Course Title: Big Java, Early Objects

Chapter Number: 13 Recursion

Question type: Multiple Choice

1) 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

Answer: c

Title: What is required in a recursive method?

Difficulty: Easy

Section Reference 1: 13.1 Triangle Numbers Revisited

2) Consider the getArea method from the textbook shown below.

public int getArea()

{

if (width <= 0) { return 0; } // line #1

else if (width == 1) { return 1; } // line #2

else

{

Triangle smallerTriangle = new Triangle(width – 1); // line #3

int smallerArea = smallerTriangle.getArea(); // line #4

return smallerArea + width; // line #5

}

}

Where is/are the recursive call(s)?

a) line #1

b) line #2

c) lines #1 and #2

d) line #4

Answer: d

Title: Where is the recursive call?

Difficulty: Medium

Section Reference 1: 13.1 Triangle Numbers Revisited

3) Consider the getArea method from the textbook shown below.

public int getArea()

{

if (width <= 0) { return 0; } // line #1

else if (width == 1) { return 1; } // line #2

else

{

Triangle smallerTriangle = new Triangle(width – 1); // line #3

int smallerArea = smallerTriangle.getArea(); // line #4

return smallerArea + width; // line #5

}

}

Where is/are the terminating condition(s)?

a) line #1

b) line #2

c) lines #1 and #2

d) line #4

Answer: c

Title: Where is/are the terminating condition(s)?

Difficulty: Medium

Section Reference 1: 13.1 Triangle Numbers Revisited

4) Consider the getArea method from the textbook shown below:

public int getArea()

{

if (width <= 0) { return 0; } // line #1

else if (width == 1) { return 1; } // line #2

else

{

Triangle smallerTriangle = new Triangle(width – 1); // line #3

int smallerArea = smallerTriangle.getArea(); // line #4

return smallerArea + width; // line #5

}

}

Assume the code in line #3 is changed to:

Triangle smallerTriangle = new Triangle(width);

This change would cause infinite recursion for which triangles?

a) Those with width equal to 0.

b) Those with width equal to 1.

c) Those with width greater than or equal to 2.

d) Triangles of any width.

Answer: c

Title: Changing line #3 code to Triangle smallerTriangle = new Triangle(width) would cause infinite recursion for which triangles?

Difficulty: Hard

Section Reference 1: 13.1 Triangle Numbers Revisited

5) Consider the getArea method from the book shown below.

public int getArea()

{

if (width <= 0) { return 0; } // line #1

else if (width == 1) { return 1; } // line #2

else

{

Triangle smallerTriangle = new Triangle(width – 1); // line #3

int smallerArea = smallerTriangle.getArea(); // line #4

return smallerArea + width; // line #5

}

}

Assume lines #1 and #2 were changed to this:

if (width == 1) { return 1; } // new line #1-2

What will happen when this code is executed?

a) A negative or zero width would cause problems.

b) Nothing - the method would still be correct.

c) We would lose our only recursive case.

d) A positive width would reduce the correct area by 1.

Answer: a

Title: What behavior would eliminating line 1 of the getArea method would result in?

Difficulty: Hard

Section Reference 1: 13.1 Triangle Numbers Revisited

6) Consider the getArea method from the textbook shown below.

public int getArea()

{

if (width <= 0) { return 0; } // line #1

else if (width == 1) { return 1; } // line #2

else

{

Triangle smallerTriangle = new Triangle(width – 1); // line #3

int smallerArea = smallerTriangle.getArea(); // line #4

return smallerArea + width; // line #5

}

}

Assume line #1 is replaced with this line:

if (width <= 0) {return width;}

What will be the result?

a) The method will still return correct results for all non-negative width triangles.

b) The method will return incorrect results for triangles with width equal to 0.

c) The method will return incorrect results for triangles with width equal to 1.

d) The method will return area to be one too high for all triangles.

Answer: a

Title: What would replacing line #1 with if (width <= 0) return width do?

Difficulty: Hard

Section Reference 1: 13.1 Triangle Numbers Revisited

7) 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

Answer: a

Title: Identify the terminating condition in code for recursive addition.

Difficulty: Easy

Section Reference 1: 13.1 Triangle Numbers Revisited

8) 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)

Answer: b

Title: Identify the terminating condition in code for Fibonacci.

Difficulty: Easy

Section Reference 1: 13.1 Triangle Numbers Revisited

9) 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

Answer: c

Title: Identify the terminating condition(s) of method mystery

Difficulty: Easy

Section Reference 1: 13.1 Triangle Numbers Revisited

10) 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

Answer: b

Title: Find the return value from the call to mystery(1,5).

Difficulty: Medium

Section Reference 1: 13.1 Triangle Numbers Revisited

11) 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

Answer: c

Title: Find the return value from the call to mystery(3,6).

Difficulty: Medium

Section Reference 1: 13.1 Triangle Numbers Revisited

12) 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

Answer: b

Title: What is printed for the call myPrint(8)?

Difficulty: Medium

Section Reference 1: 13.1 Triangle Numbers Revisited

13) Consider the recursive method myPrint 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 is printed for the call myPrint(821)?

a) 821

b) 128

c) 12

d) 10

Answer: b

Title: What is printed for the call myPrint(821)?

Difficulty: Medium

Section Reference 1: 13.1 Triangle Numbers Revisited

14) 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.

Answer: b

Title: What does the recursive method myPrint do?

Difficulty: Medium

Section Reference 1: 13.1 Triangle Numbers Revisited

15) 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));

Answer: c

Title: Complete the code for the recursive method printSum

Difficulty: Medium

Section Reference 1: 13.1 Triangle Numbers Revisited

16) Consider the code for the recursive method mysteryPrint shown in this code snippet:

public static int mysteryPrint(int n)

{

if (n == 0)

{

return 0;

}

else

{

return (n + mysteryPrint(n-1));

}

}

What will be printed with a call to mysteryPrint(-4)?

a) 0

b) -10

c) -22

d) Nothing – a StackoverflowError exception will occur

Answer: d

Title: Complete the code for the recursive method mysteryPrint

Difficulty: Medium

Section Reference 1: 13.1 Triangle Numbers Revisited

17) 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.

Answer: b

Title: Complete the code for the recursive method myPrint

Difficulty: Medium

Section Reference 1: 13.1 Triangle Numbers Revisited

18) Consider the getArea method from the textbook shown below:

public int getArea()

{

if (width <= 0) { return 0; } // line #1

if (width == 1) { return 1; } // line #2

Triangle smallerTriangle = new Triangle(width – 1); // line #3

int smallerArea = smallerTriangle.getArea(); // line #4

return smallerArea + width; // line #5

}

Which line has the recursive case?

a) line #1

b) line #2

c) line #3

d) line #4

Answer: d

Title: Which line in the getArea method has the recursive case?

