Problem 1:

Understanding the problem

We repeatedly jump to a uniformly random vertex (including our current one), adding an edge as we go. The graph is connected exactly when we've *visited every vertex at least once*. So the task is: how many steps on average until we've seen all vertices, given we start with one already seen?

Initial thoughts

I pictured the process in **stages**: when I've seen some number of distinct vertices, the next step either discovers a new one or revisits an old one. That suggested treating each "discover the next new vertex" as its own waiting period and then adding those waiting times together.

Initial strategy

- Break the process into stages: "from `m` seen vertices to `m+1` seen vertices".
- For each stage, compute the average number of steps needed to discover a *new* vertex.
- Add the stage averages to get the final expected step count.

What went wrong

- I first wrote a loop that ran all the way to the end and hit a divide-by-zero edge case.
- I read `n` as a floating point and used it as a loop bound—unnecessary and slightly risky.
- I printed with `setprecision(6)` (significant digits) instead of fixed decimal places, which can fail tight accuracy checks.

Fixes

- Stop the summation one step earlier to avoid the edge case.

- Use integer `N` for bounds; use high-precision floating only for the running sum and final answer.
- Print with `fixed` and a generous number of decimal places to satisfy the error tolerance.