ANSWER KEY

1. D	8. D	15. E
2. B	9. A	16. D
3. E	10. B	17. E
4. D	11. A	18. A
5. B	12. C	19. C
6. C	13. C	20. B
7. B	14. A	21. B

ANSWERS EXPLAINED

1. (D) Tail recursion is when the recursive call of a method is made as the last executable step of the method. Divide-and-conquer algorithms like those used in mergesort or quicksort have recursive calls *before* the last step. Thus, statement II is false.
2. (B) Code segment I is wrong because there is no base case. Code segment III is wrong because, besides anything else, `sum(n)` prevents the method from terminating—the base case `n == 1` will not be reached.
3. (E) When `stringRecur` is invoked, it calls itself irrespective of the length of `s`. Since there is no action that leads to termination, the method will not terminate until the computer runs out of memory (run-time error).
4. (D) The base case is `s.length()` $\geq$ 15. Since `s` gets longer on each method call, the method will eventually terminate. If the original length of `s` is $\geq$ 15, the method will terminate without output on the first call.
5. (B) Letting R denote the method `result`, we have

$$\begin{aligned} R(5) &= 2 * R(4) \\ &= 2 * (2 * (R(3))) \\ &= \cdots \\ &= 2 * (2 * (2 * (2 * R(1)))) \\ &= 2^5 \\ &= 32 \end{aligned}$$

6. (C) For `result(n)` there will be $(n-1)$ recursive calls before `result(1)`, the base case, is reached. Adding the initial call gives a total of n method calls.
7. (B) This method returns the nth term of an arithmetic sequence with first term `a` and common difference `d`. Letting M denote method `mystery`, we have

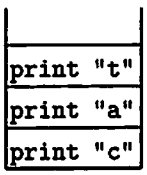
$$\begin{aligned} M(3,2,6) &= 6 + M(2,2,6) \\ &= 6 + (6 + M(1,2,6)) \quad \text{(base case)} \\ &= 6 + 6 + 2 \\ &= 14 \end{aligned}$$

8. (D) Here are the recursive calls that are made, in order: $f(6,8) \rightarrow f(6,2) \rightarrow f(4,2) \rightarrow f(2,2)$, base case. Thus, 2 is returned.

9. (A) If there is only one element in `x`, then `recur` returns that element. Having the recursive call at the beginning of the `else` part of the algorithm causes the `if` part for each method call to be stacked until `t` eventually gets assigned to `x[0]`. The pending `if` statements are then executed, and `t` is compared to each element in `x`. The largest value in `x` is returned.

10. (B) Since the recursive call is made directly following the base case, the `System.out.print...` statements are stacked up. If `printString("cat")` is called, here is the sequence of recursive calls and pending statements on the stack:

```
printString("at")  →  print "c"
printString("t")   →  print "a"
printString("")    →  print "t"
```



Execution stack

When `printString("")`, the base case, is called, the `print` statements are then popped off the stack in reverse order, which means that the characters of the string will be printed in reverse order.

11. (A) The required code is for a negative `expo`. For example, `power(2, -3)` should return $2^{-3} = 1/8$. Notice that

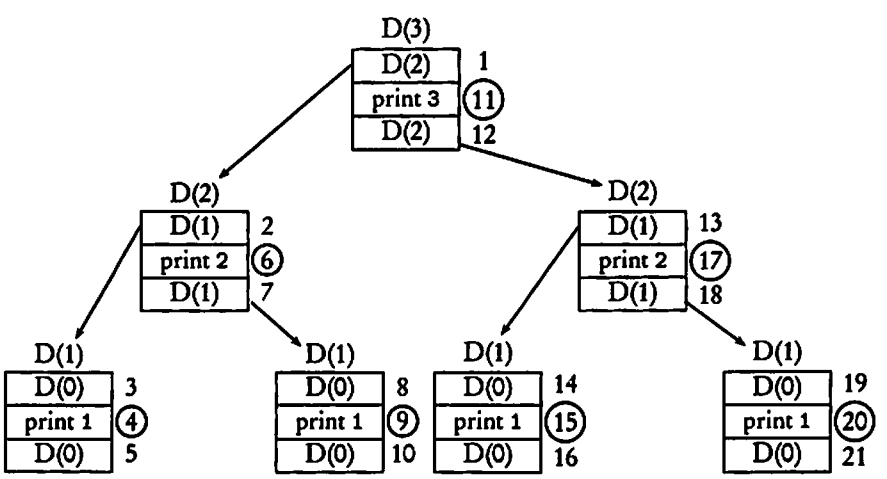
$$2^{-3} = \tfrac{1}{2}\left(2^{-2}\right)$$
$$2^{-2} = \tfrac{1}{2}\left(2^{-1}\right)$$
$$2^{-1} = \tfrac{1}{2}\left(2^{0}\right)$$

