CS 61B Exam 2
July 31, 2008

**Read and fill in this page now.**
**Do NOT turn the page until you are told to do so**

Your name: _____

Your login name: _____

Your discussion section day and time: _____

Your discussion section t.a.: _____

Name of the person sitting to your left: _____

Name of the person sitting to your right: _____

Problem 0: _____ Total : _____ / 50

Problem 1: _____ Problem 5: _____ _____

Problem 2: _____ _____ Problem 6: _____

Problem 3: _____ _____ _____

Problem 4: _____ _____

This is an open-book test. You have approximately one hour and fifty minutes to complete it. You may consult any books, notes, or other paper-based inanimate objects available to you. To avoid confusion, read the problems carefully. If you find it hard to understand a problem, ask us to explain it. If you have a question during the test, please come to the front or the side of the room to ask it. To help us understand your solution or code, please state all assumptions and provide explanatory comments next to your code.

This exam comprises 12.5% of the points on which your final grade will be based. Partial credit may be given for wrong answers. Your exam should contain 7 problems (numbered 0 through 6) on thirteen pages. Please write your answers in the spaces provided in the test; in particular, we will not grade anything on the back of an exam page unless we are clearly told on the front of the page to look there.

You need not rewrite a function that appears in any of the textbooks or handouts; merely cite the page in the textbook, the reading material, or the handout in which the function appears. Some students are taking this exam late. Please do not talk to them, mail them information, or post anything about the exam to news groups or discussion forums until after Friday.

Relax—this exam is not worth having heart failure about.

**Problem 0 (1 point)**

Put your login name on each page. Also make sure you have provided the information requested on the front page.

**Problem 1 (10 points)**

a) If `q` is Queue, what does the following code print

```
q.enqueue("a");
q.enqueue("b");
q.enqueue("c");
q.dequeue();
q.enqueue("d");
System.out.println(q.dequeue() + q.dequeue() + q.dequeue());
```

Ans: ____bcd_____

b) If `s` is a Stack, what does the following code print

```
s.push("a");
s.push("b");
s.push("c");
System.out.println(s.pop() + s.pop() + s.pop());
```

Ans: _____ cba_____

c) In terms of *m* and *n*, how many times does this snippet print hello:

```
for (i = m; i > 0;i--){
      for (j = n; j > 0; j--){
         System.out.println("hello");
      }
}
```

Ans: ___m*n_____

d) In terms of m, how many times does this snippet print hello:

```
for (i = m; i > 1 ; i /= 3){
      System.out.println("hello");
}
```

Ans: __$O(\log_3 m)$_____

e) Recall the definition for *Completely Balanced Binary Tree*:

*"An empty tree and a binary tree containing just one node are both completely balanced; any other binary tree is completely balanced if and only if the height of its left child is equal to the height of its right child, and its left and right children are also completely balanced."*

How many nodes does a completely balanced binary tree of depth *n* contain (a binary tree consisting of a single node would be depth 1)?

A completely balanced tree will be full on every level.
# of nodes at depth 1: 2^0 = 1
# of nodes at depth 2: 2^1 = 2
# of nodes at depth 3: 2^2 = 4
# of nodes at depth n: 2^(n-1)

Summing from level 1 to n 1+ 2 + 4 + … + 2 ^(n-1) can be simplified to (2^n) - 1.

f) Recall the *isMaximallyBalanced* method we asked you to code in one of the labs:

*"The isMaximallyBalanced method returns true either when the tree is completely balanced, or when all the leaves are on level k and level k+1 for some k, level k is full, and the leaves at level k+1 are as far left as possible, and returns false otherwise. (The level of a leaf is its distance from the root.)"*

What's the fewest number of nodes a maximally balanced tree of depth *n* can have (a tree consisting of a single node would be depth 1) ?

The smallest maximally balanced tree of depth n is a completelybalanced tree of depth n-1 plus one more node. So using the formula from above we get 2^(n-1).

g) Describe what effect would the following hash code method have on a hash table with keys of type `SomeClass`.

```
public class SomeClass{
      ...

      public int hashCode {
            //return a random number under 10.
            return Math.random(10);
      }
}
```

This is not a valid hash function since it is nondeterministic, meaning it can return different values when called on the same object multiple times. This can be a problem because you may put it in one bucket and search for it in another. This means the hash table won't work at all.

**Problem 2 (4 points)**

a) Is $2^{n+1} = O(2^n)$? Explain why or why not.

Yes. $2^{n+1} = 2 * 2^n$ is_in $O(2^n)$

b) Is $2^{2n} = O(2^n)$? Explain why or why not.

