Chapter 1 Test Bank

Java Data Structures

1. How many solutions are possible for a problem?

a. Multiple

b. 1

c. 0

d. One for each possible Big O category

Analysis:

1. Correct. There are many approaches for implementing an algorithm to solve a problem. See Module 1: Algorithms and Complexities, Lesson 1.1: Developing Our First Algorithm.
2. Incorrect. There is the most efficient algorithm, but other algorithms for a problem are also possible. See Module 1: Algorithms and Complexities, Lesson 1.1: Developing Our First Algorithm.
3. Incorrect. There is always an algorithm for a problem, although some algorithms are very inefficient. See Module 1: Algorithms and Complexities, Lesson 1.1: Developing Our First Algorithm.
4. Incorrect. Different algorithms exist and have different Big O performance, but not all possible Big O performance have an algorithm. See Module 1: Algorithms and Complexities, Lesson 1.1: Developing Our First Algorithm.

2. What is used to determine the right algorithm for a problem?

a. Performance and memory requirements

b. Runtime platform

c. Programming language

d. Type of data input

Analysis:

1. Correct. The best metric is performance, and then memory efficiency, in choosing an algorithm for a problem. See Module 1: Algorithms and Complexities, Lesson 1.1: Developing Our First Algorithm.
2. Incorrect. The metrics of an algorithm are independent of the runtime platform. See Module 1: Algorithms and Complexities, Lesson 1.1: Developing Our First Algorithm.
3. Incorrect. Any programming language can implement an algorithm. See Module 1: Algorithms and Complexities, Lesson 1.1: Developing Our First Algorithm.
4. Incorrect. An algorithm is described and evaluated by the Big O performance and memory requirements, not the type of data processed. See Module 1: Algorithms and Complexities, Lesson 1.1: Developing Our First Algorithm.

3. Which of the following is a disadvantage of the Minimum Distance algorithm?

a. The algorithm is inefficient for huge amount of data

b. The algorithm uses too much memory

c. The algorithm needs a faster runtime platform

d. It computes the maximum distance

Analysis:

1. Correct. The Minimum Distance algorithm cannot handle or process huge amount of input data – it becomes inefficient. See Module 1: Algorithms and Complexities, Lesson 1.2: Measuring Algorithmic Complexity with Big O Notation.
2. Incorrect. The algorithm performance is slow and is not related to memory use. See Module 1: Algorithms and Complexities, Lesson 1.2: Measuring Algorithmic Complexity with Big O Notation.
3. Incorrect. The algorithm is inefficient because of increased input data size and is not related to the runtime platform. See Module 1: Algorithms and Complexities, Lesson 1.2: Measuring Algorithmic Complexity with Big O Notation.
4. Incorrect. The minimum distance is computed, but the algorithm becomes slow and inefficient for increased input data size. See Module 1: Algorithms and Complexities, Lesson 1.2: Measuring Algorithmic Complexity with Big O Notation.

4. What is the effect of increased load on an algorithm?

a. Efficient algorithms will not increase their use of resources

b. Performance degrades

c. Algorithm has a different Big O complexity

d. Memory use increases

Analysis:

1. Correct. The most efficient algorithm will slowly increase their use of resources while not degrading in performance and speed. See Module 1: Algorithms and Complexities, Lesson 1.2: Measuring Algorithmic Complexity with Big O Notation.
2. Incorrect. It depends on the algorithm – some algorithms perform better with increased load. See Module 1: Algorithms and Complexities, Lesson 1.2: Measuring Algorithmic Complexity with Big O Notation.
3. Incorrect. The Big O complexity of an algorithm for the best, average, and worst cases is immutable. See Module 1: Algorithms and Complexities, Lesson 1.2: Measuring Algorithmic Complexity with Big O Notation.
4. Incorrect. It usually depends on the algorithm if more memory is required for a larger load. See Module 1: Algorithms and Complexities, Lesson 1.2: Measuring Algorithmic Complexity with Big O Notation.

5. Which of the following is a method for quickly determining the efficiency of an algorithm?

a. Plot the relation between the load size and the resource use for an algorithm

b. Run the algorithm

c. Check memory use

d. Compare with the best known algorithm

Analysis:

1. Correct. A plot of problem size to time quickly reveals the functional curve that identifies the algorithm efficiency. See Module 1: Algorithms and Complexities, Lesson 1.2: Measuring Algorithmic Complexity with Big O Notation.
2. Incorrect. Running the algorithm tests it, but the data for size and time are needed to compare with a plot. See Module 1: Algorithms and Complexities, Lesson 1.2: Measuring Algorithmic Complexity with Big O Notation.
3. Incorrect. Memory utilization is a characteristic of an algorithm, but time performance is the most important factor. See Module 1: Algorithms and Complexities, Lesson 1.2: Measuring Algorithmic Complexity with Big O Notation.
4. Incorrect. The best known algorithm is only the best known until a better algorithm is created. See Module 1: Algorithms and Complexities, Lesson 1.2: Measuring Algorithmic Complexity with Big O Notation.

6. What is described by a runtime complexity of O(1)?

a. Algorithm efficiency is independent of the problem size, and is the fastest

b. There is a limit on performance

c. Memory use is inefficient

d. Constant 1 needs to be removed

Analysis:

1. Correct. A constant time algorithm is the fastest regardless of the size of the data input. See Module 1: Algorithms and Complexities, Lesson 1.2: Measuring Algorithmic Complexity with Big O Notation.
2. Incorrect. Linear O(1) algorithms can scale to any problem size. See Module 1: Algorithms and Complexities, Lesson 1.2: Measuring Algorithmic Complexity with Big O Notation.
3. Incorrect. Linear O(1) algorithms' performance is not necessarily related to input size of the problem. See Module 1: Algorithms and Complexities, Lesson 1.2: Measuring Algorithmic Complexity with Big O Notation.
4. Incorrect. O(1) is constant with no higher terms in the Big O notation. See Module 1: Algorithms and Complexities, Lesson 1.2: Measuring Algorithmic Complexity with Big O Notation.

7. Which of the following are polynomial algorithm runtime complexities?

a. O(n^3) and O(n^4)

b. O(log n) and O(n)

c. O(1) and O(n)

d. O(1) and O(log n)

Analysis:

1. Correct. Many algorithms are polynomial in performance but have different performance and speed when executed. See Module 1: Algorithms and Complexities, Lesson 1.2: Measuring Algorithmic Complexity with Big O Notation.
2. Incorrect. A polynomial complexity is denoted as O(n^k), where k is non-fractional integer. See Module 1: Algorithms and Complexities, Lesson 1.2: Measuring Algorithmic Complexity with Big O Notation.
3. Incorrect. A polynomial complexity is denoted as O(n^k), where k is non-fractional integer. See Module 1: Algorithms and Complexities, Lesson 1.2: Measuring Algorithmic Complexity with Big O Notation.
4. Incorrect. A polynomial complexity is denoted as O(n^k), where k is non-fractional integer. See Module 1: Algorithms and Complexities, Lesson 1.2: Measuring Algorithmic Complexity with Big O Notation.

