CS61B Midterm 1 Review

1. Algorithmic Analysis: Give Θ bound if you can, otherwise Ω and O bounds.

   (a)
```
static int bar(int[] A) {
    int N = A.length;
    int S;
    S = 0;
    for (int i = 0; i < N; i += 1) {
        for (int j = i+1; j < N; j += 1) {
            if (A[j] == A[i]) {
                S += 1;
                break;
            }
        }
    }
    return S;
}
```
   $\Omega(N), O(N^2)$

   (b)
```
static boolean sump(int[] A, int S) {
    return sump1(A, S, 0);
}

private static boolean sump1(int[] A, int S, int k) {
    int N = A.length;
    if (S == 0)
        return true;
    else if (k >= N)
        return false;
    else if (S >= A[k] && sump1(A, S-A[k], k+1))
        return true;
    else return sump1(A, S, k+1);
}
```
   $\Omega(1), O(2^N)$

2. Bitwise arithmetic
```
/** Returns x, rotated left by k bits.
 *  For example, rotate(-92, 3) = 35
 *  Note -92 in binary is 10100100,
 *  and 37 is 00100101.
 *  Use only one statement. */
byte rotate(byte x, int k) {
    return (x << k) | (x >>> (8 - k));
}
```

3. Extra hard problem for those who have finished the rest of the sheet:
   Let p(n) represent the number of different ways in which n coins can be separated into piles. For example, five coins can separated into piles in exactly seven different ways, so p(5)=7.
   (separations: OOOOO, OOOO O, OOO OO, OOO O O, OO OO O, OO O O O, O O O O O)
   Write a program to find the least value fo n for which p(n) is divisible by one million.

4. Typing and Inheritance
The following program compiles correctly. What does the main program (in D) print?

```
class A {
    int z = 2;
    void f() { this.g(); }
    void g() { System.out.printf("A:%d%n", z); }
    int h() { return z; }
}

class B extends A {
    int z = 15;
    void g() { System.out.printf("h:%d z:%d%n", h(), z); }
}

class C extends A {
    int z = 42;
    void f() { this.g(); }
}

class D {
    public static void main (String[] args) {
        A c1 = new C();
        C c2 = new C();
        A b1 = new B();
        B b2 = new B();
        System.out.println("Before modification");
        c1.f(); c2.f(); b1.f(); b2.f();
        c1.z = 23;
        c2.z = 25;
        b1.z = 47;
        b2.z = 49;
        System.out.println("After modification");
        c1.f(); c2.f(); b1.f(); b2.f();
    }
}
```

```
Before modification
A:2
A:2
h:2 z:15
h:2 z:15
After modification
A:23
A:2
h:47 z:15
h:2 z:49
```

5. IntLists

```
/** Set each R[k] to a sublist of L such that R[k] contains
 *  <=k+1 elements and the concatenation of all the R[k] in order
 *  gives a prefix of the original list L. Each list R[k] is made
 *  as large as possible subject to these rules, with earlier lists
 *  taking precedence. For example, if the original L contains
 *  [ 1, 2, 3, 4, 5, 6, 7 ], and R has 6 elements, then on return R
 *  contains [ [1], [2,3], [4,5,6], [7], [], []]. If R had only 2
 *  elements, then on return it would contain [[1], [2,3]].
 *  May destroy the original contents of the IntList objects in L,
 *  but does not create any new IntList objects. */
static void triangularize (IntList[] R, IntList L) {
    // One of many possible solutions.
    int i, k; /* i: index into R; k: number of items in R[i] */
    i = 0; k = 0;
    while (i < R.length) {
        if (k == 0)
            R[i] = L;
        if (L == null) {
            i += 1; k = 0;
        } else if (k == i) {
            IntList next = L.tail;
            L.tail = null; L = next;
            i += 1; k = 0;
        } else {
            L = L.tail; k += 1;
        }
    }
}
```