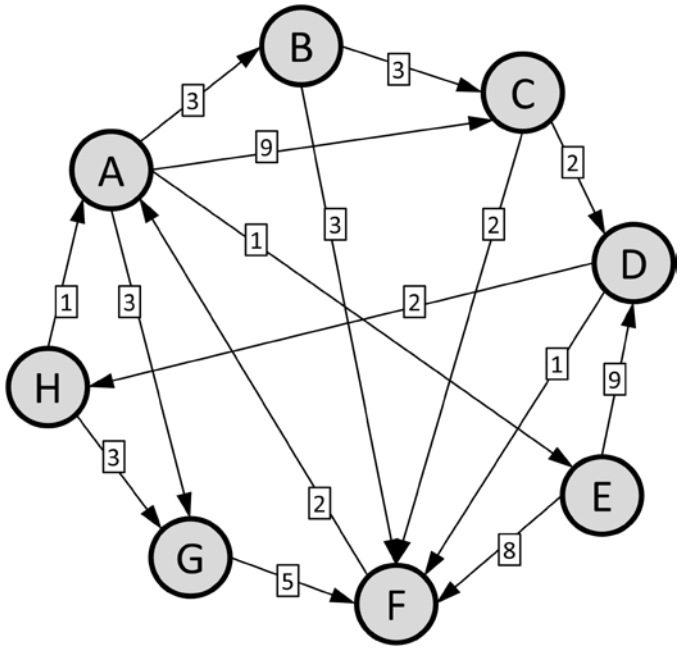
KEY
FINAL EXAMINATION
MAY 2012

1. Graph Traversal (12 points)

a.) Consider the directed graph at right. Simulate Dijkstra's shortest path algorithm starting at node H. In your results, give (i) the order in which the nodes are visited, (ii) all the cost labels computed for each node as the algorithm runs, and (iii) the homeward pointing edge associated with each of those costs. (For example, for node X you might say that the initial infinite cost was replaced with a cost of 12 via node Y, then with a cost of 8 via node Z, and finally with a cost of 7 via node W.)



In order of visitation:

- *H has cost 0*
- *A has cost 1 via H*
- *E has cost 2 via A*
- *G has cost 3 via H*
- *B has cost 4 via A*
- *C has cost ~~(10 via A)~~ 7 via B*
- *F has cost ~~(10 via E)~~ ~~(8 via G)~~ 7 via B*
- *D has cost ~~(11 via E)~~ 9 via C*

A few people gave answers as though the graph was undirected.

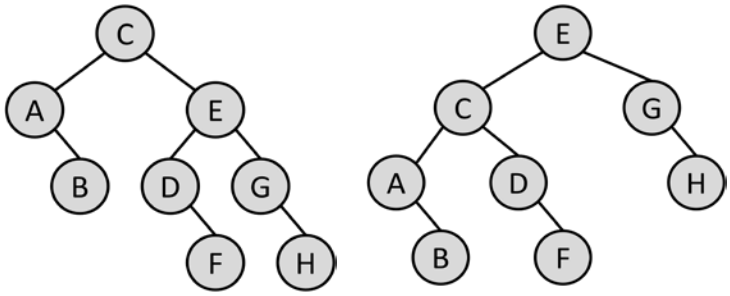
b.) For which starting nodes could node E appear in the fourth position of a breadth-first traversal? You may break ties in any manner you please. For example, BFAECGDFH is a valid breadth-first traversal that starts at B and has E in the fourth position, so you would list B as part of your answer.

Egg all over my face: the traversal I gave as an example is NOT a valid breadth-first traversal, because C has to be visited before A. In fact, there is actually no valid BFT that starts at B and has E in the fourth spot. Only one person noticed this (for full credit). I did not take off points for people who listed B in their answers. However, many people made a different dort of mistake: they listed all the nodes that could have E in the fourth position for a depth-first traversal, whereas the question asked about breadth-first. Here are the answers (note that when we have several nodes to choose from, we put E fourth):

A: Yes (ACGE…). B: No (Despite what I said in the problem…). C: No. D: No. E: No. F: Yes (FAGE… or FACE…). G: Yes (GFAE…). H: Yes (HAGE…)

2. **Trees** (12 points)

a.) Consider the tree at right. Draw the tree that would result from a left-rotation of the root.



b.) List the order in which the tree nodes would be visited in a postorder traversal.

BAFDHGEC

c.) What changes would you need to make to the original tree, if any, to make it a valid binary search tree (assuming the ordering relation on nodes is alphabetical)?