8. Which of the following algorithm complexity is slower than polynomial complexity?

a. O(k^n)

b. O(n)

c. Product of two linear algorithms

d. O(1)

Analysis:

1. Correct. The slower algorithms are exponential and factorial algorithms. See Module 1: Algorithms and Complexities, Lesson 1.2: Measuring Algorithmic Complexity with Big O Notation.
2. Incorrect. Linear O(n) algorithms have faster efficiency. See Module 1: Algorithms and Complexities, Lesson 1.2: Measuring Algorithmic Complexity with Big O Notation.
3. Incorrect. The product of two linear algorithms is O(n^2), which is polynomial hence not necessarily slower. See Module 1: Algorithms and Complexities, Lesson 1.2: Measuring Algorithmic Complexity with Big O Notation.
4. Incorrect. A constant O(1) algorithm is faster even for a large input data size. See Module 1: Algorithms and Complexities, Lesson 1.2: Measuring Algorithmic Complexity with Big O Notation.

9. Which algorithm efficiency is slower on a smaller input?

a. Logarithmic algorithm efficiency

b. Constant algorithm efficiency

c. Linear algorithm efficiency

d. Polynomial algorithm efficiency

Analysis:

1. Correct. Often, the more efficient algorithms are terrible for small problem sizes. See Module 1: Algorithms and Complexities, Lesson 1.2: Measuring Algorithmic Complexity with Big O Notation.
2. Incorrect. A constant O(1) algorithm is faster for smaller input. See Module 1: Algorithms and Complexities, Lesson 1.2: Measuring Algorithmic Complexity with Big O Notation.
3. Incorrect. A linear algorithm is faster for smaller input. See Module 1: Algorithms and Complexities, Lesson 1.2: Measuring Algorithmic Complexity with Big O Notation.
4. Incorrect. Polynomial is a class of Big O algorithms but is not a specific efficiency. See Module 1: Algorithms and Complexities, Lesson 1.2: Measuring Algorithmic Complexity with Big O Notation.

10. Can a logarithmic algorithm O(log n) outperform a linear algorithm O(n)?

a. Yes, for a large enough problem size

b. Yes, for a small problem size

c. Yes, for sorted input data

d. Yes, for duplicate input data

Analysis:

1. Correct. An example is sorting algorithms where until a certain size is reached, the less efficient algorithm is faster; for example, insertion sort or bubble sort compared to quick sort. See Module 1: Algorithms and Complexities, Lesson 1.2: Measuring Algorithmic Complexity with Big O Notation.
2. Incorrect. For a small problem size, a linear algorithm will outperform a logarithmic one. See Module 1: Algorithms and Complexities, Lesson 1.2: Measuring Algorithmic Complexity with Big O Notation.
3. Incorrect. The data input being sorted is irrelevant to the performance comparison between a logarithmic and linear algorithm. See Module 1: Algorithms and Complexities, Lesson 1.2: Measuring Algorithmic Complexity with Big O Notation.
4. Incorrect. Duplicate input data does not affect the performance. See Module 1: Algorithms and Complexities, Lesson 1.2: Measuring Algorithmic Complexity with Big O Notation.

11. What does the Big O complexity notation O(n) mean?

a. It means the algorithm efficiency is proportionate to the problem size of the input data or n

b. It means the algorithm requires data input of size n

c. It means the algorithm uses memory of size n

d. It means the algorithm has an integer n performance

Analysis:

1. Correct. The linear algorithms are O(n), where algorithm performance is tied to problem size. See Module 1: Algorithms and Complexities, Lesson 1.2: Measuring Algorithmic Complexity with Big O Notation.
2. Incorrect. The data input size is n, but O(n) indicates the performance relative to the input data size. See Module 1: Algorithms and Complexities, Lesson 1.2: Measuring Algorithmic Complexity with Big O Notation.
3. Incorrect. The Big O complexity indicates the algorithm performance relative to the input data size. See Module 1: Algorithms and Complexities, Lesson 1.2: Measuring Algorithmic Complexity with Big O Notation.
4. Incorrect. The Big O complexity is expressed in terms of the data input size, not a specific integer size value. See Module 1: Algorithms and Complexities, Lesson 1.2: Measuring Algorithmic Complexity with Big O Notation.

12. How is the average case efficiency determined?

a. By calculating the average result over all possible inputs

b. By calculating the average of the best and worst case

c. By calculating the square root of the best case

d. By calculating the difference between the best case and worst case

Analysis:

1. Correct. Calculating the average case is easier in practice than in theoretical analysis at times. See Module 1: Algorithms and Complexities, Lesson 1.2: Measuring Algorithmic Complexity with Big O Notation.
2. Incorrect. The average case is calculated from the results of all possible input sizes. See Module 1: Algorithms and Complexities, Lesson 1.2: Measuring Algorithmic Complexity with Big O Notation.
3. Incorrect. The best case has nothing to do with determining the average case. See Module 1: Algorithms and Complexities, Lesson 1.2: Measuring Algorithmic Complexity with Big O Notation.
4. Incorrect. The average case is calculated from the results of all possible input sizes. See Module 1: Algorithms and Complexities, Lesson 1.2: Measuring Algorithmic Complexity with Big O Notation.

13. Why is the average case for algorithm efficiency useful?

a. When there is a chance of the worse case occurring in practice

b. To find the best and worst case

c. To determine the usage of memory

d. To know all the different performance cases for an algorithm

Analysis:

1. Correct. The average case gives the general performance of an algorithm. See Module 1: Algorithms and Complexities, Lesson 1.2: Measuring Algorithmic Complexity with Big O Notation.
2. Incorrect. The best and worst cases are determined independently of the average case. See Module 1: Algorithms and Complexities, Lesson 1.2: Measuring Algorithmic Complexity with Big O Notation.
3. Incorrect. Memory use is independent of the average case of an algorithm. See Module 1: Algorithms and Complexities, Lesson 1.2: Measuring Algorithmic Complexity with Big O Notation.
4. Incorrect. The average case is determined independently of the best and worst cases. See Module 1: Algorithms and Complexities, Lesson 1.2: Measuring Algorithmic Complexity with Big O Notation.

14. What case of algorithm efficiency is used when measuring and comparing different algorithms?

a. Worst case

b. Average case

c. Infinite case

d. Best case

Analysis:

1. Correct. The worst case Big O of an algorithm gives the slowest, most inefficient performance of that algorithm. See Module 1: Algorithms and Complexities, Lesson 1.2: Measuring Algorithmic Complexity with Big O Notation.
2. Incorrect. Average case is used to understand an algorithm, but the worst possible scenario is used to measure an algorithm. See Module 1: Algorithms and Complexities, Lesson 1.2: Measuring Algorithmic Complexity with Big O Notation.
3. Incorrect. There is no infinite case of algorithm efficiency. See Module 1: Algorithms and Complexities, Lesson 1.2: Measuring Algorithmic Complexity with Big O Notation.
4. Incorrect. The best case is used to understand the maximum possible performance of an algorithm. See Module 1: Algorithms and Complexities, Lesson 1.2: Measuring Algorithmic Complexity with Big O Notation.