In general:

$$2^{n} = \tfrac{1}{2}(2^{n+1}) \quad \text{whenever} \quad n < 0$$

This is equivalent to `(1 / base) * power(base, expo + 1)`.

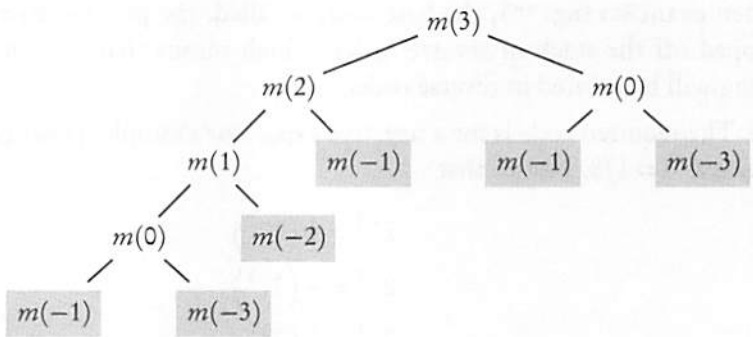
12. (C) Each box in the diagram below represents a recursive call to `doSomething`. The numbers to the right of the boxes show the order of execution of the statements. Let D denote `doSomething`.



The numbers in each box refer to that method call only. D(0) is the base case, so the statement immediately following it is executed next. When all statements in a given box (method call) have been executed, backtrack along the arrow to find the statement that gets executed next. The circled numbers represent the statements that produce output. Following them in order, statements 4, 6, 9, 11, 15, 17, and 20 produce the output in choice C.

13. (C) Since even numbers are printed *before* the recursive call in segment I, they will be printed in the order in which they are read from the keyboard. Contrast this with the correct choice, segment III, in which the recursive call is made before the test for evenness. These tests will be stacked until the last number is read. Recall that the pending statements are removed from the stack in reverse order (most recent recursive call first), which leads to even numbers being printed in reverse order. Segment II is wrong because all numbers entered will be printed, irrespective of whether they are even or not. Note that segment II would work if the input list contained only even numbers.

14. (A) Let `mystery(3)` be denoted $m(3)$. Picture the execution of the method as follows:

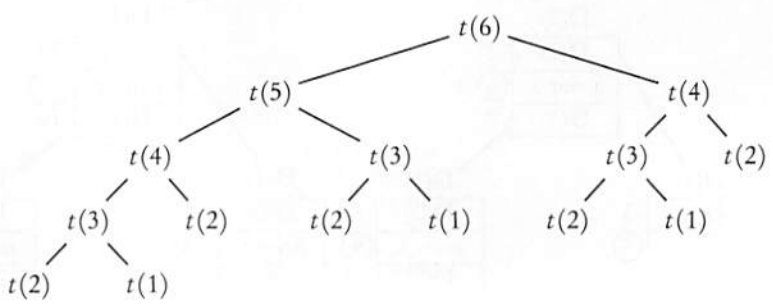


The base cases are shaded. Note that each of the six base case calls returns 2, resulting in a total of 12.

15. (E) The method generates a sequence. The first two terms, $t(1)$ and $t(2)$, are 2 and 4. Each subsequent term is generated by subtracting the previous two terms. This is the sequence: 2, 4, 2, −2, −4, −2, 2, 4, ... Thus, $t(5) = -4$. Alternatively,

$$\begin{aligned} t(5) &= t(4) - t(3) \\ &= [t(3) - t(2)] - t(3) \\ &= -t(2) \\ &= -4 \end{aligned}$$

16. (D) 15. Count them! (Note that you stop at $t(2)$ since it's a base case.)



