**Briefly describe the artifact. What is it? When was it created?**

The artifact I chose is a Trivia app. The trivia application is an app built with HTML/CSS/Javascript. This trivia app was designed and built in a coding bootcamp course I completed in 2022. The app is a quiz that gives the user ten random questions regarding music albums, the user then has to select one of four multiple choices to answer which artist the album belongs to. A total score is given to the user at the end of the quiz, and the user can restart it after completion.

**Justify the inclusion of the artifact in your ePortfolio. Why did you select this item? What specific components of the artifact showcase your skills and abilities in algorithms and data structure? How was the artifact improved?**

I chose this artifact because it represents a comprehensive application of data structures and algorithm, specifically the use of the Binary Search Tree for managing and organizing quiz questions to enhance the functionality and performance of the application.

One of the key aspects of this artifact is the implementation of a Binary Search Tree to store and retrieve quiz questions and display them in order. The questions are inserted into the tree based on the lexicographical order of the question text, and an in-order traversal ensures that the questions are presented to the user in alphabetical order. This implementation showcases my understanding of data structures and the ability to manipulate them for practical use.

The artifact improved in many ways. I implemented this BST to dynamically sort the trivia questions in alphabetical order based on their text. This addition showcased the use of algorithms and data structures to solve a real-world problem. With the implementation of this BST, the app is now more scalable, allowing for the addition of new questions without affecting the overall efficiency. With the app efficiently organizing and retrieving questions in a lexicographical order, the time complexity of insertion and sorting is handled more efficiently compared to manual sorting methods. The in-order traversal of the BST guarantees that questions are presented in the correct order, improving both the user experience and the underlying performance.

**Did you meet the course outcomes you planned to meet with this enhancement in Module One? Do you have any updates to your outcome-coverage plans?**The course outcome I planned to meet was: Design and evaluate computing solutions that solve a given problem using algorithmic principles and computer science practices and standards appropriate to its solution while managing the trade-offs involved in design choices.

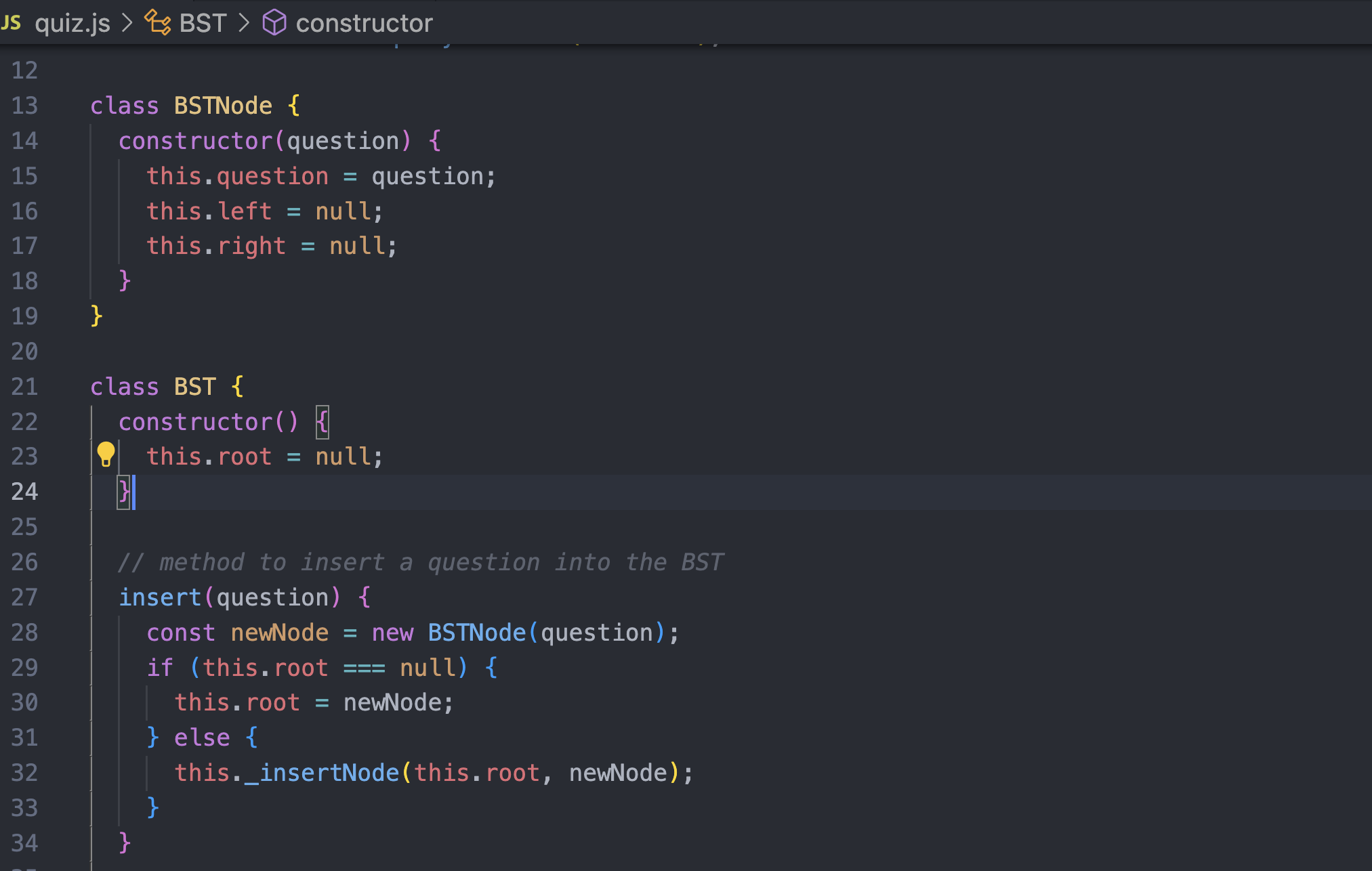
I met this outcome by implementing a BST that leverages key algorithmic concepts, such as inserting elements based on their order and traversing the tree in order to retrieve the sorted list.