15. What are the two rules for determining the Big O efficiency of an algorithm?

a. Drop any constants in the equation and use only the highest order term

b. Find the maximum case and then the minimum case

c. Multiply and simplify the terms and constants

d. Add the smallest terms and subtract constants

Analysis:

1. Correct. These two rules help to simplify algorithmic Big O expressions. See Module 1: Algorithms and Complexities, Lesson 1.2: Measuring Algorithmic Complexity with Big O Notation.
2. Incorrect. The maximum and minimum cases are extremes of input data. See Module 1: Algorithms and Complexities, Lesson 1.2: Measuring Algorithmic Complexity with Big O Notation.
3. Incorrect. Only the highest term is used. See Module 1: Algorithms and Complexities, Lesson 1.2: Measuring Algorithmic Complexity with Big O Notation.
4. Incorrect. We remove constants and use the highest term. See Module 1: Algorithms and Complexities, Lesson 1.2: Measuring Algorithmic Complexity with Big O Notation.

16. What is the Big O efficiency for the expression n(1 + log n + n)?

a. O(n^2)

b. O(n+n log n + n^2)

c. O(1)

d. O(n log n)

Analysis:

1. Correct. First, drop the expression constant 1 to get n(log n + n). Then, multiply to get n log n + n^2. Drop the lower order term n log n to get n^2. See Module 1: Algorithms and Complexities, Lesson 1.2: Measuring Algorithmic Complexity with Big O Notation.
2. Incorrect. The Big O notation uses only the highest order term of O(n^2). See Module 1: Algorithms and Complexities, Lesson 1.2: Measuring Algorithmic Complexity with Big O Notation.
3. Incorrect. The Big O notation removes constants from the expression. See Module 1: Algorithms and Complexities, Lesson 1.2: Measuring Algorithmic Complexity with Big O Notation.
4. Incorrect. The Big O notation removes constants and uses the highest term, not the lowest. See Module 1: Algorithms and Complexities, Lesson 1.2: Measuring Algorithmic Complexity with Big O Notation.

17. What is the Big O for a constant performance algorithm efficiency?

a. O(1)

b. O(0)

c. O(n)

d. O(n^2)

Analysis:

1. Correct. This is the best and simplest Big O, often stated as O(c) for constant time performance. There are no higher order terms to use, and there is only a constant being added or multiplied with a higher order term. Thus the constant 1 is not dropped. See Module 1: Algorithms and Complexities, Lesson 1.2: Measuring Algorithmic Complexity with Big O Notation.
2. Incorrect. This Big O notation makes no sense and is not possible. See Module 1: Algorithms and Complexities, Lesson 1.2: Measuring Algorithmic Complexity with Big O Notation.
3. Incorrect. This is the Big O for a linear performance algorithm. See Module 1: Algorithms and Complexities, Lesson 1.2: Measuring Algorithmic Complexity with Big O Notation.
4. Incorrect. A polynomial Big O does not show constant performance. See Module 1: Algorithms and Complexities, Lesson 1.2: Measuring Algorithmic Complexity with Big O Notation.

18. Which of the following characteristics can be used to identify a linear algorithm?

a. It typically involves a single pass through the input

b. The input data size is n

c. It has one loop

d. It is not recursive

Analysis:

1. Correct. A single pass indicates processing the input data elements only once. See Module 1: Algorithms and Complexities, Lesson 1.3: Identifying Algorithms with Different Complexities.
2. Incorrect. The input data size for any algorithm is n. See Module 1: Algorithms and Complexities, Lesson 1.3: Identifying Algorithms with Different Complexities.
3. Incorrect. A linear algorithm can use multiple loops in implementation. See Module 1: Algorithms and Complexities, Lesson 1.3: Identifying Algorithms with Different Complexities.
4. Incorrect. A linear algorithm can be recursive in operation. See Module 1: Algorithms and Complexities, Lesson 1.3: Identifying Algorithms with Different Complexities.

19. How is the input data size significant for a linear algorithm?

a. Work done grows in direct proportion to the problem size

b. The input data has no duplicate elements

c. Memory is doubled with increasing problem size

d. The data input is sorted

Analysis:

1. Correct. A linear algorithm is efficient but for peta- and tera-byte sized data input, more efficiency is needed. See Module 1: Algorithms and Complexities, Lesson 1.3: Identifying Algorithms with Different Complexities.
2. Incorrect. Duplicate elements are immaterial to the problem size, which is used to determine the linear algorithm performance. See Module 1: Algorithms and Complexities, Lesson 1.3: Identifying Algorithms with Different Complexities.
3. Incorrect. A linear algorithm does not necessarily use double the memory with increasing problem size. See Module 1: Algorithms and Complexities, Lesson 1.3: Identifying Algorithms with Different Complexities.
4. Incorrect. A linear algorithm does not necessarily require or sort data input. See Module 1: Algorithms and Complexities, Lesson 1.3: Identifying Algorithms with Different Complexities.

20. When are quadratic O(n^k) algorithms performant?

a. On smaller input data sizes

b. On input data size of zero

c. On input data size of one element

d. On duplicate data input

Analysis:

1. Correct. Smaller data input size leads to more efficient quadratic algorithms. See Module 1: Algorithms and Complexities, Lesson 1.3: Identifying Algorithms with Different Complexities.
2. Incorrect. For no input data, there is no reason to use an algorithm. See Module 1: Algorithms and Complexities, Lesson 1.3: Identifying Algorithms with Different Complexities.
3. Incorrect. Using an algorithm on one element does nothing to the input data. See Module 1: Algorithms and Complexities, Lesson 1.3: Identifying Algorithms with Different Complexities.
4. Incorrect. Duplicate data that is sufficiently large enough will lead to the worst-case scenario. See Module 1: Algorithms and Complexities, Lesson 1.3: Identifying Algorithms with Different Complexities.

21. What is the overall best case for space complexity?

a. An algorithm that uses zero or no memory

b. An algorithm that uses the same space as the input data size

c. An algorithm working on data size of one element

d. An algorithm working on duplicate elements

Analysis:

1. Correct. In practice, no algorithm will ever use zero memory space since then there would be no data to process. See Module 1: Algorithms and Complexities, Lesson 1.3: Identifying Algorithms with Different Complexities.
2. Incorrect. A Big O of O(n) for the input data size is actual but not theoretically the best case. See Module 1: Algorithms and Complexities, Lesson 1.3: Identifying Algorithms with Different Complexities.
3. Incorrect. The best case is no memory, which means no elements or data input of length zero. See Module 1: Algorithms and Complexities, Lesson 1.3: Identifying Algorithms with Different Complexities.
4. Incorrect. The best case is no memory, which means no elements or data input of length zero. See Module 1: Algorithms and Complexities, Lesson 1.3: Identifying Algorithms with Different Complexities.

