FINAL EXAMINATION KEY
DECEMBER 2016
CSC 212 ♦ SECTION 01
INSTRUCTOR: NICHOLAS R. HOWE

Vocabulary (14 points)

In the code example below, give the line number of at least one example of each of the following terms, or say "not present" if none exists.

a. Qualifier: *4,13,17,20*
b. Field declaration: *4*
c. Assignment: *6,10,18,22,24,26*
d. Allocation: *10,22*
e. Initialization: 6,22,24,26
f. Javadoc comment: *2*
g. Inline comment: *12,16*
h. Block comment: *8*
i. Call signature of a method: *13,17,20*
j. Reference copy: *24*
k. Shallow copy: *not present*
l. Deep copy: *26*
m. Local variable: *21*
n. Method argument: *10,23,25,27*

```
import java.awt.Color;                                  1
/** A class for the exam */                            2
public class Vocab {                                    3
    private Color c;                                    4
    Vocab() {                                           5
       c = Color.BLACK;                                 6
    }                                                   7
    /* constructor */                                   8
    Vocab(Vocab v) {                                    9
       this.c = new Color(v.c.getRGB());               10
    }                                                  11
    // accessor                                        12
    public Color getColor() {                          13
       return c;                                       14
    }                                                  15
    // manipulator                                     16
    public void setColor(Color c) {                    17
       this.c = c;                                     18
    }                                                  19
    public static void main(String[] args) {           20
       Vocab v1, v2, v3;                               21
       v1 = new Vocab();                               22
       v1.setColor(Color.RED);                         23
       v2 = v1;                                        24
       v2.setColor(Color.BLUE);                        25
       v3 = new Vocab(v2);                             26
       v1.setColor(v2.getColor());                     27
    }                                                  28
}                                                      29
```

Sorting (12 points)

Assume that the letters shown below are stored in an array, show each successive step (after a swap) for the indicated sorting algorithms. For example, in bubble sort the next configuration would be CDHJAFIBEG.

D	C	H	J	A	F	I	B	E	G

a.) Insertion Sort: *CDHJAFIBEG, CDHAJFIBEG, CDAHJFIBEG, CADHJFIBEG, ACDHJFIBEG, ACDHFJIBEG, ACDFHJIBEG, ACDFHIJBEG, ACDFHIBJEG, ACDFHBIJEG, ACDFBHIJEG, ACDBFHIJEG, ACBDFHIJEG, ABCDFHIJEG, ABCDFHIEJG, ABCDFHEIJG, ABCDFEHIJG, ABCDEFHIJG, ABCDEFHIGJ, ABCDEFHGIJ, ABCDEFGHIJ*

b.) Selection Sort: *ACHJDFIBEG, ABHJDFICEG, ABCJDFIHEG, ABCDJFIHEG, ABCDEFIHJG, ABCDEFGHJI, ABCDEFGHIJ*

c.) Heap Sort: *HCDJAFIBEG, HJDCAFIBEG, JHDCAFIBEG, JHFCADIBEG, JHICADFBEG, JHIEADFBCG, JHIEGDFBCA, AHIEGDFBCJ, IHAEGDFBCJ, IHFEGDABCJ, CHFEGDABIJ, HCFEGDABIJ, HGFECDABIJ, HGFECDABIJ, BGFECDAHIJ, GBFECDAHIJ, GEFBCDAHIJ, AEFBCDGHIJ, FEABCDGHIJ, FEDBCAGHIJ, AEDBCFGHIJ, EADBCFGHIJ, ECDBAFGHIJ, ACDBEFGHIJ, DCABEFGHIJ, BCADEFGHIJ, CBADEFGHIJ, ABCDEFGHIJ, BACDEFGHIJ, ABCDEFGHIJ*

Recursion (12 points)

The method below unfortunately runs forever when called. Examine the method and answer the questions that follow.

```
private void drawDragon(int rank, Point p1, Point p2, Graphics g) {
   if (rank == 0) {
       g.drawLine(p1.x,p1.y,p2.x,p2.y);
   } else {
       int dx = (p2.x-p1.x)/4;
       int dy = (p2.y-p1.y)/4;
       Point pa = new Point(p1.x-dy+dx,p1.y+dx+dy);
       Point pb = new Point(p2.x+dy-dx,p2.y-dx-dy);
       drawDragon(rank-2,p1,pa,g);
       drawDragon(rank,pa,pb,g);
       drawDragon(rank-2,pb,p2,g);
   }
}
```

a.) Why does it run forever? What recursion guideline is being ignored?
The method does not make progress towards termination, since the rank does not decrease. Also, the stop condition should be rank <= 0 since the rank decreases in some places by more than one at a time.