Difficulty: Easy

Section Reference 1: 13.1 Triangle Numbers Revisited

19) Consider the recursive method shown below:

public static int strangeCalc(int bottom, int top)

{

if (bottom > top)

{

return -1;

}

else if (bottom == top)

{

return 1;

}

else

{

return bottom \* strangeCalc(bottom + 1, top);

}

}

What value will be returned with a call to strangeCalc(4,7)?

a) 1

b) -1

c) 120

d) 840

Answer: c

Title: Which value is returned with a call to strangeCalc(4,7)?

Difficulty: Medium

Section Reference 1: 13.1 Triangle Numbers Revisited

20) Consider the recursive method shown below:

public static int strangeCalc(int bottom, int top)

{

if (bottom > top)

{

return -1;

}

else if (bottom == top)

{

return 1;

}

else

{

return bottom \* strangeCalc(bottom + 1, top);

}

}

What value will be returned with a call to strangeCalc(2,3)?

a) 1

b) 2

c) 6

d) 24

Answer: b

Title: Which value is returned with a call to strangeCalc(2,3)?

Difficulty: Medium

Section Reference 1: 13.1 Triangle Numbers Revisited

21) Insert the missing code in the following code fragment. This fragment is intended to recursively computexn, where x and n are both non-negative integers:

public int power(int x, int n)

{

if (n == 0)

{

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

}

else

{

return x \* power(x, n - 1);

}

}

a) return 1;

b) return x;

c) return power(x, n - 1);

d) return x \* power(x, n - 1);

Answer: a

Title: Complete this code to recursively compute x to the power n.

Difficulty: Medium

Section Reference 1: 13.1 Triangle Numbers Revisited

Section Reference 2: Common Error 13.1 Infinite Recursion

22) If recursion does not have a special terminating case, what error will occur?

a) Index out of range

b) Illegal argument

c) Stack overflow

d) Out of memory

Answer: c

Title: Which error occurs if recursion does not terminate?

Difficulty: Easy

Section Reference 1: 13.1 Triangle Numbers Revisited

Section Reference 2: Common Error 13.1 Infinite Recursion

23) A recursive method without a special terminating case would \_\_\_\_\_\_\_\_\_

a) end immediately.

b) not be recursive.

c) be more efficient.

d) never terminate.

Answer: d

Title: What happens when a recursive method doesn’t have a terminating case?

Difficulty: Easy

Section Reference 1: 13.1 Triangle Numbers Revisited

Section Reference 2: Common Error 13.1 Infinite Recursion

24) 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 parameter values for n would cause an infinite recursion problem in the following method?

a) n == 0

b) n == 1

c) all n with n < 0

d) all n with n >= 0

Answer: c

Title: What parameter values would cause an infinite recursion problem for method mystery?

Difficulty: Medium

Section Reference 1: 13.1 Triangle Numbers Revisited

Section Reference 2: Common Error 13.1 Infinite Recursion

25) \_\_\_\_ recursion can occur when a recursive algorithm does not contain a special case to handle the simplest computations directly.

a) Mutual

b) Non-mutual

c) Terminating condition

d) Infinite

Answer: d

Title: \_\_ recursion can occur when a recursive algorithm has no terminating case.

Difficulty: Easy

Section Reference 1: 13.1 Triangle Numbers Revisited

Section Reference 2: Common Error 13.1 Infinite Recursion

26) If a recursive method does not simplify the computation within the method and the base case is not called, what will be the result?

a) The terminating condition will be executed and recursion will end.

b) The recursion calculation will occur correctly regardless.

c) This cannot be determined.

d) Infinite recursion will occur.

Answer: d

Title: What happens if a recursive method does not simplify the computation within the method?

Difficulty: Easy

Section Reference 1: 13.1 Triangle Numbers Revisited

Section Reference 2: Common Error 13.1 Infinite Recursion

27) Consider the getArea method from the textbook shown below:

public int getArea()

{

if (width <= 0) { return 0; } // line #1

Triangle smallerTriangle = new Triangle(width – 1); // line #2

int smallerArea = smallerTriangle.getArea(); // line #3

return smallerArea + width; // line #4

}

If line#1 was removed, what would be the result?

a) The recursive method would cause an exception for values below 0.

b) The recursive method would construct triangles whose width was negative.

c) The recursive method would terminate when the width reached 0.

d) The recursive method would correctly calculate the area of the original triangle.

Answer: b

Title: If line#1 in the getArea method was removed, what would be the result?

Difficulty: Easy

Section Reference 1: 13.1 Triangle Numbers Revisited

Section Reference 2: Common Error 13.1 Infinite Recursion

28) When a recursive method is called, and it does not perform recursion, what must be true?

a) The terminating condition was true.

b) One recursive case condition was true.

c) All recursive case conditions were true.

d) An exception will occur in the method.

Answer: a

Title: Why would a recursive method not perform recursion?

Difficulty: Easy

Section Reference 1: How To 13.1

29) Consider the getArea method from the textbook shown below.

public int getArea()

{

if (width <= 0) { return 0; } // line #1

if (width == 1) { return 1; } // line #2

Triangle smallerTriangle = new Triangle(width – 1); // line #3

int smallerArea = smallerTriangle.getArea(); // line #4

return smallerArea + width; // line #5

}

Assume that line #2 is changed to this:

if (width == 1) { return 2; }

How would this affect calls to getArea?

a) It would add 1 to all calls except those where width <= 0.

b) It would make no difference to any call.

c) It would double every triangle area.

d) It would subtract 1 from all calls.

Answer: a

Title: How would changing line #2 affect calls to getArea?

Difficulty: Medium

Section Reference 1: How To 13.1

30) Consider the getArea method from the textbook shown below:

public int getArea()

{

if (width <= 0) { return 0; } // line #1

if (width == 1) { return 1; } // line #2

Triangle smallerTriangle = new Triangle(width – 1); // line #3

int smallerArea = smallerTriangle.getArea(); // line #4

return smallerArea + width; // line #5

}

Assume that line #3 is changed to this:

Triangle smallerTriangle = new Triangle(width);

This would cause infinite recursion for \_\_\_\_.

a) triangles with width equal to 0

b) triangles with width equal to 1

c) triangles with width greater than or equal to 2

d) triangles of any width

Answer: c

Title: Changing line #3 would cause infinite recursion for which triangles?

Difficulty: Medium

Section Reference 1: How To 13.1

31) Consider the getArea method from the textbook shown below:

public int getArea()

{

if (width <= 0) { return 0; } // line #1

if (width == 1) { return 1; } // line #2

Triangle smallerTriangle = new Triangle(width – 1); // line #3

int smallerArea = smallerTriangle.getArea(); // line #4

return smallerArea + width; // line #5

}

Assume line #3 is changed to this:

Triangle smallerTriangle = new Triangle(width + 1)

When calling the getArea method on a Triangle object with width = 4, what result will be produced?

a) area for all triangles will be computed to be too high

b) area for all triangles will be computed to be too low

c) area will only be incorrect for a triangle objects with width = 1

d) infinite recursion will occur for triangle objects with width >= 2

Answer: d

Title: Changing line #3 and calling the getArea method will produce what result?

