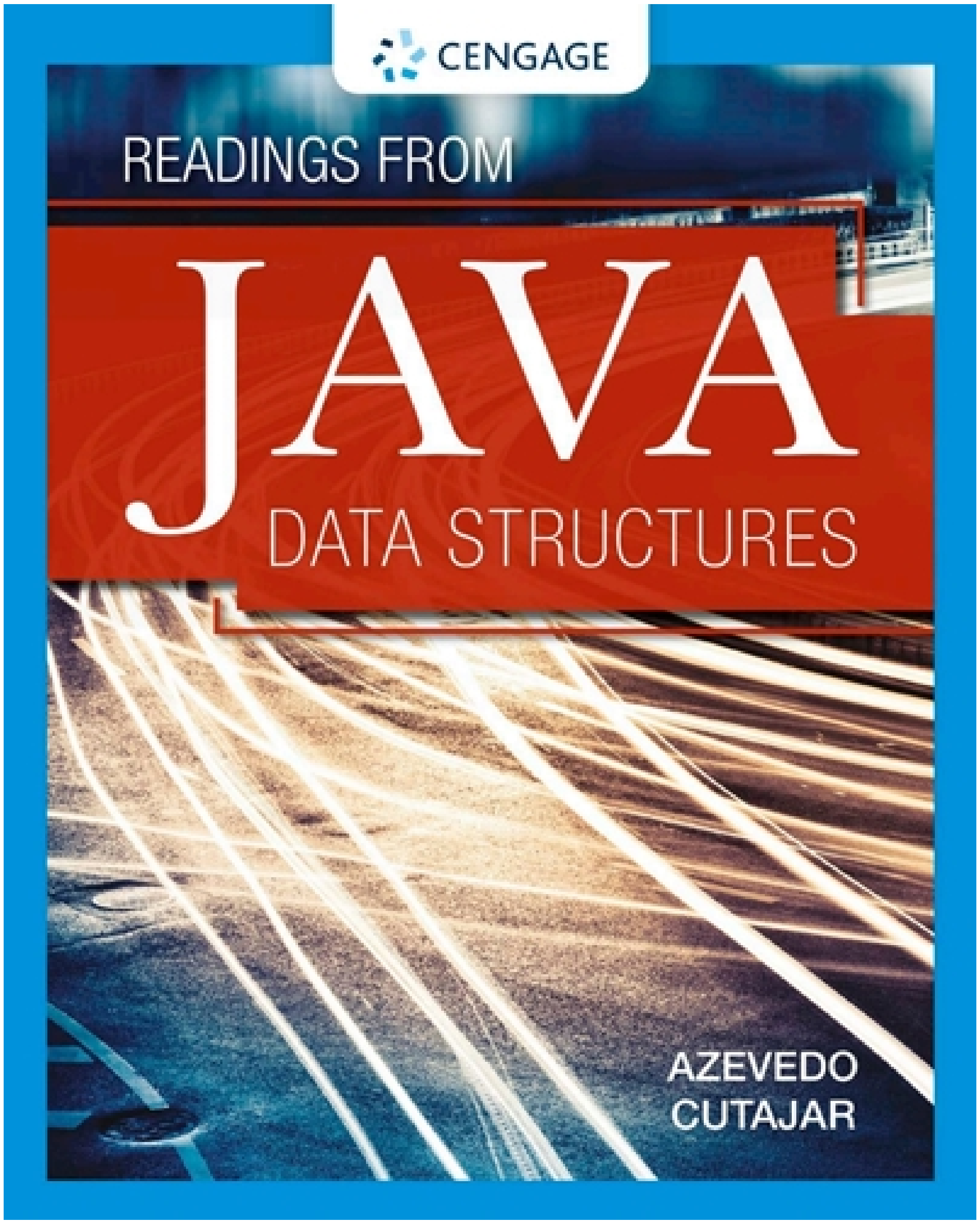

CENGAGE
READINGS FROM
JAVA
DATA STRUCTURES
AZEVEDO
CUTAJAR

Solutions

Java Data Structures
Module Quiz
Module 1: Algorithms and Complexities

1. Out of the following list, which runtime complexity scales the worst?
a. *O(1)*
b. *O(n)*
c. *O(n^2)*
d. *O(log n)*

Analysis:

a. Incorrect. See Module 1: Algorithms and Complexities, Lesson 1.2: Measuring Algorithmic Complexity with Big O Notation.
b. Incorrect. See Module 1: Algorithms and Complexities, Lesson 1.2: Measuring Algorithmic Complexity with Big O Notation.
c. Correct. See Module 1: Algorithms and Complexities, Lesson 1.2: Measuring Algorithmic Complexity with Big O Notation.
d. Incorrect. See Module 1: Algorithms and Complexities, Lesson 1.2: Measuring Algorithmic Complexity with Big O Notation.

