Practice Questions - Recursion

Q1. Define the terms

a. Recursion
b. Iteration
c. Infinite recursion

Q2. Outline, but do not implement, a recursive solution for finding the smallest value in an array.

Q3. Outline, but do not implement, a recursive solution for sorting an array of numbers.
Hint: First find the smallest value in the array.

Q4. Outline, but do not implement, a recursive solution for generating all subsets of the set {1, 2, . . . , n}.

Q5. Write a recursive definition of xn, where n ≥ 0, similar to the recursive definition of the Fibonacci numbers.
Hint: How do you compute xn from xn – 1? How does the recursion terminate?

Q6. Write a recursive definition of n! = 1 × 2 × . . . × n, similar to the recursive definition of the Fibonacci numbers.

Q7. Write a recursive method void reverse() that reverses a sentence. For example:

```
Sentence greeting = new Sentence("Hello!");
greeting.reverse();
System.out.println(greeting.getText());
prints the string "!olleH".
```

Implement a recursive solution by removing the first character, reversing a sentence consisting of the remaining text, and combining the two.

Q8. Use recursion to implement a method int indexOf(String t) that returns the starting position of the first substring of the text that matches t. Return –1 if t is not a substring of s. For example,

```
Sentence s = new Sentence("Mississippi!");
int n = s.indexOf("sip"); // Returns 6
```

Hint: This is a bit trickier than the preceding problem, because you must keep track of how far the match is from the beginning of the sentence. Make that value a parameter of a helper method.

Q9. Using recursion, compute the sum of all values in an array.

Q10. What is required to make a recursive method successful?

I special cases that handle the simplest computations directly
II a recursive call to simplify the computation
III a mutual recursion

a) I
b) II
c) I and II
d) I, II, and III

Q11. Consider the following code snippet for recursive addition:

```
int add(int i, int j)
{
   // assumes i >= 0
   if (i == 0)
   {
      return j;
   }
   else
   {
      return add(i - 1, j + 1);
   }
}
```

Identify the terminating condition in this recursive method.

a) `if (i == 0)`
b) `return j`
c) `return add(i - 1, j + 1)`
d) there is no terminating condition

Q12. Consider the following code snippet for calculating Fibonacci numbers recursively:

```
int fib(int n)
{
   // assumes n >= 0
   if (n <= 1)
   {
      return n;
   }
   else
   {
      return (fib(n - 1) + fib(n - 2));
   }
```

```
}
```

Identify the terminating condition.

a) `n < 1`
b) `n <= 1`
c) `fib(n - 1)`
d) `fib(n - 1) + fib(n - 1)`

Q13. Consider the following recursive code snippet:

```
public static int mystery(int n, int m)
{
   if (n <= 0)
   {
      return 0;
   }
   if (n == 1)
   {
      return m;
   }
   return m + mystery(n - 1, m);
}
```

Identify the terminating condition(s) of method `mystery`?

a) `n <= 0`
b) `n == 1`
c) `n <= 0 or n == 1`
d) `n > 0`

Q14. Consider the following recursive code snippet:

```
public int mystery(int n, int m)
{
   if (n == 0)
   {
      return 0;
   }
   if (n == 1)
   {
      return m;
   }
   return m + mystery(n - 1, m);
}
```

What value is returned from a call to `mystery(1,5)`?

a) 1

b) 5
c) 6
d) 11

Q15. Consider the following recursive code snippet:

```
public int mystery(int n, int m)
{
   if (n == 0)
   {
      return 0;
   }
   if (n == 1)
   {
      return m;
   }
   return m + mystery(n - 1, m);
}
```

What value is returned from a call to `mystery(3,6)`?

a) 3
b) 6
c) 18
d) 729

Q16. Consider the recursive method `myPrint`:

```
public void myPrint(int n)
{
   if (n < 10)
   {
      System.out.print(n);
   }
   else
   {
      int m = n % 10;
      System.out.print(m);
      myPrint(n / 10);
   }
}
```

What is printed for the call `myPrint(8)`?

a) 10
b) 8
c) 4

d) 21

Q16. Consider the recursive method `myPrint` shown in this code snippet:

```
public void myPrint(int n)
{
   if (n < 10)
   {
      System.out.print(n);
   }
   else
   {
      int m = n % 10;
      System.out.print(m);
      myPrint(n / 10);
   }
}
```

What does this method do?

a) Prints a positive `int` value forward, digit by digit.
b) Prints a positive `int` value backward, digit by digit.
c) Divides the `int` by 10 and prints out its last digit.
d) Divides the `int` by 10 and prints out the result.

Q17. Complete the code for the recursive method `printSum` shown in this code snippet, which is intended to return the sum of digits from 1 to n:

```
public static int printSum(int n)
{
   if (n == 0)
   {
      return 0;
   }
   else
   {
      _____________________________
   }
}
```

a) `return (printSum(n - 1));`
b) `return (n + printSum(n + 1));`
c) `return (n + printSum(n - 1));`
d) `return (n - printSum(n - 1));`

Q18. Consider the code for the recursive method `myPrint` shown in this code snippet:

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

To avoid infinite recursion, which of the following lines of code should replace the current terminating case?

a) `if (n == -1)`
b) `if (n <= 0)`
c) `if (n >= 0)`
d) The terminating case as shown will avoid infinite recursion.

Q19. Consider the method `powerOfTwo` shown below:

```
public boolean powerOfTwo(int n)
{
   if (n == 1)   // line #1
   {
      return true;
   }
   else if (n % 2 == 1) // line #2
   {
      return false;
   }
   else
   {
      return powerOfTwo(n / 2);  // line #3
   }
}
```

How many recursive calls are made from the original call `powerOfTwo(63)` (not including the original call)?

a) 6
b) 4
c) 1
d) 0

Q20. Complete the code for the `calcPower` recursive method shown below, which is intended to raise the base number passed into the method to the exponent power passed into the method:

```
public static int calcPower(int baseNum, int exponent)
{
   int answer = 0;
   ____________________________
   {
      answer = 1;
   }
   else
   {
      answer = baseNum * calcPower (baseNum, exponent - 1);
   }
   return answer;
}
```

a) `if (exponent == 0)`
b) `if (exponent == 1)`
c) `if (exponent == -1)`
d) `if (exponent != 1)`