b.) What simple change(s) to the code would ensure that the method always terminates?
*Change **rank** to **rank-1**; change conditional test to **rank <= 0***

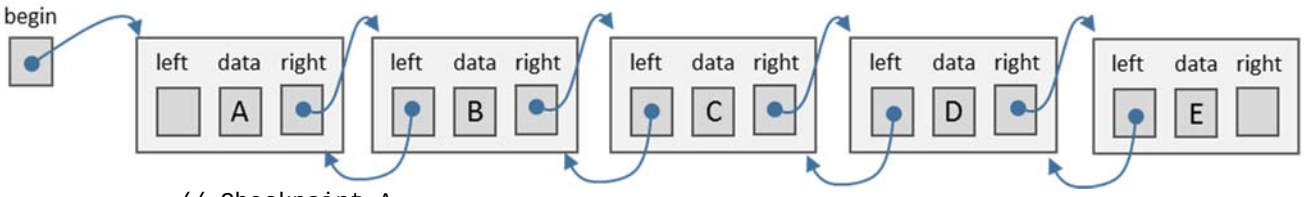
Java Memory (16 points)

Consider the short Java program shown below. Draw a diagram showing the state of memory (call stack and heap) at Checkpoint A and at Checkpoint B.

```
public class JavaMemory {
```

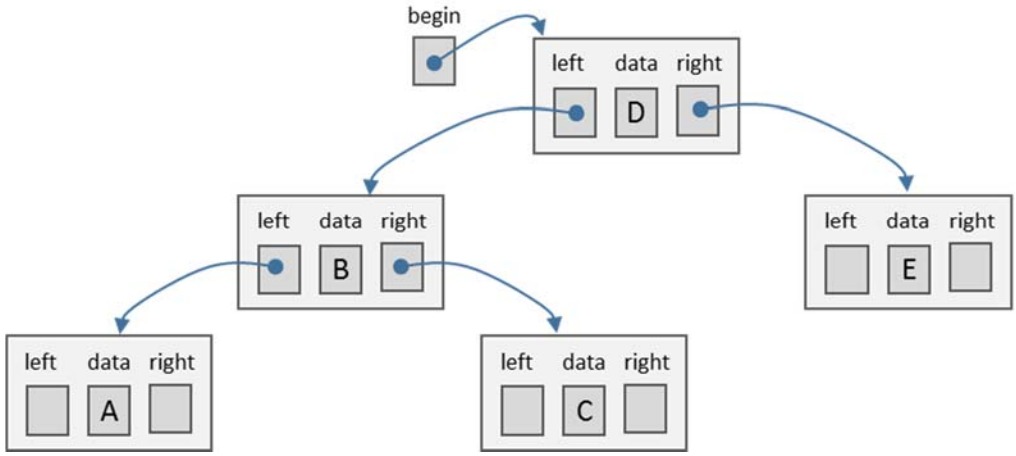
```
    static class Item {
       String data;
       Item left;
       Item right;
       Item(String data, Item left, Item right) {
           this.data = data;
           this.left = left;
           this.right = right;
       }
    }

    public static void main(String[] args) {
       Item begin = new Item("C",null,new Item("D",null,null));
       begin.right.left = begin;
       begin.right.right = new Item("E",begin.right,null);
       begin.left = new Item("B",new Item("A",null,null),begin);
       begin.left.left.right = begin.left;
       begin = begin.left.left;
```




```
       // Checkpoint A

       begin = begin.right.right.right;
       begin.left = begin.left.left;
       begin.left.right.right = null;
       begin.left.right.left = null;
       begin.left.left.right = null;
       begin.right.left = null;
       // Checkpoint B
```




```
    }
}
```

Trees (16 points)

Consider the symbol frequency table shown at right. Draw an optimal Huffman coding tree based upon these frequencies, assuming that there are no symbols that must be encoded other than the ones shown. Then answer the questions below.

