

**This Test Is a KEY
DO NOT WRITE ON THIS TEST**

This test includes program segments, which are not complete programs. Answer such questions with the assumption that the program segment is part of a correct program.

Objective 1 - Recursion Fundamentals

01. The process of a method calling itself is

- ### (A) recursion.
(B) iteration.
(C) compiling.
(D) error checking.

02. You must have an exit or _____ to stop the recursive process.

- (A) compiler
(B) stack
(C) index pointer
(D) base case

03. Which of the following is a data structure that accesses information using the “LIFO” principle?

- (A) compiler
(B) stack
(C) index pointer
(D) base case

04. The program execution sequence is handled by the

- (A) compiler.
(B) stack.
(C) index pointer.
(D) base case.

05. Consider the **count** method below.

```
public static void count(int a, int b)
{
    if (a <= b)
    {
        System.out.print(a + " ");
        count(a+1, b);
    }
}
```

What will be displayed by the method call *count(10,20)*?

- ####
- (A) 10 11 12 13 14 15 16 17 18 19 20
 - (B) 10 11 12 13 14 15 16 17 18 19
 - (C) 20 19 18 17 16 15 14 13 12 11 10
 - (D) 19 18 17 16 15 14 13 12 11 10
 - (E) 20 19 18 17 16 15 14 13 12 11 10 10 11 12 13 14 15 16 17 18 19 20

06. Consider the **count** method below.

```
public static void count(int a, int b)
{
    if (a <= b)
    {
        count(a+1, b);
        System.out.print(a + " ");
    }
}
```

What will be displayed by the method call *count(10,20)*?

- (A) 10 11 12 13 14 15 16 17 18 19 20
- (B) 10 11 12 13 14 15 16 17 18 19
- #### (C) 20 19 18 17 16 15 14 13 12 11 10
- (D) 19 18 17 16 15 14 13 12 11 10
- (E) 20 19 18 17 16 15 14 13 12 11 10 10 11 12 13 14 15 16 17 18 19 20

07. Consider the **count** method below.

```
public static void count(int a, int b)
{
    if (a <= b)
    {
        System.out.print(a + " ");
        count(a+1, b);
        System.out.print(a + " ");
    }
}
```

What will be displayed by the method call **count(10,20)** ?

- (A) 10 11 12 13 14 15 16 17 18 19 20
- (B) 10 11 12 13 14 15 16 17 18 19
- (C) 20 19 18 17 16 15 14 13 12 11 10
- ### (D) 10 11 12 13 14 15 16 17 18 19 20 20 19 18 17 16 15 14 13 12 11 10
- (E) 20 19 18 17 16 15 14 13 12 11 10 10 11 12 13 14 15 16 17 18 19 20

08. Consider the **count** method below.

```
public static void count(int a, int b)
{
    if (a < b)
    {
        System.out.print(a + " ");
        count(a+1, b);
    }
}
```

What will be displayed by the method call **count(10,20)** ?

- (A) 10 11 12 13 14 15 16 17 18 19 20
- ### (B) 10 11 12 13 14 15 16 17 18 19
- (C) 20 19 18 17 16 15 14 13 12 11 10
- (D) 19 18 17 16 15 14 13 12 11 10
- (E) 20 19 18 17 16 15 14 13 12 11 10 10 11 12 13 14 15 16 17 18 19 20

Objective 2 – Evaluating Recursive Return Methods

09. What value does **mystery1(5)** return ?

- (A) 0
- ### (B) 1
- (C) 4
- (D) 5
- (E) 6

```
public int mystery1(int a)
{
    if (a==1)
        return 1;
    else
        return mystery1(a-1);
}
```

10. What value does **mystery2(7)** return ?

- (A) 0
- (B) 1
- (C) 4
- ### (D) 8
- (E) 13

```
public int mystery2(int n)
{
    if (n <= 1)
        return 1;
    else
        if (n % 2 == 1)
            return n + mystery2(n-1);
        else
            return n - mystery2(n-1);
}
```

11. What value does **mystery3(5,1)** return ?

- (A) 1
- (B) 4
- (C) 5
- (D) 7
- ### (E) 8

```
public int mystery3(int a, int b)
{
    if (a < b)
        return a;
    else
        return b + mystery3(a-1,b+1);
}
```

12. What value does **mystery4(123,82)** return ?

- (A) 1
- ### (B) 41
- (C) 82
- (D) 121
- (E) 242

```
public int mystery4(int A, int B)
{
    if (A % B == 0)
        return B;
    else
        return mystery4(B, A % B);
}
```

13. What value does **mystery1(13)** return ?

- (A) 0
- ### (B) 1
- (C) 12
- (D) 13
- (E) 14

```
public int mystery1(int a)
{
    if (a == 1)
        return 1;
    else
        return mystery1(a-1);
}
```