Swap F and E.

3. **Programming** (8 points)

Rewrite the snippets of Java code below either to make them more efficient or more in line with the style guidelines promoted in this course. The functionality should remain unchanged.

a.) Simplify the redundant if/else:

```
if (x==7) {
    return true;
} else {
    return false;
}
```

```
public static boolean equals7ans(int x) {
        return (x==7);
    }
```

b.) Combine several methods into one more general method:

```
public int addOne(int x) {
    return x+1;
}

public int addTwo(int x) {
    return x+2;
}

public int addThree(int x) {
    return x+3;
}
```

```
public int addN(int x, int n) {
    return x+n;
}
```

c.) Use arguments and return values to pass information, instead of modifying global fields.

```
public class Cheer {
    public static int cheerNumber;

    public static void method1() {
        cheerNumber = (int)Math.floor(Math.random()*3)+1;
    }

    public static void method2() {
        for (int i = 0; i < cheerNumber; i++) {
            System.out.println("Hip Hip Hooray!");
        }
    }

    public static void main(String[] args) throws IOException {
        method1();
        method2();
    }
}
```

```
public class CheerAnswer{
    public static void howManyCheers() {
        return (int)Math.floor(Math.random()*3)+1;
    }

    public static void doCheers(int cheerNumber) {
        for (int i = 0; i < cheerNumber; i++) {
            System.out.println("Hip Hip Hooray!");
        }
    }

    public static void main(String[] args) throws IOException {
        doCheers(howManyCheers());
    }
}
```

d.) Use descriptive variable names.

```
BufferedReader input = new BufferedReader(new InputStreamReader(System.in));
    System.out.println("Please enter your age in years.");
    String temp1 = input.readLine();
    int temp2 = Integer.valueOf(temp1)*12;
    System.out.println("That is "+temp2+" months!");
```

```
System.out.println("Please enter your age in years.");
String ageInYears = input.readLine();
int ageInMonths = Integer.valueOf(temp1)*12;
System.out.println("That is "+ageInMonths+" months!");
```

4. **Hash Tables** (12 points)

Consider the hash table shown below, which uses the simple hash function h(k) = k **mod** 7 and handles collisions via simple linear probing. Answer the questions that follow.

Key	**Value**
49	Tango
13	Alpha
73	Charlie
24	Bravo
20	Foxtrot

a.) List all the (key,value) pairs that are not stored at their home position in the table.

(13, Alpha) and (24,Bravo)

b.) List all the (key,value) pairs which, if removed, would cause other (key,value) pairs to change their position.

(49,Tango),(73,Charlie), and (20,Foxtrot)

c.) If the table was initially empty, and the (key,value) pairs you see were added in some sequence without any other intervening operations, what can you infer about their relative order of insertion? List all sets of (key,value) for which you can determine that one must have been inserted before the other, and give the ordering.

(49,Tango) and(20,Foxtrot) went in before (13,Alpha).

(73,Charlie) went in before (24,Bravo).

5. **Program Analysis** (16 points)

Give the asymptotic runtime performance ("big-O notation") of the following sets of operations, in terms of the problem size **n**.

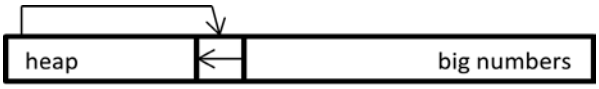
a.) All the elements in an unsorted list of length **n** are added one at a time to a rebalancing binary search tree (such as a red-black tree).

Tree insertion takes O(log n). Since we are doing it for n elements, the overall time is O(n log n).

b.) All the elements in a sorted array of length **n** are added to an ordinary binary search tree using a recursive method that adds the middle element of the array at the root, and then recursively builds the left and right subtrees using the left and right halves of the array.

In this case, insertion is O(1) because we build the tree recursively. We must still touch each element once, so the whole thing is O(n).

c.) An unsorted array is converted to a heap in place. Next the heap is shrunk by popping the largest element and moving it to the vacated point in the array, as shown in the picture at right.



This is heapsort. It takes O(n log n).

d.) Given an array of **n** different integers $a_0 \ldots a_{n-1}$, a double nested loop examines each pair (a_i,a_j) to determine whether they are relatively prime. (Assume for simplicity that testing relative primeness is a constant-time operation.) The result is a boolean value, which is stored in a hash table using the pair of numbers as the key.

We process each pair of numbers. Since there are n numbers, this takes $O(n^2)$ operations.

