UNIT 2 · SELECTION AND ITERATION

2.1 Algorithms with Selection and Repetition

Self-Study Guide · About 45 minutes

By the end you will be able to

- Name the three building blocks of algorithms (sequencing, selection, and repetition) and find each one in an everyday algorithm.
- Explain how selection uses a true-or-false decision to choose what happens next, and trace an algorithm with a decision for different inputs.
- Explain how repetition repeats a process until a goal is reached, trace a loop pass by pass, and spot a loop that never ends.
- Represent algorithms with decisions and loops in numbered written steps and as flowcharts with decision diamonds.
- Predict how changing the order of sequencing, selection, and repetition changes an algorithm's result.

What to remember from earlier

- An algorithm is a set of ordered steps that gets a job done. (Topic 1.1: Introduction to Algorithms, Programming, and Compilers, 1.1.1 What Is an Algorithm?)
- An algorithm can be written as numbered steps or drawn as a flowchart. (Topic 1.1: Introduction to Algorithms, Programming, and Compilers, 1.1.2 Representing Algorithms: Words and Diagrams)
- Sequencing means doing steps one at a time, in order, and the order matters. (Topic 1.1: Introduction to Algorithms, Programming, and Compilers, 1.1.3 Sequencing: Order Matters)
- Math.abs is a ready-made method that gives the absolute value of a number. (Topic 1.11: Math Class, 1.11.2 Absolute Value)
- indexOf returns the index where text first appears, or -1 if it never does. (Topic 1.15: String Manipulation, 1.15.8 Finding Text with indexOf)

2.1.1 Sequencing, Selection, and Repetition

Think about how you actually get ready for school. You do not follow the exact same list every day. If it is raining, you grab an umbrella; if it is not, you leave it. You brush your teeth until they feel clean, however long that takes. Your routine reacts to the situation.

Every algorithm you have seen so far could not do that. In Topic 1.1, an algorithm was a list of steps done one at a time, in order: **sequencing**. Every Java program in Unit 1 worked the same way: each statement ran exactly once, from top to bottom, no matter what. A pure sequence has no way to say "only sometimes" or "keep going until."

Real algorithms need two more kinds of steps. Together with sequencing, they are the three **building blocks** of algorithms, the only pieces you need to describe any algorithm:

- **Sequencing:** steps are done one after another, in order.
- **Selection:** the algorithm makes a decision and chooses which step to do next. "If it is raining, take an umbrella" is selection. Another name for it is **decision making**.

- **Repetition:** a step, or a group of steps, is done again and again until a goal is reached. "Brush until your teeth are clean" is repetition. Another name for it is **looping**, and the repeated steps are called a **loop**.

KEY IDEA

Algorithms are built from three building blocks: sequencing (steps in order), selection (choosing what to do next with a decision), and repetition (repeating steps with a loop until a goal is reached).

WORKED EXAMPLE **Finding the building blocks in a routine**

Here is a morning routine written as numbered steps:

1. Get dressed.
2. If it is raining, take an umbrella.
3. Brush your teeth until they are clean.
4. Walk to school.

Go through it step by step and ask what kind of step each one is:

Step	Building block	How you can tell
1. Get dressed.	Sequencing	Always done, once, in its place in the order
2. If it is raining, ...	Selection	Whether the umbrella step happens depends on a decision
3. Brush ... until clean.	Repetition	Brushing continues again and again until a goal ("clean") is reached
4. Walk to school.	Sequencing	Always done, once, after step 3

Notice that the whole list is still a sequence: step 2 comes after step 1, and step 3 comes after step 2. Selection and repetition live *inside* that order. Sequencing is the backbone that holds the other two in place.

Now change one thing: the day. On a sunny day, the umbrella step is skipped, so the routine does steps 1, 3, and 4. On a day when brushing takes longer, step 3 simply repeats more times. The written algorithm is the same on every day, but what actually happens changes with the situation. A pure sequence could never do that; it always does the same steps the same number of times.

