COS 126	General Computer Science	Spring 2006

Exam 1 Solutions

1. **Number systems.**

   (a) 57

   $32 + 16 + 8 + 1$

   (b) -7

   We identify $x = 111001_2$ as a negative 6-bit two's complement integer since it's leading bit is 1. To compute $-x$, we flip the bits and add 1: $-x = 000111$. Thus, $-x = 7$ so $x = -7$.

   (c) $14D_{16}$

   (d) $2^{31} - 1$

2. **Strings, booleans, and conditionals.**

   (a) `ababb`

   (b) `ababbab` (12 or 18), `abb` (10, 11, 13, 14, 16, 17, 19, or 20), `bbb` (7, 8, 22, or 23), `bbbbbbb` (6, 9, 21, 24).

   Note that the program will never generate two consecutive `a`'s since the statement `s = s + s` can never be executed. For completeness, the program can also generate `bab` (1), `bababba` (3), `ababb` (15), and `bbb` (22).

3. **Nested loops and conditionals.**

   (a)
   ```
   % java Pattern 2
   A B B B A
   A A B A A
   A A A A A
   A A B A A
   A B B B A
   ```

   (b) 72,000,000 bytes.

   The same technique for analyzing and predicting the running time of an algorithm applies to analyzing and predicting the size of its output. The number of characters is proportional to N^2. If the input size triples, then the output will increase by a factor of $3^2 = 9$.

4. **Debugging and arrays.**

   (a) Rearrange the elements of `a[]` in descending order.

   (b) Line 13: `double temp = a[i];`

   (c) Line 7: `for (int j = i + 1; j < a.length; j++) {`

   (d) The function leaves the array `a[]` unchanged since `max` will always equal `i` so it will always swap the element `a[i]` with itself.

   (e) Line 9: `max = j;`

5. **Java basics and standard input.**

```
public class Streak {
    public static void main (String[] args) {
        int longest = 0;     // length of longest streak so far
        int inarow  = 0;     // length of current streak
        int prev    = 0;     // previous value

        // read in positive integers, keeping track of longest streak
        while (!StdIn.isEmpty()) {
           int x = StdIn.readInt();
           if (x == prev) inarow++;
           else           inarow = 1;
           if (longest < inarow) longest = inarow;
           prev = x;
        }

        System.out.println(longest);
    }
}
```

   By initializing `prev` to `0` (a value different from any input value), we avoid having a special case to handle inputs of size 0 or 1.

   One common mistake was to reset `inarow` to `0` instead of `1`.

   Another common mistake was to only update `longest` when the new value `x` is different from the old value `prev`. This fails when the streak occurs at the very end of the input.

   Using arrays is unnecessary and complicated, since you don't know the size of the input.

6. **Functions and binary numbers.**

   Here are two possible solutions. The latter uses a logical shift (`>>>`) instead of an arithmetic shift (`>>`) to properly deal with negative integers.

```
public static int bitCount(int x) {
   int count = 0;
   while (x > 0) {
      if (x % 2 == 1) count++;
      x = x / 2;
   }
   return count;
}

public static int bitCount(int x) {
   if (x == 0) return 0;
   return (x & 1) + bitCount(x >>> 1);
}
```

   In practice, use `Integer.bitCount(x)`.

7. **Functions and recursion.**

   (a)
```
***
**
*
*
**
***
```

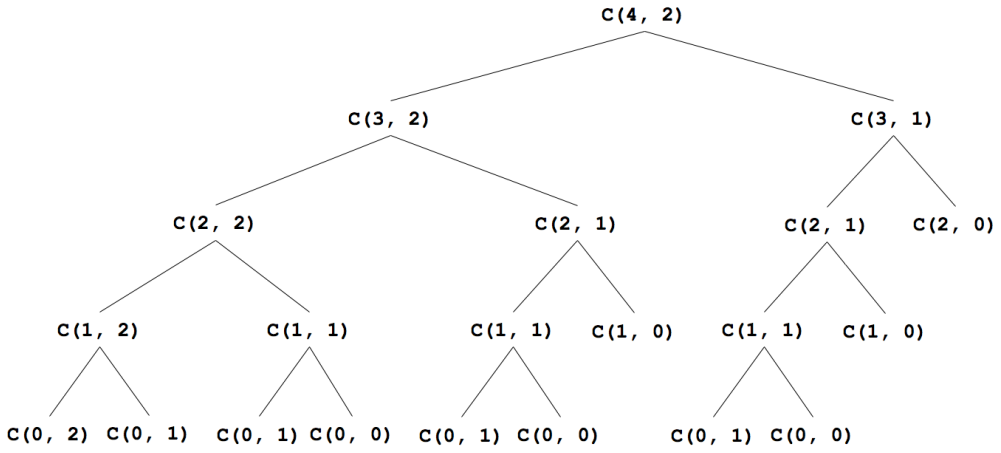
   (b)
```
****
***
**
*
*
**
***
****
```

   (c) v

   It won't go into an infinite loop because eventually `N` will wrap-around from -2^{31} to $2^{31} - 1$. Long before this happens, you will get a `StackOverflowException`.

8. **Recursion, analysis of algorithms, and number representation.**

(a)



(b) $C(4, 2) = 6$.

(c) The function is spectacularly inefficient (like the Fibonacci function) because it recomputes values from scratch each time (as you noticed when drawing the recursion tree).

The function computes the wrong answer because the result overflows a Java `int`. The function would correctly compute `C(36, 18)` if we used `long` instead of `int`.

(d) 30 minutes.

To compute `C(37, 18)`, you need to compute `C(36, 18)` and `C(36, 17)`. `C(36, 18)` takes approximately 15 minutes according to (c); `C(36, 17)` takes roughly the same amount of time (but just slightly less).

(We also accepted 15 minutes without deduction.)

9. **TOY.**

```
11:  7251    R[2] <- 51
12:  8350    R[3] <- mem[50]
15:  2423    R[4] <- R[2] - R[3]
17:  A502    R[5] <- mem[R[2]]
1D:  F015    R[0] <- pc; pc <- 15
```