The BST has an average time complexity of O(log n) for insertions and O(n) for traversals, which is efficient for the small dataset I’m working with. One of the trade-offs I had to manage was, that if the dataset grows, considerations about balanced trees may need to be made to avoid worst-case O(n) behavior during insertions. This BST fits the scale of the application, managing the trade-off between simplicity and efficiency.

**Reflect on the process of enhancing and modifying the artifact. What did you learn as you were creating it and improving it? What challenges did you face?**

Implementing a BST taught me the benefits of choosing the right data structure based on the problem. Sorting the questions alphabetically through the BST’s in-order traversal is a clear example of how data structures can help organize and manage data more efficiently than manually coding for such tasks.

I’ve never used a BST before so implementing it was challenging at first, mainly the logic for insertion and ensuring the tree balanced itself during operation. Understanding how to properly traverse the tree in order to display the questions in a sorted manner was a key challenge that required multiple redo’s to get right.

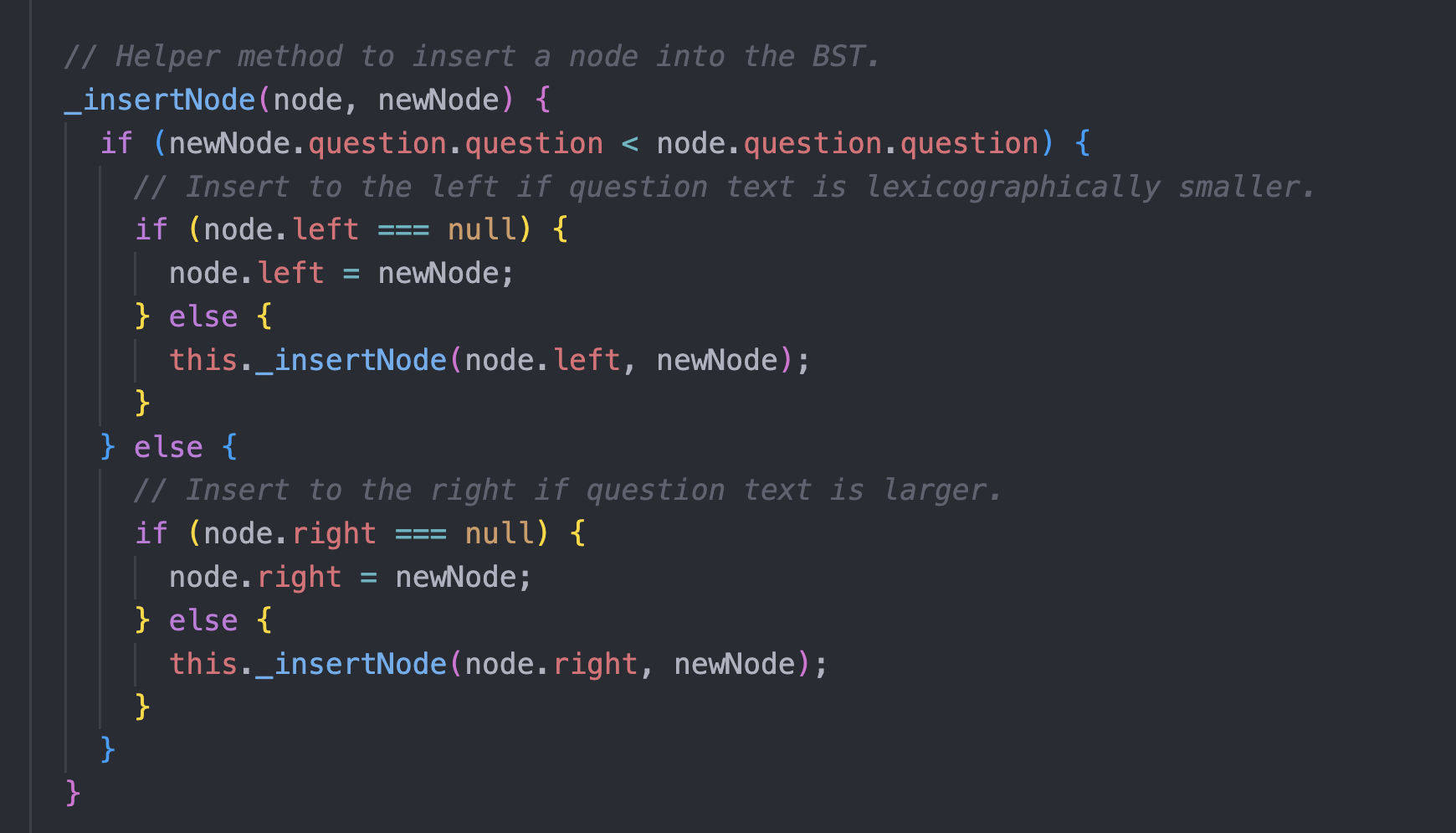