WATCH OUT

"Writing a step several times is repetition." A student writes a recipe like this:

1. Stir the batter.
2. Stir the batter.
3. Stir the batter.

That is still pure sequencing: three separate steps, each done once. It also guesses that three stirs is right. Real repetition is a single instruction to repeat until a goal is reached, such as "Stir the batter until there are no lumps." That version works whether the batter needs two stirs or twenty, because the goal decides when to stop, not a guess.

PAUSE AND TRY

Question 1 Multiple choice

A set of directions for a board game includes the step "If you roll a 6, take another turn." Which building block does this step show?

A. Sequencing, because it is one of the steps in the directions
B. Selection, because whether you take another turn depends on a decision
C. Repetition, because rolling dice happens many times in the game
D. None, because a game rule is not an algorithm

Question 2 Short answer

Here are directions for washing dishes: (1) Fill the sink with warm water. (2) Wash a dish. (3) Repeat step 2 until no dirty dishes are left. (4) If the water is still clean, leave it for later; otherwise, drain the sink. Name the building block shown by step 1, by step 3, and by step 4, and give a reason for each.

2.1.2 Selection: Deciding with a True-or-False Question

Selection lets an algorithm handle different situations. But think back to Topic 1.1: a computer is a literal follower. It cannot weigh options or use judgment. So how can it "decide" anything?

The trick is to make every decision a question with exactly two possible answers: **true** or **false**. "Is it raining?" is either true or false. A literal follower can check it and then do what the algorithm says to do for that answer. **Selection** happens when the choice of how an algorithm continues is based on a true-or-false decision. The question being checked is called the **condition**.

In written language, selection usually looks like one of these two shapes:

- **Do something, or skip it:** "If *condition*, then do this step." When the condition is false, the step is skipped and the algorithm moves on.
- **Choose between two steps:** "If *condition*, then do this step; otherwise, do that step." Exactly one of the two steps happens.

Either way, after the decision is made, the algorithm continues with the next step in the sequence.

KEY IDEA

Selection chooses how an algorithm continues based on a condition: a question whose answer is either true or false.

WORKED EXAMPLE **A movie ticket price**

A theater uses this algorithm to find a ticket price:

1. Ask for the customer's age.
2. If the age is less than 13, the price is $8; otherwise, the price is $12.
3. Print the ticket.

The condition is "the age is less than 13." To trace a decision, try it with different inputs and follow the path each one takes:

Age	Is the age less than 13?	Step chosen	Price
9	true	"the price is $8"	$8
15	false	"the price is $12"	$12
13	false (13 is not less than 13)	"the price is $12"	$12

The row for age 13 matters most. The number 13 sits right on the edge between the two choices, so it is the input most likely to be handled wrong. "Less than 13" is false for 13, so a 13-year-old pays $12. If the theater wanted 13-year-olds to pay $8, the condition would have to be "the age is 13 or less." When you check a decision, always test a value right at the edge.

Now a decision you have already used without seeing it. `Math.abs` from Topic 1.11 returns a number's absolute value: its distance from zero. You can describe the result it produces as an algorithm with a decision:

1. If the number is less than 0, the answer is the number with its sign flipped; otherwise, the answer is the number itself.

Test that description against the real method:

```
System.out.println(Math.abs(-7));
System.out.println(Math.abs(4));
System.out.println(Math.abs(0));
```

```
7
4
0
```

Number	Is the number less than 0?	Answer
-7	true	flip it: 7
4	false	keep it: 4
0	false (0 is not less than 0)	keep it: 0

All three outputs match the written algorithm. We are not looking inside `Math.abs` here, only describing what it produces. The point is that a method you called in Unit 1 behaves like an algorithm with selection: its result depends on a true-or-false decision about its input.

WATCH OUT

A "decision" that is not a true-or-false question. A student writes: "Decide what to wear based on the weather." A literal follower cannot carry this out. "The weather" is not true or

false; it could be sunny, cloudy, snowy, or windy, and the step never says what to do for any of them.

Fix it by rewriting the decision as a question with only two answers and saying what to do for each: "If the temperature is below 50 degrees, wear a jacket; otherwise, wear a T-shirt." Now the follower checks one true-or-false condition and knows exactly which step to take.