2. What is the runtime complexity of the following code?

```
public boolean strEqual(String a, String b) {
   for (int i = 0; i < Math.min(b.length(), a.length());
i++)
       if (a.charAt(i) != b.charAt(i)) return false;
   return a.length() == b.length();
}
```

a. *O(10^n)*
b. *O(n)*
c. *O(n log n)*

d. *O(1)*

Analysis

a. Incorrect. See Module 1: Algorithms and Complexities, Lesson 1.3: Identifying Algorithms with Different Complexities.
b. Correct. See Module 1: Algorithms and Complexities, Lesson 1.3: Identifying Algorithms with Different Complexities.
c. Incorrect. See Module 1: Algorithms and Complexities, Lesson 1.3: Identifying Algorithms with Different Complexities.
d. Incorrect. See Module 1: Algorithms and Complexities, Lesson 1.3: Identifying Algorithms with Different Complexities.

3. The binary search algorithm has a big O of which of the following?
a. *O(log n)*
b. $O(2^n)$
c. *O(n log n)*
d. *O(1)*

Analysis:

a. Correct. See Module 1: Algorithms and Complexities, Lesson 1.3: Identifying Algorithms with Different Complexities.
b. Incorrect. See Module 1: Algorithms and Complexities, Lesson 1.3: Identifying Algorithms with Different Complexities.
c. Incorrect. See Module 1: Algorithms and Complexities, Lesson 1.3: Identifying Algorithms with Different Complexities.
d. Incorrect. See Module 1: Algorithms and Complexities, Lesson 1.3: Identifying Algorithms with Different Complexities.

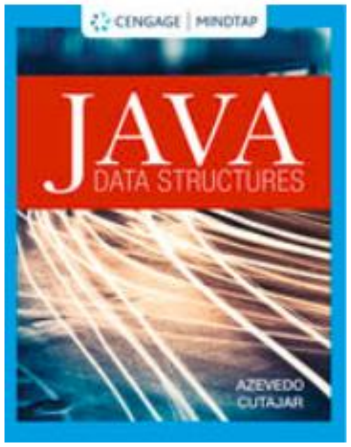
4. If we developed an algorithm that performs *$5 + 2 \log n + n$* operations, we can say that the algorithm has a complexity of which of the following?
a. *$O(n^2)$*
b. *$O(5)$*
c. *$O(\log n)$*
d. *$O(n)$*

Analysis

a. Incorrect. See Module 1: Algorithms and Complexities, Lesson 1.2: Measuring Algorithmic Complexity with Big O Notation.
b. Incorrect. See Module 1: Algorithms and Complexities, Lesson 1.2: Measuring Algorithmic Complexity with Big O Notation.
c. Incorrect. See Module 1: Algorithms and Complexities, Lesson 1.2: Measuring Algorithmic Complexity with Big O Notation.
d. Correct. See Module 1: Algorithms and Complexities, Lesson 1.2: Measuring Algorithmic Complexity with Big O Notation.

Java Data Structures and Algorithms
ISBN MindTap: 9780357114841

MindTap®



Welcome to *Java Data Structures and Algorithms*. This Instructor's Manual will help you navigate the unique activities that are included in the MindTap, which will better enable you to include the exercises in your curriculum. While the content included in this MindTap is specific to the discipline and course, the functionality will act the same as you move from product to product.

For additional resources on our MindTap platform, please click HERE. At this site, you will find User Guides, Self-Training Videos, Training Webinars, and Podcasts. We also include Resources that are specific to your campus's LMS, should additional information be needed. Student versions of the same resources are located HERE. This link can be shared with your students directly, should they have any questions about the product.

At a Glance

Instructor's Manual Table of Contents

Course Learning Design

In creating the digital learning path, we aimed to provide your students with a coherently structured experience that:

- supports and aligns the learning objectives with the course content, instructional strategies, and assessments;
- addresses individual learner differences and preferences;
- welcomes learners of all abilities and backgrounds; and
- enhances learner motivation by providing them with relevant, applicable learning experiences consistent with their own learning and professional goals.

We're excited to present you with the digital course experience and want to draw your attention to some of the design decisions we made as part of ensuring your confidence in our ability to create an effective, quality learning experience.

