Thanks to past students for parts of this solution.

1. (a) \( \text{SUM} = 2^1 + 2^2 + \ldots + 2^{10} = 2046 \). By special argument: This is binary 111 1111 1110. If we add \( 2 \) to it, it rolls over to 1000 0000 0000 = \( 2^{11} = 2K = 2048 \), so it must be 2046.

By geometric series: \( \text{SUM} = 2^1 + 2^2 + \ldots + 2^{10} = 2 \times (2^0 + \cdots + 2^9) \)

\[ \text{SUM} = 2 \times (\frac{1 - 2^{10}}{1 - 2}) = 2046 \]

(b) \( \text{SUM} = \frac{2}{3} + \frac{4}{9} + \frac{8}{27} + \frac{16}{81} + \ldots \)

Need to subtract the “to infinity ...” part out.

\[ \frac{3}{2} \times \text{SUM} = 1 + \frac{2}{3} + \frac{8}{27} + \frac{16}{81} + \ldots \text{ and therefore:} \]

\[ \frac{3}{2} \times \text{SUM} - \text{SUM} = 1 + \left( \frac{2}{3} + \frac{8}{27} + \ldots - \frac{2}{3} - \frac{8}{27} - \ldots \right). \]

The part in parentheses cancels out, we’re left with: \( 1/2 \text{SUM} = 1 \). Thus, \( \text{SUM} = 2 \)

2. A number \( N \) has \( \text{floor}(\log_2(N)) + 1 \) binary digits, where \( \text{floor}(x) \) denotes the largest integer not greater than \( x \). So:

\[ \text{floor}(\log_2(2^{100})) + 1 = \text{floor}(100) + 1 = 101, \]

\[ \text{floor}(\log_2(5^{100})) + 1 = \text{floor}(100\log_2(5)) + 1 = 232 + 1 = 233 \]

\[ \text{floor}(\log_2(10^{100})) + 1 = \text{floor}(100\log_2(10)) + 1 = 332 + 1 = 333. \]

How are these answers related? \( 2^{100} \times 5^{100} = 10^{100} \) and \( 101 + 233 = 334 = \text{approx.} 333 \)

3. We have \( \log_B(N) = \log_2(N)/\log_2(B) \) by the change-of-base formula, for any base \( B \).

So \( \log_6(N) = \log_2(N)/\log_2(6) \) – relate base \( b \) to base 2

and \( \log_a(N) = \log_2(N)/\log_2(a) \) – relate base \( a \) to base 2

and thus \( \log_b(N)/\log_a(N) = \log_2(a)/\log_2(b) = \text{const.} \)

We have \( \log_{10}(N) = \log_2(N)/\log_2(10) \), and \( 1/\log_2(10) = 0.3010 \), so \( \log_{10}(N) = 0.3010 \times \log_2(N) \)

Number of decimal digits = \( \text{ceiling}(\log_{10}(N)) \) (within 1 of \( \log_{10}(N) \))

Number of binary digits = \( \text{ceiling}(\log_2(N)) \) (within 1 of \( \log_2(N) \))

So numbers in base 10 are about 3/10 length of the same numbers in base 2.

4. (a) Functions ranked in order of increasing growth rate. Most of these are easily decided by looking at the limit of the ratio as \( N \) grows. Numerical evidence isn’t really needed.

- \( 2/N = O(1/N) \) – does not grow at all, it shrinks as \( N \to \infty \)
- \( 38 = O(1) \) – constant
- \( \sqrt{N} = N^{0.5} \)
- \( N \)
- \( N \log \log N \). \( \log \log N \) grows very slowly. It’s just \( 10 \) when \( N = 2^{1000} \).
- \( N \log N \). This and the next are tied
- \( N \log(N^2) = 2N \log N \)
- \( N^{1.5} \)
• $N^2$
• $N^2 \log N$
• $N^2 (\log N)^2$
• $N^3$
• $2^{N/2}$
• $2^N$ this is not a tie with $2^{(N/2)}$

(b) $\log N$ and $\log(N^2)$ are tied because $\log(N^2) = 2 \log N$. $\log^2 N = \log N \cdot \log N$ grows faster than $\log N$. You can divide the two and get $\log N >> const$.

