

2.12 Informal Run-Time Analysis

Lesson Plan

Pacing: 2 class periods of 45 minutes

Learning objectives: 2.12.A

Essential knowledge: 2.12.A.1

Materials

- Projector and the Count Every Step slide deck
- Whiteboard and markers
- Mini whiteboards or scrap paper
- IDE on every computer (Day 1)
- Printed Count Every Step puzzle worksheet (answer key for you) and exit ticket

Agenda

Minutes	Activity
5	Day 1 warm-up — who claps more? (slide 2)
12	Day 1 direct instruction — execution counts and the parts of a for loop (slides 3–6)
5	Day 1 pause and try — doubling up (slide 7)
12	Day 1 direct instruction — reading the header, steps other than 1 (slides 8–11)
8	Day 1 guided practice — predict header counts, then check with a counter
3	Day 1 closure — one pass too many
5	Day 2 warm-up — how many times does each line run?
12	Day 2 direct instruction — ifs and nested loops (slides 12–15)
5	Day 2 pause and try — the inner loop gets shorter (slide 16)
10	Day 2 direct instruction — comparing run times (slides 17–19)
5	Day 2 pause and try — predict the bigger input (slide 20)
3	Day 2 closure — multiply or add? (slides 21–22)
5	Day 2 exit ticket

Overview

A statement's execution count is how many times it runs while the program runs (2.12.A.1). Day 1 is counting: tracing a loop to count its statements, and reading a `for` or `while` header to know how many times the body runs, including steps other than 1 and loops that never run. Day 2 is structure: an `if` inside a loop gives two counts, nested loops multiply, and informally comparing two pieces of code by how many statements each executes and how that count grows when the input grows.

Stay inside the boundary: counts are numbers for given inputs. Do not introduce Big-O notation, formulas for sums, or logarithms by name (the halving loop is only "count the passes").

TEACHER NOTE

The slides are the Count Every Step deck (22 slides); slide numbers refer to it. Day 1 uses slides 2–11, Day 2 slides 12–22. The worksheet (10 questions, about 35 minutes) is homework: questions 1, 2, 3, and 4 after Day 1; questions 5 to 10 after Day 2. Question 9 settles the Think First guess about doubling the input.

Demonstrations are in the topic's `examples/` folder (`.snippet.java` files hold only the statements for `main`): `LapCount`, `StepTotal`, `HeaderCounts`, `ByFiveCount`, `OffByOne`, `NeverRuns` (Day 1); `EvenOdd`, `SeatingChart`, `NoBraces`, `NestedLess`, `ShortButBusy`, `SearchSteps`, `GrowthTable`, `Halving` (Day 2).

Day 1 warm-up: who claps more?

Show slide 2. Students predict which of the two claps more without running anything, and say what they would count to decide. Record the word "count" on the board; the lesson is about counting statement executions instead of timing a program.

Day 1 direct instruction: counting executions

Slide 5: trace a short loop and write how many times each statement runs. Slide 6: in a `for` loop, the initialization runs once, the condition one time more than the body (the last check is false), and the update as often as the body. Run `LapCount` or `StepTotal` so students can confirm counts with a counter variable.

MISCONCEPTION Assuming every loop body runs at least once

A `while` or `for` loop checks its condition first. Run `NeverRuns`: `score` starts at 120, the condition `score < 100` is false at once, and the body runs 0 times. The condition was still checked once.

Day 1 pause and try

Slide 7: students count the executions in the doubling code and write the final values before revealing.

Day 1 direct instruction: reading the header

Slide 9: for count-by-1 headers, count the values the variable takes: `k = 0`; `k < 12` is 12 values; `k = 1`; `k <= 12` is 12 values; `k = 4`; `k <= 12` is 9. Slide 10: for other steps, list the values (`k = 0`; `k < 12`; `k += 3` gives 0, 3, 6, 9, so 4 passes). Slide 11: check both ends.

MISCONCEPTION Off-by-one counts

Students mix up `<` with `<=` or forget the first value, so the count is one too many or too few. Have them write the first value and the last value that still passes the condition, then count the list. Run `OffByOne`.

Day 1 guided practice

Show four headers from worksheet question 2 on the board. Students write the number of passes on mini whiteboards, then run `HeaderCounts` (or their own loops with a counter) to check.

Day 1 independent practice

Puzzle worksheet questions 1, 2, 3, and 4 are homework: the loop that counts down by 7, the header table, the code tiles that show the condition checked one more time than the body runs, and the snippet pool for counts 8, 5, and 12.

Day 1 closure

Cold-call: "How many times does the body run for `for (int k = 12; k > 0; k -= 5) ?`" (3: k is 12, 7, 2.)

Day 2 warm-up

Show `for (int k = 1; k <= 6; k++) { if (k % 2 == 0) { evens++; } }` on the board. Students say how many times the condition `k % 2 == 0` is evaluated (6) and how many times `evens++` runs (3).

Day 2 direct instruction: ifs and nested loops

Slide 13: an `if` in a loop has two counts: the test runs on every pass, the guarded statement only when the condition is true. Slide 14: nested loops multiply (the inner body runs outer passes times inner passes). Slide 15: loops one after another add. Run `SeatingChart` (12 stars, 3 `println` calls) and `NestedLess` (10, from $0 + 1 + 2 + 3 + 4$).

MISCONCEPTION Adding instead of multiplying, or trusting the indentation

For a seating chart with 3 rows of 4 seats, the common answer is $3 + 4 = 7$. Adding fits loops one after another; a loop inside a loop gives $3 \text{ times } 4 = 12$. Separately, a loop without braces repeats only the one statement after its header however it is indented; run `NoBraces` (`cheers` is 1).

Day 2 pause and try

Slide 16: the inner loop gets shorter; students add the inner passes for each outer pass (for example 0, 1, 2, 3, 4) rather than multiplying.

Day 2 direct instruction: comparing run times

Slide 18: two segments reach the same answer, one checks every letter and the other stops when it finds a match; compare counts. Slide 19: when `n` doubles, a single loop does twice as many passes and a nested loop four times as many. Run `GrowthTable` and `ShortButBusy` to show that short code can do a lot of work (10,000 passes).

MISCONCEPTION Judging speed by how long the code looks

Fewer lines does not mean less work. `ShortButBusy` is five lines and runs its inner body 10,000 times. Counting executions, not reading length, compares two pieces of code.

Day 2 pause and try

Slide 20: students predict the work for the bigger input before the table is revealed, then explain the growth in one sentence.

Day 2 independent practice

Puzzle worksheet questions 5 to 10 are homework (the count match-up, the condition-versus-body face-off, which segment does less work, the one-token fix, two truths and a lie, and the wrap-up).

Day 2 closure and exit ticket

Show slides 21 (vocabulary) and 22: partners explain how to count the inner body of a nested loop. Give the exit ticket (about five minutes). Question 1 checks counting a nested loop whose inner bound depends on the outer variable, question 2 reading a header with a step, and question 3 counting several statements in nested loops (2.12.A.1).