› Answers and Explanations

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

1. The answer is D.

- This is a `for-each` loop. Read it like this: "For each int (which I will call val) in values. . . ."
- The loop will go through each element of the array values and add it to number. Notice that number starts at 13.

```
13 + 0 + 1 + 2 + 3 + 4 + 5 = 28
```

2. The answer is D.

- values[total] = values[3] = -2. Remember to start counting at 0.
- 3 + -2 = 1, now total = 1.
- values [total] = values[1] = 6.
- 1 + 6 = 7 and that is what is printed.

3. The answer is C.

- Let's lay out the array along with its indices.

index	0	1	2	3	4	5	6	7
contents	7	9	4	1	6	3	8	5

- The first time through the loop, i = 3.
  - values[3–2] = values [1] = 9 Print it.
- Next time through the loop, i = 5.
  - values [5–2] = values [3] = 1 Print it.
- Next time through the loop, i = 7.
  - Remember that even though the last index is 7, the length of the array is 8.
  - values [7–2] = values[5] = 3 Print it.
- We exit the loop having printed "913".

4. The answer is B.

- Segment I checks if the index of the element is even, not the element itself.
- Segment II correctly checks if the element is even.
- Segment III correctly checks if the element is even, but because the increment is +2, it doesn't check each element of the array.

5. The answer is A.

- The loop will shift each of the elements to the right starting at the second to last element.

6. The answer is D.

- On entering the loop, nums = {0, 0, 1, 1, 2, 2, 3, 3} and i = 3. Set nums[4] to 3.
- Now nums = {0, 0, 1, 1, 3, 2, 3, 3} and i = 4. Set nums[5] to 4.
- Now nums = {0, 0, 1, 1, 3, 4, 3, 3} and i = 5. Set nums[6] to 5.
- Now nums = {0, 0, 1, 1, 3, 4, 5, 3} and i = 6. Set nums[7] to 6.
- Now nums = {0, 0, 1, 1, 3, 4, 5, 6} and i = 7. nums.length − 1 = 7, so exit the loop.

7. The answer is E.

- Since the condition arr[0] > arr[k] holds true for each element, element arr[0] must be the largest.

8. The answer is D.

   - The `indexOf` method returns the index of the first occurrence of the string parameter.
   - If the string parameter is not found the value –1 is returned.

9. The answer is C.

   - Since we need to access the index of an element a standard `for` loop must be used, not a `for-each` loop.
   - Only elements that are above `high` should be printed, so the inequality > must be used.

10. Count the number of occurrences.

Algorithm:
Step 1: Initialize a counter to 0
Step 2: Look at each value in the list
Step 3: If the value is less than or equal to lower, increment the counter
Step 4: If the value is greater than or equal to upper, increment the counter
Step 5: Continue until you reach the end of the list
Step 6: Return the counter

Pseudocode:

```
create and initialize a temporary counter variable
for (iterate through all the values in the array)
{
    if (value <= 250)
        add 1 to counter
    else if (value >= 750)
        add 1 to counter
}
return counter
```

Java code:

```
public int countBetween(double [ ] values, double lower, double upper)
{
      int count = 0;
      for (double element : values)
      {
            if (element <= lower)
               count ++;
            else if (element >= upper)
               count ++;
      }
      return count;
}
```

11. Determine the relative strength index of a stock.

Algorithm:
Step 1: Create an array called `temp` that is the same length as the parameter array
Step 2: Look at each of the scores in the array
Step 3: If the score > 70, then set the corresponding element of the temp array to true; otherwise set it to false
Step 4: Continue until you reach the end of the list
Step 5: Return temp

Pseudocode:

```
create a boolean array called temp that is the same length as the parameter array
for (iterate through all the elements in the parameter array)
{
    if (parameter array[index] > 70)
    {
        temp[index] = true
    }
    else
    {
        temp[index] = false;
    }
}
return temp
```

Java code:

```
public boolean[] overpriced(double[] rsiValues)
{
    boolean[] temp = new boolean[rsivalues.length];

    for (int i = 0; i < rsivalues.length; i++)
    {
        if (rsivalues[i] >= 70)
        {
            temp[i] = true;
        }
        else
        {
            temp[i] = false;
        }
    }
    return temp;
}
```

12. Fill an array with randomly chosen even numbers. There are several ways to do this. Here is one.

Algorithm:

Step 1: Create an array called `temp` that is the same length as the parameter array
Step 2: Loop as many times as the length of the parameter array
Step 3: Generate a random number in the appropriate range
Step 4: While the random number is not even, generate a random number in the appropriate range
Step 5: Put the random number in the array
Step 6: Continue until you complete the correct number of iterations
Step 7: Return temp

Pseudocode:

```
create an integer array called temp that is the same length as the parameter array
for (iterate as many times as the length of the parameter array)
{
    create a random number in the interval from 0 to parameter range
    while (the newly created random number is not even)
    {
        generate a new random value and assign it to the random number
    }
    temp[index] = random number
 }
 return temp
```

Java code:

```
public int[] onlyEvens(int arraySize, int range)
{
    int[] evens = new int[arraySize];

    for (int i = 0; i < arraySize; i++)
    {
        int number = (int)(Math.random() * (range + 1);
        while (number % 2 != 0)                    // as long as the number is odd
        {
            number = (int)(Math.random() * (range + 1); // generate a new number
        }
        evens[i] = number;                  // add the even number to the array of evens
    }
    return evens;
}
```