› Answers and Explanations

Bullets mark each step in the process of arriving at the correct solution.

1. The answer is B.

- The first time we evaluate the `while` condition, value = 31, which is >= 10, so the loop executes.
  - In the loop, we subtract count from value which becomes 30,
  - and then we add 3 to count, which becomes 4.
- Then we go back to the top of the `while` loop and reevaluate the condition. 30 >= 10, so we execute the loop again.
  - This time we subtract 4 from value, since count is now 4. value = 26
  - and count = 7.
- Back up to the top, 26 >= 10, so on we go.
  - value = 27 − 7 = 19 and count = 10.
- Since 10 >= 10, we will execute the loop one more time.
  - value = 19 − 10 = 9 and count = 13.
- This time when we evaluate the condition, it is no longer true, so we exit the loop and print value, which equals 9.

2. The answer is D.

- When we first enter the `for` loop count = 5 and i = 3.
- The first time through the loop, we execute count += i, which makes count = 8, and we increment i by 2, which makes i = 5.
- i < 7, so we execute the loop again; count = 8 + 5 = 13, and i = i + 2 = 7.
- This time the condition i < 7 fails and we exit the loop. 13 is printed.

3. The answer is A.

- The first time through the loop, incr = 1 and i = 0. 0 − 1 = -1, which is printed; then `incr` is increased to 2. The increment portion of this `for` loop is a little unusual. Instead of i++ or i = i + 2, it's i = i + incr, so, since incr = 2 and i = 0, i becomes 2.
- 2 < 10 so we continue. 2 − 2 = 0, which is printed, incr = 3 and i = 2 + 3 = 5.
- 5 < 10 so we continue. 5 − 3 = 2, which is printed, incr = 4 and i = 5 + 4 = 9.
- 9 < 10 so we continue. 9 − 4 = 5, which is printed, incr = 5 and i = 9 + 5 = 14.
- This time we fail the condition, so our loop ends having printed -1 0 2 5

4. The answer is E.

- The first option is the classic way to write a `for` loop. If you always start at i = 0 and go until i < n, you know the loop will execute n times without having to think about it (using i++ of course). So this first example prints eight "$".
- The second example is the exact same loop as the first example, except written as a `while` loop instead of a `for` loop. You can see the i = 0, the i < 8, and the i++. So this example also prints eight "$".
- The third example shows why we always stick to the convention used in the first example. Here we have to reason it out: i will equal 7, 10, 13, 16, 19, 22, 25, 28 before finally failing at 31. That's eight different values, which means eight times through the loop. So this example also prints eight "$".

5. The answer is E.

- `Integer.MAX_VALUE` is a constant that represents the greatest value that can be stored in an `int`. In this problem, we need to make sure that `a + b` is not greater than that value.

- Solution b seems like the logical way to do this, but if a + b is too big to fit in an int, then we can't successfully add them together to test if they are too big. a + b will overflow.
- We need to do some basic algebra and subtract a from both sides giving us:

```
b < Integer.MAX_VALUE - a
```

- The left and right sides of the expression have been switched, so we need to flip the inequality. Flipping the < gives us >=.

6. The answer is D.

- The outer loop is going to execute 10 times: j = 0 through j = 9.
- The inner loop is trickier. The number of times it executes depends on j.
  - The first time through, j = 0, so k = 10, 9, 8 . . . 1 (10 times).
  - The second time through, j = 1, so k = 10, 9, 8 . . . 2 (9 times).
  - The third time through, j = 2, so k = 10, 9, 8 . . . 3 (8 times).
  - There's a clear pattern here, so we don't need to write them all out. We do have to be careful that we stop at the right time though.
  - The last time through the loop, j = 9, so k = 10 (1 time).
  - Adding it all together: 10 + 9 + 8 + 7 + 6 + 5 + 4 + 3 + 2 + 1 = 55

7. The answer is A.

- b1 and b2 are both true. The OR (||) in the `while` condition requires that at least one of them be true, so we enter the loop for the first time.
  - 7 > 4 so we execute b2 = ! b2. That's a tricky little statement that is used to flip the value of a `boolean`. Since b2 is true, !b2 is false, and we assign that value back into b2, which becomes false.
  - We skip the `else` clause and subtract one from x.
  - At the bottom of the loop: b1 = true, b2 = false, x = 6.
- The `while` condition now evaluates to (true || false), which is still true, so we execute the loop.
  - 6 > 4 so we flip b2 again. b2 is now true.
  - Skip the `else` clause, subtract one from x.
  - At the bottom of the loop: b1 = true, b2 = true, x = 5.
- The `while` condition now evaluates to (true || true), which is true, so we execute the loop.
  - 5 > 4 so we flip b2 again. b2 is now false.
  - Skip the `else` clause, subtract one from x.
  - At the bottom of the loop: b1 = true, b2 = false, x = 4.
- The `while` condition now evaluates to (true || false), which is still true, so we execute the loop.
  - This time the `if` condition is false, so we execute the `else` clause, which flips the condition of b1 instead of b2. b1 is now false.
  - Subtract one from x.
  - At the bottom of the loop: b1 = false, b2 = false, x = 3.
- Now when we go to evaluate the `while` condition, we have (false || false). Do not execute the loop. Print the value of x, which is 3.