17. (E) This is an example of *mutual recursion*, where two methods call each other.

$$\begin{aligned} f_1(5,3) &= 5 + f_2(4,3) \\ &= 5 + (4 + f_1(2,3)) \\ &= 5 + (4 + (2 + f_2(1,3))) \\ &= 5 + (4 + (2 + 4)) \\ &= 15 \end{aligned}$$

Note that $f_2(1,3)$ is a base case.

18. (A) foo(3) = 3 (This is a base case). Also, foo(4) = 4 × foo(3) = 12. So you need to find foo(foo(3) + foo(4)) = foo(15).

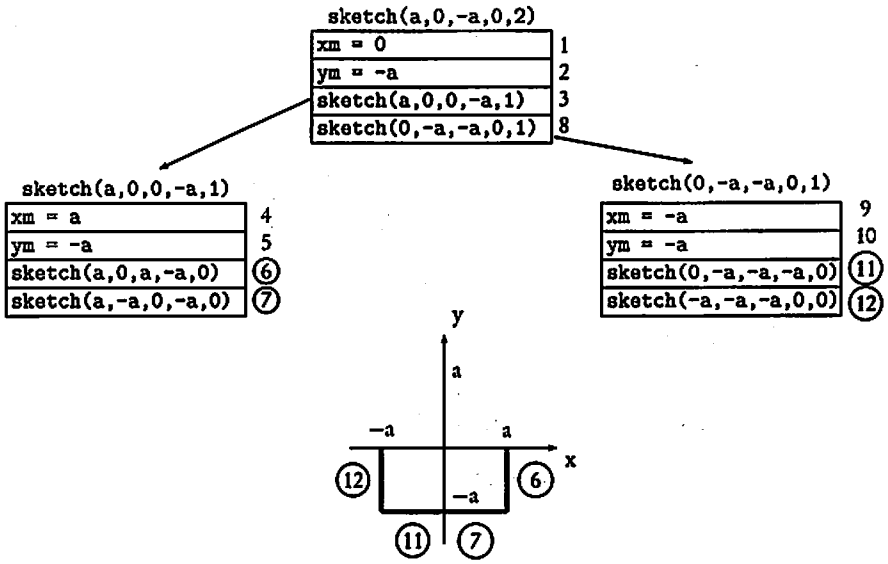
$$\begin{aligned} \text{foo}(15) &= 15 \times \text{foo}(14) \\ &= 15 \times (14 \times \text{foo}(13)) \\ &= \cdots \\ &= 15 \times 14 \times \cdots \times 4 \times \text{foo}(3) \\ &= 15 \times 14 \times \cdots \times 4 \times 3 \\ &= (15)!/(2!) \end{aligned}$$

19. (C) Suppose that $n = 365051$. The method call `writeWithCommas(365051)` will write `051` and then execute the call `writeWithCommas(365)`. This is a base case, so `365` will be written out, resulting in `051,365`. A number like 278278 (two sets of three identical digits) will be written out correctly, as will a "symmetrical" number like 251462251. Also, any $n < 1000$ is a base case and the number will be written out correctly as is.

20. (B) The cause of the problem is that the numbers are being written out with the sets of three digits in the wrong order. The problem is fixed by interchanging `writeThreeDigits(n % 1000)` and `writeWithCommas(n / 1000)`. For example, here is the order of execution for `writeWithCommas(365051)`.

    `writeWithCommas(365)` → Base case. Writes `365`
    `System.out.print(",");` → `365,`
    `writeThreeDigits(051)` → `365,051` which is correct

21. (B) Here is the "box diagram" for the recursive method calls, showing the order of execution of statements. Notice that the circled statements are the base case calls, the only statements that actually draw a line. Note also that the first time you reach a base case (see circled statement 6), you can get the answer: The picture in choice B is the only one that has a line segment joining `(a,0)` to `(a,-a)`.


sketch(a,0,-a,0,2)
xm = 0
ym = -a
sketch(a,0,0,-a,1)
sketch(0,-a,-a,0,1)
1
2
3
8
sketch(a,0,0,-a,1)
xm = a
ym = -a
sketch(a,0,a,-a,0)
sketch(a,-a,0,-a,0)
4
5
6
7
sketch(0,-a,-a,0,1)
xm = -a
ym = -a
sketch(0,-a,-a,-a,0)
sketch(-a,-a,-a,0,0)
9
10
11
12
y
a
-a
a
x
-a
12
6
11
7