PAUSE AND TRY

Question 3 Tracing

An online store uses this algorithm:

1. Add up the prices of the items in the cart.
2. If the total is at least $35, shipping is free; otherwise, shipping costs $6.
3. Display the total plus shipping.

What amount is displayed for a cart totaling $20, and for a cart totaling $35?

Question 4 Multiple choice

Which step is written so that a literal follower can make the decision?

A. "If the soup is ready, serve it."
B. "If the soup has been simmering for at least 20 minutes, serve it."
C. "Serve the soup when it seems right."
D. "Choose how to serve the soup."

2.1.3 Repetition: Repeating Until the Goal Is Reached

Suppose you are saving up for $50 headphones, and you put $15 from your paycheck into savings each week. How many weeks will it take? You could work it out first and write that many steps. But if the price or the paycheck changes, the steps are wrong. In Unit 1, the only way to do something several times in Java was to write the statement several times, just like the "stir, stir, stir" recipe from 2.1.1.

Repetition means doing the same steps over and over until you get the result you were aiming for. Instead of guessing how many times, the algorithm states the goal and checks it: "Add $15 to savings until savings is at least $50." The steps that repeat form a **loop**. Each time through the loop, the follower does the repeated steps and then checks whether the goal has been reached. If not, it goes around again. If so, the loop ends and the algorithm moves on to the next step.

Look closely at that check: "Is savings at least $50?" It is a true-or-false question, just like the condition in a selection. Every loop contains a decision about whether to keep going.

KEY IDEA

Repetition repeats a process until a desired outcome is reached. A true-or-false check of that outcome decides whether the loop goes around again or stops.

WORKED EXAMPLE **Saving for headphones**

1. Set savings to $0.
2. Repeat: add $15 to savings, until savings is at least $50.
3. Buy the headphones.

Trace the loop one pass at a time. A **pass** here means one time through the repeated step, followed by the check:

Pass	Savings after adding $15	Is savings at least $50?	What happens next
1	$15	false	Go around again
2	$30	false	Go around again
3	$45	false	Go around again
4	$60	true	Stop; go to step 3

The loop runs **4 times**, and savings ends at **$60**. Notice that the algorithm never said "four." The number of passes came from the goal.

Now vary the starting point. Suppose you already have $20 when you begin, so step 1 says "Set savings to $20." The first pass brings savings to $35 (false), and the second brings it to $50 (true; $50 is at least $50). The loop stops after only **2 passes**. Same loop, same goal, different situation, different number of passes. That is exactly what repetition is for.

Sometimes the goal is a count. "Do 10 jumping jacks" can be written as a loop that keeps track of how many are done: (1) Set count to 0. (2) Repeat: do a jumping jack, then add 1 to count, until count is 10. The goal "count is 10" is reached after exactly 10 passes.

WATCH OUT

A loop that never reaches its goal. Look at this algorithm:

```
1. Set savings to $0.
2. Repeat: spend $5 from savings, until savings is at least $50.
```

Each pass makes savings smaller: -$5, -$10, -$15, and so on. Savings can never reach $50, so the check is false every single time. A literal follower would keep going forever. This is called a **never-ending loop** (programmers also say **infinite loop**).

Students write never-ending loops when they focus on the goal and forget to check that each pass actually moves toward it. To fix one, make sure the repeated steps change things in the direction of the goal: "add $15," not "spend $5."

PAUSE AND TRY

Question 5 Tracing

A baker has 23 cookies and follows this algorithm:

1. Repeat: put 6 cookies into a new bag, until fewer than 6 cookies are left.
2. Eat the cookies that are left.

How many bags are filled, and how many cookies does the baker eat? Show a trace table.

Question 6 Multiple choice

Which algorithm is a never-ending loop?

A. Set the water level to 0 cm. Repeat: add 2 cm of water, until the water level is at least 10 cm.
B. Set the temperature to 80 degrees. Repeat: lower the temperature by 3 degrees, until the temperature is below 70 degrees.
C. Set the count to 0. Repeat: clap once, then add 1 to the count, until the count is 5.
D. Set the gas in the tank to 10 gallons. Repeat: drive 30 miles, which uses 1 gallon, until the tank holds at least 12 gallons.