8. The answer is C.

- Option I
  - As we enter the loop for the first time, i = 1, and that's what gets printed. Then we multiply by 2, i = 2.
  - We print 2, multiply by 2, i = 4.
  - We print 4, multiply by 2, i = 8.
  - We print 8, multiply by 2, i = 16, which fails the loop condition so we exit the loop.
  - Final output: 1 2 4 8

- Option II
  - We can see by examining the `for` loop that we will execute the loop 4 times: i = 4, 3, 2, 1.
  - Each time through the loop we will print (4 − i) * 2 + 1.
  - (4 − 4) * 2 + 1 = 1
  - (4 − 3) * 2 + 1 = 3
  - (4 − 2) * 2 + 1 = 5
  - (4 − 1) * 2 + 1 = 7
  - Final output: 1 3 5 7
- Option III
  - We can see by examining the `for` loop that we will execute the loop 4 times: i = 0, 1, 2, 3.
  - Each time through the loop we will print `(int)Math.pow(2, i)`. (`Math.pow` returns a double, so we cast to an `int` before printing.)
  - $2^0 = 1$
  - $2^1 = 2$
  - $2^2 = 4$
  - $2^3 = 8$
  - Final output: 1 2 4 8
- Options I and III have the same output.

9. The answer is A.

- The first time through the loop: count = 6.0 and num = 0.
  - Looking at the `if` condition: 6.0 != 0, but 0 / 6 is 0 → false, so we skip the `if` clause and go back to the start of the loop.
- Back to the top of the loop: count is still 6.0, but num = 1.
  - Looking at the `if` condition: 6.0 ! = 0 && 1 / 6.0 > 0 → true, so we execute the `if` clause and count = 6.0 − 1 = 5.0.
- Back to the top of the loop: count = 5.0 and num = 2.
  - Looking at the `if` condition: 5.0 ! = 0 && 1 / 5.0 > 0 → true, so we execute the `if` clause and count = 5.0 − 2 = 3.0.
- Back to the top of the loop: count = 3.0 and num = 3.
  - Looking at the `if` condition: 3.0 ! = 0 && 3 / 3.0 > 0 → true, so we execute the `if` clause and count = 3.0 − 3 = 0.0.
- Back to the top of the loop: count = 0.0 and num = 4.
  - Looking at the `if` condition: Here's where it gets interesting. If we execute 4 / 0.0, we are going to crash our program with an `ArithmeticExceptionError`, but that's not going to happen. The first part of the condition, count != 0 is false, and Java knows that false AND anything is false, so it doesn't even bother to execute the second part of the condition. That is called short-circuiting, and it is often used for exactly this purpose. The condition is false; the `if` clause doesn't get executed.
- Back to the top of the loop: count = 0.0 and num = 5, so our loop is complete and we print the value of count, which is 0.0.

10. The answer is A.

- The only thing we can do is to trace the code carefully line by line until we have the answer.
- n = 0. Print 0 % 4, which is 0
  - 0 % 5 = 0 so we execute the `else`, now n = 3.
- Print 3 % 4, which is 3
  - 3 % 5 = 3 so we execute the `else`, now n = 6.
- Print 6 % 4, which is 2
  - 6 % 5 = 1 so we execute the `else`, now n = 9.
- Print 9 % 4, which is 1
  - 9 % 5 = 1 so we execute the `else`, now n = 12.

- Print 12 % 4, which is 0
  - 12 % 5 = 2 so finally we execute the `if`, now n = 16.
- Print 16 % 4, which is 0
  - 16 % 5 = 1 so we execute the `else`, now n = 19.
- Print 19 % 4, which is 3
  - 19 % 5 is 4 so we execute the `else`, now n = 22, which completes the loop.
- We have printed: 0 3 2 1 0 0 3

11. The answer is D.

- When we enter the loop, tabulate = 100 and repeat = 1
  - tabulate = 100 − Math.pow(1, 2) = 99; increment repeat to 2
- 99 > 20 so we enter the loop again
  - tabulate = 99 − Math.pow(2, 2) = 95; increment repeat to 3
- 95 > 20
  - tabulate = 95 − Math.pow(3, 2) = 86; increment repeat to 4
- 86 > 20
  - tabulate = 86 − Math.pow(4, 2) = 70; increment repeat to 5
- 70 > 20
  - tabulate = 70 − Math.pow(5, 2) = 45; increment repeat to 6
- 45 > 20
  - tabulate = 45 − Math.pow(6, 2) = 9; increment repeat to 7
- 9 < 20 so we exit the loop and print 9

12. The answer is B.

- The value can be found by using the starting and stopping values of the index.
- The inner loop begins at n, ends at i+1, and is incremented by 1.
- So, the number of times will be n – (i+1) – 1.