FINAL EXAMINATION
MAY 2017
CSC 212 ♦ SECTION 02
INSTRUCTOR: NICHOLAS R. HOWE

YOU MAY USE TWO 8.5"x11" SHEETS OF NOTES ON THIS EXAM.
YOU MAY NOT USE THE TEXTBOOK, A COMPUTER, OR ANY OTHER INFORMATION SOURCE BESIDES YOUR TWO PAGES OF NOTES.

All work should be written in the exam booklet.
Partial credit will be granted where appropriate if intermediate steps are shown.

Vocabulary (16 points)
Explain the function of each of the follow Java keywords in 1-2 sentences. Include what they apply to (e.g., methods, variables)

1. new
2. static
3. public
4. throw
5. extends
6. this
7. super
8. interface

Inheritance (8 points)

Given the following classes, what gets printed when Inherit.main() is invoked?

```
public class Big {
    private int x = 0;

    public Big() {
        x = 1;
    }

    public Big(int b) {
        x = b;
    }

    public int getNumber() {
        return x;
    }
}

public class Middle extends Big {

    public Middle() {
        super(2);
    }

    public Middle(int m) {
        super(m);
    }

}
```

```
public class Small extends Middle {
    public int x = 0;

    public Small() {
        super(3);
    }

    public Small(int s) {
        super();
        x = s;
    }

    public int getNumber() {
        return super.getNumber() + x;
    }
}

public class Inherit {

    public static void main(String[] args) {
        Big b1 = new Big();
        Big b2 = new Big(10);
        Small s1 = new Small();
        Small s2 = new Small(20);

        System.out.println(b1.getNumber());
        System.out.println(b2.getNumber());
        System.out.println(s1.getNumber());
        System.out.println(s2.getNumber());
    }
}
```

Lists and Iterators (8 points)

Consider the sequence A B C D E stored in a **`LinkedList<String>`** and the code snippets shown below. Assuming all begin with the same configuration shown above, give the final contents of the list after the code shown has executed. If the code would generate an exception, specify what it would be.

```
a.)
   ListIterator<String> mark = list.listIterator();
   mark.next(); mark.next(); mark.add(mark.previous());
b.)
   ListIterator<String> mark = list.listIterator(3);
   mark.previous(); mark.remove();
c.)
   ListIterator<String> mark = list.listIterator();
   while (mark.hasNext()&&!mark.next().equals("B")){
       if (mark.next().equals("C")||mark.next().equals("D")) {
         mark.set("Z");
       }
   }
d.)
   list.removeFirst();
   ListIterator<String> mark = list.listIterator(list.size());
   mark.add(list.getFirst());
   list.addLast("F");
```

Graphs (10 points)

Genetic programming represents competing solutions to a problem as sequences of bits, and apply a forced evolutionary process to identify solutions with high "fitness". In each generation, new solutions (bit strings) are created out of older ones by applying one of several mutations. The genome space can be viewed as a graph, where each possible bit string is a vertex, and possible mutations are edges. Suupose that a particular project allows two types of mutations: a **flip**, which changes one bit in the string from a 0 to a 1 or vice versa, and a **shift**, which cycles all the bits one position to the left or right, with the first or last bit wrapping around to the other end. (For example, 00110 might become 01110 via a flip, or it might become 01100 via a shift.)

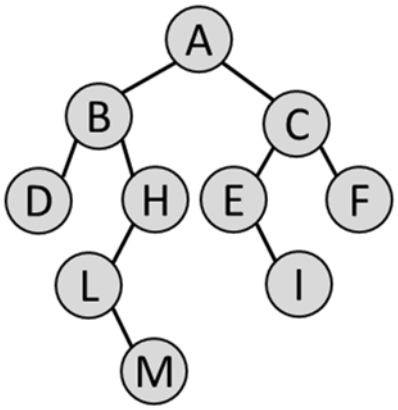
a.) Draw the graph showing the genome space for bit strings of length three. There will be a node for each solution, and edges for each possible mutation.

b.) What is the degree of this graph (the maximum degree of any node)?

c.) Which traversal algorithm (DFT or BFT) could be used with slight modification to compute the number of mutations required to transform any one individual into some other?

d.) Suppose that flips have a cost of 1.0, while shifts have a cost of 1.5. What is the least expensive way to transform 100 into 011?

Trees and Recursion (8 points)



Consider the tree shown at right and the code below, which appears in a subclass of **`BinaryTree<String>`**. The programmer had intended to create three methods implementing preorder, inorder, and postorder traversal. Unfortunately, the recursive calls got mixed up, so that each method calls one of the others instead of calling itself!