The **BSTNode** class represents a single node in the binary search tree. Each node contains a question object, which holds the trivia question and its details (image, answer choices, correct answer).

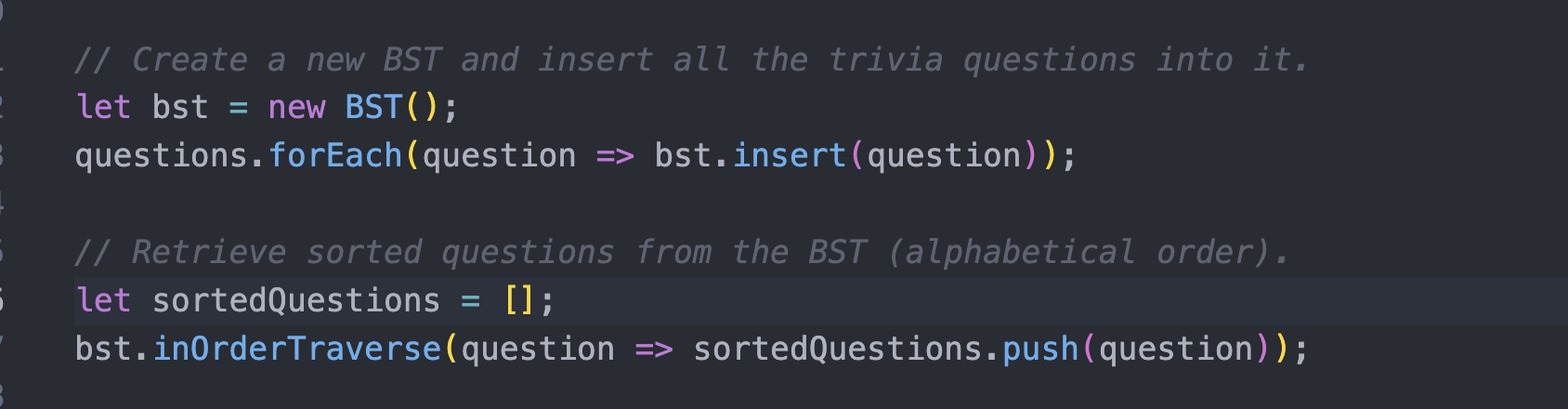
The left and right properties point to the node's left and right children and represent other lexicographically smaller (left) or larger (right) questions.

The **BST** class manages the entire tree structure, starting with an empty root.

The **insert** method inserts a new question into the BST. If the tree is empty, the new question becomes the root node, otherwise, it calls a helper method to find the correct position in the tree.



This method ensures the new question is inserted in the correct position in the tree based on lexicographical order. If the new question's text is smaller than the current node's text, it goes to the left. If it's larger, it goes to the right. It recursively traverses the tree to find the appropriate spot.



When the app starts, all questions are inserted into the BST using the insert(question) method. Then the questions are stored in an ordered fashion based on their question text.

The in-order traversal ensures that the questions are sorted alphabetically, the BST stores the questions in a way that makes it easy to retrieve in this order.