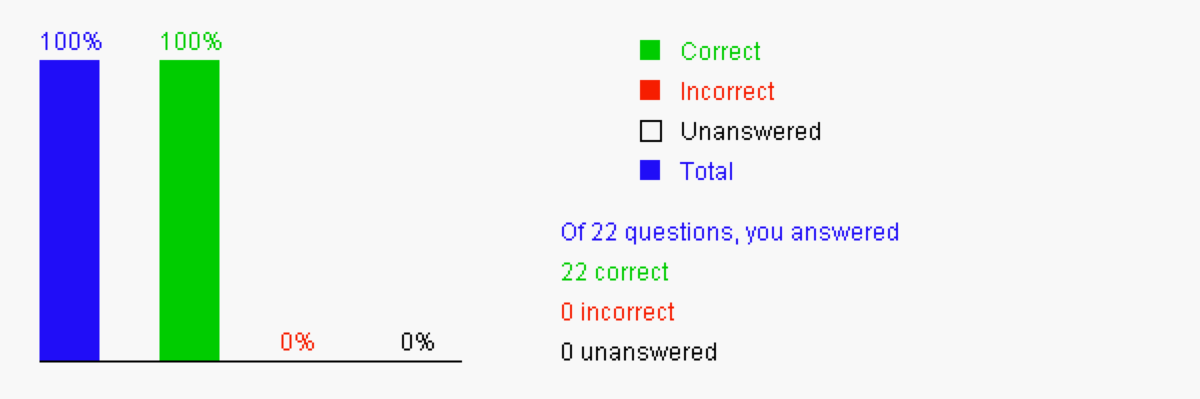
Introduction to Java Programming, Includes Data Structures, Eleventh Edition, Y. Daniel Liang

This quiz is for students to practice. A large number of additional quiz is available for instructors using Quiz Generator from the Instructor's Resource Website. Videos for Java, Python, and C++ can be found at https://yongdanielliang.github.io/revelvideos.html.

Chapter 18 Recursion



Of 22 questions, you answered
22 correct
0 incorrect
0 unanswered

Please send suggestions and errata to Dr. Liang at y.daniel.liang@gmail.com. Indicate which book and edition you are using. Thanks!

Section 18.2 Example: Factorials

18.1 Which of the following statements are true?

☑ A. Every recursive method must have a base case or a stopping condition.
☑ B. Every recursive call reduces the original problem, bringing it increasingly closer to a base case until it becomes that case.
☑ C. Infinite recursion can occur if recursion does not reduce the problem in a manner that allows it to eventually converge into the base case.
☐ D. Every recursive method must have a return value.
☐ E. A recursive method is invoked differently from a non-recursive method.

Your answer is correct ✓

18.2 Fill in the code to complete the following method for computing factorial.

```
/** Return the factorial for a specified index */
public static long factorial(int n) {
  if (n == 0) // Base case
    return 1;
  else
    return _____________; // Recursive call
}
```

☐ A. n * (n - 1)
☐ B. n
☑ C. n * factorial(n - 1)
☑ D. factorial(n - 1) * n

Your answer is correct ✓

18.3 What are the base cases in the following recursive method?

```
public static void xMethod(int n) {
  if (n > 0) {
    System.out.print(n % 10);
    xMethod(n / 10);
  }
}
```

○ A. n > 0
● B. n <= 0
○ C. no base cases
○ D. n < 0

Your answer is correct ✓

18.4 Analyze the following recursive method.

```
public static long factorial(int n) {
  return n * factorial(n - 1);
}
```

○ A. Invoking factorial(0) returns 0.
○ B. Invoking factorial(1) returns 1.

○ C. Invoking factorial(2) returns 2.
○ D. Invoking factorial(3) returns 6.
● E. The method runs infinitely and causes a StackOverflowError.

Your answer is correct ✓

18.5 How many times is the factorial method in Listing 18.1 invoked for factorial(5)?

○ A. 3
○ B. 4
○ C. 5
● D. 6

Your answer is correct ✓

Section 18.3 Example: Fibonacci Numbers

18.6 Which of the following statements are true?

● A. The Fibonacci series begins with 0 and 1, and each subsequent number is the sum of the preceding two numbers in the series.
○ B. The Fibonacci series begins with 1 and 1, and each subsequent number is the sum of the preceding two numbers in the series.
○ C. The Fibonacci series begins with 1 and 2, and each subsequent number is the sum of the preceding two numbers in the series.
○ D. The Fibonacci series begins with 2 and 3, and each subsequent number is the sum of the preceding two numbers in the series.