6. **Recursion** (12 points)

Consider the recursive method in the box.

a.) What output would be generated by a call to printRecursive(4)?

```
public static void printRecursive(int x) {
    if (x > 0) printRecursive(x-1);
    System.out.print(x+" ");
    if (x > 0) printRecursive(x-1);
}
```

0 1 0 2 0 1 0 3 0 1 0 2 0 1 0 4 0 1 0 2 0 1 0 3 0 1 0 2 0 1 0

b.) How many times in total does printRecursive get called in the course of executing printRecursive(4)?

31 times. (Each time prints out exactly one number in its middle line.)

c.) If the second recursive call (the last line of the method body) is commented out, what output would be generated by a call to printRecursive(4)?

0 1 2 3 4

7. **Abstract Data Types** (8 points)

Give the abstract data type description for a list iterator: Describe the operations it must implement, any data it must keep track of, the effects of the operations on the data, and the inputs and outputs for the operations. You may omit the operations that modify the list.

The list iterator has access to a list $A_0 \ldots A_{n-1}$, and keeps track of a position j in the list, $0 \le j \le n$.

It must implement the following operations:

next : increment j and return A_{j-1} (Error if j==n initially)

previous : decrement j and return A_j (Error if j==0 initially)

hasNext: return true iff j<n

hasPrevious: return true iff j>0

nextIndex: return j (Error if j==n)

previousIndex: return j-1 (Error if j==0)

8. **Inheritance** (8 points)

Consider the program at right. For each item below, write a line of Java code that you could place within the main() method to accomplish the task described, or write "not possible" if no such command exists. Use casts only when necessary.

I should have initialized the two variables in main:

n1 = new Nested1();

n2 = new Nested2();

I did not take off points if people omitted this step.

a.) Cause n1.toString() to return "Crow Hen" when called from main(). *Not possible. (Method M1 will change the a1 field of the static field n1, not thelocal variable called n1 in main. Thus n1.toString will still return "Dove Hen".*

b.) Cause n1.toString() to return "Robin Hen" when called from main(). *n1.M2();*

c.) Cause n1.toString() to return "Dove Velociraptor" when called from main(). *Not possible.*

d.) Cause n2.toString() to return "Dove Squirrel Auk Kiwi" when called from main(). *n2.M3();*

e.) Cause n2.toString() to return "Dove Hen Squirrel Kiwi" when called from main(). *Not possible.*

f.) Cause n2.toString() to return "Dove VelociraptorAuk Kiwi" when called from main(). *Not possible*

g.) Cause n2.toString() to return "Dove Hen Velociraptor Kiwi" when called from main(). *N2.M2();*

h.) Cause n2.toString() to return “Dove Hen Auk Squirrel” when called from main().

N2.M3(“Squirrel”);

9. **Sorting** (12 points)

Write a short essay summarizing all the sorting algorithms we have studied this semester: their asymptotic runtime, a brief description of how they work, and any distinctive features (what kind of extra storage space they require, if any, and whether thay can perform a stable sort, for example).

Selection sort: Chooses lowest remaining unsorted element and adds it to the sorted sequence at the end. $O(n^2)$, stable, in place.

Insertion sort: Chooses the next remaining unsorted element and adds it to the sorted sequence in its correct position. $O(n^2)$, stable, in place.

Merge sort: Combine small sorted lists into successively larger ones. O(n log n).

Quick sort: Recursively subdivide array into smaller and larger portions. O(n log n), stable, in place.

Heap sort: Convert array to heap; convert heap to sorted array. O(n log n), in place.

```
public class Inheritance {
    public static Nested1 n1 = new Nested1();

    public static void main(String[] args) {
        Nested1 n1;
        Nested2 n2;
    }

    public static class Nested1 {
        public String a1;
        public String a2;

        public Nested1() {
            a1 = "Dove";
            a2 = "Hen";
        }
        public static void M1() {
            n1.a1 = "Crow";
        }
        public void M2() {
            a1 = "Robin";
        }
        public void M3() {
            a2 = "Squirrel";
        }
        public String toString() {
            return a1+" "+a2;
        }
    }

    public static class Nested2 extends Nested1 {
        public String a2;
        public String a3;

        public Nested2() {
            super();
            a2 = "Auk";
            a3 = "Kiwi";
        }
        public void M2() {
            a2 = "Velociraptor";
        }
        public void M3(String s) {
            a3 = s;
        }
        public String toString() {
            return super.toString()+" "+a2+" "+a3;
        }
    }
}
```