1. If 4^3 <= 82 < 5^3, then you know that the person is correct. The person is correct in this case.
2. Yes. This is because the minimum value $\sqrt[r]{n}$ can take is root, and the maximum value is less than root + 1. Therefore, the floor of $\sqrt[r]{n}$ must equal root
3. There is no reason to try a guess where g < 0, because n cannot be less than 0 and $\sqrt[r]{n}$ will not be less than 0. There may be a reason to try g > n because if $\sqrt[r]{n}$ is a decimal number, for example, 1.1, you'll find that $\sqrt[r]{n}$ > g when g = 1, but $\sqrt[r]{n}$ < g when g = 2, so you would have to attempt g = 2 before you know that you can floor $\sqrt[r]{n}$ to be equal to 1.
4. lowEnough = 0, tooHigh = n + 1
5. I would set lowEnough = 0 and tooHigh = 47227 and start off with a guess of ⌊tooHigh / 2⌋ = 23613, which I'll call g. I would raise g to the "r"th (5th, in this case) power, and if g^r > 47226, then I'll lower tooHigh to g. If g^r is <= 47226, I'll set lowEnough equal to g. Everytime I change low enough, I will check if (g+1)^r > 47226, in which case, I know that g is my answer.
6.

```
/**
 * Returns the {@code r}-th root of {@code n}.
 *
 * @param n
 *            the number to which we want to apply the root
 * @param r
 *            the root
 * @return the root of the number
 * @requires n >= 0 and r > 0 and n ^ (r) <= Integer.MAX_VALUE
 * @ensures root ^ (r) <= n < (root + 1) ^ (r)
 */
private static int root(int n, int r) {
     int lowEnough = 0;
     int tooHigh = n + 1;
     int g = (tooHigh + lowEnough) / 2;
     int lowGuess = power(g, r);
     int highGuess = power(g + 1, r);
     while(lowGuess > n || highGuess <= n) {
          if(lowGuess <= n) {
               lowEnough = g;
          } else {
               tooHigh = g;
          }
          g = (tooHigh + lowEnough) / 2;
          lowGuess = power(g, r);
```

```
        highGuess = power(g + 1, r);
    }
    return g;
}
```