5. (a) The outer loop is executed $n$ times, and the inner loop, in the worst case, is also executed $n$ times giving time complexity of $O(n^2)$.

(b) In terms of big-O both functions are $O(N^2)$. When absolute time (in seconds) is considered the second version could be a bit faster since we are doing fewer multiplications. It is possible however that due to compiler optimizations both versions will be equally fast.

(c) ```
int mysterySum(int n)
{
    int i, j, s=0;
    for(i=0; i<n; i++)
        s += i*i*i;
}
``` And the time complexity is $O(n)$.

(d) The term is $\sum(i^3) = (n^2(n+1)/2)^2$. This formula is available in the cs220 textbook. Proof by induction: True for 1 (result is 1) and 2 (result is 9 = $(2 \cdot 3/2)^2$). Suppose it’s true for $n-1$, that is $\sum(i-1)^3 = (n-1)^2(n+1)/2$. Let’s show that by adding $i^3$ we’ll get $(n^2(n+1)/2)^2$. $(n^2(n+1)/2)^2$ opens to $(n^4 - n^3 + n^2)/4$. $(n^2(n+1)/2)^2$ opens to $(n^4 + 2n^3 + n^2)/4$. Subtracting the former from the latter gives us $4 \cdot n^3/4 = n^3$. Exactly what we said we had to add. QED.

6. Calculating power2

```
int power2(int n)
{
    if (n==0) return 1;
    return power2(n-1)+power2(n-1);
}
``` We have a double recursion here, so obviously the runtime is bad. The formula is $T(n) = C + 2 \cdot T(n-1)$. So each run doubles the number of recursive calls while doing constant time operations outside the recursions. This is exponential, so $T(N) = O(2^N)$. It can be shown easily by expanding the next expression $T(n - 1) = C + 2T(n-2)$, etc. The factors of 2 accumulate so that at the $k^{th}$ stage we have $2^k \cdot C$. Overall we have $n$ stages.

A linear time algorithm would be the following:

```
int power2(int n)
{
    if (n==0) return 1;
    return 2*power2(n-1);
}
``` We simply run the recursion once and multiply by 2... You can easily show that the recurrence formula is linear, since it’s the same as the factorial function we showed in class: $T(n) = C + T(n-1)$. Another way: a simple for loop (maybe not such a small change, though):
```java
int power2(int n)
{
    int power = 1;
    for (int i=0; i < n; i++) {
        power *=2;
    }
}
```

7. Code for combinational lock. Note there are no getters or setters for a, b, c, since the combination is “secret”. To change the combination, you need to supply the old combination.

```java
/*
 * CombinationLock.java
 * @author -- Ruchi Dubey
 * ruchi@cs.umb.edu
 */

class CombinationLock
{
    private int a,b,c;

    CombinationLock(int a1,int a2, int a3)
    {
        a = a1;
        b = a2;
        c = a3;
    }

    public boolean open(int x,int y, int z)
    {
        return x == a && y == b && z == c;
    }

    /* The old combination is xyz and the new to be changed is pqr */
    public boolean changeCombo(int x, int y, int z, int p, int q, int r)
    {
        if(this.open(x, y, z))
        {
            a = p;
            b = q;
            c = r;
            return true;
        }
        else
            return false;
    }
}
```

8. (a) equals, hashCode, toString.
   (b) if (s.equals(t))

   This equals call compares 'a' vs. 'a', and since they are equal, goes on to compare 'b' vs. 'b', and since equal, goes on to compare 'c' vs 'x' and returns false. For (s == t), the string references are compared (memory addresses), which must be different, so again evaluates to false (in this case).

   (c) hashCode value for "abc" is the int 96354
   toString value for "abc" is "abc"
9. An interface is a Java file that provides an API, a set of method descriptions (technically, method headers) of public methods, but no implementation whatsoever of that API. It also can specify constants and nested interfaces. More technically, it provides "abstract methods", but this just means method declarations without implementation.

10. methods, constants, (and nested interfaces, but not expected as an answer here)

11. ```java
   // It would be good to put comments above each of these methods--
   public interface UnionFind {
     void union(int p, int q);
     int find(int p);
     boolean connected(int p, int q);
     int count();
   }

   // new top line for UF.java:
   public class UF implements UnionFind {
```