Second Exam– SAMPLE
Computer Programming 338
Dr. St. John
Lehman College
City University of New York
Tuesday, 5 April 2011

Exam Rules

- Show all your work. Your grade will be based on the work shown.
- The exam is closed book and closed notes.
- When taking the exam, you may have with you pens or pencils, and an 8 1/2" x 11" piece of paper filled with notes, programs, etc.
- You may not use a computer or calculator.
- All books and bags must be left at the front of the classroom during this exam.
- All pseudocode is from `http://en.wikipedia.org/wiki/` unless otherwise noted.
- **Do not open this exams until instructed to do so.**

1. Given the following pseudocode:

```
mystery(s)
  error = false
  for i = 0 to length(s)
    if s == 'a' or s == 'b'
      stack.push(s)
    if s == 'A'
      n = stack.pop()
      if n != 'a'
        error = true
        break
    if s == 'B'
      n = stack.pop()
      if n != 'a'
        error = true
        break
  if !stack.empty()
    error = true
  return error
```

   (a) What is `mystery("aA")`?
   (b) What is `mystery("abBA")`?
   (c) What is `mystery("aaa")`?
   (d) What is `mystery("aaabBAAA")`?
   (e) What does this method do?

2. Suppose we have numbers between 1 and 1000 in a binary search tree.

   (a) We want to search for the number 767. Which of the following could *not* be the sequence of nodes examined?

      i. 2000, 1000, 2, 90, 800, 600, 700, 770, 767
      ii. 20, 1100, 302, 900, 480, 610, 690, 766, 767
      iii. 32, 900, 44, 333, 404, 550, 666, 777, 767
      iv. 400, 1000, 409, 999, 500, 790, 797, 770, 767

   (b) Choose one of the sequences above that represents a sequence of nodes that could a search path and draw a binary search trees on which that search is performed.

3. Given the following array:

50	40	45	20	25	30	36	11	13	17

   Assume that it represents a binary tree with the following convention:
   `Parent(i) = i/2`, `Left(i) = 2i`, `Right(i) = 2i+1`

   (a) Draw the tree.
   (b) What is the height of the tree? Justify your answer.
   (c) Is this a binary search tree? Justify your answer.
   (d) Is this a heap? Justify your answer.

4. Given the following `Node` class and `insert` method:

```
class Node {
  String data;
  int depth;
  Node left;
  Node right;
  Node(String x, int d) {
    data = x;  depth = d;
    left = right = null;
  }
}
```

```
public static void insert(Node n, String x) {
  if (x.compareTo(n.data))
    if (n.left == null)
      n.left = new Node(x,n.depth+1);
    else insert(n.left, x);
  else if (n.right == null)
    n.right = new Node(x, n.depth+1);
  else insert(n.right, x);
}
```

   (a) Draw a picture of memory after the following:

```
Node r = new Node("first", 0);
for (int i=1; i < 5; i++) {
  insert(r, "Kid"+i);
```

   (b) Continuing from above, draw a picture of memory after the following:

```
for (int i=3; i > 0; i--) {
  insert(r, "Student"+i);
```

   (c) What is the height of the tree?
   (d) How does the variable `depth` of a node correspond to the height?

5. Given the `node` class above, what do the following methods do:

```
mystery(Node r)
  if (r.left == null AND r.right == null)
    return 1
  else if r.left == null
    return 2*mystery1(r.right)
  else if r.right == null
    return 2*mystery1(r.left)
  else
    return 2*max(mystery(r.left),mystery(r.right));
```

```
mystery2(Node r)
  print("(")
  if (r.left != null)
    mystery2(r.left)
  print(",")
  if (r.right != null)
    mystery2(r.right)
  print(")");
```

6. What is the asymptotic running time of the following methods (that is, give a "big Oh" analysis):

   (code from: `http://javaj2eesolution.blogspot.com/2008/11/avl-tree-using-java.html`)