Difficulty: Medium

Section Reference 1: How To 13.1

32) Consider the getArea method from the textbook shown below:

public int getArea()

{

if (width <= 0) { return 0; } // line #1

if (width == 1) { return 1; } // line #2

Triangle smallerTriangle = new Triangle(width – 1); // line #3

int smallerArea = smallerTriangle.getArea(); // line #4

return smallerArea + width; // line #5

}

If line #1 was eliminated from the method, what would be the result when calling getArea?

a) A negative or zero width would cause problems.

b) Nothing - the method would still work correctly.

c) We would lose our only recursive case.

d) A positive width would reduce the correct area by 1.

Answer: a

Title: What would be the result of eliminating line 1 of the getArea method?

Difficulty: Easy

Section Reference 1: How To 13.1

33) How many recursive calls to the fib method shown below would be made from an original call to fib(4)? (Do not count the original call)

public int fib(int n)

{ // assumes n >= 0

if (n <= 1)

{

return n

}

else

{

return (fib(n - 1) + fib(n - 2));

}

}

a) 1

b) 2

c) 4

d) 8

Answer: d

Title: Count the recursive calls to fib.

Difficulty: Medium

Section Reference 1: How To 13.1

34) Would switching the special case order affect the return value of the following method?

public int mystery(int n, int m)

{

if (n == 0) // special case #1

{

return 0;

}

if (n == 1) // special case #2

{

return m;

}

return m + mystery(n - 1, m);

}

a) No

b) Yes

c) It is impossible to tell.

d) An exception will be thrown.

Answer: a

Title: Would switching the special case order affect the return value of method what?

Difficulty: Easy

Section Reference 1: How To 13.1

35) Which of the following options could be used as a terminating condition for a recursive method that finds the middle character of a String with any number of characters?

I the length of the String is 1

II first and last String characters match

III the String is not empty

a) I

b) II

c) I, II and III

d) I and III

Answer: a

Title: Identifying possible terminating conditions.

Difficulty: Medium

Section Reference 1: How To 13.1

36) Consider the problem of arranging matchsticks so as to form a row of rectangles, as shown below.

\_ \_ \_ \_ \_ \_

|\_ \_|\_ \_|\_ \_|

Complete the recursive method below, which is designed to return the number of matchsticks needed to form n rectangles.

public static int matchsticks(int rectangles)

{

if (rectangles == 1) // 1 square can be formed with 6 matchsticks

{

return 6;

}

else

{

return \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

}

}

a) 6 + matchsticks(rectangles – 1);

b) 5 + matchsticks(rectangles – 1);

c) 6 \* rectangles;

d) matchsticks(rectangles + 6);

Answer: b

Title: Complete recursive method to compute number of matchsticks needed

Difficulty: Medium

Section Reference 1: 13.1 Triangle Numbers

37) Consider the method below, which implements the exponentiation operation recursively. Select the statement that should be used to complete the method, so that it handles the special case correctly.

public static double power(int base, int exponent)

{

if (exponent == 0)

{

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

}

else

{

reurn base \* power(base, exponent – 1);

}

}

a) return 1;

b) return base;

c) return 0;

d) return 1 \* power(base, exponent – 1);

Answer: a

Title: Complete recursive exponentiation method with special case

Difficulty: Easy

Section Reference 1: 13.1 Triangle Numbers

38) Consider the method below, which displays the characters from a String in reverse order. Each character appears on a separate line. Select the statement that should be used to complete the method, so that it performs a recursive method call correctly.

public static void printReverse(String word)

{

if (word.length() > 0)

{

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

System.out.println(word.charAt(0));

}

}

a) printReverse(word);

b) printReverse(new String(word.charAt(1)));

c) printReverse(word.substring(1));

d) printReverse(word.length() - 1);

Answer: c

Title: Complete recursive method

Difficulty: Medium

Section Reference 1: 13.1 Triangle Numbers

39) Consider the method below, which prints the digits of an arbitrary integer in reverse order, one digit per line. The method should print the last digit first. Then, it should recursively print the integer obtained by removing the last digit. Select the statements that should be used to complete the method.

public static void printReverse(int value)

{

if (value > 0)

{

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ // print last digit

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ // recursive call to print value without last digit

}

}

a)

System.out.println(value / 10);

printReverse(value / 10);

b)

System.out.println(value % 10);

printReverse(value / 10);

c)

System.out.println(value / 10);

printReverse(value % 10);

d)

System.out.println(value % 10);

printReverse(value % 10);

Answer: b

Title: Select statements to complete recursive method to print digits in reverse

Difficulty: Medium

Section Reference 1: 13.1 Triangle Numbers

40) 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

}

}

What is the best interpretation of line #1?

a) One is a power of two

b) One is not a power of two

c) Any multiple of one is a power of two

d) 1 is an invalid choice for n

Answer: a

Title: Analyze the powerOfTwo method.

Difficulty: Medium

Section Reference 1: 13.2 Recursive Helper Methods

41) 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

Answer: d

Title: How many recursive calls are made from the original call of powerOfTwo(63)?

Difficulty: Medium

Section Reference 1: 13.2 Recursive Helper Methods

42) 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 of powerOfTwo(64) (not including the original call)?

a) 8

b) 6

c) 4

d) 2

Answer: b

Title: How many recursive calls are made from powerOfTwo(64)?

Difficulty: Medium

Section Reference 1: 13.2 Recursive Helper Methods

43) Complete the code for the myFactorial recursive method shown below, which is intended to compute the factorial of the value passed to the method:

public int myFactorial(int anInteger)

{

if (anInteger == 1)

{

return 1;

}

else

{

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

}

}

a) return(anInteger \* (myFactorial(anInteger)));

b) return ((anInteger - 1) \* (myFactorial(anInteger)));

c) return(anInteger \* (myFactorial(anInteger - 1)));

d) return((anInteger - 1)\*(myFactorial(anInteger - 1)));

Answer: c

Title: Complete the code for the myFactorial recursive method.

Difficulty: Medium

Section Reference 1: 13.2 Recursive Helper Methods

44) Complete the code for the myFactorial recursive method shown below, which is intended to compute the factorial of the value passed to the method:

public int myFactorial(int anInteger)

{

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

{

return 1;

}

else

{

return(anInteger \* myFactorial(anInteger - 1));

}

}

a) if (anInteger == 1)

b) if ((anInteger - 1) == 1)

c) if (anInteger \* (anInteger - 1) == 1)

d) if (myFactorial(anInteger) == 1)

Answer: a

Title: Complete the code for the myFactorial recursive method.

Difficulty: Medium

Section Reference 1: 13.2 Recursive Helper Methods

45) Complete the code for the myFactorial recursive method shown below, which is intended to compute the factorial of the value passed to the method:

public int myFactorial(int anInteger)

{

if (anInteger == 1)

{

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

}

else

{

return (anInteger \* myFactorial(anInteger - 1));

}

}

a) return 0;

b) return -anInteger;

c) return 1;

d) return myFactorial(anInteger);

Answer: c

Title: Complete the code for the myFactorial recursive method.