22. How is the internal operation of a logarithmic algorithm detected?

a. If an algorithm divides the input data in several steps

b. If an algorithm doubles the amount of memory

c. If an algorithm uses a minimum and maximum value

d. If an algorithm sorts the input data

Analysis:

1. Correct. The logarithmic algorithm divides and sub-divides the input data often recursively. See Module 1: Algorithms and Complexities, Lesson 1.3: Identifying Algorithms with Different Complexities.
2. Incorrect. Memory is unrelated to performance, and the memory use in such a case is exponential O(k^n). See Module 1: Algorithms and Complexities, Lesson 1.3: Identifying Algorithms with Different Complexities.
3. Incorrect. The minimum and maximum elements of the input data are not relevant to algorithm performance, which is logarithmic. See Module 1: Algorithms and Complexities, Lesson 1.3: Identifying Algorithms with Different Complexities.
4. Incorrect. For the binary search algorithm, the input data needs to be sorted, but this is not a logarithmic algorithm. See Module 1: Algorithms and Complexities, Lesson 1.3: Identifying Algorithms with Different Complexities.

23. How is the data input size determined for a logarithmic O(log n) algorithm?

a. By using the inverse of the logarithm function

b. By using the logarithm function

c. By using a polynomial function

d. By multiplying the time and input data size

Analysis:

1. Correct. For input size n with log(n) = m, the data size is 2^m. See Module 1: Algorithms and Complexities, Lesson 1.3: Identifying Algorithms with Different Complexities.
2. Incorrect. We use the inverse function to a logarithm, which is the exponential function. See Module 1: Algorithms and Complexities, Lesson 1.3: Identifying Algorithms with Different Complexities.
3. Incorrect. We use the inverse function to a logarithm, which is the exponential function. See Module 1: Algorithms and Complexities, Lesson 1.3: Identifying Algorithms with Different Complexities.
4. Incorrect. We use the inverse function to a logarithm, which is the exponential function. See Module 1: Algorithms and Complexities, Lesson 1.3: Identifying Algorithms with Different Complexities.

24. Which of the following happens when the input data size grows for a logarithmic algorithm of O(log n) performance?

a. Performance degradation slows

b. Memory use increases

c. The algorithm becomes slower

d. The input data size is reduced

Analysis:

1. Correct. As the input gets bigger, the rate of performance degradation gets smaller. See Module 1: Algorithms and Complexities, Lesson 1.3: Identifying Algorithms with Different Complexities.
2. Incorrect. The memory use of a logarithmic algorithm is not necessarily related to the Big O performance efficiency. See Module 1: Algorithms and Complexities, Lesson 1.3: Identifying Algorithms with Different Complexities.
3. Incorrect. A logarithmic algorithm performs more efficiently with increasing input size. See Module 1: Algorithms and Complexities, Lesson 1.3: Identifying Algorithms with Different Complexities.
4. Incorrect. The input data size remains a constant parameter. See Module 1: Algorithms and Complexities, Lesson 1.3: Identifying Algorithms with Different Complexities.

25. Which of the following happens when the input data size grows for an exponential algorithm of O(k^n) performance?

a. Performance degradation greatly increases

b. The performance averages out

c. Memory use decreases

d. The algorithm becomes faster

Analysis:

1. Correct. The classic, intractable problems of computer science often have only an exponential algorithm as a solution, such as the traveling salesman problem. See Module 1: Algorithms and Complexities, Lesson 1.3: Identifying Algorithms with Different Complexities.
2. Incorrect. For an exponential algorithm, the performance quickly degrades with increasing input size. See Module 1: Algorithms and Complexities, Lesson 1.3: Identifying Algorithms with Different Complexities.
3. Incorrect. The memory use of an exponential algorithm is not necessarily related to the worst case performance. See Module 1: Algorithms and Complexities, Lesson 1.3: Identifying Algorithms with Different Complexities.
4. Incorrect. An exponential algorithm has bad performance metrics, more so for a larger input data size. See Module 1: Algorithms and Complexities, Lesson 1.3: Identifying Algorithms with Different Complexities.

26. When is an exponential algorithm of O(k^n) performance used?

a. If the input data size is small enough and the worst case is not guaranteed to occur

b. If we require average case performance

c. If we require non-exponential memory use

d. If an algorithm is needed quickly

Analysis:

1. Correct. For a small problem size, an exponential algorithm can be used reasonably. See Module 1: Algorithms and Complexities, Lesson 1.3: Identifying Algorithms with Different Complexities.
2. Incorrect. The exponential algorithm has the worst Big O efficiency. See Module 1: Algorithms and Complexities, Lesson 1.3: Identifying Algorithms with Different Complexities.
3. Incorrect. An exponential algorithm is the worst option for Big O efficiency and performance. See Module 1: Algorithms and Complexities, Lesson 1.3: Identifying Algorithms with Different Complexities.
4. Incorrect. Many algorithms exist for a problem that have better efficiency in most cases. See Module 1: Algorithms and Complexities, Lesson 1.3: Identifying Algorithms with Different Complexities.

27. Why is the constant O(1) algorithm desirable?

a. Such algorithms are very efficient and fast

b. Because the constant can be removed

c. Because the data input size is constant

d. Because there is only one data element

Analysis:

1. Correct. No matter how large the input data, such an algorithm remains constant in performance. See Module 1: Algorithms and Complexities, Lesson 1.3: Identifying Algorithms with Different Complexities.
2. Incorrect. The Big O of a constant algorithm is as simple as possible. See Module 1: Algorithms and Complexities, Lesson 1.3: Identifying Algorithms with Different Complexities.
3. Incorrect. The Big O of a constant algorithm is for an input data of size n. See Module 1: Algorithms and Complexities, Lesson 1.3: Identifying Algorithms with Different Complexities.
4. Incorrect. The input size is n for a constant algorithm of Big O O(n). See Module 1: Algorithms and Complexities, Lesson 1.3: Identifying Algorithms with Different Complexities.

28. Which of the following is an example of an exponential or Big O (k^n) algorithm?

a. The traveling salesman problem

b. The bubble sort algorithm

c. The binary search algorithm

d. Array access

Analysis:

1. Correct. An exponential algorithm is often an exhaustive generation of possible solutions that are checked as generated. See Module 1: Algorithms and Complexities, Lesson 1.3: Identifying Algorithms with Different Complexities.
2. Incorrect. The bubble sort is a quadratic O(n^2) algorithm. See Module 1: Algorithms and Complexities, Lesson 1.3: Identifying Algorithms with Different Complexities.
3. Incorrect. The binary search is a logarithmic O(log n) algorithm. See Module 1: Algorithms and Complexities, Lesson 1.3: Identifying Algorithms with Different Complexities.
4. Incorrect. Array access is a constant O(1) algorithm. See Module 1: Algorithms and Complexities, Lesson 1.3: Identifying Algorithms with Different Complexities.