Course Learning Design	
Course Description	Algorithms and data structures are crucial for application performance. MindTap for *Java Data Structures and Algorithms* teaches problem-solving skills for building efficient applications. It starts with an introduction to algorithms and explains bubble, merge, quicksort, and other popular programming patterns. Coverage also includes data structures, such as binary tree, hash table, and graphs. The course progresses to advanced concepts, such as algorithm design paradigms and graph theory.
Course Approach (6 modules in course)	This course teaches students how to write systematic code in Java and improve application efficiency with hands-on practice, step-by-step instruction, and provides immediate feedback and troubleshooting support on their code. Students will develop skills that are in-demand by employers by completing authentic, real-world coding projects that can be added to their GitHub portfolios.
Module Approach	Each module is broken into 2–6 lessons—within each lesson are activities that align to meet specific learning objectives that are concrete and actionable. Within each lesson, the student will read some narrative and follow up with hands-on learning. There are four types of online labs in this course: 1. **Practice Exercises** (Ungraded) provide an opportunity to practice a new concept in a short coding activity. Students are provided with guided instructional materials alongside a live computing environment. There will typically be 1–3 practice labs in each lesson and there are on average around 5 lessons per module (5–15 practice/module). 2. **Lab Activities** (Auto-Graded) are coding activities that are completed by a student and contain auto-grading that feeds directly to the gradebook. Learners demonstrate an understanding of numerous concepts by completing tasks. Tasks are verified using unit tests, I/O tests, image and webpage comparison, debugging tests, and many other checks. There will be a lab assessment for every lesson and there are on average around 5 lessons per module (around 5 labs per module). 3. **Module Lab Assessments** (Auto- and Manual-Graded) encompass all the learning objectives in the module. Students are asked to complete a larger, authentic assignment with many tasks. Some tasks will be verified using unit tests, I/O tests, image and webpage comparison, debugging tests but other tasks will be unique to each student's project and will require manual grading. The goal of these assignments is to prove that students have mastered the learning objectives in the module and in doing so have also created a program for their GitHub portfolio (1 Module Lab Assessment per module). 4. **Capstone Lab Assessment (**Auto- and Manual-Graded) is a final project that is the summative assessment. The goal of this assignment is to prove that students have mastered the course objectives and in doing so have also created a program for their GitHub portfolio (1 Capstone Lab Assessment per course).

Learning Path Activities	How many in course	What is it?	Why it matters?	Seat time
Welcome to Your Course	1	This is a brief overview of the course objectives that will be covered in the modules of this MindTap.	Students will gain a clear understanding of the course objectives and will explore how this course offers the opportunity to not only read but watch videos, engage in critical-thinking simulations and hands-on trainings, teach them how to use the technology, and take quizzes to practice and check their understanding.	5 minutes
Getting Started Resources	1	This section includes videos that provide an overview of the MindTap platform and the Coding IDE. There are 3 lab Pre-Requisites, 2 of which count toward the grade	Students will learn how to use MindTap to its fullest potential, which will help them excel in the course. They'll also be introduced to the IDE's functionality in 6 brief videos. They'll then complete 3 Lab Pre-Requisites, 1 is practice and 2 count toward their grade.	30 minutes
Pre- and Post-Course Assessments	25 questions each assessment	Brief survey to-assess students' knowledge of the subject matter before and after completing the course.	**For students:** It creates awareness around what they will learn (pre) and how much they have learned (post). **For instructors:** It establishes a baseline of what students already know (pre) and demonstrates how much they learned (post). **For administrators:** Coupling the pre- and post-course assessment provides data on how much the students learned and the overall impact of the course.	40 minutes
Module Content (6 modules total)				
Readings for each module lesson; 2–6 lessons per module	~11 Short readings per module (66 total in course)	Readings reinforce learning objectives.	Students will read succinct, focused excerpts vs long chapters (then move into an interactive activity).	55 minutes
Practice Exercises	~4 per module (21 total in course)	Short coding exercises in an IDE (non-graded)	Students complete step-by-step coding exercises that offer a practical, hands-on approach to acquiring and retaining new concepts and skills.	2-5 minutes
Lab Activity (Graded)	~4 per module (20 total in course)	Scenario-based coding labs in an IDE (auto-graded)	These scenario-based activities bring together skills learned throughout the topics and lessons to solve real-world problems.	30 minutes
Reflection	~3 per module (17 total in course)	Essay question	The reflection prompt challenges students to develop higher-level thinking and promotes problem-solving. This is also an opportunity for you to confirm that tricky topics are understood	15 minutes
Module Quizzes	~1 per module (6 total in course)	Includes 6–9 multiple-choice questions at the end of each module.	The student can integrate material across the entire lesson and check their understanding before moving on to the next lesson.	10 minutes
Module Lab Assessment (Auto & Manual Grading)	1 per module (6total in course)	A larger coding project in our IDE that assesses whether students have mastered the Learning Objectives in the module.	A larger lab with an authentic development project with many tasks. Upon completion, students will have 6 large coding projects for their GitHub portfolios.	1–2 hours
Capstone Lab Assessment	1 per course	Final coding project in our IDE that assesses whether students have mastered the Course Objectives.	A larger lab with an authentic development project with many tasks. Upon completion, students will have 1 additional coding project to add to their GitHub portfolio.	2–5 hours
Instructor Test Bank	1 per module (6 total in course)	An exam of @400 objective-based questions based on each module available in the CNOW app.	The Test Bank evaluates the student on their mastery of that module.	30 minutes

