These are just the final answers and a few key things to notice. Be sure that you know how to derive these answers in detail.

1. $O(n^3)$.
2. $O(n^3)$.
3. $O(n^2)$.
   Even though 10000 is a large number, the r-loop is $O(1)$. This makes sense because the time to execute it does not depend on anything — it’s always the same.
4. $O(n^3)$.
   An easy way to see this is to notice that the r-loop iterates at most $n$ times (when $p$ is 1). You can do a more fine-grained analysis, but it will still come out to $O(n^3)$ once you do the simplifications that big-oh allows. Exercise: Try this.
5. $O(n^3)$.
The r-loop iterates about $6n$ times (give or take one). We don’t need to figure out the exact number of iterations because a plus or minus one will get cancelled out in the end. In fact, we don’t even need the 6: we can just say that it iterates $O(n)$ times.
6. $O(n + \log m)$
   This one is tricky. On the first iteration of the for-loop, the while-loop iterates $O(\log m)$ times, doing $O(1)$ work each time; so that time the while-loop is $O(\log m)$. And on that first iteration of the for-loop we also do some $O(1)$ things after the while-loop. So in total, the first iteration of the for-loop is $O(\log m)$.
   But notice that the variable shrink is never reset. So every other time we reach the while-loop, it doesn’t iterate at all. That means the second and subsequent iterations of the for-loop take only $O(1)$ time each.
   So in total, we have $O(\log m)$ time due to the while-loop, and $O(n)$ time due to the 2nd and subsequent iterations of the for-loop (plus some $O(1)$ work before the for-loop).