29. What is quick sort an example of?

a. Algorithm

b. Pseudocode

c. Computer program

d. None of the above

Analysis:

1. Correct. Quick sort is one of many sorting algorithms. See Module 1: Algorithms and Complexities, Lesson 1.1: Developing Our First Algorithm.
2. Incorrect. Pseudocode is a plain English statement for better understanding of what should be programmed – it is an intermediary between an algorithm and a computer program. See Module 1: Algorithms and Complexities, Lesson 1.1: Developing Our First Algorithm.
3. Incorrect. A computer program is often an implementation of the algorithm, not the algorithm itself. See Module 1: Algorithms and Complexities, Lesson 1.1: Developing Our First Algorithm.
4. Incorrect. The correct answer is algorithm. See Module 1: Algorithms and Complexities, Lesson 1.1: Developing Our First Algorithm.

30. What is the “Big O” an example of?

a. Algorithm complexity notation

b. Sorting algorithm

c. Data structure

d. String matching algorithm

Analysis:

1. Correct. Big O is one way of describing the complexity of an algorithm. See Module 1: Algorithms and Complexities, Lesson 1.2: Measuring Algorithmic Complexity with Big O Notation.
2. Incorrect. Big O is the name of the notation used for describing algorithm complexity. See Module 1: Algorithms and Complexities, Lesson 1.2: Measuring Algorithmic Complexity with Big O Notation.
3. Incorrect. Big O is the name of the notation used for describing algorithm complexity. A data structure is, for example, List or Map. See Module 1: Algorithms and Complexities, Lesson 1.2: Measuring Algorithmic Complexity with Big O Notation.
4. Incorrect. Big O is the name of the notation used for describing algorithm complexity. See Module 1: Algorithms and Complexities, Lesson 1.2: Measuring Algorithmic Complexity with Big O Notation.

31. What is the difference between an algorithm with a higher complexity and an algorithm with lower complexity?

a. It is less efficient

b. It is more efficient

c. It is harder to implement

d. It is easier to implement

Analysis:

1. Correct. You should keep algorithm complexity as low as possible. See Module 1: Algorithms and Complexities, Lesson 1.2: Measuring Algorithmic Complexity with Big O Notation.
2. Incorrect. Higher complexity means lower efficiency. See Module 1: Algorithms and Complexities, Lesson 1.2: Measuring Algorithmic Complexity with Big O Notation.
3. Incorrect. Complexity has no relation to difficulty of implementation of a given algorithm. See Module 1: Algorithms and Complexities, Lesson 1.2: Measuring Algorithmic Complexity with Big O Notation.
4. Incorrect. Complexity has no relation to difficulty of implementation of a given algorithm. See Module 1: Algorithms and Complexities, Lesson 1.2: Measuring Algorithmic Complexity with Big O Notation.

32. Which of the following is an example of an algorithm?

a. Cooking recipe

b. Conversation between people

c. Result of computation

d. Algebraic expression

Analysis:

1. Correct. An algorithm is a set of instructions to perform a particular task. See Module 1: Algorithms and Complexities, Lesson 1.2: Measuring Algorithmic Complexity with Big O Notation.
2. Incorrect. An algorithm is a set of instructions to perform a particular task. See Module 1: Algorithms and Complexities, Lesson 1.2: Measuring Algorithmic Complexity with Big O Notation.
3. Incorrect. An algorithm is a set of instructions to perform a particular task. See Module 1: Algorithms and Complexities, Lesson 1.2: Measuring Algorithmic Complexity with Big O Notation.
4. Incorrect. An algorithm is a set of instructions to perform a particular task. See Module 1: Algorithms and Complexities, Lesson 1.2: Measuring Algorithmic Complexity with Big O Notation.

33. What is algorithm complexity?

a. A way to describe the efficiency of an algorithm, often by equation

b. Number of single instructions used in the algorithm description

c. Name of an algorithm for sorting numbers

d. Algorithm written with pseudocode

Analysis:

1. Correct. Complexity gives us information on how many steps are needed to be performed to finish a given task. See Module 1: Algorithms and Complexities, Lesson 1.2: Measuring Algorithmic Complexity with Big O Notation.
2. Incorrect. In fact, the number of steps/instructions used in the description doesn't matter. What is important for us is efficiency (number of operations done by the computer). See Module 1: Algorithms and Complexities, Lesson 1.2: Measuring Algorithmic Complexity with Big O Notation.
3. Incorrect. There is no algorithm named such. See Module 1: Algorithms and Complexities, Lesson 1.2: Measuring Algorithmic Complexity with Big O Notation.
4. Incorrect. By writing an algorithm with pseudocode, we are just getting one step closer to the final implementation – a computer program. See Module 1: Algorithms and Complexities, Lesson 1.2: Measuring Algorithmic Complexity with Big O Notation.

34. If an algorithm needs 4 seconds to run on a set of 2 input data, 9 seconds for 3, 16 for 4, and so on, then what is the complexity of this algorithm?

a. Quadratic

b. Linear

c. Logarithmic

d. This can't be determined

Analysis:

1. Correct. 2^2 = 4 and 3^2 = 9. See Module 1: Algorithms and Complexities, Lesson 1.2: Measuring Algorithmic Complexity with Big O Notation.
2. Incorrect. Such an algorithm has quadratic runtime performance. See Module 1: Algorithms and Complexities, Lesson 1.2: Measuring Algorithmic Complexity with Big O Notation.
3. Incorrect. An algorithm with logarithmic complexity needs y=log x operations/time to finish the task. See Module 1: Algorithms and Complexities, Lesson 1.2: Measuring Algorithmic Complexity with Big O Notation.
4. Incorrect. This algorithm has quadratic complexity, because all sets (2-4, 3-9, and 4-16) are correct for the equation y=x^2. See Module 1: Algorithms and Complexities, Lesson 1.2: Measuring Algorithmic Complexity with Big O Notation.

35. Order the given Big O complexity values by efficiency.

a. O(1) -> O(log n) -> O(n) -> O(3^n)

b. O(n) -> O(3^n) -> O(log n) -> O(1)

c. O(3^n) -> O(n) -> O(log n) -> O(1)

d. O(n) -> O(3^n) -> O(1) -> O(log n)

Analysis:

1. Correct. That's the correct order. See Module 1: Algorithms and Complexities, Lesson 1.2: Measuring Algorithmic Complexity with Big O Notation.
2. Incorrect. The correct order is O(1) -> O(log n) -> O(n) -> O(3^n). See Module 1: Algorithms and Complexities, Lesson 1.2: Measuring Algorithmic Complexity with Big O Notation.
3. Incorrect. The correct order is O(1) -> O(log n) -> O(n) -> O(3^n). See Module 1: Algorithms and Complexities, Lesson 1.2: Measuring Algorithmic Complexity with Big O Notation.
4. Incorrect. The correct order is O(1) -> O(log n) -> O(n) -> O(3^n). See Module 1: Algorithms and Complexities, Lesson 1.2: Measuring Algorithmic Complexity with Big O Notation.