Topic/Chapter	Assignments
Module 1 Algorithms and Complexities	Lessons 1.1–1.3 Reading Practice Labs Lab Activities Reflection Practice Quiz Module Lab Assessment Module Quiz
Module 2 Sorting Algorithms and Fundamental Data Structures	Lessons 2.1–2.4 Reading Practice Labs Lab Activities Reflection Practice Quiz Module Lab Assessment Module Quiz
Module 3 Hash Tables and Binary Search Trees	Lessons 3.1–3.2 Reading Practice Labs Lab Activities Reflection Practice Quiz Module Lab Assessment Module Quiz
Module 4 Algorithm Design Paradigms	Lessons 4.1–4.3 Reading Practice Labs Lab Activities Reflection Practice Quiz Module Lab Assessment Module Quiz
Module 5 String Matching Algorithms	Lessons 5.1–5.3 Reading Practice Labs Lab Activities Reflection Practice Quiz Module Lab Assessment Module Quiz
Module 6 Graphs, Prime Numbers, and Complexity	Lessons 6.1–6.6 Reading Practice Labs Lab Activities Reflection Practice Quiz Module Lab Assessment Module Quiz
Capstone Lab Assessment: Creating a Naval Navigation	

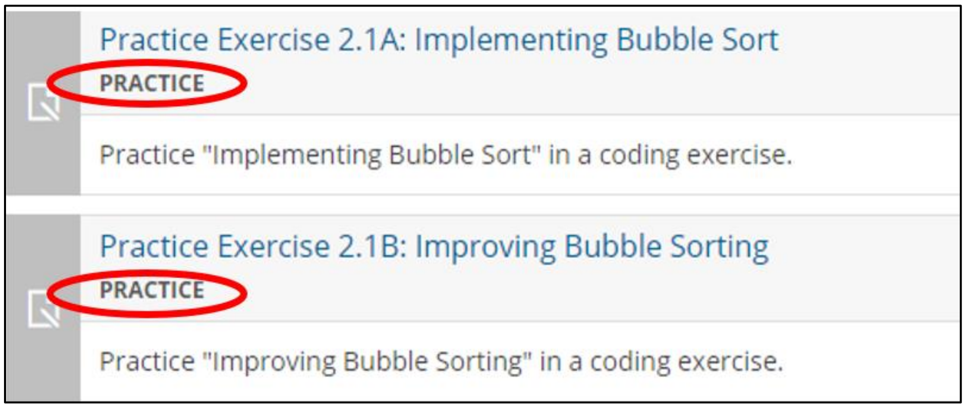
Lab Details

There are 21 Practice Exercises, 19 Lab Activities, 6 Module Lab Assessments, and 1 Capstone Lab Assessment across 6 modules.

Lab Types

Practice Exercises:

- Practice Exercises are coding lab assignments within the IDE that allow you to practice writing and running code.
- Practice Exercises are not graded and are not captured in the Progress App. These are designated in the learning path:

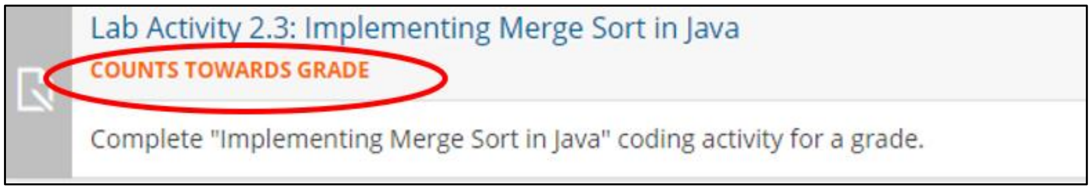


Practice Exercise 2.1A: Implementing Bubble Sort
PRACTICE
Practice "Implementing Bubble Sort" in a coding exercise.

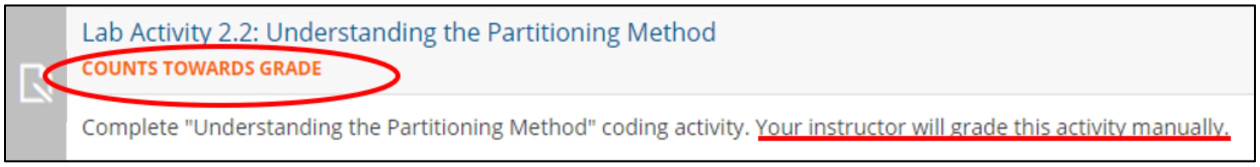
Practice Exercise 2.1B: Improving Bubble Sorting
PRACTICE
Practice "Improving Bubble Sorting" in a coding exercise.

Lab Activities:

- Lab Activities are coding lab assignments within the IDE that run tests against your code to ensure that the objectives in the activity have been satisfied.
- Lab Activities are automatically graded unless otherwise noted in the learning path:

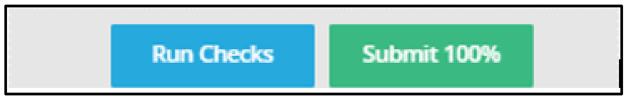


Lab Activity 2.3: Implementing Merge Sort in Java
COUNTS TOWARDS GRADE
Complete "Implementing Merge Sort in Java" coding activity for a grade.



Lab Activity 2.2: Understanding the Partitioning Method
COUNTS TOWARDS GRADE
Complete "Understanding the Partitioning Method" coding activity. Your instructor will grade this activity manually.

