**Project: EduTech Interactive Tutorials on Data Structures and Algorithms**

**Overview:**

This series of interactive tutorials will introduce students to fundamental data structures and algorithms. Students will learn key concepts through engaging video lectures, coding exercises, quizzes, and demonstrations. The tutorials will cover arrays, linked lists, stacks, queues, trees, sorting, and searching algorithms, helping students build a strong foundation in this core area of computer science.

**Learning Objectives:**

By the end of this tutorial series, students will be able to:

* Understand and implement basic data structures such as arrays, linked lists, stacks, and queues.
* Apply various sorting and searching algorithms and evaluate their time complexity.
* Analyze real-world problems and select appropriate data structures and algorithms to solve them efficiently.
* Write and optimize code in Python to implement common algorithms and data structures.

**Tutorial Structure:**

**Week 1: Introduction to Data Structures and Arrays**

* **Interactive Video Lecture**:
  + Overview of Data Structures (linear vs non-linear)
  + Introduction to arrays: definition, operations, and use cases
* **Coding Exercise**:
  + Implement an array in Python and perform basic operations (insertion, deletion, traversal)
* **Quiz**:
  + Multiple-choice questions on array properties and basic operations
* **Assessment Criteria**:
  + Ability to implement and manipulate arrays correctly.
* **Video Demonstration**:
  + Show how to create and manipulate arrays step-by-step, providing explanations of time complexity.

**Week 2: Linked Lists**

* **Interactive Video Lecture**:
  + Introduction to linked lists: singly linked list, doubly linked list
  + Memory management and the advantages/disadvantages of linked lists
* **Coding Exercise**:
  + Implement a singly linked list in Python, including insertions, deletions, and traversals.
* **Quiz**:
  + Test on linked list properties and traversal techniques.
* **Assessment Criteria**:
  + Correct implementation of linked list operations and understanding of memory handling.
* **Video Demonstration**:
  + Walkthrough of linked list creation and operations, with comparisons to arrays in terms of efficiency.

**Week 3: Stacks and Queues**

* **Interactive Video Lecture**:
  + Introduction to stacks and queues: LIFO (Last-In-First-Out) and FIFO (First-In-First-Out) structures
  + Real-world applications of stacks and queues
* **Coding Exercise**:
  + Implement a stack and a queue in Python, using both list and linked list structures.
* **Quiz**:
  + Questions on stack operations (push, pop, peek) and queue operations (enqueue, dequeue).
* **Assessment Criteria**:
  + Proficiency in implementing stack and queue operations and understanding their applications.
* **Video Demonstration**:
  + Demonstrate stack and queue usage in solving practical problems (e.g., balancing parentheses with a stack).

**Week 4: Trees and Binary Search Trees (BST)**

* **Interactive Video Lecture**:
  + Introduction to tree structures: binary trees, binary search trees (BST), and tree traversal methods (in-order, pre-order, post-order)
  + Explanation of BST properties and applications
* **Coding Exercise**:
  + Implement a binary search tree in Python with functions for insertion, deletion, and searching.
* **Quiz**:
  + Multiple-choice questions on tree properties, traversal methods, and operations.
* **Assessment Criteria**:
  + Correct implementation of binary search tree operations and understanding of traversal algorithms.
* **Video Demonstration**:
  + Show how to construct a binary search tree and traverse it using different methods.

**Week 5: Sorting Algorithms**

* **Interactive Video Lecture**:
  + Overview of sorting algorithms: bubble sort, insertion sort, merge sort, quicksort
  + Explanation of time complexity (Big-O notation) for each algorithm
* **Coding Exercise**:
  + Implement bubble sort, merge sort, and quicksort in Python.
* **Quiz**:
  + Test on the time complexity and working principles of sorting algorithms.
* **Assessment Criteria**:
  + Proficiency in implementing sorting algorithms and analyzing their efficiency.
* **Video Demonstration**:
  + Step-by-step walkthrough of sorting an array using different algorithms and comparing their performance.

**Week 6: Searching Algorithms**

* **Interactive Video Lecture**:
  + Introduction to searching algorithms: linear search and binary search
  + Efficiency and use cases for each searching method
* **Coding Exercise**:
  + Implement linear search and binary search in Python, with input validation and performance analysis.
* **Quiz**:
  + Questions on binary search algorithm and its application in sorted arrays.
* **Assessment Criteria**:
  + Ability to correctly implement searching algorithms and evaluate when to use each.
* **Video Demonstration**:
  + Live demonstration of binary search in action, with a discussion on its logarithmic time complexity.

**Week 7: Advanced Algorithms: Graphs and Dynamic Programming**

* **Interactive Video Lecture**:
  + Introduction to graph algorithms: depth-first search (DFS) and breadth-first search (BFS)
  + Overview of dynamic programming concepts
* **Coding Exercise**:
  + Implement graph traversal algorithms (DFS and BFS) in Python.
  + Solve a dynamic programming problem (e.g., Fibonacci sequence with memoization).
* **Quiz**:
  + Questions on graph traversal techniques and dynamic programming principles.
* **Assessment Criteria**:
  + Understanding and implementation of advanced algorithms.
* **Video Demonstration**:
  + Demonstration of solving a graph problem using DFS and BFS, followed by a dynamic programming example.

**Assessment Plan:**

1. **Quizzes** (30%): Weekly multiple-choice questions to test understanding of key concepts.
2. **Coding Exercises** (50%): Hands-on coding assignments after each lecture to implement data structures and algorithms.
3. **Final Project** (20%):
   * Develop a real-world application that requires implementing multiple data structures (e.g., a route planner using a graph, a task scheduler using a queue, or a sorting algorithm for large datasets).
   * Assessment Criteria: Proper selection and implementation of data structures, optimization of code, and clarity in commenting and documentation.