Difficulty: Medium

Section Reference 1: 13.2 Recursive Helper Methods

46) Complete the code for the recursive method shown below, which is intended to compute the sum of the first *n* integers:

public int s(int n)

{

if (n == 1)

{

return 1;

}

else

{

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

}

}

a) return n + (n - 1);

b) return s(n) + n - 1;

c) return n + s(n - 1);

d) return n + s(n + 1);

Answer: c

Title: Complete the code for this recursive method.

Difficulty: Medium

Section Reference 1: 13.2 Recursive Helper Methods

47) 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)

Answer: a

Title: Complete the code for the calcPower recursive method.

Difficulty: Medium

Section Reference 1: 13.2 Recursive Helper Methods

48) 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;

if (exponent == 0)

{

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

}

else

{

answer = baseNum \* calcPower (baseNum, exponent - 1);

}

return answer;

}

a) answer = 0;

b) answer = 1;

c) answer = -1;

d) answer = calcPower(1);

Answer: b

Title: Complete the code for the calcPower recursive method.

Difficulty: Medium

Section Reference 1: 13.2 Recursive Helper Methods

49) 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;

if (exponent == 0)

{

answer = 1;

}

else

{

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

}

return answer;

}

a) answer = baseNum \* calcPower (baseNum -1, exponent);

b) answer = baseNum \* calcPower (baseNum, exponent - 1);

c) answer = baseNum \* calcPower (baseNum, exponent);

d) answer = baseNum \* calcPower (baseNum -1, exponent - 1);

Answer: b

Title: Complete the code for the calcPower recursive method.

Difficulty: Medium

Section Reference 1: 13.2 Recursive Helper Methods

50) Given the following class code:

public class RecurseSample

{

public static void main(String[] args)

{

recurse(3);

}

public static int recurse(int n)

{

int total = 0;

if (n == 0)

{

return 0;}

else

{

total = 3 + recurse(n - 1);

}

System.out.println(total);

return total;

}

}

What values will be printed?

a) 1, 3, and 6

b) 1, 3, 6, and 9

c) 3, 6, and 9

d) 3, 6, 9, and 12

Answer: c

Title: What values will this recursive code sample print?

Difficulty: Medium

Section Reference 1: 13.2 Recursive Helper Methods

51) Given the following class code:

public class RecurseSample

{

public static void main(String[] args)

{

System.out.println(recurse(3));

}

public static int recurse(int n)

{

int total = 0;

if (n == 0)

{

return 0;

}

else

{

total = 3 + recurse(n - 1);

}

return total;

}

}

What values will be printed when this code is executed?

a) 6

b) 9

c) 3, 6, and 9

d) 1, 3, 6, and 9

Answer: b

Title: What values will this recursive code sample print?

Difficulty: Medium

Section Reference 1: 13.2 Recursive Helper Methods

52) Given the following class code:

public class RecurseMore

{

public static void main(String[] args)

{

recurse(4);

}

public static int recurse(int n)

{

int total = 0;

if (n == 0)

{

return 0;

}

else

{

total = 4 + recurse(n - 2);

}

System.out.println(total);

return total;

}

}

What values will be printed when this code is executed?

a) 0, 4, and 8

b) 4 and 8

c) 4

d) 8

Answer: b

Title: What values will this code sample print?

Difficulty: Medium

Section Reference 1: 13.2 Recursive Helper Methods

53) Given the following code snippet:

public static int newCalc(int n)

{

if (n < 0)

{

return -1;

}

else if (n < 10)

{

return n;

}

else

{

return (n % 10) + newCalc(n / 10);

}

}

What value will be returned when this code is executed with a call to newCalc(15)?

a) 2

b) 2.5

c) 6

d) 6.5

Answer: c

Title: What value will be returned with a call to newCalc(15)?

Difficulty: Medium

Section Reference 1: 13.2 Recursive Helper Methods

54) Given the following code snippet:

public static int newCalc(int n)

{

if (n < 0)

{

return -1;

}

else if (n < 10)

{

return n;

}

else

{

return (n % 10) + newCalc(n / 10);

}

}

What value will be returned when this code is executed with a call to newCalc(5)?

a) 2

b) 2.5

c) 5

d) 5.5

Answer: c

Title: What value will be returned with a call to newCalc(5)?

Difficulty: Medium

Section Reference 1: 13.2 Recursive Helper Methods

55) Given the following code snippet:

public static int newCalc(int n)

{

if (n < 0)

{

return -1;

}

else if (n < 10)

{

return n;

}

else

{

return (1 + newCalc(n / 10));

}

}

What value will be returned when this code is executed with a call to newCalc(5)?

a) 1

b) 1.5

c) 5

d) 5.5

Answer: c

Title: What value will be returned with a call to newCalc(5)?

Difficulty: Medium

Section Reference 1: 13.2 Recursive Helper Methods

56) Given the following code snippet:

public static int newCalc(int n)

{

if (n < 0)

{

return -1;

}

else if (n < 10)

{

return n;

}

else

{

return (1 + newCalc(n / 10));

}

}

What value will be returned when this code is executed with a call to newCalc(15)?

a) 2

b) 2.5

c) 5

d) 5.5

Answer: a

Title: What value will be returned with a call to newCalc(15)?

Difficulty: Medium

Section Reference 1: 13.2 Recursive Helper Methods

57) Given the following class code:

public class RecurseMore

{

private static int total;

public static void main(String[] args)

{

System.out.println(recurse(4));

}

public static int recurse(int n)

{

int total = 0;

if (n == 0)

{

return 0;

}

else

{

total = 4 + recurse(n - 2);

}

return total;

}

}

What values will be printed when this code is executed?

a) 0, 4, and 8

b) 4 and 8

c) 4

d) 8

Answer: d

Title: What values will this code sample print?

Difficulty: Medium

Section Reference 1: 13.2 Recursive Helper Methods

58) Complete the following code snippet, which is intended to be a recursive method that will find the smallest value in an array of double values from index to the end of the array:

public static double minVal(double[] elements, int index)

{

if (index == elements.length - 1)

{

return elements[index];

}

double val = \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_;

if (elements[index] < val)

{

return elements[index];

}

else

{

return val;

}

}

a) minVal(elements, index - 1)

b) minVal(index – 1)

c) minVal(elements, index + 1)

d) minVal(index + 1)

Answer: c

Title: Complete the code for this recursive method

Difficulty: Hard

Section Reference 1: 13.2 Recursive Helper Methods

59) Complete the following code snippet, which is intended to be a recursive method that will find the smallest value in an array of double values from index to the end of the array:

public static double minVal(double[] elements, int index)

{

if (index == elements.length - 1)

{

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

}

double val = minVal(elements, index + 1);

if (elements[index] < val)

{

return elements[index];

}

else

{

return val;

}

}

a) return elements[index];

b) return 1;

c) return 0;

d) return elements[0];

Answer: a

Title: Complete the code for this recursive method

Difficulty: Hard

Section Reference 1: 13.2 Recursive Helper Methods

60) Complete the following code snippet, which is intended to be a recursive method that will find the sum of all elements in an array of double values from index to the end of the array:

// return the sum of all elements in arr[]

public static double findSum(double arr[], int index)

{

if (index == 0)

{

return arr[index];

}

else

{

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

}

}

Assume that this method would be called using an existing array named myArray as follows:

findSum(myArray, myArray.length - 1);

a) return (findSum(arr, index + 1));

b) return (findSum(arr, index - 1));

c) return (arr[index] + findSum(arr, index + 1));

d) return (arr[index] + findSum(arr, index - 1));

Answer: d

Title: Complete the code for this recursive method

Difficulty: Hard

Section Reference 1: 13.2 Recursive Helper Methods

61) Complete the following code snippet, which is intended to be a recursive method that will find the sum of all elements in an array of double values from index to the end of the array:

// return the sum of all elements in arr[]

public static double findSum(double arr[], int index)

{

if (index == 0)

{

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

}

else

{

return (arr[index] + findSum(arr, index - 1));

}

}

Assume that this method would be called using an existing array named myArray as follows:

findSum(myArray,myArray.length - 1);

a) return arr[index];

b) return arr[index + 1];

c) return arr[1];

d) return arr[index - 1];

Answer: a

Title: Complete the code for this recursive method

Difficulty: Hard

Section Reference 1: 13.2 Recursive Helper Methods

62) Complete the following code snippet, which is intended to be a recursive method that reverses a String value:

public static String reverseIt(String s)

{

if (s.length() <= 1)

{

\_\_\_\_\_\_\_\_\_

}

else

{

return reverseIt(s.substring(1)) + s.charAt(0);

}

}

a) return s;

b) return 0;

c) return s.charAt(0);

d) return s.substring(1);

Answer: a

Title: Complete the code for this recursive method

Difficulty: Hard

Section Reference 1: 13.2 Recursive Helper Methods

63) Complete the following code snippet, which is intended to be a recursive method that reverses a String value:

public static String reverseIt(String s)

{

if (s.length() <= 1)

{

return s;

}

else

{

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
 }

}

a) return reverseIt(s.substring(0)) + s.charAt(1);

b) return reverseIt(s.substring(0)) + s.charAt(0);

c) return reverseIt(s.substring(1)) + s.charAt(1);

d) return reverseIt(s.substring(1)) + s.charAt(0);

Answer: d

Title: Complete the code for this recursive method

Difficulty: Hard

Section Reference 1: 13.2 Recursive Helper Methods

64) Consider 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) // line #1

{

return 0; // line #

}

else

{

return (n + printSum(n)); //line #3

}

}

Which of the following statements is correct?

a) line #1 is incorrect, and should be changed to if (n <= 1)

b) line #3 is incorrect, and should be changed to return (printSum (n - 1));

c) line #3 is incorrect, and should be changed to return (n + printSum (n + 1));

d) line #3 is incorrect, and should be changed to return (n + printSum (n - 1));

Answer: d

Title: Which statement about the recursive method printSum is correct?

Difficulty: Medium

Section Reference 1: 13.2 Recursive Helper Methods

65) A palindrome is a word or phrase that reads the same forward or backward. Consider the methods palindrome and isPal shown below:

public boolean palindrome(String string)

{

return isPal(string, 0, string.length() - 1);

}

private boolean isPal(String string, int left, int right)

{

if (left >= right)

{

return true;

}

else if (string.charAt(left) == string.charAt(right))

{

return isPal(string, left + 1, right - 1);

}

else

{

return false;

}

}

The method palindrome as shown here would be considered to be a \_\_\_\_ method.

a) recursive

b) terminating

c) helper

d) static

Answer: c

Title: The method shown is considered to be a \_\_\_\_ method.

Difficulty: Easy

Section Reference 1: 13.2 Recursive Helper Methods

66) A palindrome is a word or phrase that reads the same forward or backward. Consider the following code snippet:

public boolean palindrome(String string)

{

return isPal(string, 0, string.length() - 1);

}

private boolean isPal(String string, int left, int right)

{

if (left >= right)

{

return true;

}

else if (string.charAt(left) == string.charAt(right))

{

return isPal(string, left + 1, right - 1);

}

else

{

return false;

}

}

What is the purpose of the palindrome method?

a) Return the palindrome to the calling method.

b) Provide the string, along with its first and last indexes to the recursive isPal method.

c) Send the recursive isPal method its terminating condition.

d) Recursively call itself.

Answer: b

Title: What is the purpose of the palindrome method?

Difficulty: Medium

Section Reference 1: 13.2 Recursive Helper Methods

67) A palindrome is a word or phrase spelled which reads the same forward or backward. Consider the following code snippet:

public boolean palindrome(String string)

{

return isPal(string, 0, string.length() - 1);

}

private boolean isPal(String string, int left, int right)

{

if (left >= right)

{

return true;

}

else if (string.charAt(left) == string.charAt(right))

{

return isPal(string, left + 1, right - 1);

}

else

{

return false;

}

}

What does the method palindrome return?

a) true

b) false

c) a palindrome not found exception

d) the Boolean value returned from the isPal method

Answer: d

Title: What does the method palindrome return?

Difficulty: Easy

Section Reference 1: 13.2 Recursive Helper Methods

68) A palindrome is a word or phrase that reads the same forward or backward. Consider the following code snippet:

public boolean palindrome(String string)

{

return isPal(string, 0, string.length() - 1);

}

private boolean isPal(String string, int left, int right)

{

if (left >= right)

{

return true;

}

else if (string.charAt(left) == string.charAt(right))

{

return isPal(string, left + 1, right - 1);

}

else

{

return false;

}

}

What does the condition left >= right refer to?

a) An empty or one-character string is considered a palindrome.

b) The string is not a palindrome.

c) It cannot be determined if the string is a palindrome.

d) You have reached the middle of the string.

Answer: a

Title: What does the condition left >= right in the isPal method refer to?

Difficulty: Medium

Section Reference 1: 13.2 Recursive Helper Methods

69) What is the purpose of a recursive helper method?

a) Shield the user of the recursive method from the recursive details.

b) Speed up the execution.

c) Eliminate the recursion.

d) Add another base case.

Answer: a

Title: What is the purpose of a recursive helper method?

Difficulty: Easy

Section Reference 1: 13.2 Recursive Helper Methods

70) Consider the recursive square method shown below. It takes a non-negative int argument. Then it recursively adds the number n to itself n times to produce the square of n. Complete the correct code for the square helper method.

public int square(int n)

{

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_;

}

public int square(int c, int n)

{

if (c == 1)

{

return n;

}

else

{

return n + square(c - 1, n);

}

}

a) return square(n, n)

b) return square(n)

c) return square(n - 1, n)

d) return square(n, n - 1)

Answer: a

Title: Identify the correct code for the square helper method

Difficulty: Medium

Section Reference 1: 13.2 Recursive Helper Methods

71) Consider the recursive square method shown below that takes a non-negative int argument:

public int square(int n)

{

return square(n, n);

}

public int square(int c, int n)

{

if (c == 1)

{

return n;

}

else

{

return n + square(c - 1, n);

}

}

Assume that the last return statement is changed to this:

return square(c - 1, n);

What would a call to square(7) return?

a) 7

b) 13

c) 14

d) 49

Answer: a

Title: What would square(7) return after this code change?