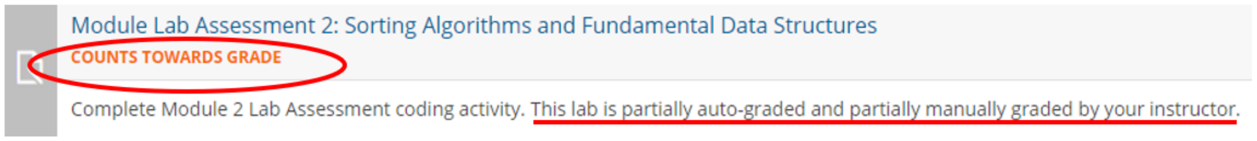
- You will work through the Lab Activities and "Run Checks" as you work through the problems. Once you have completed the assignment, you can "Submit," which will send your lab to your instructor.

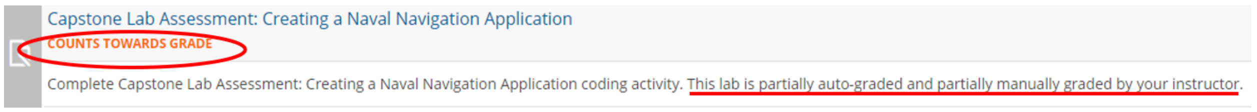


- Note that instructors have the capability to review code submissions and alter grades as they see fit. Grade submissions are not final.

Module Lab Assessments and Capstone Lab Assessment:

- The Lab Assessments are coding lab assignments within the IDE that provide you with an authentic scenario to test your coding skills.
- There is one Module Lab Assessment per module, and one Capstone Lab Assessment for the entire course.
- Lab Assessments are partially automatically graded and partially manually graded by your instructor. These are designated in the learning path:


Module Lab Assessment 2: Sorting Algorithms and Fundamental Data Structures
COUNTS TOWARDS GRADE
Complete Module 2 Lab Assessment coding activity. This lab is partially auto-graded and partially manually graded by your instructor.


Capstone Lab Assessment: Creating a Naval Navigation Application
COUNTS TOWARDS GRADE
Complete Capstone Lab Assessment: Creating a Naval Navigation Application coding activity. This lab is partially auto-graded and partially manually graded by your instructor.

- Note that instructors have the capability to review code submissions and alter grades as they see fit. Grade submissions are not final.

List of Coding Labs

Coding IDE Lab Prerequisite for Practice Exercises	Practice
Coding IDE Lab Prerequisite for Lab Activities	Auto-Grade
Coding IDE Lab Prerequisite for Module and Capstone Lab Assessments	Auto/Manual Grade
Module 1	
Lab Activity 1.1: Writing an Algorithm to Convert Numbers from Octal to Decimal	Auto-Grade
Lab Activity 1.2.A: Developing a Timing Table Using the Exponential Algorithm	Practice
Practice Exercise 1.2: Identify the Best and Worst Performance of an Algorithm	Practice
Lab Activity 1.2.B: Converting Expressions to Big O Notations	Practice