2.1.4 Showing Decisions and Loops in Words and Diagrams

In Topic 1.1 you represented algorithms in two ways: as numbered written steps and as a flowchart, where each step is a box and the arrows show the order. Those flowcharts were straight lines: one arrow in, one arrow out, from Start to Stop. A decision breaks that pattern, because after it the algorithm can go two different ways. How do you show that?

In a flowchart, a decision is drawn as a diamond, called a **decision diamond**. The condition is written inside it, and two arrows leave it: one labeled **true** and one labeled **false**. In the diagrams below, the diamond is drawn with pointed ends like `< this >`, regular steps are in square brackets, and Start and Stop are in parentheses. Here is the umbrella step from 2.1.1:

```
        ( Start )
            |
            v
   < Is it raining? >
     |            |
    true        false
     |            |
     v            |
[Take umbrella]   |
     |            |
     v            v
    [Walk to school]
            |
            v
        ( Stop )
```

Follow it on a rainy day: the condition is true, so take the left arrow to "Take umbrella," then continue to "Walk to school." On a dry day, take the false arrow, which goes straight to "Walk to school," skipping the umbrella. Both paths come back together, because after the decision the algorithm continues in sequence.

A loop is shown with an arrow that points **back up** to an earlier step. The decision diamond checks the goal: one arrow leaves the loop, and the other goes back around. Here is the savings algorithm from 2.1.3:

```
       ( Start )
           |
           v
  [Set savings to $0]
           |
           v
       [Add $15] <------+
           |            |
           v            | false
< Savings at least $50? >
           |
           | true
           v
       ( Stop )
```

The false arrow leaves the right side of the diamond and climbs back to "Add $15." That backward arrow *is* the repetition. When the condition is finally true, the downward arrow leads out of the loop.

In written language, you have already seen "If ..., then ...; otherwise, ..." and "Repeat ... until ...." A third way, which matches a flowchart arrow for arrow, writes the backward arrow as a "go back" instruction:

1. Set savings to $0.
2. Add $15 to savings.
3. If savings is less than $50, go back to step 2.
4. Stop.

KEY IDEA

In a flowchart, a decision is a diamond with a true arrow and a false arrow, and a loop is an arrow that goes back to an earlier step. In written steps, the same ideas are "if ... otherwise ...," "repeat ... until ...," or "go back to step ...."

WORKED EXAMPLE **Checking that two representations match**

Compare the written version to the diagram, one piece at a time:

Written step	Flowchart piece
1. Set savings to $0.	First box after Start
2. Add $15 to savings.	The box the backward arrow points to
3. If savings is less than $50, go back to 2.	The diamond; its false arrow goes back to "Add $15"
4. Stop.	The true arrow out of the diamond leads to Stop

Notice that step 3 asks "less than $50?" while the diamond asks "at least $50?" They are opposite questions, so the arrows are swapped too: written step 3 goes back when its answer is **true**, and the diamond goes back when its answer is **false**. Both send the follower back in exactly the same situations (whenever savings is under $50), so they represent the same algorithm. When you compare two representations, check which answer leads where, not just whether the words look alike.

WATCH OUT

A backward arrow that points to the wrong step. Suppose the false arrow from the diamond went back to "Set savings to $0" instead of "Add $15." Then every pass would reset savings to $0 and add $15, so savings would be $15 at every check. The goal is never reached, and the loop never ends.

Students make this mistake because the arrow looks like it goes "back to the start of the loop." Ask yourself which steps should repeat. Setting savings to $0 should happen once, before the loop, so the backward arrow must point to the first step that repeats, "Add $15."

PAUSE AND TRY

Question 7 Tracing

Trace this algorithm and give the final value of *total* and the number of times step 2 is done.

1. Set total to 1.
2. Double total.
3. If total is less than 20, go back to step 2.
4. Stop.

Question 8 Multiple choice

In this flowchart, the diamond's false arrow goes back up to "Take one step forward."

```
          ( Start )
              |
              v
  [Take one step forward] <---+
              |               |
              v               | false
    < Are you at the door? >--+
              |
              | true
              v
       [Open the door]
              |
              v
          ( Stop )
```

Which written algorithm represents the same algorithm?

