**1. What is a recursive method?**

A recursive method is the one that call itself.

**Describe the characteristics of recursive methods.**

One or more base cases (the simplest case) are used to stop recursion.

Every recursive call reduces the original problem,

brining it increasingly close to base case until it becomes that case.

**2. What is a cause of a stack-overflow exception?**

When a method is invoked, its content are placed into a stack,

if the method is recursively invoked, it is possible that the stack space is exhausted.

This cases is stack overflow.

**3. Which of the following statements are true?**

a. Any non-recursive method can be converted into a recursive method. **True.**

b. Non-recursive methods take more resources to execute than recursive methods. **False**

c. Recursive methods without selection statement lead to infinite recursion. **True.**

d. Recursive definition does not guarantee that a recursive code is the best way to solve a problem. **False.**

**4. Consider ComputeFactorial.java program in the main textbook, how many times is the factorial method invoked for factorial(5)? What is the base case?**

Six times (base case factorial(0)). Base case is n = 0.

**5. Consider ComputeFibonacci.java program in the main textbook, how many times is the fibmethod invoked for fib(3)? What are the base cases?**

Five times. Base cases are index = 0 and index =1

**6. Consider RecursivePalindromeUsingSubstring.java program in the main textbook, how many times is the isPalindrome method invoked for isPalindrome(abdxcxdba)? What are the base cases?**

Five times.  
Base cases are s.length() <= 1 and s.charAt(0) != s.charAt(s.length() - 1)

**7. Consider TowersOfHanoi.java program in the main textbook, how many times is the moveDisks method invoked for moveDisks(5, 'A', 'B', 'C')?**

2^5 - 1 = 31 times.

### . Show the output of the following programs. Also, identify base cases and recursive calls.

public class Test {

public static void main(String[] args) {

System.out.println("GCD is " + xMethod(48, 18));

}

public static int xMethod(int n1, int n2) {

int remainder;

remainder = n1 % n2;

if (remainder == 0)

return n2;

else

return xMethod(n2, remainder);

}

}

Answer: GCD is 6

public class Test {

public static void main(String[] args) {

System.out.println("Exponent is " + xMethod(3, 4));

}

public static long xMethod(int base, int exponent) {

if (exponent == 0)

return 1;

else

return base \* xMethod(base, exponent - 1);

}

}

Answer: Exponent is 81

## 9. What is wrong in the following recursive solution?

public class Test {

public static void main(String[] args) {

System.out.println("Factorial = " + factorial(4.0));

}

public static long factorial(double n) {

if (n == 0)

return 1;

else

return n \* factorial(n - 1);

}

}

The return type of factorial should be double to prevent loss of precision.

public class Test {

public static void main(String[] args) {

System.out.println("Exponent is " + power(3, 4));

}

public static long power (int base, int exponent) {

return base \* power(base, exponent - 1);

}

}

The base case of power is missing.

### 10. Write a recursive definition for printing numbers from n to 0.

public class Test {

public static void main(String[] args) {

printNumbers(5); // Example

}

// Solution 1

public static void printNumbers (int n) {

if(n == 0) {

System.out.println(0);

} else {

System.out.println(n);

printNumbers(n-1);

}

}

// Solution 2

public static void printNumbers (int n) {

System.out.println(n);

if(n > 0) {

System.out.println(n-4)

}

}

}

### 11. Write a recursive definition for printing numbers from 0 to n.

public class Test {

public static void main(String[] args) {

printNumbers(5); // Example

}

// Solution 1

public static void printNumbers (int n) {

if(n == 0) {

System.out.println(end);

} else {

printNumbers(n - 1);

System.out.println(start);

}

}

// Solution 2

// Solution 2

public static void printNumbers (int n) {

if(n > 0) {

System.out.println(n-4)

}

System.out.println(n);

}

}

### 12. Write a recursive mathematical definition for computing the sum of the squares of the first n positive integers.

public class Test {

public static void main(String[] args) {

System.out.println(sumOfSquares(5)); // Example

}

public static int sumOfSquares (int n) {

if(n == 1) {

return 1;

} else {

return (n \* n) + sumOfSquares(n-1);

}

}

}

### 13. Write a recursive mathematical definition for computing the power xn for a positive integer n and a real number x.

public class Test {

public static void main(String[] args) {

System.out.println(power(5, 2)); // Should display 25

}

public static int power (int base, int exponent) {

if(exponent == 0) {

return 1;

} else {

return base \* power(base, exponent - 1);

}

}

}

## 14. Write a recursive definition, public static void binaryPrint(int x) to print out the non-negative integer x as a binary number. For example, if the value of x were 19 then the binary output should be 10011.

public class Test {

public static void main(String[] args) {

binaryOf(10); // Should display 25

}

public static int binaryOf (int n) {

int divided = n / 2;

int remainder = n % 2;

binaryOf(divided);

if(divided > 0) {

System.out.println(remainder);

}

}

}