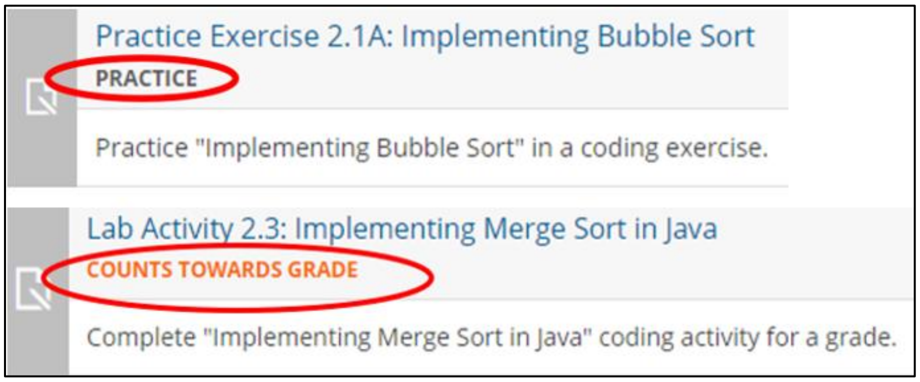
Lab Activity 1.3: Developing a Faster Intersection Algorithm	Auto-Grade
Module Lab Assessment 1: Algorithms and Complexities	Auto/Manual Grade
Module 2	
Practice Exercise 2.1.A: Implementing Bubble Sort	Practice
Practice Exercise 2.1.B: Improving Bubble Sorting	Practice
Lab Activity 2.1: Implementing Selection Sort in Java	Auto-Grade
Practice Exercise 2.2.A: Implementing Binary Search Recursively	Practice
Lab Activity 2.2: Understanding the Partitioning Method	Manual-Grade
Practice Exercise 2.2.B: Implementing QuickSort	Practice
Practice Exercise 2.3: Implementing Merge Sort	Practice
Lab Activity 2.3: Implementing Merge Sort in Java	Auto-Grade
Practice Exercise 2.4.A: Converting the Linked List to a Doubly Linked List Structure	Practice
Lab Activity 2.4.A: Traversing the Linked List	Auto-Grade
Practice Exercise 2.4.B: Adding and Deleting the Elements from the Queue	Practice
Practice Exercise 2.4.C: Reversing a String	Practice
Practice Exercise 2.4.D: Safe Enqueing in an Array	Practice
Lab Activity 2.4.B: Evaluating the Postfix Expression	Auto-Grade
Module Lab Assessment 2: Sorting Algorithms and Fundamental Data Structures	Auto/Manual Grade
Module 3	
Practice Exercise 3.1.A: Carrying out the Linear Probing Search Operation	Practice
Practice Exercise 3.1.B: Implementing the Multiplication Method for a Hash Table	Practice
Lab Activity 3.1: Implementing Open Addressing	Auto-Grade
Practice Exercise 3.2.A: Searching for a Minimum Key in a Binary Tree	Practice
Lab Activity 3.2.A: Implementing BFS in Java	Auto-Grade
Practice Exercise 3.2.B: Applying Right Tree Rotation	Practice
Lab Activity 3.2.B: Retrieving the Successor of an Element When the Tree Is Traversed in Inorder	Auto-Grade
Module Lab Assessment 3: Hash Tables and Binary Search Trees	Auto/Manual Grade
Module 4	
Practice Exercise 4.1.A: Solving the Activity Selection Problem	Practice
Practice Exercise 4.1.B: Developing an Algorithm to Generate Code Words Using Huffman Coding	Practice
Lab Activity 4.1: Implementing a Greedy Algorithm to Compute Egyptian Fractions	Auto-Grade
Lab Activity 4.2: Solving the Maximum Subarray Problem	Auto-Grade
Practice Exercise 4.3: Solving the 0-1 Knapsack Problem Using Recursion	Practice
Lab Activity 4.3: The Coin Change Problem	Auto-Grade
Module Lab Assessment 4: Algorithm Design Paradigms	Auto/Manual Grade
Module 5	

Practice Exercise 5.1: Developing the String Matching Algorithm in Java	Practice
Lab Activity 5.2: Implementing the Bad Character Rule	Auto-Grade
Practice Exercise 5.2: Implementing the Boyer-Moore Algorithm	Practice
Practice Exercise 5.3: Applying the Rabin-Karp Algorithm	Practice
Module Lab Assessment 5: String Matching Algorithms	Auto/Manual Grade
Module 6	
Practice Exercise 6.1: Writing a Java Code to Add Weights to the Directed Graph	Practice
Lab Activity 6.1: Building the Adjacency Matrix Representation of a Weighted Undirected Graph	Auto-Grade
Lab Activity 6.2: Using BFS to Find the Shortest Path Out of a Maze	Auto-Grade
Lab Activity 6.3: Improving Floyd-Warshall's Algorithm to Reconstruct the Shortest Path	Auto-Grade
Lab Activity 6.4: Implementing the Sieve of Eratosthenes	Auto-Grade
Module Lab Assessment 6: Graphs, Prime Numbers, and Complexity Classes	Auto/Manual Grade
Capstone Lab Assessment	Auto/Manual Grade

A Note to Instructors:

COUNTS TOWARD GRADE/PRACTICE: Whether a lab COUNTS TOWARD GRADE or is PRACTICE, as indicated in the Learning Path, is preset and cannot be changed. Changing the Gradeable field within MindTap will not change the gradeability of the actual labs. We recommend not changing these default settings to avoid discrepancies between the MindTap plank description and the actual lab.

- COUNTS TOWARD GRADE = lab is automatically or manually graded and the score is captured in the Progress App
- PRACTICE = lab is not graded and the score is not captured in the Progress App



Practice Exercise 2.1A: Implementing Bubble Sort
PRACTICE
Practice "Implementing Bubble Sort" in a coding exercise.

Lab Activity 2.3: Implementing Merge Sort in Java
COUNTS TOWARDS GRADE
Complete "Implementing Merge Sort in Java" coding activity for a grade.

Module 1

Algorithms and Complexities

Module Objectives

- Define an algorithm with an example
- Measure algorithmic complexity
- Identify algorithms with different complexities
- Assess various examples with different runtime complexities

Module Notes

1. It's important for the students to understand that an algorithm is just a small part of an application used to solve a well-defined problem. Examples such as sorting a list of numbers, finding the shortest route, or word prediction are all correct. Big software applications, such as email clients or an operating system are improper.

Lesson 1.1 Developing Our First Algorithm

Lesson 1.1.1 Algorithm for Converting Binary Numbers to Decimal

Note
Walk through the code in *Snippet 1.1* line by line, ideally starting from the two initial variables, and then give details about the loop. Describe how we're processing the binary string in reverse inside the loop.

Lesson 1.2 Measuring Algorithmic Complexity with Big O Notation

Lesson 1.2.1 Complexity Example

Note
Explain Snippet 1.2 by describing how the algorithm is going through every single pair combination.

