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 \cdot (2^0 + \ldots + 2^9) \sum = a \cdot (1 + r + r^2 + \ldots + r^n) = a \cdot (1 - r^{n+1}) / (1 - r) \\text{sum formula where here} \ r = 2, \ a=2, \ n=9 \ \text{so} \ \text{SUM} = 2 \cdot (1 - 2^{10}) / (1 - 2) = 2 \cdot (1K - 1) = 2046

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

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

3/2 \cdot \text{SUM} = 1 + 2/3 + 8/27 + 16/81 + \ldots \text{ and therefore:}

3/2 \cdot \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} \cdot 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_4(N) = \log_2(N) / \log_2(4) \ - \ relate \ base \ 4 \ to \ base \ 2

and \ \log_5(N) = \log_2(N) / \log_2(5) \ - \ relate \ base \ 5 \ to \ base \ 2

and thus \ \log_6(N) / \log_4(N) = \log_2(6) / \log_2(4) = \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 \cdot \log_2(N)

Number of decimal digits = ceiling(\log_{10}(0)) \ (within \ 1 \ of \ \log_{10}(0))

Number of binary digits = 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. Doubling the input size:

(a) When you double the size the runtime goes from \ n^2 \ to \ (2n)^2 = 4n^2, \ so \ the \ runtime \ increases \ four \ times.

(b) When you double the size the runtime goes from \ n^3 \ to \ (2n)^3 = 8n^3, \ so \ the \ runtime \ increases \ eight \ times.

(c) This is like (a) above when it comes to doubling the size, since it’s a quadratic algorithm.

(d) When doubling the size the runtime becomes \ 2n \ log(2n) = 2n(\log n + 1) = 2n \ log n + 2n.

(e) When doubling the size the runtime becomes \ 2^{2n} = (2^n)^2. \ That’s \ quite \ a \ big \ increase, \ since \ the \ runtime \ is \ squared.
Adding one to the input size:

(a) The runtime goes from $n^2$ to $(n + 1)^2 = n^2 + 2n + 1$.

(b) The runtime goes from $n^3$ to $(n + 1)^3 = n^3 + 3n^2 + 3n + 1$.

(c) The runtime goes from $100n^2$ to $100(n + 1)^2 = 100n^2 + 200n + 100$.

(d) The runtime becomes $(n + 1) \log(n + 1) = n \log(n + 1) + \log(n + 1)$. So the runtime increases by a little bit more than $\log n$, since as $n$ grows larger, the difference between $\log n$ and $\log(n + 1)$ is small.

(e) The runtime becomes $2^{n+1} = 2 \cdot 2^n$. The runtime doubles itself.

5. 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. Remember that a polynomial always beats a logarithm. The rank is: $\sqrt{2n}, n + 10, n^2 \log n, n^2, n^5, 10n, 100^n$.

6. 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.

7. 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.

8. (a) We should use the modulo and integer division as follows:

```java
int reverse(int num) {
    int result = 0, tmp = num;
    while(tmp > 0) {
        result = result*10 + tmp % 10; // get the next digit
        tmp = tmp / 10; // eliminate said digit
    }
    return result;
}
```

(b) Just check if a number is equal to its reverse (using a):

```java
boolean palindrome(int num) {
    int rev = reverse(num);
    return (rev == num);
}
```

(c) Use integer arithmetics:

```java
int intValue(String s) {
    int len = s.length();
    int sum = 0;
    for(int i=0;i<len;i++)
        sum = sum*10 + s[i] - '0';
    return sum;
}
```