Your answer is correct ✓

18.7 How many times is the fib method in Listing 18.2 invoked for fib(5)?

○ A. 14
● B. 15
○ C. 25
○ D. 31
○ E. 32

Your answer is correct ✓

Explanation: Hint: number of time fib is invoked in fib(5) = 1 + number of time fib is invoked in fib(3) + number of time fib is invoked in fib(4) = 1 + 5 + 9 = 15

18.8 Fill in the code to complete the following method for computing a Fibonacci number.

```
public static long fib(long index) {
  if (index == 0) // Base case
    return 0;
  else if (index == 1) // Base case
    return 1;
  else // Reduction and recursive calls
    return __________________;
}
```

☐ A. fib(index - 1)
☐ B. fib(index - 2)
☑ C. fib(index - 1) + fib(index - 2)
☑ D. fib(index - 2) + fib(index - 1)

Your answer is correct ✓

Section 18.4 Problem Solving Using Recursion

18.9 In the following method, what is the base case?

```
static int xMethod(int n) {
  if (n == 1)
    return 1;
  else
    return n + xMethod(n - 1);
}
```

● A. n is 1.
○ B. n is greater than 1.
○ C. n is less than 1.
○ D. no base case.

Your answer is correct ✓

18.10 What is the return value for xMethod(4) after calling the following method?

```
static int xMethod(int n) {
  if (n == 1)
    return 1;
  else
    return n + xMethod(n - 1);
}
```

○ A. 12
○ B. 11
● C. 10
○ D. 9

Your answer is correct ✓
Explanation: 4 + 3 + 2 + 1 = 10

18.11 Fill in the code to complete the following method for checking whether a string is a palindrome.

```
public static boolean isPalindrome(String s) {
  if (s.length() <= 1) // Base case
    return true;
  else if _____________________________
    return false;
  else
    return isPalindrome(s.substring(1, s.length() - 1));
}
```

● A. (s.charAt(0) != s.charAt(s.length() - 1)) // Base case
○ B. (s.charAt(0) != s.charAt(s.length())) // Base case
○ C. (s.charAt(1) != s.charAt(s.length() - 1)) // Base case
○ D. (s.charAt(1) != s.charAt(s.length())) // Base case

Your answer is correct ✓

18.12 Analyze the following code:

```
public class Test {
  public static void main(String[] args) {
    int[] x = {1, 2, 3, 4, 5};
    xMethod(x, 5);
  }

  public static void xMethod(int[] x, int length) {
    System.out.print(" " + x[length - 1]);
    xMethod(x, length - 1);
  }
}
```

○ A. The program displays 1 2 3 4 6.
○ B. The program displays 1 2 3 4 5 and then raises an ArrayIndexOutOfBoundsException.
○ C. The program displays 5 4 3 2 1.
● D. The program displays 5 4 3 2 1 and then raises an ArrayIndexOutOfBoundsException.

Your answer is correct ✓
Explanation: xMethod(x, 5) is invoked, then xMethod(x, 4), xMethod(x, 3), xMethod(x, 2), xMethod(x, 1), xMethod(x, 0). When invoking xMethod(x, 0), a runtime exception is raised because System.out.print(' '+x[0-1]) causes array out of bound.

Section 18.5 Recursive Helper Methods

18.13 Fill in the code to complete the following method for checking whether a string is a palindrome.

```
public static boolean isPalindrome(String s) {
  return isPalindrome(s, 0, s.length() - 1);
}

public static boolean isPalindrome(String s, int low, int high) {
  if (high <= low) // Base case
    return true;
  else if (s.charAt(low) != s.charAt(high)) // Base case
    return false;
  else
    return _______________________________;
}
```

○ A. isPalindrome(s)
○ B. isPalindrome(s, low, high)
○ C. isPalindrome(s, low + 1, high)
○ D. isPalindrome(s, low, high - 1)

● E. isPalindrome(s, low + 1, high - 1)

Your answer is correct ✓

18.14 Fill in the code to complete the following method for sorting a list.

```
public static void sort(double[] list) {
  ___________________________;
}

public static void sort(double[] list, int high) {
  if (high > 1) {
    // Find the largest number and its index
    int indexOfMax = 0;
    double max = list[0];
    for (int i = 1; i <= high; i++) {
      if (list[i] > max) {
        max = list[i];
        indexOfMax = i;
      }
    }

    // Swap the largest with the last number in the list
    list[indexOfMax] = list[high];
    list[high] = max;

    // Sort the remaining list
    sort(list, high - 1);
  }
}
```

○ A. sort(list)

○ B. sort(list, list.length)

● C. sort(list, list.length - 1)