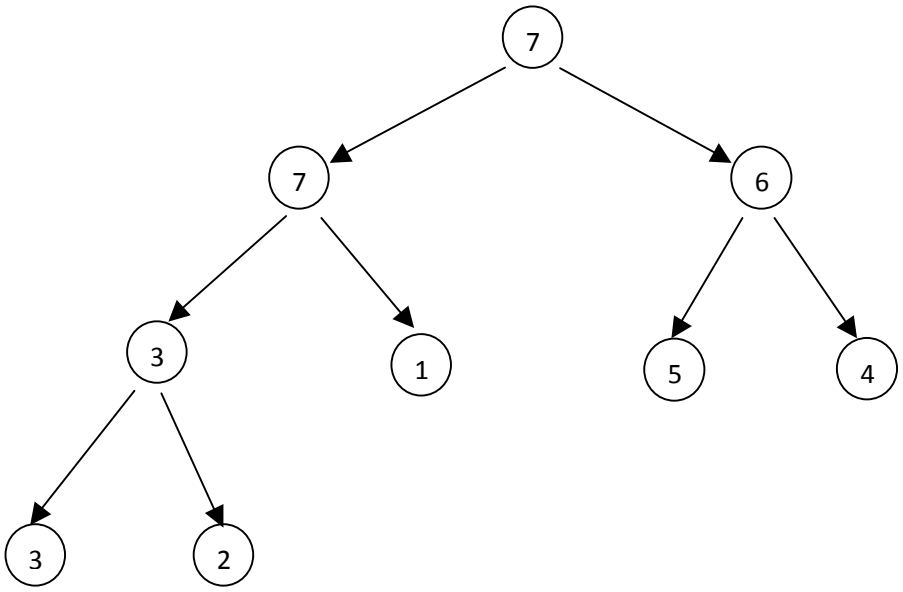
No. $2^{2n} = 2^n * 2^n$ is_not_in $O(2^n)$

**Problem 3 (11 points)**

Consider the *max heap* below, represented in an array

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 7 | 7 | 6 | 3 | 1 | 5 | 4 | 3 | 2 |

a) Draw this max heap as a binary tree



b) Delete the top of the max heap, using the standard deletion algorithm we discussed in class. Indicate, step by step, the change made to the array to perform the deletion in the boxes below (you won't need to use all the boxes).

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 7 | 7 | 6 | 3 | 1 | 5 | 4 | 3 | 2 |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| ***2*** | 7 | 6 | 3 | 1 | 5 | 4 | 3 |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 7 | ***2*** | 6 | 3 | 1 | 5 | 4 | 3 |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 7 | 3 | 6 | ***2*** | 1 | 5 | 4 | 3 |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 7 | 3 | 6 | 3 | 1 | 5 | 4 | ***2*** |

c) Given the following input array

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 6 | 3 | 3 | 7 | 4 | 2 | 3 | 5 | 1 |

Show step by step, the changes made to the array using the `bottomUpHeap` method we discussed in class to turns the above array into a *max heap* (you won't need all the boxes).

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 6 | ***7*** | 3 | ***3*** | 4 | 2 | *3* | *5* | 1 |

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 6 | 7 | 3 | ***5*** | 4 | 2 | *3* | ***3*** | 1 |

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| ***7*** | ***6*** | 3 | 3 | 4 | 2 | *3* | *5* | 1 |

**Problem 4 (6 points)**

Suppose we are designing a hash function for the following class:

```
public class myIntArray {
      int[] elems;

      ...

      int hashCode1() {
            return elems[0];
      }

      int hashCode2() {
            int total = 0;
            for(int i = 0; i<elems.length; i++) {
                  total += elems[i];
            }
            return total;
      }

      int hashCode3() {
            int total = 0;
            int scale = 1;
            for(int i = 0; i<elems.length; i++) {
                  total = total + scale * elems[i];
                  scale *= 11;
            }
            return total;
      }

      int hashCode4() {
            int total = 0;
            int scale = 1;
            for(int i = 0; i<elems.length; i++) {
                  total = total + scale * elems[i];
                  scale *= 10;
            }
            return total;
      }
}
```

Suppose `myIntArray` objects containing the following arrays are added to a very large hash table that uses *chaining* to deal with collisions. Also assume that no equal `myIntArray` objects are ever inserted into the hash table.

```
myIntArray A = {1,4,7}
```

```
myIntArray B = {1,8,0,3}
myIntArray C = {1,3,9,4}
myIntArray D = {9,8}
```

a) Show the content of all non-empty buckets as well as how they are chained together, after adding A, B, C, and D (in this order) to the table using the different hashing functions.

*Example:*
*Consider the following hash code:*

```
int hashCode() {
      return elems.length;
}
```

*After inserting A, B, C, and D into our hash table, the content of the hash table can be expressed as:*

*D, A, B -> C*

i) `hashCode1`

A->B->C, D

ii) `hashCode2`
A->B,C->D

iii) `hashCode3`

D,A,B,C

iv) `hashCode4`

D,A,B,C

b) Is it possible for `hashCode3` to have any collisions? Explain why or why not?

Yes. An example: A = {1, 11}  B = {11,1}

**Problem 5 (8 Points)**

a) For each of the arrays below, determine if the array could result from an application of the partition step in Quicksort for some given array. If so, explain which array elements could have been the pivot for the partition. If not, explain why not.

| *array* | *result of quick sort partition? yes/no* | *explanation* |
|---|---|---|
| 3 1 4 5 2 7 6 9 8 | No | No element x divides the array into right and left arrays where elems on right >= element and elems oN left <= element |
| 3 4 2 5 1 6 9 7 8 | Yes | 6 – everything on the left of 6 is < 6 Everything on the right of 6 is > 6 |

b) For each of the arrays below, determine if it could result after some number of iterations of the Selection Sort algorithm. If so, explain which element would be the *last possible* element to be placed in its correct position. If not, explain why not.