Difficulty: Medium

Section Reference 1: 13.2 Recursive Helper Methods

72) Consider the recursive square method shown below that takes a non-negative int argument.

public int square(int n)

{

return square(n, n);

}

public int square(int c, int n)

{

if (c == 1)

{

return n;

}

else

{

return n + square(c - 1, n);

}

}

Assume that the last return statement is changed to this:

return n \* square(c - 1, n);

What would a call to square(4) return?

a) 4

b) 16

c) 64

d) 256

Answer: d

Title: What would square(4) return after this code change?

Difficulty: Medium

Section Reference 1: 13.2 Recursive Helper Methods

73) Consider the helper method reversePrint, which uses recursion to display in reverse the elements in a section of an array limited by the firstIndex and lastIndex arguments. What statement should be used to complete the recursive method?

public static void reversePrint(int[] array, int firstIndex, int lastIndex)

{

if (firstIndex < lastIndex)

{

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

}

System.out.println(array[firstIndex]);

}

public static void main(String[] args)

{

int [] numbers = { 4, 7, 1, 0, 2, 7 };

reversePrint(numbers, 0, numbers.length – 1);

}

a) reversePrint(array, firstIndex, lastIndex + 1);

b) reversePrint(array, firstIndex, lastIndex - 1);

c) reversePrint(array, firstIndex + 1, lastIndex - 1);

d) reversePrint(array, firstIndex + 1, lastIndex);

Answer: d

Title: Select statement to complete recursive helper method to print array elements in reverse

Difficulty: Hard

Section Reference 1: 13.2 Recursive Helper Methods

74) Assume that recursive method search returns true if argument value is one of the elements in the section of the array limited by the firstIndex and lastIndex arguments. What statement can be used in main to determine if the value 7 is one of the elements in array values?

public static boolean search(int value, int[] array, int firstIndex, int lastIndex)

{

if (firstIndex <= lastIndex)

{

if (array[firstIndex] == value)

{

return true;

}

else

{

return search(value, array, firstIndex + 1, lastIndex);

}

}

return false;

}

public static void main(String[] args)

{

int [] values = { 4, 7, 1, 0, 2, 7 };

if ( \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ )

{

System.out.println("7 is in the array");

}

}

a) search(7, values, 0, values.length)

b) search(7, values, 1, values.length)

c) search(7, values, 0, values.length - 1)

d) search(7, values, 1, values.length - 1)

Answer: c

Title: Select statement that invokes recursive helper method to search array

Difficulty: Easy

Section Reference 1: 13.2 Recursive Helper Methods

75) Consider the problem of displaying a pattern of asterisks which form a triangle of height h, as shown below for h = 4:

\*

\*\*

\*\*\*

\*\*\*\*

\*\*\*

\*\*

\*

The problem can be solved with the recursive helper method shape below. The method takes two parameters: low and high. It first prints a row of asterisks corresponding to parameter low. If high is higher than low, it recursively generates a shape in which low is incremented by one, and then prints another row of low asterisks. Select a statement to complete method triangle, so that the helper method shape is invoked with the correct arguments in order to display a triangle with parameter height.

public static void shape(int low, int high)  
{  
 if (high >= low)  
 {  
 asterisksRow(low);  
 if (high > low)  
 {  
 shape(low + 1, high);  
 asterisksRow(low);  
 }  
 }  
}

public static void asterisksRow(int n) // Method to display a row of n stars

{

for (int j = 1; j < n; ++j)

{

System.out.print("\*");

}

System.out.println("\*");

}

public static void triangle(int height)  
{  
 if (height > 1)  
 {  
 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
 }  
}

a) shape(0, height);

b) shape(0, height - 1);

c) shape(1, height);

d) shape(0, height - 1);

Answer: c

Title: Select statement that invokes recursive helper method to display triangle shape

Difficulty: Medium

Section Reference 1: 13.2 Recursive Helper Methods

76) Why does the best recursive method usually run slightly slower than its iterative counterpart?

a) Testing the terminating condition takes longer.

b) Each recursive method call takes processor time.

c) Multiple recursive cases must be considered.

d) Checking multiple terminating conditions take more processor time.

Answer: b

Title: Why does the best recursive method usually run slightly slower than its iterative counterpart?

Difficulty: Medium

Section Reference 1: 13.3 The Efficiency of Recursion

77) Consider the fib method from the textbook shown below:

public static long fib(int n)

{

if (n <= 2)

{

return 1;

}

else

{

return fib(n – 1) + fib(n – 2);

}

}

Computing the 7th fibonacci number, fib(7), recursively computes fib(6), fib(5), and fib(4) \_\_\_ times respectively.

a) 1, 2, and 3

b) 6, 5, and 4

c) 4, 5, and 6

d) 3, 2, and 1

Answer: a

Title: Computing the 7th fibonacci number, fib(7)

Difficulty: Medium

Section Reference 1: 13.3 The Efficiency of Recursion

78) Consider the fib method from the textbook shown below.

public static long fib(int n)

{

if (n <= 2)

{

return 1;

}

else

{

return fib(n – 1) + fib(n – 2);

}

}

Calling fib(3) will trigger \_\_\_ recursive call(s) and execute the terminating condition \_\_\_ time(s), respectively.

a) 1, 1

b) 2, 2

c) 1, 2

d) 2, 1

Answer: b

Title: Calling fib(3) will trigger \_\_\_ recursive call(s) and execute the terminating condition \_\_\_ time(s).

Difficulty: Medium

Section Reference 1: 13.3 The Efficiency of Recursion

79) Consider the fib method from the textbook shown below:

public static long fib(int n)

{

if (n <= 2)

{

return 1; // line #1

}

else

{

return fib(n – 1) + fib(n – 2); // line #2

}

}

Assume line #1 is changed to this:

if (n <= 2) { return 2; }

How will this change affect the result of calling fib(7)?

a) It will add 2 to the return from fib(7)

b) It will add 4 to the return from fib(7)

c) It will add 8 to the return from fib(7)

d) It will double the return from fib(7)

Answer: d

Title: In the fib method, how would changing line #1 affect the result?

Difficulty: Hard

Section Reference 1: 13.3 The Efficiency of Recursion

80) Consider the fib method from the textbook shown below:

public static long fib(int n)

{

if (n <= 2)

{

return 1; // line #1

}

else

{

return fib(n – 1) + fib(n – 2); // line #2

}

}

Assume line #2 is changed to this:

else { return 2 \* fib(n – 1) + 2 \* fib(n – 2); }

What effect will this change have?

a) This will return 5 from fib(4)

b) This will return 8 from fib(4)

c) This will return 10 from fib(4)

d) This will cause an exception on a call fib(4)

Answer: c

Title: In the fib method, what effect will this code change have?

