1. Number systems.
   
   (a) $54_{10}$
   
   (b) $1101111_2$
   
   (c) $AB_{16}$
   
   (d) $a = 0$, $b = -2^{31}$. Note that $2^{31}$ is not representable in 32-bit two’s complement notation; it gets wrapped around to $-2^{31}$. More generally, any pair of integers $a$ and $b$ that satisfy (i) $a \geq 0$, (ii) $b < 0$, and (iii) $a - b \geq 2^{31}$ will also overflow a 32-bit int and lead to the same result.

2. Debugging.

   (a) Line 6: array should be of size 100 to accommodate entries between 0 and 99; otherwise you will get an array out-of-bounds exception if the user enters 99.

   Lines 8–11: need curly braces around body of while loop or the $a[i]++$ statement only gets executed once (after the loop is finished).

   Line 12: remove semicolon at the end of the line. Otherwise, the if statement only gets executed once (after the loop is finished).

   (b) It will print out the smallest such one.

3. Loops and conditionals.

   0 1 2 3 4 5
   5 0 1 2 3 4
   4 5 0 1 2 3
   3 4 5 0 1 2
   2 3 4 5 0 1
   1 2 3 4 5 0

   Remark: a recent immunity challenge on Survivor asked the contestants to arrange copies of 4 elements in a 4-by-4 grid so that no row or column contained two or more copies of the same element. This program produces an N-by-N solution.
4. Java basics.

```java
public class SignalAnalyzer {
    public static void main(String[] args) {
        double sum = 0.0; // sum of absolute values
        int N = 0; // number of inputs
        while (!StdIn.isEmpty()) {
            double x = StdIn.readDouble();
            sum += Math.abs(x);
            N++;
        }
        System.out.println(sum / N);
    }
}
```

5. Recursive graphics.

(a) ii  
(b) v  
(c) iii 
(d) i  
(e) iv  
(f) vi  

6. TOY.

(a) 00: 60  
    01: BE  

(b) Sorts the two integers in ascending order. Note: it may fail if the integers are allowed to be negative (e.g., see question 1d).

(c) 00: 000D  
    01: 0060  
    02: 00BE  

(d) Sorts the three integers in ascending order.
7. Functions.

```java
class FunctionSolution {
    public static boolean majority(boolean a, boolean b, boolean c) {
        return (a && b) || (a && c) || (b && c);
    }
}
```

8. Arrays.

(a) 2 0 1 4 5 3

(b) int[] binv = new int[N];
    for (int i = 0; i < N; i++)
        binv[b[i]] = i;

(c) (ainv[i] < ainv[j])

(d) int tau = 0;
    for (int i = 0; i < N; i++) {
        for (int j = i + 1; j < N; j++) {
            boolean a = (ainv[i] < ainv[j]); // does i appear before j in a?
            boolean b = (binv[i] < binv[j]); // does i appear before j in b?
            if (a != b) tau++;
        }
    }

9. Input, output.

The body of the loop counts the number of consecutive occurrences of each integer, and prints out that number followed by the digit. This is a crude form of data compression known as run-length encoding (RLE); it is effective when the input contains lots of runs of the same digit.

(a) 3 1 3 2 5 3 3 6 6 1
(b) 1 3 1 1 1 3 1 2 1 5 2 3 2 6 1 1
(c) 1 1 1 3 3 1 1 3 1 1 2 1 1 1 5 1 2 1 3 1 2 1 6 2 1

Remark: if you start the sequence with the value 1, and repeatedly pipe the results through java Conway, you obtain Conway’s look-and-say sequence: 1, 11, 21, 1211, 111221, 312211, 13112221, 111312211, which has some rather amazing properties.