14. What value does **mystery2(8)** return ?

- ### (A) 0
- (B) 1
- (C) 7
- (D) 8
- (E) 9

```
public int mystery2(int n)
{
    if (n <= 1)
        return 1;
    else
        if (n % 2 == 1)
            return n + mystery2(n-1);
        else
            return n - mystery2(n-1);
}
```

15. What value does **mystery3(1,5)** return ?

- ### (A) 1
- (B) 4
- (C) 5
- (D) 7
- (E) 8

```
public int mystery3(int a, int b)
{
    if (a < b)
        return a;
    else
        return b + mystery3(a-1,b+1);
}
```

16. What value does **mystery4(29,13)** return ?

- ### (A) 1
- (B) 13
- (C) 26
- (D) 29
- (E) 58

```
public int mystery4(int a, int b)
{
    if (a % b == 0)
        return b;
    else
        return mystery4(b, a % b);
}
```

17. What is the action of the method **mystery1** ?

- (A) $a + b$
- #### (B) $a * b$
- (C) a^b
- (D) b^a
- (E) $a!$ (a factorial)

```
public int mystery1(int a, int b)
{
    if (b == 1)
        return a;
    else
        return a + mystery1(a,b-1);
}
```

18. What is the action of the method **mystery2** ?

- (A) $a + b$
- (B) $a * b$
- #### (C) a^b
- (D) b^a
- (E) $a!$ (a factorial)

```
public int mystery2(int a, int b)
{
    if (b == 1)
        return a;
    else
        return a * mystery2(a,b-1);
}
```

19. What is the action of the method **mystery3** ?

- (A) $a + b$
- (B) $a * b$
- (C) a^b
- #### (D) b^a
- (E) $a!$ (a factorial)

```
public int mystery3(int a, int b)
{
    if (a == 1)
        return b;
    else
        return b * mystery3(a-1,b);
}
```

20. What is the action of the method **mystery4** ?

- (A) $a + b$
- (B) $a * b$
- (C) a^b
- (D) b^a
- #### (E) $a!$ (a factorial)

```
public int mystery4(int a, int b)
{
    if (a == 0)
        return 1;
    else
        return a * mystery4(a-1,b);
}
```

Objective 3 – Manipulating Parameters of Recursive Methods

Use the "iterative" *linear1* method and the three "recursive" **linearX** methods below .

```
public static int linear1(int list[], int key)
{
    boolean found = false;
    int k = 0;
    while (k < list.length && !found)
    {
        if (list[k] == key)
            found = true;
        else
            k++;
    }
    if (found)
        return k;
    else
        return -1;
}
```

```
public static int linear2(int list[], int key)
{
    int k = 0;
    if (k == list.length)
        return -1;
    else
        if (list[k] == key)
            return k;
        else
        {
            k++;
            return linear2(list,key);
        }
}
```

```
public static int linear3(int list[], int key, int k)
{
    if (k == list.length)
        return -1;
    else
        if (list[k] == key)
            return k;
        else
            return linear3(list,key,k+1);
}
```

```
public static int linear4(int list[], int key, int k)
{
    if (k == list.length)
        return -1;
    else
        if (list[k] == key)
            return k;
        else
        {
            k++;
            return linear2(list,key,k);
        }
}
```

21. Which of the "recursive" **linearX** methods performs the same searching actions as the "iterative" **linear1** method?

- (A) linear2 only
- (B) linear3 only
- (C) linear4 only
- (D) linear2 and linear3 only

(E) linear3 and linear4 only

22. Method **binary1** is a correct, iterative *Binary Search* method. Which of the recursive method(s) perform(s) the same action as the **binary1method**?

```
public static int binary1(int list[], int key)
{
    int lo = 0;
    int hi = list.length-1;
    int mid = 0;
    boolean found = false;
    while (lo <= hi && !found)
    {
        mid = (lo + hi) /2;
        if (list[mid] == key)
            found = true;
        else
            if (key > list[mid])
                lo = mid + 1;
            else
                hi = mid - 1;
    }
    if (found)
        return mid;
    else
        return -1;
}
```

```
public static int binary2(int list[], int key, int lo, int hi)
{
    int mid = 0;
    if (lo > hi)
        return -1;
    else
    {
        mid = (lo + hi) /2;
        if (list[mid] == key)
            return mid;
        else
            if (key > list[mid])
                return binary2(list,key,mid+1,hi);
            else
                return binary2(list,key,lo,mid-1);
    }
}
```

```
public static int binary3(int list[], int key)
{
    int lo = 0;
    int hi = list.length-1;
    int mid = 0;
    if (lo > hi)
        return -1;
    else
    {
        mid = (lo + hi) /2;
        if (list[mid] == key)
            return mid;
        else
            if (key > list[mid])
            {
                lo = mid + 1;
                return binary3(list,key);
            }
            else
            {
                hi = mid - 1;
                return binary3(list,key);
            }
    }
}
```

```
public static int binary4(int list[], int key, int lo, int hi)
{
    int mid = 0;
    if (lo > hi)
        return -1;
    else
    {
        mid = (lo + hi) /2;
        if (list[mid] == key)
            return mid;
        else
            if (key > list[mid])
            {
                lo = mid + 1;
                return binary4(list,key,lo,hi);
            }
            else
            {
                hi = mid - 1;
                return binary4(list,key,lo,hi);
            }
    }
}
```