A. Take one step forward. If you are at the door, open the door. Stop.
B. Repeat: take one step forward, until you are at the door. Then open the door.
C. Repeat: open the door, until you are at the door. Then take one step forward.
D. If you are not at the door, open the door; otherwise, take one step forward.

2.1.5 Order Matters When You Combine Them

Back in Topic 1.1, you saw that the same steps in a different order can give a different result: socks before shoes, not after. That was about sequencing alone. Now an algorithm can have decisions and loops too, and the question becomes: does it matter *where* they go?

It does. Where you place each step, each decision, and each loop helps decide what the algorithm produces. A decision checks the situation at the moment it is made, so a step placed before the decision can change the answer. A loop's check can come before or after the repeated steps, and that changes how many times they run.

KEY IDEA

The order of sequencing, selection, and repetition affects the outcome. The same steps and decisions, arranged differently, can produce a different result.

WORKED EXAMPLE **A coupon before or after shipping**

A store has a coupon: $10 off any order of at least $30. Shipping costs $5. Here are two algorithms with the same steps in a different order:

Algorithm A

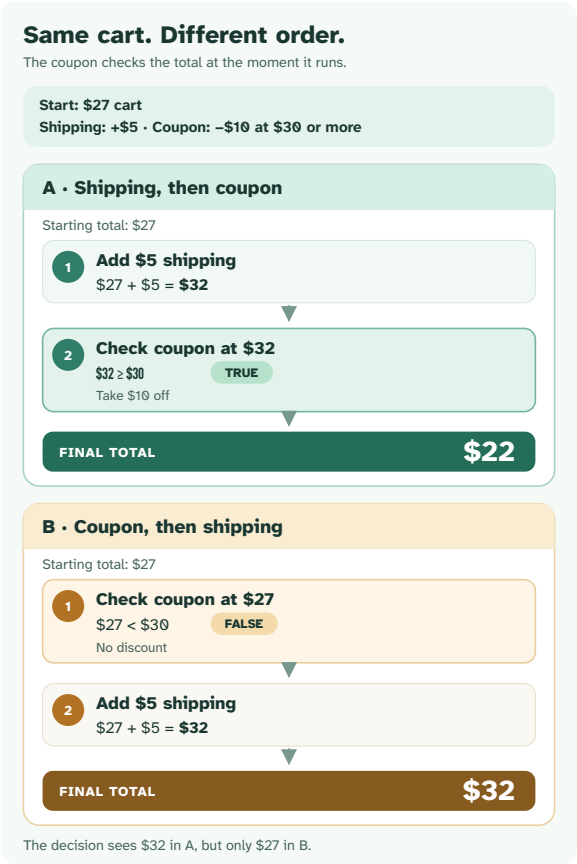
1. Add $5 shipping to the total.
2. If the total is at least $30, take $10 off.

Algorithm B

1. If the total is at least $30, take $10 off.
2. Add $5 shipping to the total.

Trace both for an order of $27:

Algorithm	After step 1	Is the total at least $30?	Final total
A	$27 + $5 = $32	true ($32): take $10 off	**$22**
B	not yet checked: $27	false ($27): no coupon	**$32**



For Algorithm B, the decision happens first, while the total is still $27. Then shipping is added: $27 + $5 = $32. Algorithm A adds shipping first, so the total has grown to $32 by the time the decision checks it. The same $27 order costs $22 in one algorithm and $32 in the other. The steps are identical; only the order changed, and with it the moment the decision looked at the total.

Now vary the building block: move a loop's check. Compare two ways to water a plant:

- **Check after:** Repeat: pour a cup of water, until the soil is wet.
- **Check first:** Check whether the soil is wet; if it is not, pour a cup of water and check again.

When the soil starts out dry, both pour until it is wet. But suppose it rained last night and the soil is already wet. The check-after version pours a cup anyway, because it does the repeated step before it ever checks. The check-first version pours **zero** cups. Placing the check before or after the repeated step changes the outcome in exactly this kind of edge case.