Difficulty: Hard

Section Reference 1: 13.3 The Efficiency of Recursion

81) Consider the fib method from the textbook shown below:

public static long fib(int n)

{

if (n <= 2)

{

return 1; // line #1

}

else

{

return fib(n – 1) + fib(n – 2); // line #2

}

}

Assume line #1 is changed to this:

if (n <= 2) { return n; }

What effect will this change have?

a) This will return 21 from fib(6)

b) This will return 13 from fib(6)

c) This will return 8 from fib(6)

d) This will return 6 from fib(6)

Answer: b

Title: In the fib method, what effect will this code change have?

Difficulty: Hard

Section Reference 1: 13.3 The Efficiency of Recursion

82) Consider the recursive version of the fib method from the textbook shown below:

public static long fib(int n)

{

if (n <= 2)

{

return 1;

}

else

{

return fib(n – 1) + fib(n – 2);

}

}

How many recursive calls to fib(2) will be made from the original call of fib(6)?

a) 2

b) 3

c) 4

d) 5

Answer: d

Title: How many recursive calls to fib(2) will be made from the call of fib(6)?

Difficulty: Medium

Section Reference 1: 13.3 The Efficiency of Recursion

83) Consider the recursive version of the fib method from the textbook shown below:

public static long fib(int n)

{

if (n <= 2)

{

return 1;

}

else

{

return fib(n – 1) + fib(n – 2);

}

}

How many total recursive calls (not counting the original call) to fib will be made from the original call of fib(6)?

a) 6

b) 12

c) 14

d) 20

Answer: c

Title: How many total recursive calls to fib will be made from the call of fib(6)?

Difficulty: Medium

Section Reference 1: 13.3 The Efficiency of Recursion

84) Consider the recursive version of the fib method from the textbook shown below:

public static long fib(int n)

{

if (n <= 2)

{

return 1;

}

else

{

return fib(n – 1) + fib(n – 2);

}

}

How many total recursive calls (not counting the original call) to fib will be made from the original call of fib(7)?

a) 12

b) 24

c) 30

d) 32

Answer: b

Title: How many total recursive calls to fib will be made from the original call of fib(7)?

Difficulty: Medium

Section Reference 1: 13.3 The Efficiency of Recursion

85) Consider the recursive version of the fib method from the textbook shown below:

public static long fib(int n)

{

if (n <= 2)

{

return 1;

}

else

{

return fib(n – 1) + fib(n – 2);

}

}

How many more recursive calls to fib will be made from the original call of fib(7) than from the original call of fib(6) (not counting the original calls)?

a) 1

b) 2

c) 5

d) 10

Answer: d

Title: How many more recursive calls (not counting the original call) to fib will be made from the original call of fib(7) than fib(6)?

Difficulty: Medium

Section Reference 1: 13.3 The Efficiency of Recursion

86) The method below implements the exponentiation operation recursively by taking advantage of the fact that, if the exponent n is even, then xn = (xn/2)2. Select the expression that should be used to complete the method so that it computes the result correctly.

public static double power(double base, double exponent)

{

if (exponent % 2 != 0) // if exponent is odd

{

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

}

else if (exponent > 0)

{

double temp = \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ ;

return temp \* temp;

}

return base;

}

a) power(base, exponent / 2)

b) power(base, exponent / 2) \* power(base, exponent / 2)

c) power(base, exponent / 2) + power(base, exponent / 2)

d) power(base, exponent) / 2

Answer: a

Title: Complete recursive exponentiation method

Difficulty: Medium

Section Reference 1: 13.3 The Efficiency of Recursion

87) Consider the iterative version of the fib method from the textbook shown below:

public static long fib(int n)

{

if (n <= 2)

{

return 1;

}

long fold = 1;

long fold2 = 1;

long fnew = 1;

for (int i = 3; i <= n; i++)

{

fnew = fold + fold2;

fold2 = fold;

fold = fnew;

}

return fnew;

}

How many iterations of the for loop will there be for the call fib(6)?

a) 6

b) 5

c) 4

d) 3

Answer: c

Title: Comparing iteration to recursion.

Difficulty: Easy

Section Reference 1: 13.3 The Efficiency of Recursion

88) Suppose we wrote a new version of fib called newFib. What does the call newFib(6) return?

public static long newFib(int n)

{

if (n <= 3)

{

return 1;

}

else

{

return newFib(n – 1) + newFib(n – 2) + newFib(n – 3);

}

}

a) 3

b) 5

c) 7

d) 9

Answer: d

Title: Suppose we wrote a method called newFib. What does the call newFib(6) return?

Difficulty: Medium

Section Reference 1: 13.3 The Efficiency of Recursion

89) Suppose we wrote a new version of method fib, called newFib. Compare newFib to the original fib shown below:

public static long newFib(int n)

{

if (n <= 3)

{

return 1;

}

else

{

return newFib(n – 1) + newFib(n – 2) + newFib(n – 3);

}

}

public static long fib(int n)

{

if (n <= 2)

{

return 1;

}

else

{

return fib(n – 1) + fib(n – 2);

}

}

For which values of the integer n does newFib(n) always returns a value greater than fib(n)?

a) any positive n

b) any value of n

c) n >= 3

d) n > 3

Answer: d

Title: Comparing iteration to recursion.

Difficulty: Medium

Section Reference 1: 13.3 The Efficiency of Recursion

90) Which statement(s) about recursion are true?

I Recursion is faster than iteration

II Recursion is often easier to understand than iteration

III Recursive design has an economy of thought

a) I

b) II

c) II and III

d) I and III

Answer: c

Title: Identify the characteristics of recursion.

Difficulty: Medium

Section Reference 1: 13.3 The Efficiency of Recursion

91) In recursion, the recursive call is analogous to a loop \_\_\_\_.

a) call

b) iteration

c) termination

d) return

Answer: b

Title: In recursion, the recursive call is analogous to a loop \_\_\_\_.

Difficulty: Easy

Section Reference 1: 13.3 The Efficiency of Recursion

92) In recursion, the non-recursive case is analogous to a loop \_\_\_\_.

a) call

b) iteration

c) termination

d) condition

Answer: c

Title: In recursion, the non-recursive case is analogous to a loop \_\_\_\_.

Difficulty: Easy

Section Reference 1: 13.3 The Efficiency of Recursion

93) In recursion, the terminating condition is analogous to a loop \_\_\_\_\_\_\_\_\_.

a) call

b) iteration

c) termination condition

d) initialization condition

Answer: c

Title: In recursion, the terminating condition is analogous to a loop \_\_\_\_\_\_\_\_\_\_.

Difficulty: Easy

Section Reference 1: 13.3 The Efficiency of Recursion

94) Complete the following code snippet, which is intended to print out all permutations of the string generate by using a permutation generator object.

public class PermutationGeneratorTester

{

public static void main(String[] args)

{

PermutationGenerator generator

= new PermutationGenerator("generate");

ArrayList<String> permutations

= generator.getPermutations();

for (String s : permutations)

{

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

}

}

}

a) generator.print();

b) generator.setPermutations();

c) generator.calculate();

d) System.out.println(s);

Answer: d

Title: Insert the missing statement in code using a permutation generator.

