CS109 Lecture 17: LLM Learning Guide

Algorithmic Analysis

Summer 2026

Lecture 17: LLM Learning Guide

Before lecture: read the Lecture 17 slides (algorithmic analysis). Then open your favorite LLM and work through the concepts below **in order**. Each concept has a *Learn* prompt and a *Test me* prompt. Attempt the "Test me" questions yourself before asking the LLM to grade you.

How to get the most out of this:

- Tell the LLM your level: "I'm a student in Stanford's CS109 (probability for computer scientists). Explain at that level."
- After any answer, ask "why?" at least once and ask it to show each step.
- When it states a formula, ask it to define every symbol before continuing.
- Attempt practice yourself first, then say: "Here is my reasoning: _____. Where exactly am I wrong, if anywhere?"

Concept 1: Conditional expectation

Learn: > "Define conditional expectation for discrete random variables: $E[X \mid Y = y] = \sum_x x\, P(X = x \mid Y = y)$. Show how to compute it from a joint PMF table (condition on a row, renormalize, take the expectation). Also explain the distinction between $E[X \mid Y = y]$ (a number) and $E[X \mid Y]$ (a function of the random variable Y, hence itself random)."

Test me: > "Give me a small joint PMF table for two random variables and make me compute $E[X \mid Y = y]$ for each value of y, showing the renormalization. Then ask me whether $E[X \mid Y]$ is a number or a random variable, and why."

Concept 2: The law of total expectation

Learn: > "State the law of total expectation $E[X] = \sum_y E[X \mid Y = y]\, P(Y = y)$ (equivalently $E[X] = E[E[X \mid Y]]$), and walk through its derivation from the definitions using the chain rule, swapping the order of summation, and marginalization. Explain the intuition: a weighted average of per-case averages, weighted by how likely each case is. Compare it to the law of total probability."

Test me: > "Give me a problem where a quantity depends on which of 2 or 3 scenarios occurs (with given probabilities and per-scenario conditional expectations) and make me compute the overall expectation. Then make me verify it once directly from the joint distribution so I trust the theorem."

Concept 3: Expected runtime of code

Learn: > "Explain how the law of total expectation is used to analyze the expected runtime of a program whose cost depends on a random condition — for example, a request served from a local cache (fast, high probability) versus a remote server (slow, low probability). Work an example with three cases, and then discuss why expected runtime can be a misleading summary when the distribution has a long tail (percentiles vs. means in real systems)."

Test me: > "Give me a code snippet or system description where runtime depends on 2–3 random cases with given probabilities and costs. Make me define the right random variables, compute $E[T]$, and compute a conditional expectation like $E[T \mid \text{cache miss}]$. Then ask me one conceptual question about when the expectation is *not* the number I should report."

Concept 4: Analyzing recursive code

Learn: > "Explain the technique for computing the expected return value of *recursive* randomized code: let the unknown be $\mu = E[X]$, condition on the first random choice using the law of total expectation, write μ in terms of itself, and solve the linear equation. Work one example, like a function that with some probability returns a constant and otherwise returns a constant plus a recursive call. Point out the connection to expected values of geometric random variables, and mention when the equation would have no finite solution."

Test me: > "Write a short recursive function with randomness (like `randomInt` branches where some branches return a constant and others return a constant plus a recursive call) and make me set up the total-expectation equation and solve for $E[X]$. Check each coefficient. Then change the probabilities so the expected number of recursive calls is at least 1, and ask me what happens."

Concept 5: Indicator random variables

Learn: > "Explain the indicator-variable method for computing expectations of counts: to find $E[X]$ where X counts how many of n events occur, write $X = \sum X_i$ with $X_i \in \{0, 1\}$, use $E[X_i] = P(E_i)$, and apply linearity of expectation. Emphasize that linearity requires *no independence*. Work an example like the expected number of empty servers/buckets when requests are assigned randomly, and one with indicators over *pairs* (expected number of collisions)."

Test me: > "Give me two or three counting problems (empty buckets in a hash table, matching birthdays, fixed points of a random permutation) and make me solve each with indicators. Ask me at each step where I used linearity and whether independence was ever needed."

Concept 6: The coupon collector's problem

Learn: > "Explain the coupon collector's problem: collecting all n distinct types when each draw is uniform. Decompose the total draws as $X = X_0 + \cdots + X_{n-1}$ where $X_i \sim \text{Geo}(\frac{n-i}{n})$ is the wait for a new type after owning i types, so $E[X] = \sum \frac{n}{n-i} = nH_n \approx n \ln n$. Explain the intuition for why the last few coupons dominate the cost."

Test me: > "Give me a coupon-collector scenario (gacha games, hash-table coverage, promo caps) with a specific n and make me set up the geometric decomposition, compute $E[X]$, and compute the expected draws to reach a *partial* collection. Then ask me why the cost is front-loaded toward the final items and what the growth rate is in n."

Wrap-up prompt (after all six concepts)

> "Give me one multi-part CS109-style problem in which I must (1) compute conditional expectations from a joint table, (2) use the law of total expectation to find the expected runtime of code with random branches, (3) solve for the expected return value of a recursive randomized function, and (4) use indicator variables to compute an expected count. Let me solve every part, then grade each part, tell me which concept it tested, and tell me the single concept I should review most."

Note: these tools — especially total expectation and indicators — are the workhorses of algorithm analysis (expected runtime of quicksort and hashing), systems performance modeling, and epidemiology-style branching processes. They also show up constantly on the final.

Bring any sticking points to lecture; the TAs and instructor will help you work through them on the worksheet.