Lesson 1.2.2 Understanding Complexity

Java Data Structures
Pre-Course Assessment

1. Order the following algorithms in the ascending order of their best-case complexity:

Bubble sort
Selection sort
Quick sort
Merge sort

a. Bubble sort < Quick sort == Merge sort < Selection sort

b. Quick sort < Bubble sort == Merge sort < Selection sort

c. Selection sort < Quick sort < Merge sort < Bubble sort

d. Selection sort < Bubble sort < Merge sort < Quick sort

Analysis:

a. Correct. Bubble sort has a complexity of O(n) in the best-case scenario. Quick sort and merge sort have O(n log(n)), and selection sort has O(n^2). See Module 2: Sorting Algorithms and Fundamental Data Structures, Lesson 2.3: Using Merge Sort.
b. Incorrect. Bubble sort has a complexity of O(n) in the best-case scenario. Quick sort and merge sort have O(n log(n)), and selection sort has O(n^2). See Module 2: Sorting Algorithms and Fundamental Data Structures, Lesson 2.3: Using Merge Sort.
c. Incorrect. Bubble sort has a complexity of O(n) in the best-case scenario. Quick sort and merge sort have O(n log(n)), and selection sort has O(n^2). See Module 2: Sorting Algorithms and Fundamental Data Structures, Lesson 2.3: Using Merge Sort.
d. Incorrect. Bubble sort has a complexity of O(n) in the best-case scenario. Quick sort and merge sort have O(n log(n)), and selection sort has O(n^2). See Module 2: Sorting Algorithms and Fundamental Data Structures, Lesson 2.3: Using Merge Sort.

2. Which of the following statements is true for a bubble sort structure?
a. There are two for loops, one nested in the other.

b. There is a while loop.

c. There are three for loops, all of them separate.

d. There is a single for loop.

Analysis:

a. Correct. Bubble sort is based on two nested for loops. See Module 2: Sorting Algorithms and Fundamental Data Structures, Lesson 2.1: Introducing Bubble Sorting.
b. Incorrect. Bubble sort is based on two nested for loops. See Module 2: Sorting Algorithms and Fundamental Data Structures, Lesson 2.1: Introducing Bubble Sorting.
c. Incorrect. Bubble sort is based on two nested for loops. See Module 2: Sorting Algorithms and Fundamental Data Structures, Lesson 2.1: Introducing Bubble Sorting.
d. Incorrect. Bubble sort is based on two nested for loops. See Module 2: Sorting Algorithms and Fundamental Data Structures, Lesson 2.1: Introducing Bubble Sorting.

3. What is the result of the following expression in postfix notation?
40 10 / 4 * 20 +

a. 36

b. 6

c. 90

d. 10

Analysis:

a. Correct. It can be written in infix notation as: 40 / 10 *4 + 20. See Module 2: Sorting Algorithms and Fundamental Data Structures, Lesson 2.4: Getting Started with Fundamental Data Structures.
b. Incorrect. It can be written in infix notation as: 40 / 10 *4 + 20. See Module 2: Sorting Algorithms and Fundamental Data Structures, Lesson 2.4: Getting Started with Fundamental Data Structures.
c. Incorrect. It can be written in infix notation as: 40 / 10 *4 + 20. See Module 2: Sorting Algorithms and Fundamental Data Structures, Lesson 2.4: Getting Started with Fundamental Data Structures.
d. Incorrect. It can be written in infix notation as: 40 / 10 *4 + 20. See Module 2: Sorting Algorithms and Fundamental Data Structures, Lesson 2.4: Getting Started with Fundamental Data Structures.

4. What will be the output of the following program?

```
import java.util.HashMap;
import java.util.Map;

public class Program {
   public static void main(String... args) {
        Key k1 = new Key();
        Key k2 = new Key();

        Map<Key, String> map = new HashMap<>();

        map.put(k1, "value1");
        map.put(k2, "value2");

        System.out.println(map.get(k1));
        System.out.println(map.get(k2));
    }
}

class Key {
    public int hashCode() {
        return 1;
    }
}
```

a. value1

value2

b. null

null

c. value2

value2

d. A Runtime Exception will be thrown.

Analysis:

a. Correct. It doesn't matter that hashCode returns a constant value. HashMap still uses the equals method to compare the keys. The only problem here is that this code is inefficient. See Module 3: Hash Tables and Binary Search Trees, Lesson 3.1: Introducing Hash Tables.
b. Incorrect. It doesn't matter that hashCode returns a constant value. HashMap still uses the equals method to compare the keys. The only problem here is that this code is inefficient. See Module 3: Hash Tables and Binary Search Trees, Lesson 3.1: Introducing Hash Tables.
c. Incorrect. It doesn't matter that hashCode returns a constant value. HashMap still uses the equals method to compare the keys. The only problem here is that this code is inefficient. See Module 3: Hash Tables and Binary Search Trees, Lesson 3.1: Introducing Hash Tables.
d. Incorrect. There is no reason for which a Runtime Exception could be thrown from this program. See Module 3: Hash Tables and Binary Search Trees, Lesson 3.1: Introducing Hash Tables.