36. What is the complexity of the expression 5n + 3mn converted to Big O notation?

a. O(mn)

b. O(n)

c. O(5n + 3mn)

d. O(3mn)

Analysis:

1. Correct. The highest-order part here is 3mn, so the complexity is O(mn). See Module 1: Algorithms and Complexities, Lesson 1.2: Measuring Algorithmic Complexity with Big O Notation.
2. Incorrect. We should drop everything except the highest-order part, which is 3mn – so the complexity is O(mn). See Module 1: Algorithms and Complexities, Lesson 1.2: Measuring Algorithmic Complexity with Big O Notation.
3. Incorrect. We should drop everything except the highest-order part and write it without multiplication constants. See Module 1: Algorithms and Complexities, Lesson 1.2: Measuring Algorithmic Complexity with Big O Notation.
4. Incorrect. Multiplication constants do not count in Big O notation. See Module 1: Algorithms and Complexities, Lesson 1.2: Measuring Algorithmic Complexity with Big O Notation.

37. Which of the following conversions do not follow the dropping all constants rule?

a. 3^n + 5(n^2) + 8 -> 5(n^2)

b. 3mn -> 3mn

c. 5n + 44(n^2) + 4 -> 5n + 44(n^2)

d. 4 + 5 log n -> 5 log n

Analysis:

1. Correct. The expression 3^n should not be dropped. See Module 1: Algorithms and Complexities, Lesson 1.2: Measuring Algorithmic Complexity with Big O Notation.
2. Incorrect. It follows the rule. See Module 1: Algorithms and Complexities, Lesson 1.2: Measuring Algorithmic Complexity with Big O Notation.
3. Incorrect. It follows the rule. See Module 1: Algorithms and Complexities, Lesson 1.2: Measuring Algorithmic Complexity with Big O Notation.
4. Incorrect. It follows the rule. See Module 1: Algorithms and Complexities, Lesson 1.2: Measuring Algorithmic Complexity with Big O Notation.

38. Which of the following conversions do not follow the dropping all except the highest order rule?

a. 7^n + 4(n^2) -> O(n^2)

b. 5 log n -> O(log n)

c. 3mn -> O(mn)

d. 5n + 44(n^2) -> O(n^2)

Analysis:

1. Correct. The highest order is 7^n. See Module 1: Algorithms and Complexities, Lesson 1.2: Measuring Algorithmic Complexity with Big O Notation.
2. Incorrect. It follows the rule. See Module 1: Algorithms and Complexities, Lesson 1.2: Measuring Algorithmic Complexity with Big O Notation.
3. Incorrect. It follows the rule. See Module 1: Algorithms and Complexities, Lesson 1.2: Measuring Algorithmic Complexity with Big O Notation.
4. Incorrect. It follows the rule. See Module 1: Algorithms and Complexities, Lesson 1.2: Measuring Algorithmic Complexity with Big O Notation.

39. What is the time complexity of the following program?

double a = 0, b = 0;

for (int x = 0; x <= M; x++ ) {

a += Math.random();

}

for (int x = 0; x <= N; x++ ) {

b += Math.random();

}

a. O(N+M)

b. O(N\*M)

c. O(N)

d. O(M)

Analysis:

1. Correct. The first loop is O(N) and the second loop is O(M). Since we don’t know which is bigger, we can say this is O(N + M). This can also be written as O(max(N, M)). See Module 1: Algorithms and Complexities, Lesson 1.3: Identifying Algorithms with Different Complexities.
2. Incorrect. It would have O(N\*M) complexity if those were nested loops. See Module 1: Algorithms and Complexities, Lesson 1.3: Identifying Algorithms with Different Complexities.
3. Incorrect. The first loop is O(N) and the second loop is O(M). Since we don’t know which is bigger, we can say this is O(N + M). This can also be written as O(max(N, M)). See Module 1: Algorithms and Complexities, Lesson 1.3: Identifying Algorithms with Different Complexities.
4. Incorrect. The first loop is O(N) and the second loop is O(M). Since we don’t know which is bigger, we can say this is O(N + M). This can also be written as O(max(N, M)). See Module 1: Algorithms and Complexities, Lesson 1.3: Identifying Algorithms with Different Complexities.

40. What is the space complexity of the following program?

double a = 0, b = 0;

for (int x = 0; x <= M; x++ ) {

a += Math.random();

}

for (int x = 0; x <= N; x++ ) {

b += Math.random();

}

a. O(1)

b. O(N+M)

c. O(N\*M)

d. O(N)

Analysis:

1. Correct. Since there is no additional space being utilized, the space complexity is constant – O(1). See Module 1: Algorithms and Complexities, Lesson 1.3: Identifying Algorithms with Different Complexities.
2. Incorrect. Since there is no additional space being utilized, the space complexity is constant – O(1). See Module 1: Algorithms and Complexities, Lesson 1.3: Identifying Algorithms with Different Complexities.
3. Incorrect. Since there is no additional space being utilized, the space complexity is constant – O(1). See Module 1: Algorithms and Complexities, Lesson 1.3: Identifying Algorithms with Different Complexities.
4. Incorrect. Since there is no additional space being utilized, the space complexity is constant – O(1). See Module 1: Algorithms and Complexities, Lesson 1.3: Identifying Algorithms with Different Complexities.

41. How can we determine which algorithm is the right choice to use when there is more than one solution?

a. By algorithm complexity

b. By implementation difficulty

c. By number of steps in the algorithm

d. By number of read/write operations performed by the algorithm

Analysis:

1. Correct. That is what we take under consideration when deciding which algorithm to choose. See Module 1: Algorithms and Complexities, Lesson 1.3: Identifying Algorithms with Different Complexities.
2. Incorrect. We implement the algorithm once and the code is run multiple times. Difficulty of implementation shouldn't be a metric. See Module 1: Algorithms and Complexities, Lesson 1.3: Identifying Algorithms with Different Complexities.
3. Incorrect. Number of steps in an algorithm doesn't relate to its complexity. See Module 1: Algorithms and Complexities, Lesson 1.3: Identifying Algorithms with Different Complexities.
4. Incorrect. Some computers (databases) are designed to perform such operations quickly, but they are not good for other tasks. See Module 1: Algorithms and Complexities, Lesson 1.3: Identifying Algorithms with Different Complexities.

