MIDTERM EXAMINATION KEY
CSC 212 ♦ SPRING 2013

You may use one 8.5"x11" sheet of notes on this exam. You may not consult other sources of information. You will have the entire lab period (110 minutes) to complete your work. All work should be written in the exam booklet. Partial credit will be granted where appropriate if intermediate steps are shown.

Data Access (12 points). Consider the short program fragment below. Without actually compiling the program, indicate for each line or set of lines A-H below either the value that would be output, or if a compilation error would prevent the line (or a method it calls) from being included as written.

```
public class Data {
    public int x = 2;
    private int y = 3;
    public static int z = 5;
    public static int getX() {
       int x = 7;
       return this.x;
    }
    public int getY() {
       return y;
    }
    public int getZ() {
       return this.z;
    }
}
```

```
public class DataAccess {
    public static void main(String args[]) {
       Data d = new Data();
       System.out.println(d.x);          // A
       System.out.println(d.y);          // B
       System.out.println(d.z);          // C
       System.out.println(d.getX());     // D
       System.out.println(d.getY());     // E
       System.out.println(d.getZ());     // F
       System.out.println(Data.x);       // G
       System.out.println(Data.y);       // H
       System.out.println(Data.z);       // I
       System.out.println(Data.getX());  // J
       System.out.println(Data.getY());  // K
       System.out.println(Data.getZ());  // L
    }
}
```

A: 2; B: error; C: 5; D: error; E: 3; F: 5; G: error; H: error; I: 5; J: error; K: error; L: error;

Linked Lists (16 points). For each of the tasks below, state whether it can be most simply accomplished using (i) iterators, (ii) end operations like getFirst(), removeFirst(), etc., or (iii) a for-each loop.

a.) Traverse two lists in tandem, producing output that combines elements from each.
Iterators

b.) Push all the elements of a list in order onto a stack, leaving the original list unchanged.
For-each

c.) Take all the elements out of one list and add them to either of two other lists, depending on whether they are bigger or smaller than a pivot value.
End operations

d.) Compute the product of all the elements in a list.
For-each

e.) Remove from a list all the elements that are divisible by 13.
Iterators

f.) Merge two sorted lists into a single list, also sorted.
End operations

g.) Move the first n elements of one list into another.
End operations

h.) Count the number of elements in a list that are divisible by 13.
For-each

Class Design (16 points). We mentioned in class that MVC (Model-View-Controller) is a common software pattern. Explain briefly each of the MVC roles and their interactions, and explain why this paradigm fulfills some of the principles of good class design. Give reference to a specific example of each role from the homework assignments.

The Model organizes the data to be displayed. The View organizes aspects of the display and how the raw data will be rendered. The Controller initiates action and coordinates between the Model and the View. Each role is organized around a strong concept, and the interfaces between them can be relatively simple. The map assignments use this paradigm. MapGrid is the Model, MapViewer is the View, and MapGUI is the Controller.

Stacks (16 points). Many programs allow the user to undo their last action. Some programs allow multiple undo operations, where the entire history of user actions can be undone one operation at a time. A user may undo some actions, do something else, and then undo that, etc. Are stacks a suitable data structure for this sort of functionality? Why or why not? In your answer, make specific reference to the data access pattern of stacks, and how each of the key stack operations might be used in such a program. (Underline the access pattern and operation names in your answer.)

Stacks are well suited for this functionality, because operations are undone in ***LIFO*** *fashion (last in, first out) just like the elements of a stack. Each time the user performs an operation, a record of that operation (or the state just prior to it) can be* ***push****ed on the stack. When the user wishes to undo, the most recent operation is* ***pop****ped of the stack and the information is used to roll back the state.*

Now consider that the same program also offers a redo option, whereby the user may restore any changes that were undone – at least until they make any new edits. Describe the role stacks might play in implementing such additional functionality.

We can also use a stack for redo functionality. Each time the user undoes an action, that action is pushed on the redo stack. Selecting a redo operation pops the last action from the redo stack and causes it to be redone. The redo stack only lasts until the user begins making new edits. At this time, it is thrown away and redo actions are no longer possible.

Sorting (8 points). First, write detailed pseudocode for insertion sort. In class, I mentioned that it is interesting that we often tend to use insertion sort when sorting objects by hand, since our analysis of the insertion sort algorithm suggested that it is a slower $\mathcal{O}(\boldsymbol{n}^2)$ algorithm while other available methods are $\mathcal{O}(\boldsymbol{n} \log \boldsymbol{n})$. However, one can make an argument that insertion sort performed physically with the eyes and hands is actually $\mathcal{O}(\boldsymbol{n} \log \boldsymbol{n})$. Provide such an argument, and explain why the physical situation differs from the computational one. Your answer should demonstrate complete comprehension of the insertion sort algorithm.

Select an item from the unsorted.
Scan the sorted items from biggest to smallest. When you find one smaller than the selected item, insert after it. If there are no smaller items, insert it at the beginning.