Given the three methods as written below, what would be the full string returned by a call to **`inString()`** on the tree shown at right?

```
public String preString() {
    String left = "";
    String right = "";
    if (getLeft() != null) { left = getLeft().inString(); }
    if (getRight() !=null) { right = getRight().inString(); }
    return "{"+getData()+left+right+"}";
}
public String inString() {
    String left = "";
    String right = "";
    if (getLeft() != null) { left = getLeft().postString(); }
    if (getRight() !=null) { right = getRight().postString(); }
    return "("+left+getData()+right+")";
}
public String postString() {
    String left = "";
    String right = "";
    if (getLeft() != null) { left = getLeft().preString(); }
    if (getRight() !=null) { right = getRight().preString(); }
    return "["+left+right+getData()+"]";
}
```

Hash Tables (10 points)

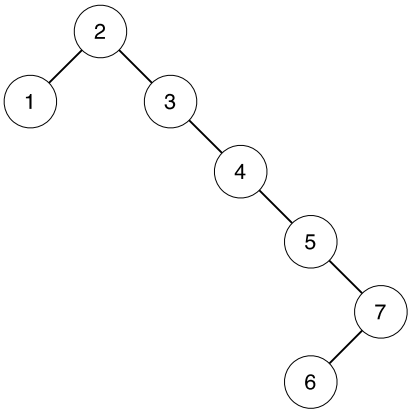
A set of eight key-value pairs is inserted into a hash table that uses open addressing with linear probing, resulting in the table shown at right.

0	(99, B)
1	
2	(42, D)
3	(13, A)
4	(12, E)
5	(3, G)
6	(73, F)
7	(16, H)
8	
9	(19, C)

a.) Assuming that the hash function is h(k) = k mod 10, draw a diagram of a new hash table storing the same elements (in the same order) that uses chaining to handle collisions instead of open addressing.

b.) A naïve programmer mistakenly modifies the hash function to return 6 for every possible key. Explain how this affects the performance of inserting and accessing new elements when using open addressing with linear probing.

Binary Search Trees (12 points)

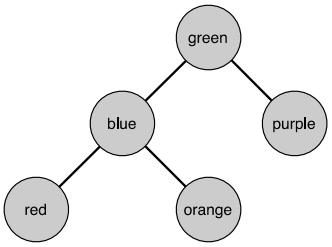


Please answer the questions below concerning BST:

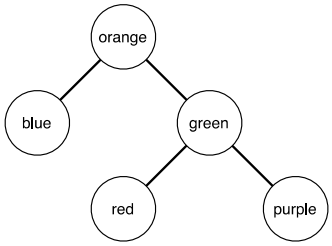
a.) Elements 1-8 are inserted one by one into a BST resulting in the tree shown at right. Assuming no post-insert balancing is done, provide a valid ordering of insertion that would produce this tree as shown.

b.) Provide a different insertion order that would instead create a completely balanced tree of height 3.

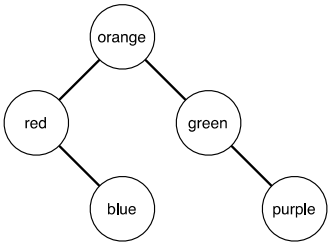
c.) The tree at right is a properly formed BST. The labels of the nodes have no relation to the hidden data fields. Given this tree, which of the following alternatively structured trees below will be valid BSTs? If they are not valid, explain why.



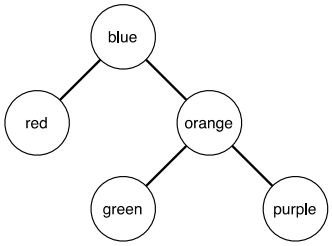
i.



ii.



iii.



Heaps (16 points)

Your boss is somewhat eccentric, and likes the number three better than the number two. He would like you to create a new data structure called a *TriHeap*, which is like an ordinary heap except that every node can have three children instead of two. Answer the following questions about TriHeaps.

a.) Suppose that the following numbers are inserted into a TriHeap that is initially empty, in the following order: 3, 9, 27, 6, 12, 15, 35, 18, 24, 42, 51, 30, 33, 81. Draw the tree that results.

b.) Beginning with the tree above, suppose that the top number is popped off the TriHeap three times. Draw the tree that results.

c.) Show how your answer to (b) above would be stored in an array indexed from 0, using the same convention as for an ordinary heap.

d.) Using your answer to (c) above, give a formula for finding the index of the parent node, in terms of the index *i* of some starting node.

Data Structures (12 points)

Your cousin Vinnie is always trying to sell you on his latest get-rich-quick scheme. Today he comes to you claiming to have invented several new data structures and algorithms that are better than anything out there. Based on our studies of data structures so far, classify each of the following ideas as either (i) not as good as an existing data structure, (ii) equal in performance to an existing data structure, or (iii) better than any existing data structure we have studied (and therefore, given Vinnie's past record, probably a hoax). If you have selected (ii), identify the comparison data structure you have in mind.

a.) Keeps a collection of arbitrary items; can insert, look up, and remove in O(log n) time, and give the entire sequence in sorted order in O(n) time.
b.) Keeps a collection of arbitrary items; can insert, look up, and remove in constant time, and give the entire sequence in sorted order in O(n log n) time.
c.) Keeps a collection of arbitrary items; can look up in constant time, insert and remove in O(log n) time, and give the entire sequence in sorted order in O(n) time.
d.) Keeps a collection of arbitrary items; can insert, look up, and remove in O(log n) time, and give the entire sequence in sorted order in O(n log n) time.
e.) Keeps a collection of items in an arbitrary sequence (not sorted); can look up or insert new items at arbitrary spots in O(log n) time.
f.) Keeps a collection of items in an arbitrary sequence (not sorted); can look up items in constant time and insert them in O(n) time.