1.
```java
/* Given a string and a character, find the number of times the character
 * appears in the string. Matches are case-sensitive.
 * Give a recursive implementation.
 */
public static int countChar(String str, char c) {
    if (str.length() == 0)
        return 0;
    int firstCharMatches = str.charAt(0) == c ? 1 : 0;
    return firstCharMatches + countChar(str.substring(1), c);
}
```

2.
```java
/* Find the maximum value in a list of Integers, using recursion.
 *
 * Hint: to keep track of which parts of the list still need to be visited,
 * which works a lot like cleanHotel(int lo, int hi)
 * use a recursive helper function:
 * recursiveMaxHelper(List<Integer> li, int lo, int hi)
 */
public static int recursiveMax(List<Integer> li) {
    if (li.size() == 1)
        return li.get(0);
    return Integer.max(li.get(0), recursiveMax(li.subList(1, li.size())));
}
```

or using a helper function

```java
public static int recursiveMax(int[] a) {
    return recursiveMaxHelper(a, 0, a.length);
}

public static int recursiveMaxHelper(int[] a, int lo, int hi) {
    if (lo + 1 == hi)
        return a[lo];
    return Integer.max(a[lo], recursiveMaxHelper(a, lo+1, hi));
}
```

3. We are given:

```java
/* Return a prime factor of n. */
public static int findAPrimeFactor(int n) {
    for (int f = 2; f < n; f++)
        if (n % f == 0)
            return f;
    return n;
}
```

```
/* Return a list of the prime factors of a given integer n, using recursion
   .
 * Your function should call findAPrimeFactor() to find a single prime
    factor
 * for you. (We're using it as a building block).
 *
 * You may assume that n is positive.
 */
public static List<Integer> recursivePrimeFactors(int n) {
    if (n == 1)
        return new ArrayList<>();
    int f = findAPrimeFactor(n);
    List<Integer> primeFactors = recursivePrimeFactors(n/f);
    primeFactors.add(f);
    return primeFactors;
}
```

4. Show that $3^{\log_4 n} = n^{\log_4 3}$ and give an identity that generalizes this equality.

   There are many valid proofs, most of which are likely using the change of base formula, as well as the fact that logs and exponential functions with the matching base are inverses of one another. Here is one possible proof.

$$\begin{aligned} 3^{\log_4 n} &= (n^{\log_n 3})^{\log_4 n} && n^x \text{ and } \log_n x \text{ are inverses of each other;} \\ &= n^{(\log_n 3)(\log_4 n)} && \text{by an exponent rule;} \\ &= n^{(\log_4 3/\log_4 n)(\log_4 n)} && \text{by the change of base formula;} \\ &= n^{\log_4 3} && \text{by algebra.} \end{aligned}$$

   More generally, we have the following log rule:

$$\boxed{a^{\log_b c} = c^{\log_b a}}$$

5. a) If $f(n)$ is $O(g(n))$, it is not necessarily true that $2^{f(n)}$ is $O(2^{g(n)})$. For example, if $f(n) = 2n$ and $g(n) = n$, then we have $2n$ is $O(n)$. However $2^{f(n)} = 4^n$, which is not $O(2^{g(n)}) = O(2^n)$.

   b) "The running time of algorithm A is at least $O(n^2)$" is meaningless because O-notation, by definition means "is at most" or "has an upper bound of".

6. In order of least dominant to most dominant function with respect to asymptotic growth rate, we have
   (a) $\log n$, $\log \sqrt{n} = (1/2)\log n$, $\log n^2 = 2\log n$
   (b) $(\log n)^2$ since $\log n > 1$, so that $(\log n)^2 > \log n$.
   (c) $\sqrt[36]{n}$
   (d) $\sqrt{n}$
   (e) n
   (f) $n \log n$, and $n \log_{10} n = (n \log n)/(\log 10)$
   (g) n^2

(h) n^4
(i) 2^n
(j) e^n, since $e > 2$
(k) 2^{n^2} since $2^{n^2} > 2^{2n} = 4^n$, and $4^n > e^n$ because $4 > e$.