   (a)
```
private AvlNode rotateWithLeftChild( AvlNode k2 )
{
  AvlNode k1 = k2.left;
  k2.left = k1.right;
  k1.right = k2;
  k2.height = Math.max( height( k2.left ), height( k2.right ) ) + 1;
  k1.height = Math.max( height( k1.left ), k2.height ) + 1;
  return k1;
}

private AvlNode rotateWithRightChild( AvlNode k1 )
{
  AvlNode k2 = k1.right;
  k1.right = k2.left;
  k2.left = k1;
  k1.height = Math.max( height( k1.left ), height( k1.right ) ) + 1;
  k2.height = Math.max( height( k2.right ), k1.height ) + 1;
  return k2;
}

private AvlNode doubleWithLeftChild( AvlNode k3 )
{
  k3.left = rotateWithRightChild( k3.left );
  return rotateWithLeftChild( k3 );
}
private AvlNode doubleWithRightChild( AvlNode k1 )
{
  k1.right = rotateWithLeftChild( k1.right );
  return rotateWithRightChild( k1 );
}
```

(b)
```
private AvlNode insert( AnyType x, AvlNode t )
{
  if( t == null )
    return new AvlNode( x, null, null );
  int compareResult = x.compareTo( t.element );
  if( compareResult < 0 ) {
    t.left = insert( x, t.left );
    if( height( t.left ) - height( t.right ) == 2 )
      if( x.compareTo( t.left.element ) < 0 )
        t = rotateWithLeftChild( t );
      else
        t = doubleWithLeftChild( t );
  }
  else if( compareResult > 0 ) {
    t.right = insert( x, t.right );
    if( height( t.right ) - height( t.left ) == 2 )
      if( x.compareTo( t.right.element ) > 0 )
        t = rotateWithRightChild( t );
      else
        t = doubleWithRightChild( t );
  }
  else
    ; // Duplicate; do nothing
  t.height = Math.max( height( t.left ), height( t.right ) ) + 1;
  return t;
}
```

7. Assuming the following basic operations for a tree have been defined:

```
Parent(i)      //returns the parent of node i
Left(i)        //returns the left child of node i
Right(i)       //returns the right child of node i
Key(i)         //returns the key (priority value) of node i
```

Write the following methods for a heap:

```
max(H)         //returns the maximum element of the heap H
heapify(H,i)   //assumes the trees rooted at Left(i) and Right(i) are heaps,
               //H[i] may be smaller than its children.  Makes H[i] a heap.
extractMax(H) //removes and returns the maximum element of the heap H
```

Make sure that upon completion of your methods that the properties of a heap are maintained.

8. Assuming the methods above, write out a method that takes a heap H and prints out the nodes in the heap in order of priority.

9. Using an array as your base, write the following operations for a stack:

```
int[] myStack = new int[20];
int top;
int bottom;

int pop()           //remove and returns the top element of the stack
void push(int x)    //pushes x onto the top of the stack
boolean isEmpty()   //returns true if the stack is empty, false otherwise
boolean isFull()    //returns true if the stack is full, false otherwise
```

10. (a) Write a class `BTreeNode` which has the following properties:
    i. Every node has m children (of type `BTreeNode`).
    ii. Every node has m-1 keys (of type integer).

    (b) Write a method for your class called:

    ```
    public static void insert(BTreeNode root, int key)
    ```

    that inserts the `key` into an existing BTree rooted at `root`. Your insert should preserve the following properties:

    i. Every node (except root) has at least $m2$ children.
    ii. The root has at least two children if it is not a leaf node.
    iii. All leaves appear in the same level, and carry information.
    iv. A non-leaf node with k children contains $k?1$ keys.

    Further, each internal node's elements act as separation values which divide its subtrees. For example, if an internal node has three child nodes (or subtrees) then it must have two separation values or elements a_1 and a_2. All values in the leftmost subtree will be less than a_1, all values in the middle subtree will be between a_1 and a_2, and all values in the rightmost subtree will be greater than a_2.