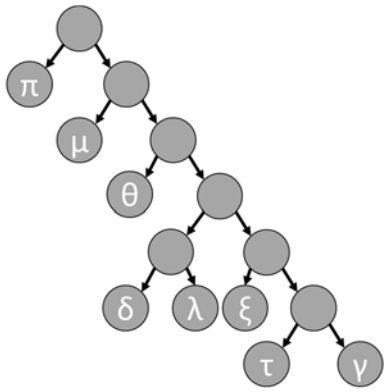
Symbol:	π	δ	λ	ξ	μ	θ	τ	γ
Frequency:	48	6	5	4	20	14	2	1

a.) [Draw the tree] *See figure at right.*

b.) Based upon this tree, what is the variable-length bitcode for μ? For τ? *10 and 111110*

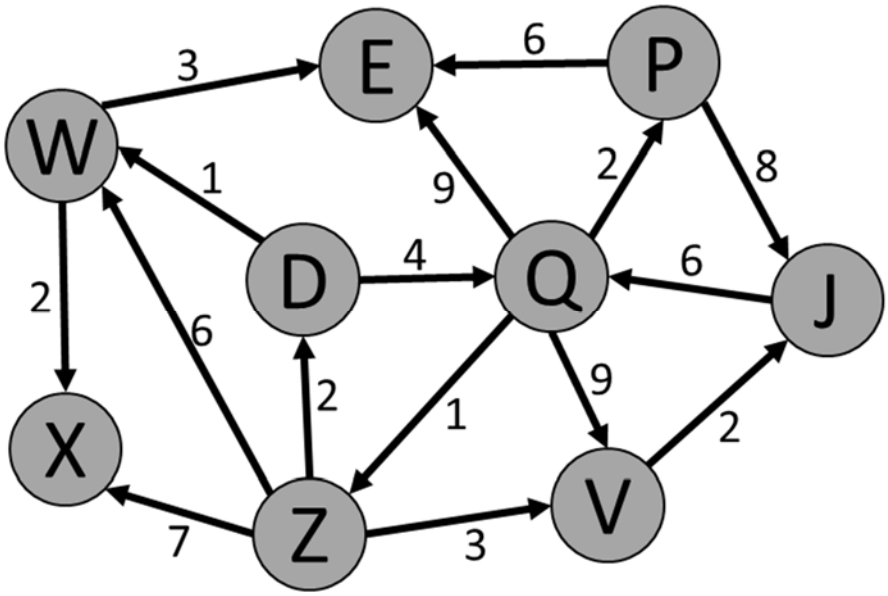
c.) How many bits would be required to represent a message containing 5 x π, 2 x δ, 3 x λ, and 4 x γ? *54 bits*

d.) If we used a fixed-length code to represent these 8 symbols instead of variable-length codes, how many bits would be required per character? $log_2 8 = 3$



Graph Traversal (12 points)

Consider the graph shown below. In what order would the nodes be visited for each of the following algorithms? In case of ties when deciding where to go next, choose the node whose label is closest to the start of the alphabet.



a.) Depth-first traversal from Z: *ZDQEPJVWX*
b.) Breadth-first traversal from P: *PEJQVZDXW*
c.) Dijkstra's shortest path algorithm starting from Q: *QZPDVWJXE*

Programming Style (10 points)

Describe the guidelines for good programming style that apply to methods that perform a boolean test and return a boolean value. Discuss as many relevant aspects as you can identify, comparing and contrasting different options for achieving the same effect using specific examples of good or bad style. You may hypothesize different scenarios as necessary within this broad description.

Ideally, the method should be given as a single boolean expression that is returned, and can usually consist of just a single line. Eschew use of a conditional (if statement) to generate the values to be returned, particularly if it involves asymmetric handling of the true and false cases or multiple return statements.

Hash Tables (8 points)

Please indicate whether the statements below are true or false. Assume that ***h(k)*** is a hash function mapping keys to table rows.

a.) Collision handling allows two different values to be stored in a hash table under the same key at the same time. *false*
b.) Collision handling allows two different keys with the same ***h(k)*** to be stored in a hash table at the same time. *true*
c.) If a hash table uses open addressing, the actual location of a key/value pair will always be greater than or equal to ***h(k)***. *false*
d.) A good choice for ***h(k)*** will distribute keys evenly across the table rows. *true*
e.) A hash table will not perform well if it is more than 10% full. *false*
f.) If ***h(k)*** is the identity function ***h(k) = k***, then the hash table is actually a lookup table. *true*
g.) If the table is overly full, lookup may require more than a constant time operation. *true*
h.) If you store just 10,000 items in a table with 1 million rows, there will most likely be no collisions. *false*