Order matters in Java too. Even in Unit 1 programs, which only use sequencing, swapping two statements can change the result. These two snippets use the compound assignment operators from Topic 1.6 on the same starting value:

```java
int points = 10;
points += 5;
points *= 2;
System.out.println("Points: " + points);
```

```
Points: 30
```

```java
int points = 10;
points *= 2;
points += 5;
System.out.println("Points: " + points);
```

```
Points: 25
```

In the first snippet, `points` goes 10 → 15 → 30: the bonus of 5 is added and then doubled along with everything else. In the second, `points` goes 10 → 20 → 25: the bonus is added after doubling, so it counts only once. Neither is wrong; they are different algorithms, and the order you choose must match what you want to happen.

WATCH OUT

"Same steps, same result." Students check that an algorithm has all the right steps and the right decisions and stop there. As the coupon example shows, that is not enough. A decision only sees the situation at the moment it runs, so you must also ask, "What has already happened by the time this check is made?" When an algorithm gives a wrong result, try tracing it with a value near the edge (like $27 or $30 for the coupon) and watch what each decision sees.

PAUSE AND TRY

Question 9 Tracing

A game starts a player with 40 points and uses this algorithm at the end of a level:

1. If the player has at least 50 points, award a gold star.
2. Add 15 bonus points.

Does the player get a gold star? Would the answer change if steps 1 and 2 were swapped? Show what the decision sees in each order.

Question 10 Debugging

A game's rules say: "At the end, **double** the player's score, and **then** add a 50-point bonus." A player finishes with 100 points, so the final score should be 250. This code prints `Final score: 300`. Find the error and write the corrected code.

```
int score = 100;
score += 50;
score *= 2;
System.out.println("Final score: " + score);
```

Putting it together

Real algorithms usually combine all three building blocks, often with a decision *inside* a loop. Here is one you already know the result of: finding where a letter first appears in a word, which is what `indexOf` from Topic 1.15 reports.

To find the first `"p"` in `"hopper"`:

1. Set position to 0.
2. Repeat these steps until the search is done:
   - If position is past the last letter, the answer is -1 and the search is done.
   - Otherwise, if the letter at position is `"p"`, the answer is position and the search is done.
   - Otherwise, add 1 to position.

The building blocks: step 1 then step 2 is **sequencing**; step 2 is **repetition**; and each pass makes **decisions** (first "is position past the end?", then "is this letter a p?"). Recall that the letters of `"hopper"` sit at indices 0 through 5:

```
index:  0 1 2 3 4 5
letter: h o p p e r
```

Trace it:

Pass	Position	Past the last letter?	Letter is "p"?
1	0	false	false ("h"): add 1
2	1	false	false ("o"): add 1
3	2	false	true: answer is 2, done

The search stops at position **2**, the first `"p"`, and never looks at the second `"p"` at index 3. Searching for `"z"` would find no match at positions 0 through 5; then position becomes 6, which is past the last letter, so the answer is -1.

Order matters here as well. The "past the last letter?" decision comes first in each pass for a reason: if the algorithm tried to look at the letter at position 6 first, there would be no letter there to look at. In Java, asking for a character that does not exist causes a run-time error, as you saw with `substring` in Topic 1.15.

Now check the written algorithm against Java:

```java
String word = "hopper";
System.out.println(word.indexOf("p"));
System.out.println(word.indexOf("z"));
```

```
2
-1
```

Both results match the trace. We are not claiming this is exactly how Java's `indexOf` is written inside; the point is that its behavior is described by an algorithm that uses sequencing, selection, and repetition together. Later in this unit, you will learn the Java statements that let you write decisions and loops like these yourself.