5. What will be the output of the following program?

```
import java.util.HashMap;
import java.util.Hashtable;
```

Java Data Structures
Post-Course Assessment

1. Which of the following statements is true?

1. Removing an element from a linked list is faster than adding an element to a linked list.
2. Traversing a linked list has the same complexity as removing an element from a list.
3. Traversing a linked list has the same complexity as adding an element to a list.
4. Removing and adding an element to a linked list has the same complexity as a traversing operation.

a. 1 and 3
b. 1
c. 2 and 4
d. 4

Analysis:

a. Correct. Removing an element has a complexity of O(1), while traversing and adding an element to a list has a complexity of O(n). See Module 2: Sorting Algorithms and Fundamental Data Structures, Lesson 2.4: Getting Started with Fundamental Data Structures.
b. Incorrect. Removing an element has a complexity of O(1), while traversing and adding an element to a list has a complexity of O(n). See Module 2: Sorting Algorithms and Fundamental Data Structures, Lesson 2.4: Getting Started with Fundamental Data Structures.
c. Incorrect. Removing an element has a complexity of O(1), while traversing and adding an element to a list has a complexity of O(n). See Module 2: Sorting Algorithms and Fundamental Data Structures, Lesson 2.4: Getting Started with Fundamental Data Structures.
d. Incorrect. Removing an element has a complexity of O(1), while traversing and adding an element to a list has a complexity of O(n). See Module 2: Sorting Algorithms and Fundamental Data Structures, Lesson 2.4: Getting Started with Fundamental Data Structures.

2. What is the maximum number of comparisons that can take place in bubble sort? Assume that there are n elements in the array.

a. (1/2)n(n-1)
b. (1/2)(n-1)
c. (1/4)n(n-1)
d. (1/4)(n-1)

Analysis:

1. Correct. This is because after each pass, the total number of comparisons is reduced by 1. See Module 2: Sorting Algorithms and Fundamental Data Structures, Lesson 2.1: Introducing Bubble Sorting.
2. Incorrect. The maximum number of comparisons is (1/2)n(n-1). This is because after each pass, the total number of comparisons is reduced by 1. See Module 2: Sorting Algorithms and Fundamental Data Structures, Lesson 2.1: Introducing Bubble Sorting.

3. Incorrect. The maximum number of comparisons is (1/2)n(n-1). This is because after each pass, the total number of comparisons is reduced by 1. See Module 2: Sorting Algorithms and Fundamental Data Structures, Lesson 2.1: Introducing Bubble Sorting.
4. Incorrect. The maximum number of comparisons is (1/2)n(n-1). This is because after each pass, the total number of comparisons is reduced by 1. See Module 2: Sorting Algorithms and Fundamental Data Structures, Lesson 2.1: Introducing Bubble Sorting.

3. What will be the output of the following program?

```
import java.util.HashMap;
import java.util.Map;

public class Program {
    public static void main(String... args) {
        Key k1 = new Key();
        Key k2 = new Key();

        Map<Key, String> map = new HashMap<>();

        map.put(k1, "value1");
        map.put(k2, "value2");

        System.out.println(map.get(k1));
        System.out.println(map.get(k2));
    }
}

class Key {
    public int hashCode() {
        return (int)(Math.random() * 100);
    }
}
```

a. null
   null
b. value1
   value2
c. value2
   value2
d. A Runtime Exception will be thrown.

Analysis:

a. Correct. As hashCode returns a different value, we won't be able to find the given keys in the HashMap. See Module 3: Hash Tables and Binary Search Trees, Lesson 3.1: Introducing Hash Tables.
b. Incorrect. As hashCode returns a different value, we won't be able to find the given keys in the HashMap. See Module 3: Hash Tables and Binary Search Trees, Lesson 3.1: Introducing Hash Tables.

c. Incorrect. As hashCode returns a different value, we won't be able to find the given keys in the HashMap. See Module 3: Hash Tables and Binary Search Trees, Lesson 3.1: Introducing Hash Tables.
d. Incorrect. There is no reason for which a Runtime Exception could be thrown from this program. See Module 3: Hash Tables and Binary Search Trees, Lesson 3.1: Introducing Hash Tables.

4. What will be the output of the following program?

```
import java.util.HashMap;
import java.util.Map;

public class Program {
    public static void main(String... args) {
        Key k1 = new Key();
        Key k2 = new Key();

        Map<Key, String> map = new HashMap<>();

        map.put(k1, "value1");
        map.put(k2, "value2");

        System.out.println(map.get(k1));
        System.out.println(map.get(k2));
    }
}

class Key {
    public boolean equals(Object o) {
        return true;
    }
    public int hashCode() {
        return 1;
    }
}
```

a. value2

value2

b. null

null

c. value1

value2

d. A Runtime Exception will be thrown.

Analysis: