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: \( \text{SUM} = 2^1 + 2^2 + \ldots + 2^{10} = 2^1 \times \frac{1 - r^{10}}{1 - r} \) sum formula where here \( r = 2 \), \( a=2 \), \( n=9 \) so \( \text{SUM} = 2^1 \times \frac{1 - 2^{11}}{1 - 2} = 2^1 \times (1 - 1) = 2046 \).

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

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

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

Therefore:

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

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 Weiss, pg. 165, for any base \( B \).

So \( \log_B(N) = \log_2(N) / \log_2(b) \) – 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. The stable matching algorithm: Given the following rankings of students and hospitals:

<table>
<thead>
<tr>
<th>Atlanta</th>
<th>Wyatt</th>
<th>Xena</th>
<th>Yousef</th>
<th>Zelda</th>
<th>Vanessa</th>
</tr>
</thead>
<tbody>
<tr>
<td>Boston</td>
<td>Xena</td>
<td>Yousef</td>
<td>Zelda</td>
<td>Vanessa</td>
<td>Wyatt</td>
</tr>
<tr>
<td>Chicago</td>
<td>Zelda</td>
<td>Vanessa</td>
<td>Yousef</td>
<td>Wyatt</td>
<td>Xena</td>
</tr>
<tr>
<td>Denver</td>
<td>Xena</td>
<td>Vanessa</td>
<td>Wyatt</td>
<td>Yousef</td>
<td>Zelda</td>
</tr>
<tr>
<td>El Paso</td>
<td>Vanessa</td>
<td>Zelda</td>
<td>Wyatt</td>
<td>Xena</td>
<td>Yousef</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Vanessa</th>
<th>Atlanta</th>
<th>Boston</th>
<th>Chicago</th>
<th>El Paso</th>
<th>Denver</th>
</tr>
</thead>
<tbody>
<tr>
<td>Wyatt</td>
<td>Boston</td>
<td>Atlanta</td>
<td>Denver</td>
<td>El Paso</td>
<td>Chicago</td>
</tr>
<tr>
<td>Xena</td>
<td>Chicago</td>
<td>Boston</td>
<td>El Paso</td>
<td>Atlanta</td>
<td>Denver</td>
</tr>
<tr>
<td>Yousef</td>
<td>Denver</td>
<td>El Paso</td>
<td>Atlanta</td>
<td>Boston</td>
<td>Chicago</td>
</tr>
<tr>
<td>Zelda</td>
<td>El Paso</td>
<td>Atlanta</td>
<td>Boston</td>
<td>Chicago</td>
<td>Denver</td>
</tr>
</tbody>
</table>
(a) No need to detail the stages, but here they are anyway:

- Atlanta offers Wyatt. He accepts.
- Boston offers Xena. She accepts.
- Chicago offers Zelda, who accepts.
- Denver offers Xena, who rejects (Boston is better)
- Denver offers Vanessa, who accepts.
- El Paso offers Vanessa, who accepts (El Paso is better than Denver).
- Denver offers Wyatt, who rejects.
- Denver offers Yousef, who accepts.

The final match is A-W, B-X, C-Z, D-Y, E-V.

(b) The role reversal is very easy, since each student has a different first choice, so they all accept right away. The final match is: A-V, B-W, C-X, D-Y, E-Z.

(c) After Vanessa falsifies her preference list the table is:

<table>
<thead>
<tr>
<th></th>
<th>Vanessa</th>
<th>Chicago</th>
<th>Denver</th>
<th>Atlanta</th>
<th>Boston</th>
<th>El Paso</th>
</tr>
</thead>
<tbody>
<tr>
<td>Wyatt</td>
<td>Boston</td>
<td>Atlanta</td>
<td>Denver</td>
<td>El Paso</td>
<td>Chicago</td>
<td></td>
</tr>
<tr>
<td>Xena</td>
<td>Chicago</td>
<td>Boston</td>
<td>El Paso</td>
<td>Atlanta</td>
<td>Denver</td>
<td></td>
</tr>
<tr>
<td>Yousef</td>
<td>Denver</td>
<td>El Paso</td>
<td>Atlanta</td>
<td>Boston</td>
<td>Chicago</td>
<td></td>
</tr>
<tr>
<td>Zelda</td>
<td>El Paso</td>
<td>Atlanta</td>
<td>Boston</td>
<td>Chicago</td>
<td>Denver</td>
<td></td>
</tr>
</tbody>
</table>

Here is the new match: A-W, B-X, C-V, D-Y, E-Z

As seen, Vanessa improved her position (as did Zelda, by chance), but not to her optimal valid partner, which is Atlanta.

5. Here is the code (obviously in real life you should reduce etc.)

```java
public class Rational {

    private int num; // the numerator
    private int den; // the denominator

    // create and initialize a new Rational object
    public Rational(int numerator, int denominator) {
        if (denominator == 0) {
            throw new ArithmeticException("denominator is zero");
        }

        // In principle we should reduce using GCD but I said not to bother.
        num = n;
        den = d;

        // needed only for negative numbers. Optional
        if (den < 0) { den = -den; num = -num; }
    }

    // return the numerator and denominator of (this)
    public int getNumerator() { return num; }
    public int getDenominator() { return den; }

    public int getNumerator() { return num; }
    public int getDenominator() { return den; }

    // return string representation of (this). Printing function
    public String toString() {
        if (den == 1) return num + "";
        else return num + "/" + den;
    }
}
```
public Rational multiply(Rational b) {
    return new Rational(num * b.num, den * b.den);
}

public Rational add(Rational b) {
    // add cross-product terms for numerator. We should have reduced in real life
    Rational s = new Rational(num * b.den + b.num * den,
                                den * b.den);
    return s;
}

// return -a
public Rational negate() {
    return new Rational(-num, den);
}

public Rational subtract(Rational b) {
    Rational a = this;
    return a.add(b.negate());
}

public Rational reciprocal() { return new Rational(den, num); }

public Rational divide(Rational b) {
    Rational a = this;
    return a.multiply(b.reciprocal());
}

6. Here is a suggested class:

public class WordUsage {
    private final String word;
    private int count;

    public WordUsage(String x, int count) {
        word = x;
        this.count = count;
    }

    public WordUsage(String x) {
        word = x;
        this.count = 1;
    }

    public void setCount(int x) {
        count = x;
    }

    public String getWord() {
        return word;
    }

    public int getCount() {
        return count;
    }

    public void increment() {
        count = count + 1;
    }
}

7. (a) A final class is a class that cannot be extended or subclass-ed. Final classes are used for several
reasons: a. Speedup (can be more efficient code). b. To prevent accidental overriding of classes that the programmer wishes to leave as is.

(b) An interface provides an API, a set of method descriptions, but no implementation whatsoever of that API. It also can specify constants and interfaces (see Map.java, pp 237-238 for an example of interface inside an interface). An abstract class may provide a default implementation by specifying some non-abstract methods, and can have fields. The interface is not allowed to provide any implementation details either in the form of data fields or implemented methods. Only public final fields and abstract public members may be in an interface, plus nested interfaces such as Entry in Map. The big advantage of interface over abstract class is that another class X can implement several interfaces but can extend only one class, abstract or not. Thus we use abstract classes only as base classes for important concrete classes we want.