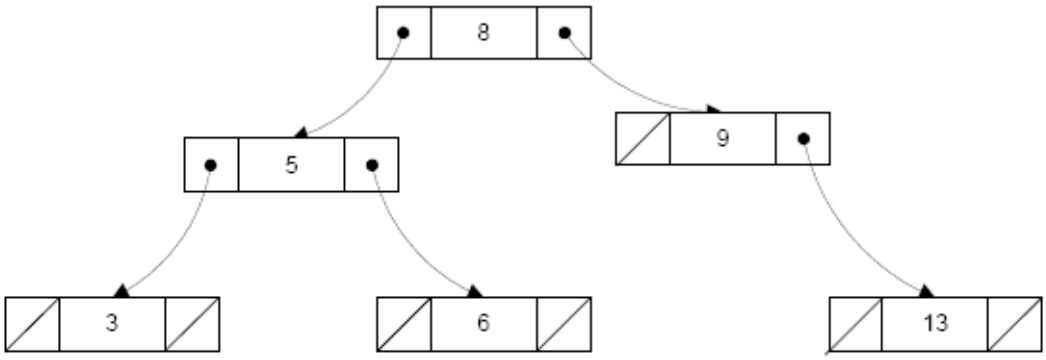
| *array* | *result of some iterations of selection sort? yes/no* | *explanation* |
|---|---|---|
| 3 4 6 5 8 9 7 | Yes | 4 |
| 3 4 5 6 2 8 9 | No | 2- the smallest element is not in its correct position |

**Problem 6 (10 points)**

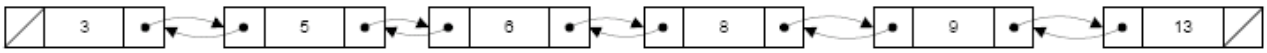
Consider the following class:

```
public class SpecialTree {

      SpecialNode theRoot;

      class SpecialNode{
            int myItem;
            SpecialNode a;
            SpecialNode b;
      }
}
```

Imagine we are using the class `SpecialTree` to represent a Binary Search Tree, with the class `SpecialNode` representing the nodes of this tree. Each `SpecialNode` contains an integer item, a `SpecialNode a` representing the left child and a `SpecialNode b` representing the right child.




Write a method `convertToDoublyLinkedList` to convert this binary tree to a *sorted doubly linked list*. For a given `SpecialNode`, `a` should point to the node before you and `b` should point the node after you. The root of the `SpecialTree` should point to the first node of this doubly linked list. You should write `convertToDoublyLinkedList` destructively -- you can declare variables of type `SpecialNode` but you may not make calls to `new SpecialNode`. Feel free to use a helper method. The above tree should become:



Solution1:

```
public void convertToDoublyLinkedList(){

      theRoot.convertToDoublyLinkedListHelper();
      while(theRoot.a != null) {
            theRoot = theRoot.a;
      }
}

public void converToDoublyLinkedListHelper() {
      // Now go to the end of a's linked list and connect it to us
      if (a != null){
            // Recursively call convert on the left node
            a.convertToDoublyLinkedList();
            SpecialNode temp = a;
            while(temp.b != null){
                  temp = temp.b;
            }
            a = temp;
            temp.b = this;
      }
      if (b != null){
            // Recursively call convert on the right node
            b.convertToDoublyLinkedList();
            // Now go to the beginning of b's linked list and connect
      it to us
            temp = b;
            while (temp.a ! = null){
                  temp = temp.a;
            }
            b = temp;
            temp.a = this;
      }
}
```

Solution 2:

```
      public void convertToDoublyLinkedList() {
       helper(theRoot);
       theRoot = theRoot.a;
      }

      // This is a SpecialTree method
      // Stores the head and tail of a sorted, doubly linked list in
      // theRoot.{a,b}.
      private void helper(SpecialNode node) {
       SpecialNode head = node, tail = node;
```

```
      if(node.b != null) {
        helper(node.b);
        node.b = theRoot.a;
        node.b.a = node;
        tail = theRoot.b;
      }
      if(node.a != null) {
        helper(node.a);
        node.a = theRoot.b;
        node.a.b = node;
        head = theRoot.a;
      }
      theRoot.a = head;
      theRoot.b = tail;
     }
```

Solution 3:

```
public void convertToDoublyLinkedList() {
 theRoot = helper(theRoot)[0];
}

// Returns the head and tail of a sorted, doubly linked list.
public static SpecialNode[] helper(SpecialNode node) {
 SpecialNode[] temp, retval = new SpecialNode[2] {node, node};
 if(node.b != null) {
   temp = helper(node.b);
   node.b = temp[0];
   node.b.a = node;
   retval[1] = temp[1];
 }
 if(node.a != null) {
   temp = helper(node.a);
   node.a = temp[1];
   node.a.b = node;
   retval[0] = temp[0];
 }
 return retval;
}
```