42. What is an algorithm?

a. Any set of instructions needed to accomplish a well-designed task

b. A set of instructions to estimate the result of a task

c. A computer program

d. A different name for “pseudocode”

Analysis:

1. Correct. That's exactly what an algorithm is. See Module 1: Algorithms and Complexities, Lesson 1.3: Identifying Algorithms with Different Complexities.
2. Incorrect. An algorithm always leads to the solution of a well-designed task. See Module 1: Algorithms and Complexities, Lesson 1.3: Identifying Algorithms with Different Complexities.
3. Incorrect. A computer program is often an implementation of an algorithm, not the algorithm itself. See Module 1: Algorithms and Complexities, Lesson 1.3: Identifying Algorithms with Different Complexities.
4. Incorrect. A pseudocode is a plain English statement for better understanding of what should be programmed; it's an intermediary between an algorithm and a computer program. See Module 1: Algorithms and Complexities, Lesson 1.3: Identifying Algorithms with Different Complexities.

43. How many operations will be performed by an algorithm of complexity O(2^n) for n=16?

a. 65,536

b. 256

c. 16

d. 20

Analysis:

1. Correct. 2^16 equals 65536 See Module 1: Algorithms and Complexities, Lesson 1.3: Identifying Algorithms with Different Complexities.
2. Incorrect. This number of operations would be performed by an algorithm with quadratic complexity. See Module 1: Algorithms and Complexities, Lesson 1.3: Identifying Algorithms with Different Complexities.
3. Incorrect. This number of operations would be performed by an algorithm with linear complexity. See Module 1: Algorithms and Complexities, Lesson 1.3: Identifying Algorithms with Different Complexities.
4. Incorrect. This number of operations would be performed by an algorithm with linearithmic (n log(n)) complexity. See Module 1: Algorithms and Complexities, Lesson 1.3: Identifying Algorithms with Different Complexities.

44. What does a complexity of O(1) mean?

a. The algorithm is executed in constant time

b. Such complexity doesn't exist

c. The algorithm works with only one dataset at time

d. The algorithm is easy to implement

Analysis:

1. Correct. The input size doesn't matter – such an algorithm is the gold standard of programming. See Module 1: Algorithms and Complexities, Lesson 1.3: Identifying Algorithms with Different Complexities.
2. Incorrect. Some programs are performed in the O(1) complexity. For example: "Hello World". See Module 1: Algorithms and Complexities, Lesson 1.3: Identifying Algorithms with Different Complexities.
3. Incorrect. The Big O notation has nothing to do with input size. See Module 1: Algorithms and Complexities, Lesson 1.3: Identifying Algorithms with Different Complexities.
4. Incorrect. The Big O notation is related to algorithm complexity – not implementation difficulty. See Module 1: Algorithms and Complexities, Lesson 1.3: Identifying Algorithms with Different Complexities.

45. What is the time complexity of the following program?

int N = 0, M = 10;

for (int x = 0; x <= M; x++ ) {

for (int y = 0; y <= N; y++ ) {

writeToDisk(""something"");

}

}

a. O(M\*N)

b. O(N)

c. O(N^2)

d. O(M+N)

Analysis:

1. Correct. For every run of the inner loop (N times), an operation will be performed. The inner loop will be executed from the beginning M times. See Module 1: Algorithms and Complexities, Lesson 1.3: Identifying Algorithms with Different Complexities.
2. Incorrect. This would be the complexity of a code that has only one loop level declared. See Module 1: Algorithms and Complexities, Lesson 1.3: Identifying Algorithms with Different Complexities.
3. Incorrect. This answer MIGHT be true if M=N was known. See Module 1: Algorithms and Complexities, Lesson 1.3: Identifying Algorithms with Different Complexities.
4. Incorrect. This would be the complexity of a program implementing two loops (on the same level). See Module 1: Algorithms and Complexities, Lesson 1.3: Identifying Algorithms with Different Complexities.

46. What is the read/write complexity of the following program?

int N = 0, M = 10;

for (int x = 0; x <= M; x++ ) {

for (int y = 0; y <= N; y++ ) {

writeToDisk(""something"");

}

}

a. O(M\*N)

b. O(N)

c. O(N^2)

d. O(M+N)

Analysis:

1. Correct. For every run of the inner loop (N times), a write operation will be performed. The inner loop will be executed from the beginning M times. See Module 1: Algorithms and Complexities, Lesson 1.3: Identifying Algorithms with Different Complexities.
2. Incorrect. This would be the complexity of a code that has only one loop level declared. See Module 1: Algorithms and Complexities, Lesson 1.3: Identifying Algorithms with Different Complexities.
3. Incorrect. This answer MIGHT be true if M=N was known. See Module 1: Algorithms and Complexities, Lesson 1.3: Identifying Algorithms with Different Complexities.
4. Incorrect. This would be the complexity of a program implementing two loops (on the same level). See Module 1: Algorithms and Complexities, Lesson 1.3: Identifying Algorithms with Different Complexities.

47. What is the time complexity of the following program?

int N = 0, M = 10;

for (int x = 0; x <= M; x++ ) {

writeToDisk(""something"");

for (int y = 0; y <= N; y++ ) {

Math.random();

}

}

a. O(M\*N)

b. O(N)

c. O(N^2)

d. O(M+N)

Analysis:

1. Correct. For every run of the inner loop (N times), an operation will be performed. The inner loop will be executed from the beginning M times. See Module 1: Algorithms and Complexities, Lesson 1.3: Identifying Algorithms with Different Complexities.
2. Incorrect. This would be the complexity of a code that has only one loop level declared. See Module 1: Algorithms and Complexities, Lesson 1.3: Identifying Algorithms with Different Complexities.
3. Incorrect. This answer MIGHT be true if M=N was known. See Module 1: Algorithms and Complexities, Lesson 1.3: Identifying Algorithms with Different Complexities.
4. Incorrect. This would be the complexity of a program implementing two loops (on the same level). See Module 1: Algorithms and Complexities, Lesson 1.3: Identifying Algorithms with Different Complexities.

48. What is the read/write complexity of the following program?

int N = 0, M = 10;

for (int x = 0; x <= M; x++ ) {

writeToDisk(""something"");

for (int y = 0; y <= N; y++ ) {

Math.random();

}

}

a. O(M)

b. O(M\*N)

c. O(N^2)

d. O(1)

Analysis:

1. Correct. For every run of the outer loop (M times), a write operation will be performed. The inner loop doesn't perform any write operation, so it doesn't matter with respect to the space complexity. See Module 1: Algorithms and Complexities, Lesson 1.3: Identifying Algorithms with Different Complexities.
2. Incorrect. This would be the complexity of a code that performs read/write operations inside the inner loop. See Module 1: Algorithms and Complexities, Lesson 1.3: Identifying Algorithms with Different Complexities.
3. Incorrect. This would be the complexity of a code that performs read/write operations inside the inner loop and if M=N was known. See Module 1: Algorithms and Complexities, Lesson 1.3: Identifying Algorithms with Different Complexities.
4. Incorrect. This would be the complexity of a code that performs a constant number of read/write operations. See Module 1: Algorithms and Complexities, Lesson 1.3: Identifying Algorithms with Different Complexities.

