Answers for Test 1

1. From the given information we concluded that $g(n)\in\Theta(n^2)$ or $g(n)\in o(n^2)$. In other words, $g(n)\in O(n^2)$. Therefore the answers are no, maybe, maybe, yes, maybe.

2. The for loop runs log n times, and the if statement takes either Θ(n log n) time or Θ(n) time, depending on which branch is taken. We are told to state the runtime using Θ notation if possible, otherwise O notation. Doing that, the runtime of the if statement is O(n log n) and the overall runtime is O(n * log n * log n), which is commonly written as O(n $\log^2$n).

3. We are told $f(n)\in O(g(n))$. From the definition of O this means that $\lim_{n\to\infty}\frac{f(n)}{g(n)}$ is 0 or some constant c. Inverting the fraction, compute that
$\lim_{n\to\infty}\frac{g(n)}{f(n)}$ is ∞ or some constant 1/c. This matches the definition of $g(n)\in\Omega(f(n))$.

4. $n^3/8$ n^3 $8n^3$ Any runtime proportional to n^3 will have this behaviour of taking 8 times as long to run when n is doubled.

5. The answer is (b). We translate SAT to set intersection, thus showing that we can make set intersection do the work of SAT.
The definition of *NP-complete* is that the problem must be in NP, and all other problems in NP must be reducible to it. Since we know SAT is NP-complete, all other problems are reducible to SAT. Therefore, when we provide the reduction in our proof (from SAT to set intersection) we have shown that all other problems in NP are reducible to set intersection. (We can reduce another problem to set intersection in two stages: first reduce it to SAT – which we know is possible because SAT is NP-complete – and then reduce from SAT to set-intersection using the reduction in our proof. Both of these reductions take polynomial time, so stringing the two of them together still takes polynomial time.)

6. We are told that someone has found a polynomial time algorithm for some NP-complete problem. Refer to that problem as "problem A".
From the definition of NP-complete, we know that all other problems in NP are reducible to A. So we can solve any problem in NP in polynomial time as follows: first reduce the problem to A (which takes polynomial time), and then execute the algorithm for A (which takes polynomial time). Executing these two steps in succession takes polynomial time.

7. To prove that Set Partition is in NP we need to show that it can be solved in polynomial time on a nondeterministic machine. Create 2^n guesses, to exhaustively enumerate all possible ways of partitioning S into two sets A and B. Each individual guess can be checked in linear time as follows. Sum up the items in A and compare to the sum of the items in B. Since there are n items total in A and B, this can be done in Θ(n) time, which is polynomial runtime.