Section 18.2 Example: Factorials

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.

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

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

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.

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

a. 3

b. 4

c. 5

d. 6

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.

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

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)

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.

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

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

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.

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)

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)

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)

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

a. 3

b. 7

c. 10

d. 14

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

a. 5

b. 10

c. 15

d. 20

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.

19. The following program draws squares recursively. Fill in the missing code.

```
import javax.swing.*;
import java.awt.*;

public class Test extends JApplet {
```

```
 public Test() {
  add(new SquarePanel());
 }

 static class SquarePanel extends JPanel {
  public void paintComponent(Graphics g) {
   super.paintComponent(g);

   int width = (int)(Math.min(getWidth(), getHeight()) * 0.4);
   int centerx = getWidth() / 2;
   int centery = getHeight() / 2;

   displaySquares(g, width, centerx, centery);
  }

  private static void displaySquares(Graphics g, int width,
    int centerx, int centery) {
   if (width >= 20) {
    g.drawRect(centerx - width, centery - width, 2* width, 2 * width);
    displaySquares(__________, width - 20, centerx, centery);
   }
  }
 }
}
```

a. getGraphics()

b. newGraphics()

c. null

d. g

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.

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

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