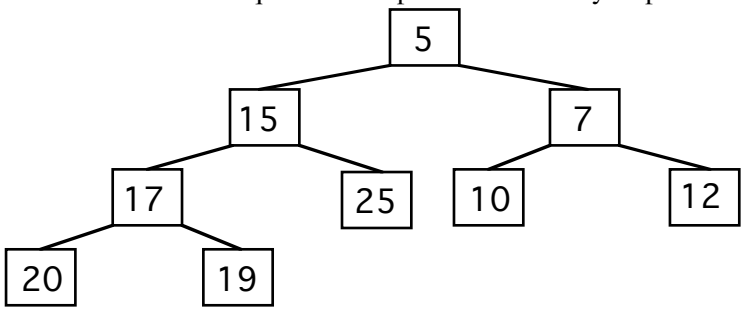
```
public void set(int n){
   value = n;
}

public int get(){
   return value;
}

public void write(int n){
   set(n);
   int temp = get();
   System.out.println("The value of n is "+temp);
}
```

Normally, executing the `write` method results in printing the value of n.

(i) What Java language construct can we use to ensure that only one thread is executing in these three methods at a time? Update the code above to show how it would be used with these methods.

(ii) What might happen if two threads could be executing in these methods at the same time? Describe a scenario in which `write` would not behave the same way as if only one thread were allowed in it at a time.

3. The following tree is a min-heap, as defined in class (i.e., each node is smaller than any of its descendents). For the rest of this question we presume the array implementation of heaps.



a. Suppose this heap is being used to represent a Priority Queue. Please illustrate (via a sequence of pictures) the steps that the computer must go through to remove the highest priority element (i.e., the one with lowest numeric value) from this heap, and restore the remaining elements to a heap. *Your pictures should be of trees rather than arrays.*

b. What is the worst-case time complexity of removing an element from a Priority Queue (and restoring it to a heap) with this heap representation if there are n elements in the heap before the removal takes place (use "O" notation). Explain your answer.

c. Please explain why this heap implementation is better overall than using our linked list implementation of regular queues and just keeping the elements in order by priority. (Your answer should compare complexities of adding and removing elements.)

4. The implementation in the text for class `BinarySearchTree` includes an instance variable root to hold a binary tree underlying the representation. However, we could instead define a binary search tree directly. We will assume the elements of the tree implement `Comparable`. Use the interface:

```
interface BSTInterface {
   // constant for empty tree
   public static BSTInterface EMPTY = new EmptyBSTree();

   // return whether tree contains a node with value
   public boolean contains(Comparable value);

   // create a binary search tree which includes all of the
   // old nodes plus a new node holding value.
   public void add(Comparable value)

   // return left subtree
   public BSTInterface left();

   // return right subtree
   public BSTInterface right();

   // update left subtree
   public void setLeft(BSTInterface newLeft);

   // update right subtree
   public void setRight(BSTInterface newRight);

}
```

Define classes `EmptyBSTree` and `NonEmptyBSTree` to implement `BSTInterface`. Class `EmptyBSTree` will have only a `parent` instance variable, and represents an empty binary search tree, while `BinaryTree` has instance variables `left`, `right`, `parent`, and `value` and represents a non-empty tree. Recall that interface `Comparable` has method `compareTo(other)`.

In defining these classes, you need only write the constructors and methods `contains` and `add`. You may assume all of the other have already been written and may use them in your code. As usual `setLeft` and `setRight` both set the parameter as the new value of the appropriate instance variable, but also set the parent link of the new child to refer to the parent.

The `add` method on an `EmptyBSTree` should result in setting its parent to refer to a new `NonEmptyBSTree` with contents value and empty left and right children. For a `NonEmptyBSTree`, it should add the new node as a leaf in a place that guarantees that the structure remains a binary search tree. The constructors for `EmptyBSTree` should take no parameters, while that for `NonEmptyBSTree` should construct a tree with a single node and should take as a parameter the value to be held at the root.

Be sure to include in your class definitions the constructors and methods `contains` and `add` for these two classes.

5. The event model in Java relies on associating listeners with objects that can generate events. Please illustrate this by completing the following program to place a button in a frame so that pushing a button results in printing "Button pushed!" on the output window. Use an "inner" class for the listener. Relevant method names are `addActionListener(ActionListener listener)` for class `JButton`, and `actionPerformed(ActionEvent evt)` for interface `ActionListener`.

```
public Exam extends JFrame
{
   protected Button printButton = new JButton("Push me");

   public static void main(String[] args){
      Exam app = new Exam();
      app.show;
   }

   public Exam(){
      super("Exam Frame");
      setLayout(new FlowLayout());
      printButton.setBackground(Color.red);
      add(printButton);
      // rest is up to you!
```

6. The `BinaryTree` class fails to override the `equals` method. We can fix that oversight by adding a method `equals` to `BinaryTree`:

```
// pre:  otherTree is a BinaryTree object
// post: returns true iff shape of receiver and otherTree
//  are same & corresponding values at nodes are equal.
public boolean equals(Object otherTree){
   return samePreorder((BinaryTree)otherTree) &&
           sameInorder((BinaryTree)otherTree)
}
```

The code is based on the fact that two `BinaryTree`'s are equal iff the corresponding values in the two preorder traversals are identical and similarly for the inorder traversals. Please fill in the code for the method `samePreorder` below (the code for `sameInorder` is similar). Hint: Use iterators on both the receiver of the message and `otherTree` in order to check equality of corresponding terms in the traversals. The name of the method of `BinaryTree` returning the appropriate iterator is `preorderIterator()`. The interface `Iterator` supports methods `hasNext()` and `next()`. Don't forget the case where one traversal ends before the other.

```
// post: returns true iff this and otherTree have exactly
// the same values in the preorder traversal
public boolean samePreorder(BinaryTree otherTree){
```