- (A) binary2 only
 (B) binary3 only
 (E) All three binaryX methods work
- (C) binary2 and binary3 only
 (D) binary2 and binary4 only
- ###

23. Is the parameter list likely to be different between iterative and recursive methods of the same algorithm?
- (A) No, the parameter list will always be the same.
 - (B) No, in most instances the parameter list will be the same, with some occasional exceptions.
 - (C) Yes, the parameter list will always be different.
 - ### (D) It depends, many methods will have the same parameter lists. However, anytime that an iterative method initializes a variable in the method body, the parameter list will be different.

24. Recursive solutions of the *Linear Search* and the *Binary Search* have shorter code than the iterative solutions. Is this a typical occurrence between recursive and iterative solutions?
- ### (A) Yes, it is typical. Most recursive solutions are shorter than iterative solutions.
 - (B) No, it is not typical. Most recursive solutions are inter than iterative solutions.
 - (C) No, it is not typical. Most recursive solutions are the same length as iterative solutions.
 - (D) No, it is not typical. It depends strictly on the algorithm.

Objective 4 - The Tower of Hanoi

25. Given that **n** is the number of disk to be moved in a *Tower of Hanoi* puzzle. Which one of the formulas below computes the number of moves to solve the puzzle?
- (A) n^2
 - (B) $n^2 - 1$
 - (C) 2^n
 - ### (D) $2^n - 1$

26. Which steps describe the general solution to the *Tower of Hanoi* puzzle?

(A) Move **n**th disk from source to destination.
Move **n-1** disks from source to temp.
Move **n-1** disks from temp to destination.

(B) Move **n-1** disks from source to temp.
Move **n**th disk from source to destination.
Move **n-1** disks from temp to destination.

(C) Move **n-1** disks from source to destination.
Move **n**th disk from source to temp.
Move **n-1** disks from temp to destination.

(D) Move **n-1** disks from source to temp.
Move **n-1** disks from temp to destination.
Move **n**th disk from source to destination.

27. Assume that disks in a *Tower of Hanoi* puzzle are numbered from top to bottom, starting with **1**. Furthermore, assume that **Peg-A** is the source location, **Peg-B** is the temporary location and **Peg-C** is the destination location. What is the solution for a three-disk problem?

(A)
Move Disk1 from Peg A to Peg B
Move Disk2 from Peg A to Peg C
Move Disk1 from Peg B to Peg C
Move Disk3 from Peg A to Peg B
Move Disk1 from Peg C to Peg A
Move Disk2 from Peg B to Peg C
Move Disk1 from Peg A to Peg C

(B)
Move Disk1 from Peg A to Peg C
Move Disk2 from Peg A to Peg B
Move Disk1 from Peg C to Peg B
Move Disk3 from Peg A to Peg C
Move Disk1 from Peg B to Peg A
Move Disk2 from Peg B to Peg C
Move Disk1 from Peg A to Peg C

(C)
Move Disk3 from Peg A to Peg C
Move Disk2 from Peg A to Peg B
Move Disk1 from Peg A to Peg B
Move Disk2 from Peg B to Peg C
Move Disk3 from Peg C to Peg B
Move Disk2 from Peg B to Peg C
Move Disk1 from Peg B to Peg C

(D)
Move Disk2 from Peg B to Peg C
Move Disk1 from Peg A to Peg C
Move Disk1 from Peg A to Peg B
Move Disk2 from Peg A to Peg C
Move Disk1 from Peg B to Peg C
Move Disk3 from Peg A to Peg B
Move Disk1 from Peg C to Peg A

28. Consider the incomplete **solveHanoi** method below.

```
public static void solveHanoi(char s; char t; char d; int n)
{
    if (n != 0)
    {
        // code segment
    }
}
```

Which of the following code segments complete the **solveHanoi** method correctly?