49. What is the time complexity of the following program?

double a = 0;

for (int x = 0; x <= 100; x++ ) {

a += Math.random();

}

a. O(1)

b. O(100)

c. O(N)

d. This cannot be determined

Analysis:

1. Correct. This program will take a constant time to execute. See Module 1: Algorithms and Complexities, Lesson 1.3: Identifying Algorithms with Different Complexities.
2. Incorrect. Constant complexity is written as O(1). See Module 1: Algorithms and Complexities, Lesson 1.3: Identifying Algorithms with Different Complexities.
3. Incorrect. This program will take a constant time to execute. See Module 1: Algorithms and Complexities, Lesson 1.3: Identifying Algorithms with Different Complexities.
4. Incorrect. This program will take a constant time to execute. See Module 1: Algorithms and Complexities, Lesson 1.3: Identifying Algorithms with Different Complexities.

50. What can be measured by algorithm complexity?

a. All of the above

b. Time needed by an algorithm to solve a problem

c. Number of read/write operations performed by an algorithm

d. Number of Internet connections performed by a program that implements an algorithm

Analysis:

1. Correct. Algorithm complexity can measure all of these metrics. See Module 1: Algorithms and Complexities, Lesson 1.3: Identifying Algorithms with Different Complexities.
2. Incorrect. Algorithm complexity can measure more metrics. See Module 1: Algorithms and Complexities, Lesson 1.3: Identifying Algorithms with Different Complexities.
3. Incorrect. Algorithm complexity can measure more metrics. See Module 1: Algorithms and Complexities, Lesson 1.3: Identifying Algorithms with Different Complexities.
4. Incorrect. Algorithm complexity can measure more metrics. See Module 1: Algorithms and Complexities, Lesson 1.3: Identifying Algorithms with Different Complexities.

51. What is the complexity of the following code?

int i, j, k = 0;

for (i = n / 2; i <= n; i++) {

for (j = 2; j <= n; j = j \* 2) {

k = k + n / 2;

}

}

a. O(nlog(n))

b. O(n)

c. O(n^2)

d. O(2^n)

Analysis:

1. Correct. The inner loop is executed in log(n) time; we need to multiple it by “n” because of the outer loop. See Module 1: Algorithms and Complexities, Lesson 1.3: Identifying Algorithms with Different Complexities.
2. Incorrect. The inner loop is executed in log(n) time; we need to multiple it by “n” because of the outer loop. See Module 1: Algorithms and Complexities, Lesson 1.3: Identifying Algorithms with Different Complexities.
3. Incorrect. The inner loop is executed in log(n) time; we need to multiple it by “n” because of the outer loop. See Module 1: Algorithms and Complexities, Lesson 1.3: Identifying Algorithms with Different Complexities.
4. Incorrect. The inner loop is executed in log(n) time; we need to multiple it by “n” because of the outer loop. See Module 1: Algorithms and Complexities, Lesson 1.3: Identifying Algorithms with Different Complexities.

52. What is the complexity of the following code?

int a = 0, i = N;

while (i > 0) {

a += i;

i /= 2;

}

a. O(nlog(n))

b. O(n)

c. O(n^2)

d. O(2^n)

Analysis:

1. Correct. On every instance of the loop being completed, we decrease the counter by half of its value. See Module 1: Algorithms and Complexities, Lesson 1.3: Identifying Algorithms with Different Complexities.
2. Incorrect. On every instance of the loop being completed, we decrease the counter by half of its value. See Module 1: Algorithms and Complexities, Lesson 1.3: Identifying Algorithms with Different Complexities.
3. Incorrect. On every instance of the loop being completed, we decrease the counter by half of its value. See Module 1: Algorithms and Complexities, Lesson 1.3: Identifying Algorithms with Different Complexities.
4. Incorrect. On every instance of the loop being completed, we decrease the counter by half of its value. See Module 1: Algorithms and Complexities, Lesson 1.3: Identifying Algorithms with Different Complexities.

53. Which interface must be implemented by a class so as to apply the Collections.sort() method on a list of objects of that class?

a. Comparable

b. Sortable

c. Such a method doesn't exist

d. Hashable

Analysis:

1. Correct. This is the name of the interface that needs to be implemented. This interface is shipped with the compareTo method. The compareTo method needs to fulfill the following contract:

If a>b, then a.compareTo(b) returns a positive integer

If a<b, then a.compareTo(b) returns a negative integer

If a=b, then a.compareTo(b) returns 0.

See Module 1: Algorithms and Complexities, Lesson 1.3: Identifying Algorithms with Different Complexities.

1. Incorrect. There is no such interface in Java out of the box. See Module 1: Algorithms and Complexities, Lesson 1.3: Identifying Algorithms with Different Complexities.
2. Incorrect. There is a static sort method in the Collections class. See Module 1: Algorithms and Complexities, Lesson 1.3: Identifying Algorithms with Different Complexities.
3. Incorrect. This isn't the interface that needs to be implemented. See Module 1: Algorithms and Complexities, Lesson 1.3: Identifying Algorithms with Different Complexities.

54. What is the time complexity of the following program?

double a = 0, b = 0;

for (int x = 0; x <= N; x++ ) {

for (int y = 0; y <= N; y++ ) {

a += Math.random();

}

}

for (int x = 0; x <= N; x++ ) {

b += Math.random();

}

a. O(N^2)

b. O(n^2 + n)

c. O(n)

d. O(1)

Analysis:

1. Correct. The first loops (inner and outer) will execute in time N^2. The second loop doesn't matter in the case of Big O notation, because n^2 (quadratic) has a bigger impact than n (linear). See Module 1: Algorithms and Complexities, Lesson 1.3: Identifying Algorithms with Different Complexities.
2. Incorrect. The first loops (inner and outer) will execute in time N^2. The second loop doesn't matter in the case of Big O notation, because n^2 (quadratic) has a bigger impact than n (linear). See Module 1: Algorithms and Complexities, Lesson 1.3: Identifying Algorithms with Different Complexities.
3. Incorrect. The first loops (inner and outer) will execute in time N^2. The second loop doesn't matter in the case of Big O notation, because n^2 (quadratic) has a bigger impact than n (linear). See Module 1: Algorithms and Complexities, Lesson 1.3: Identifying Algorithms with Different Complexities.
4. Incorrect. The first loops (inner and outer) will execute in time N^2. The second loop doesn't matter in the case of Big O notation, because n^2 (quadratic) has a bigger impact than n (linear). See Module 1: Algorithms and Complexities, Lesson 1.3: Identifying Algorithms with Different Complexities.