7. (a) $5n^3 - 7n^2 + 88$ is $O(n^3)$
(b) $(n \log n + n^2)(n^3 + 2)$ is $O(n^5)$, by taking the dominant term in each factor.
(c) $\log(n^3 + 1) + (\log n)^2$ is $O((\log n)^2)$
because the first term is $O(\log n)$, which is of lower order than the second.
(d) $(n \log n + 1)^2 + (\log n + 1)(n^2 + 1)$:
since $(n \log n + 1)^2$ is $O(n^2(\log n)^2)$ and
since $(\log n + 1)(n^2 + 1)$ is $O(n^2 \log n)$,
then, $(n \log n + 1)^2 + (\log n + 1)(n^2 + 1)$ is $O(n^2(\log n)^2)$.
(e) $n^{2n} + n^{n^2}$ is $O(n^{n^2})$ simply by comparing the exponents.
(f) $(n! + 2^n)(n^3 + \log(n^2 + 1))$ is $O(n! n^3)$ by taking the dominant term in each factor.
(g) $n(5/4)^{\log_4 n + 1}$ is $O(n^{\log_4 5})$ by the following little calculation:

$$\begin{aligned}
n \cdot (5/4)^{\log_4 n + 1} &= \frac{5}{4} \cdot n \cdot \left(\frac{5}{4}\right)^{\log_4 n} && \text{by an exponent rule} \\
&= \frac{5}{4} \cdot n \cdot \frac{5^{\log_4 n}}{4^{\log_4 n}} && \text{by an exponent rule} \\
&= \frac{5}{4} \cdot n \cdot \frac{n^{\log_4 5}}{4^{\log_4 n}} && \text{by the result of problem 4} \\
&= \frac{5}{4} \cdot n \cdot \frac{n^{\log_4 5}}{n} && \text{because } \log_4(\cdot) \text{ and } 4^{\cdot} \text{ are inverses} \\
&= \frac{5}{4} \cdot n^{\log_4 5} && \text{by algebra.}
\end{aligned}$$

8. (a) $\displaystyle\sum_{i=10}^{n} i \le \sum_{i=1}^{n} i = \frac{n(n+1)}{2}$, which is $O(n^2)$.
Note: it's okay to substitute an expression we want to simplify with a larger expression when using O-notation. (Do you see why?)
(b) $\displaystyle\sum_{i=1}^{\log_2 n} ni = n \sum_{i=1}^{\log_2 n} i = n \cdot \frac{(\log_2 n)(\log_2 n + 1)}{2}$, which is $O(n(\log_2 n)^2)$.
(c) $\displaystyle\sum_{i=1}^{n} 4^n = 4^n \sum_{i=1}^{n} = 4^n \cdot n$, which cannot be simplified any further.
(d) $\displaystyle\sum_{i=1}^{n} 4^i \le \sum_{i=0}^{n} 4^i = \frac{4^{n+1} - 1}{4 - 1}$, which is $O(4^{n+1})$ or simply $O(4^n)$ since $4^{n+1} = 4 \cdot 4^n$.
(e) $\displaystyle\sum_{i=1}^{\log_2 n} 4^i \le \sum_{i=0}^{\log_2 n} 4^i = \frac{4^{\log_2 n + 1} - 1}{4 - 1} = \frac{4 \cdot 4^{\log_2 n} - 1}{4 - 1} = \frac{4n^{\log_2 4} - 1}{3} = \frac{4n^2 - 1}{3}$, which is $O(n^2)$. Note we used the result of problem 4 again to get the second to last equality.

(f) $\sum_{i=1}^{\log_2 n} n(1/3)^i \leq \sum_{i=0}^{\log_2 n} n(1/3)^i = n \sum_{i=0}^{\log_2 n} (1/3)^i = n \, \frac{1-(1/3)^{\log_2 n+1}}{1-1/3}$, which is $O(n)$.
This is because $\lim_{n\to\infty} (1/3)^{\log_2 n+1} = 0$ (we see that because the base is a constant < 1 while the exponent grows large), which means that $\frac{1-(1/3)^{\log_2 n+1}}{1-1/3}$ is dominated by the constant 1.