- (A) solveHanoi(s, d, t, n);
System.out.println("Move Disk " + n-1 + " from Peg " + s + " to Peg " + d);
solveHanoi(t, s, d, n);
- (B) solveHanoi(t, s, d, n-1);
System.out.println("Move Disk " + n + " from Peg " + s + " to Peg " + d);
solveHanoi(s, d, t, n-1);
- (C) System.out.println("Move Disk " + n + " from Peg " + s + " to Peg " + d);
solveHanoi(s, d, t, n-1);
solveHanoi(t, s, d, n-1);
- ### (D) solveHanoi(s, d, t, n-1);
System.out.println("Move Disk " + n + " from Peg " + s + " to Peg " + d);
solveHanoi(t, s, d, n-1);

Objective 5 – Why Recursion?

29. Does a recursive solution execute faster than an iterative solution?

- (A) Yes, recursion is always faster than iteration.
- (B) Most of the time recursion is faster than iteration.
- (C) No, recursion and iteration execute at the same speed.

(D) No, recursion is anywhere from slightly slower to a lot slower than iteration.

30. Does a recursive solution use less memory than an iterative solution?

- (A) Yes, recursion uses less memory than iteration.
- (B) Most of the time recursion uses more memory than iteration.
- (C) No, recursion and iteration use the same amount of memory.

(D) No, recursion uses more memory than iteration.

31. Is a recursive solution more readable than an iterative solution for the novice programmer?

- (A) Yes, recursion is more readable than iteration.
- (B) Most of the time recursion is more readable than iteration.
- (C) No, recursion and iteration are equally readable.

(D) No, recursion is less readable than iteration.

32. Why do you learn and use recursion?

- (A) Recursion is faster than iteration.
- (B) Recursion uses less memory than iteration.

(C) Recursion is easier to code than iteration for certain problems like the *Tower of Hanoi*.

(D) Recursion is more readable than iteration.

Objective 6 - The Fibonacci Sequence

33. If the first four numbers of the *Fibonacci Sequence* are: **1 1 2 3**
Which of the following is the 13th number in the sequence?

- (A) 12
- (B) 89
- (C) 144
- ### (D) 233

34. What is true about comparing the execution efficiency of the iterative and recursive solution of the *Fibonacci Sequence* for numbers greater than the 20th number?

- (A) The recursive solution is always faster than the iterative solution.
- ### (B) The iterative solution is always faster than the recursive solution.
- (C) The recursive solution is the same speed as the iterative solution.
- (D) The iterative solution is initially faster, but the recursive solution is faster for later numbers in the sequence.

35. What does **guess(6)** return ?

- (A) 1
- (B) 6
- (C) 16
- ### (D) 32
- (E) 64

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

36. Which one of the methods below correctly returns the n^{th} number in the *Fibonacci Sequence*?
NOTE: By “correctly” we are saying it works. We are not necessarily saying it works efficiently.

I.

```
public static int fibo(int n)
{
    int temp1 = 1;
    int temp2 = 1;
    int temp3 = 1;
    int k;
    for (k=3; k<=n; k++)
    {
        temp3 = temp1 + temp2;
        temp1 = temp2;
        temp2 = temp3;
    }
    return temp3;
}
```

II.

```
public static int fibo(int N)
{
    int temp1 = 1;
    int temp2 = 1;
    int temp3;
    int K;
    for (K=1; K<=N; K++)
    {
        temp3 = temp1 + temp2;
        temp1 = temp2;
        temp2 = temp3;
    }
    return temp3;
}
```

III.

```
public static int fibo(int n)
{
    if ((N == 1) || (N == 2))
        return 1;
    else
        return Fibo(n-1) + Fibo(n-2);
}
```

IV.

```
public static int fibo(int n)
{
    if ((N == 1) || (N == 2))
        return 1;
    else
        return fibo(N) + Fibo(N-1);
}
```

V.

```
public static int fibo(int n)
{
    int list[] = new int[n+1];
    for (int j=0; j<=n; j++)
        if (j < 2)
            list[j] = j;
        else
            list[j] = list[j-1] + list[j-2];
    return list[n];
}
```

- (A) I only
(B) I & II only
(C) I & III only
(D) II & IV only
(E) I, III & V only

Objective 7 - Challenging Recursive Method

Refer to this method for questions 37-40

```
public static int mystery(int a, int b)
{
    if (a == b)
        return a * b;
    else if (a % b == 0)
        return mystery(b,b);
    else if (a < b)
        return mystery(b,a);
    else
        return mystery(1,mystery(a-1,b+1));
}
```

37. What values does **mystery(5,5)** return?

- (A) 0 (B) 1 (C) 5 (D) 10 ### (E) 25

38. What values does **mystery(65,13)** return?

- (A) 5 (B) 13 (C) 65 ### (D) 169 (E) 4175

39. What values does **mystery(7,21)** return?

- (A) 3 (B) 7 (C) 21 ### (D) 49 (E) 441

40. What values does **mystery(29,13)** return?

- ### (A) 1 (B) 13 (C) 29 (D) 169 (E) 196