In physical sorting, visual scanning is typically very fast. More importantly, we don't need to do a search of every element: in physical space we can easily jump to the middle of a sequence of sorted elements, and thus can perform binary search in $\mathcal{O}(\log \boldsymbol{n})$ time rather than linear search in $\mathcal{O}(\boldsymbol{n})$ time.

However, once we have found the correct spot for an item we can make room for it, which is not possible when using an array in memory.

Complexity Analysis (8 points). Compute a tight simplified bound on the asymptotic complexity of the short program below. (In other words, give the big-$\mathcal{O}$ bound, in terms of $\boldsymbol{n}$.)

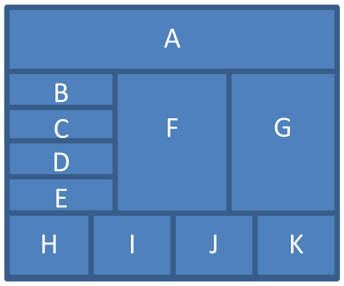
The outer loop executes n times. The inner loop executes once the first time, twice the second, etc., or n/2 times on average. Therefore the innermost line is executed n(n/2) times, and the algorithm is $\mathcal{O}(\boldsymbol{n}^2)$.

```
public class Pascal {
    public static void main(String[] args) {
       int n = Integer.valueOf(args[0]);
       int[] p = new int[n+1];

       // compute nth row of Pascal's Triangle
       p[0] = 1;
       for (int i = 0; i < n; i++) {
           p[i+1] = 1;
           for (int j = i; j > 0; j--) {
              p[j] += p[j-1];
           }
       }

       // output result
       for (int i = 0; i <= n; i++) {
           System.out.print(p[i]+" ");
       }
       System.out.println("");
    }
}
```

GUI Design (12 points). How might the interface below be created using a minimal number of panels? For each Pane or Panel in your design, specify the layout to be used (with dimensions if applicable), along with the components (lettered A through K in the diagram) to be assigned to it and their positions in the layout. Assign letter designations L, M, etc. to any panels you will create. (You don't need to write code; just describe what the code should accomplish.)



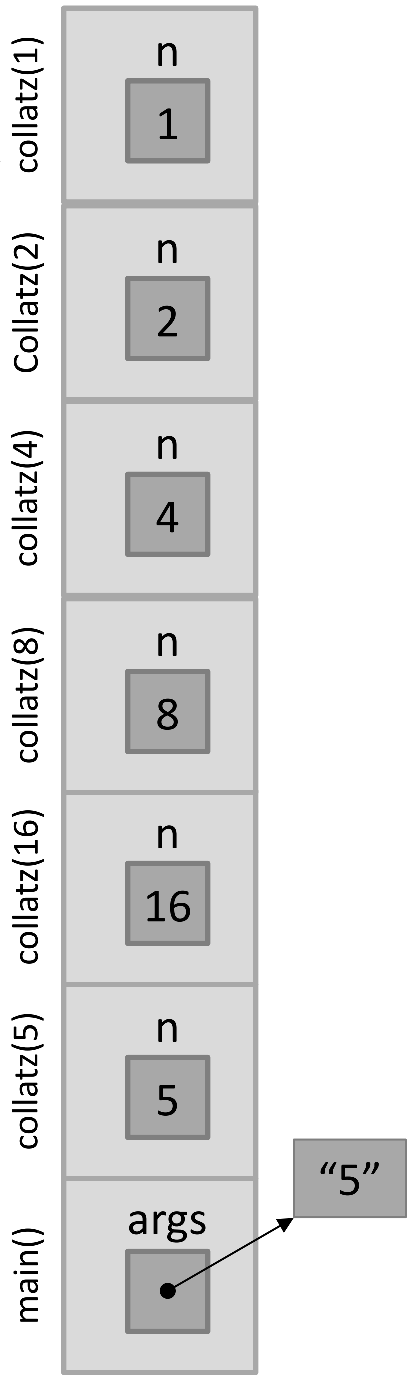
Pane: BorderLayout
North: A
East: Panel L
West: G
Center: F
South: Panel M

Panel L: GridLayout(1,4)
0: B
1: C
2: D
3: E

Panel M: GridLayout(4,1) or FlowLayout
0: H
1: I
2: J
3: K

Recursion (12 points). Draw the call stack (state of memory) for this program at the point where the CHECKPOINT is reached. Include the values of all local variables and parameters stored in the stack frame. Assume that the program was run using the following Unix command: **`java Collatz 5`**

```
public class Collatz {
    public static int collatz(int n) {
       if (n == 1) {
           return 0;  // CHECKPOINT
       } else if (n%2 == 0) {
           return collatz(n/2)+1;
       } else {
           return collatz(3*n+1)+1;
       }
    }
    public static void main(String[] args) {
       System.out.println(collatz(Integer.valueOf(args[0])));
    }
}
```


collatz(1)
n
1
Collatz(2)
n
2
collatz(4)
n
4
collatz(8)
n
8
collatz(16)
n
16
collatz(5)
n
5
main()
args
“5”