Difficulty: Easy

Section Reference 1: 13.4 Permutations

95) Which of the following strings is a palindrome?

a) "Test"

b) "B"

c) "canal"

d) "salami"

Answer: b

Title: Which of these strings is a palindrome?

Difficulty: Easy

Section Reference 1: 13.4 Permutations

96) Which of the following statements is correct?

a) The empty string is a palindrome.

b) Strings of length 0 or 1 are not palindromes.

c) The string "eat" is a palindrome.

d) The string "Adam" is a palindrome.

Answer: a

Title: Which of these statements about palindromes is correct?

Difficulty: Easy

Section Reference 1: 13.4 Permutations

97) The string "eat" has \_\_\_\_ permutations.

a) 2

b) 4

c) 6

d) 8

Answer: c

Title: The string "eat" has how many permutations?

Difficulty: Easy

Section Reference 1: 13.4 Permutations

98) A unique permutation is one that is different from any other generated permutation. How many unique permutations does the string "bee" have?

a) 2

b) 3

c) 4

d) 5

Answer: b

Title: How many unique permutations does the string "bee" have?

Difficulty: Medium

Section Reference 1: 13.4 Permutations

99) A unique permutation is one that is different from any other generated permutation. How many unique permutations does the string "aaa" have?

a) 0

b) 1

c) 2

d) 3

Answer: b

Title: How many unique permutations does the string "aaa" have?

Difficulty: Medium

Section Reference 1: 13.4 Permutations

100) Which of the following statements about palindromes is correct?

a) The empty string is not a palindrome.

b) The string "I" is not a palindrome.

c) The string "rascal" is a palindrome.

d) All strings of length 0 or 1 are palindromes.

Answer: d

Title: Which of these statements about palindromes is correct?

Difficulty: Medium

Section Reference 1: 13.4 Permutations

101) Consider the following change to the PermutationGenerator class from the textbook. Instead of adding the removed character to the front of the each permutation of the simpler word, we will add it to the end.

// Add the removed character to the end of

// each permutation of the simpler word

for (String s : shorterWordPermutations)

{

result.add(s + word.charAt(i));

}

Consider the list produced by the modified PermutationGenerator. Which best describes this list?

a) It contains all permutations.

b) It contains reversed permutations.

c) It is an incomplete list of permutations.

d) It contains strings that are not permutations.

Answer: a

Title: Effect of adding the removed character to the end of the simpler permutation.

Difficulty: Medium

Section Reference 1: 13.4 Permutations

102) Which of the following statements about recursion is correct?

a) It is not necessary to have a special terminating case in all recursions.

b) It is not necessary to simplify the argument in the recursive call.

c) A recursive solution will always run faster than an equivalent iterative solution.

d) In many cases, a recursive solution will be easier to understand and to implement than an iterative solution.

Answer: d

Title: Which statement about recursion is correct?

Difficulty: Easy

Section Reference 1: 13.4 Permutations

103) The method below generates all substrings of a String passed as argument. Assuming that the string contains no duplicate characters, select the statement to complete the method so that it prints all substrings correctly.

public static void printSubstrings(String word)

{

if (word.length() > 0)

{

for (int j = 1; j <= word.length(); j++) // print substrings that begin

{ // with the first character

System.out.println(word.substring(0, j));

}

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ // print substrings without

} // the first character

}

a) printSubstrings(word.substring(0));

b) printSubstrings(word.substring(1));

c) printSubstrings(word.substring(word.length()));

d) printSubstrings(word.substring(word.length() - 1));

Answer: b

Title: Complete recursive method to print substrings of a word

Difficulty: Medium

Section Reference 1: 13.4 Permutations

104) Consider the mutually recursive methods below. Select the method call that could be used to generate the output sequence: A5 B4 A3 B2 A1

public static void methodA(int value)

{

if (value > 0)

{

System.out.print(" A" + value);

methodB(value – 1);

}

}

public static void methodB(int value)

{

if (value > 0)

{

System.out.print(" B" + value);

methodA(value – 1);

}

}

a) methodA(5);

b) methodB(5);

c) methodA(4);

d) methodB(4);

Answer: a

Title: Identify appropriate statement to invoke mutually recursive methods

Difficulty: Easy

Section Reference 1: 13.5 Mutual Recursion

105) Recursion will take place if any of the following happen:

I method A calls method B, which calls method C

II method A calls method B, which calls method A

III method A calls method A

a) I

b) I and II

c) II

d) II and III

Answer: d

Title: Describe conditions necessary for recursion.

Difficulty: Easy

Section Reference 1: 13.5 Mutual Recursion

106) Recursion does NOT take place if any of the following happen:

I method A calls method B, which calls method C, which calls method B

II method A calls method B, which calls method A

III method A calls method B, B returns, and A calls B again

a) I

b) I and II

c) II

d) III

Answer: d

Title: Describe conditions necessary for recursion.

Difficulty: Easy

Section Reference 1: 13.5 Mutual Recursion

107) Consider the following code snippet:

public static boolean isEven(int n)

{

if (n % 2 == 0)

{

return true;

}

else

{

return (isOdd(n));

}

}

public static boolean isOdd(int n)

{

if (n % 2 == 1)

{

return true;

}

else

{

return (isEven(n));

}

}

For any given value of n, what is the maximum number of function calls that could occur?

a) 0

b) 1

c) 2

d) cannot be determined

Answer: c

Title: For any given value of n, what is the maximum number of function calls that could occur?

Difficulty: Medium

Section Reference 1: 13.5 Mutual Recursion

108) Complete the following code snippet, which is intended to determine if a value is even or odd using mutual recursion:

public static boolean isEven(int n)

{

if (n == 0)

{

return true;

}

else

{

return isOdd(Math.abs(n) - 1);

}

}

public static boolean isOdd(int n)

{

if (n == 0)

{

\_\_\_\_\_\_\_\_\_

}

else

{

return isEven(Math.abs(n) - 1);

}

}

a) return true;

b) return false;

c) return isOdd(Math.abs(n)-1);

d) return isOdd(Math.abs(n));

Answer: b

Title: Complete this code for mutual recursion.

Difficulty: Hard

Section Reference 1: 13.5 Mutual Recursion

109) Backtracking \_\_\_\_\_.

a) starts from the end of the program and works backward to the beginning.

b) starts with a partial solution and builds it up to get closer to the goal.

c) never abandons a partial solution.

d) explores only one path toward a solution

Answer: b

Title: Backtracking \_\_\_\_.

Difficulty: Easy

Section Reference 1: 13.6 Backtracking

110) \_\_\_\_ is a problem-solving technique that examines partial solutions, abandons unsuitable ones, and returns to consider other candidate solutions.

a) Debugging

b) Traceback

c) Backtracking

d) Recursion

Answer: c

Title: \_\_\_\_ is a problem-solving technique that examines partial solutions.

Difficulty: Easy

Section Reference 1: 13.6 Backtracking