○ D. sort(list, list.length - 2)

Your answer is correct ✓

18.15 Fill in the code to complete the following method for binary search.

```
public static int recursiveBinarySearch(int[] list, int key) {
  int low = 0;
  int high = list.length - 1;
  return ___________________________;
}

public static int recursiveBinarySearch(int[] list, int key,
    int low, int high) {
  if (low > high) // The list has been exhausted without a match
    return -low - 1; // Return -insertion point - 1

  int mid = (low + high) / 2;
  if (key < list[mid])
    return recursiveBinarySearch(list, key, low, mid - 1);
  else if (key == list[mid])
    return mid;
  else
    return recursiveBinarySearch(list, key, mid + 1, high);
}
```

○ A. recursiveBinarySearch(list, key)

○ B. recursiveBinarySearch(list, key, low + 1, high - 1)

○ C. recursiveBinarySearch(list, key, low - 1, high + 1)

● D. recursiveBinarySearch(list, key, low, high)

Your answer is correct ✓

Section 18.7 Tower of Hanoi

18.16 How many times is the recursive moveDisks method invoked for 3 disks?

○ A. 3

● B. 7

○ C. 10

○ D. 14

Your answer is correct ✓

18.17 How many times is the recursive moveDisks method invoked for 4 disks?

○ A. 5

○ B. 10

● C. 15

○ D. 20

Your answer is correct ✓

18.18 Analyze the following two programs:

```
A:

public class Test {
  public static void main(String[] args) {
    xMethod(5);
  }

  public static void xMethod(int length) {
    if (length > 1) {
      System.out.print((length - 1) + " ");
      xMethod(length - 1);
    }
  }
}

B:
public class Test {
  public static void main(String[] args) {
    xMethod(5);
  }

  public static void xMethod(int length) {
    while (length > 1) {
      System.out.print((length - 1) + " ");
      xMethod(length - 1);
    }
  }
}
```

○ A. The two programs produce the same output 5 4 3 2 1.

○ B. The two programs produce the same output 1 2 3 4 5.

○ C. The two programs produce the same output 4 3 2 1.

○ D. The two programs produce the same output 1 2 3 4.

● E. Program A produces the output 4 3 2 1 and Program B prints 4 3 2 1 1 1 .... 1 infinitely.

Your answer is correct ✓

Explanation: In Program B, xmethod(5) invokes xmethod(4), xmethod(4) invokes xmethod(3), xmethod(3) invokes xmethod(2), xmethod(2) invokes xmethod(1), xmethod(1) returns control to xmethod(2), xmethod(2) invokes xmethod(1) because of the while loop. This continues infinitely.

Section 18.8 Case Study: Fractals

18.19 In LiveExample 18.9, to draw three smaller triangles recursively, the program invokes:

☑ A. displayTriangles(order - 1, p1, p12, p31);

☑ B. displayTriangles(order - 1, p12, p2, p23);

☑ C. displayTriangles(order - 1, p31, p23, p3);

□ D. displayTriangles(order - 1, p12, p23, p31);

Your answer is correct ✓

Explanation: See LiveExample 18.9.

Section 18.9 Recursion versus Iteration

18.20 Which of the following statements are true?

□ A. Recursive methods run faster than non-recursive methods.

☑ B. Recursive methods usually take more memory space than non-recursive methods.

☑ C. A recursive method can always be replaced by a non-recursive method.

☑ D. In some cases, however, using recursion enables you to give a natural, straightforward, simple solution to a program that would otherwise be difficult to solve.

Your answer is correct ✓

Section 18.10 Tail Recursion

18.21 Analyze the following functions;

```
public class Test1 {
  public static void main(String[] args) {
    System.out.println(f1(3));
    System.out.println(f2(3, 0));
  }

  public static int f1(int n) {
    if (n == 0)
      return 0;
    else {
      return n + f1(n - 1);
    }
  }

  public static int f2(int n, int result) {
    if (n == 0)
      return result;
    else
      return f2(n - 1, n + result);
  }
}
```

○ A. f1 is tail recursion, but f2 is not

● B. f2 is tail recursion, but f1 is not

○ C. f1 and f2 are both tail recursive

○ D. Neither f1 nor f2 is tail recursive

Your answer is correct ✓

18.22 Show the output of the following code

```
public class Test1 {
  public static void main(String[] args) {
    System.out.println(f2(2, 0));
  }

  public static int f2(int n, int result) {
    if (n == 0)
      return 0;
    else
      return f2(n - 1, n + result);
  }
}
```

● A. 0

○ B. 1

○ C. 2

○ D. 3

Your answer is correct ✓