Vocabulary

Term	Definition	Example
building blocks	The three kinds of steps every algorithm is built from: sequencing, selection, and repetition	A routine with ordered steps, a decision, and a loop
sequencing	Doing steps one at a time, in order	Get dressed, then walk to school
selection / decision making	Choosing how an algorithm continues based on a true-or-false decision	If it is raining, take an umbrella
condition	The true-or-false question a decision checks	"Is the age less than 13?"
repetition / looping	Repeating a process until a desired outcome is reached	Add $15 until savings is at least $50
loop	The steps that repeat	"Add $15 to savings"
pass	One time through a loop's repeated steps, followed by the check	Pass 3 brings savings to $45
never-ending loop / infinite loop	A loop whose goal is never reached, so it never stops	Spend $5 until savings is at least $50
decision diamond	The flowchart shape for a decision, with a true arrow and a false arrow	`< Is it raining? >`

Chapter summary

- Every algorithm is built from three **building blocks**: **sequencing** (steps in order), **selection** (decision making), and **repetition** (looping).
- **Selection** chooses how an algorithm continues based on a **condition**, a question that is either true or false. Test decisions with a value right at the edge.
- **Repetition** repeats a process until a desired outcome is reached; a true-or-false check after each pass decides whether to go around again. If the repeated steps never move toward the goal, the loop never ends.
- Decisions and loops can be written in words ("if ... otherwise ...," "repeat ... until ...," "go back to step ...") or drawn in a flowchart with **decision diamonds** and backward arrows.
- The order of sequencing, selection, and repetition affects the outcome, because a decision only sees what has happened before it is checked.

Review questions

Question 11 Tracing

A teacher flips through a stack of quiz cards with the scores 82, 64, 91, and 70, in that order, using this algorithm:

1. Set passCount to 0.
2. Repeat until no cards are left: take the top card; if its score is at least 70, add 1 to passCount.
3. Announce passCount.

What number is announced? Show a trace table.

Question 12 Multiple choice

A student writes this algorithm for getting into a locked building: "Repeat: knock on the door, until someone opens it." Which statement about it is true?

A. It uses only sequencing, because each knock comes after the one before it.
B. It uses repetition, and the check "has someone opened the door?" is a true-or-false decision.
C. It uses selection but not repetition, because the word "until" makes a decision.
D. It cannot be an algorithm, because nobody knows in advance how many knocks it will take.

Question 13 Debugging

This algorithm is supposed to count down from 5 and then shout "Go!", but it never stops. Find the error and write the corrected step.

1. Set number to 5.
2. Say number.

3. Subtract 1 from number.
4. If number is greater than 0, go back to step 1.
5. Shout "Go!"

Question 14 Multiple choice

A pool admits swimmers using this algorithm: "If the swimmer is taller than 48 inches, allow them in the deep end; otherwise, send them to the shallow end." The pool's rule says swimmers who are **exactly** 48 inches tall may use the deep end. Which statement is correct?

A. The algorithm follows the rule, because 48 is taller than 48.
B. The algorithm breaks the rule for a swimmer who is exactly 48 inches tall; the condition should be "at least 48 inches."
C. The algorithm breaks the rule for a swimmer who is 50 inches tall.
D. The algorithm cannot be checked, because height is not a true-or-false question.

Question 15 Short answer

Two friends write algorithms for filling a glass with water:

- **Ana:** Repeat: pour a little water into the glass, until the glass is full.
- **Ben:** Check whether the glass is full. If it is not, pour a little water and check again.

Describe a situation where these two algorithms give different results, and explain why.

Question 16 Tracing

Trace this algorithm, which starts with number set to 20. List what is said and the final value of count.

1. Set count to 0.
2. Repeat until number is 1: divide number by 2, dropping any remainder; add 1 to count; say number.
3. Say count.

Reflection

Check each statement. If the answer is "not yet," reread the linked subsection.

- [ ] I can name the three building blocks of algorithms and find each one in an everyday algorithm. Reread: 2.1.1 Sequencing, Selection, and Repetition
- [ ] I can explain how selection uses a true-or-false condition and trace a decision, including an edge value. Reread: 2.1.2 Selection: Deciding with a True-or-False Question
- [ ] I can trace a loop pass by pass and explain why a loop might never end. Reread: 2.1.3 Repetition: Repeating Until the Goal Is Reached
- [ ] I can show decisions and loops in written steps and in a flowchart with decision diamonds. Reread: 2.1.4 Showing Decisions and Loops in Words and Diagrams
- [ ] I can predict how changing the order of steps, decisions, and loops changes the result. Reread: 2.1.5 Order Matters When You Combine Them