**Binary Search Tree Pseudocode**

**Main Flow:**

BEGIN

DISPLAY menu

WHILE user does not choose to exit

READ user choice

SWITCH on choice

CASE 1:

CALL loadCoursesIntoBST

CASE 2:

CALL bstInOrderTraversalToDisplay

CASE 3:

PROMPT user for course ID

CALL findAndDisplayCourseInfo

CASE 9:

EXIT

END SWITCH

END WHILE

END

**Loading Courses into Binary Search Tree:**

FUNCTION loadCoursesIntoBST(filePath, bst)

OPEN file at filePath

FOR each line in file

PARSE line into courseID, title, and prerequisites

CREATE new Course object with parsed data

CALL bst.insert(Course object)

END FOR

CLOSE file

END FUNCTION

**Displaying All Courses In Order:**

FUNCTION bstInOrderTraversalToDisplay(bst)

CALL bst.inOrderTraversal()

END FUNCTION

FUNCTION inOrderTraversal(node)

IF node is not null

CALL inOrderTraversal(node.left)

PRINT node.courseID and node.title

CALL inOrderTraversal(node.right)

END FUNCTION

**Finding and Displaying Course Information:**

FUNCTION findAndDisplayCourseInfo(bst, courseID)

SET current to bst.root

WHILE current is not null

IF current.courseID equals courseID

PRINT current.title and current.prerequisites

RETURN

ELSE IF courseID < current.courseID

current = current.left

ELSE

current = current.right

PRINT "Course not found."

END FUNCTION

**Runtime Analysis Chart**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Data Structure | Operation | Line Cost | # Times Executes | Total Cost | Runtime |
| Binary Search Tree | Load Courses into BST | 1 | log(n) | log(n) | O(log(n))(avg) / O(n) (worst) |
|  | In-Order Traversal (Sort) | 1 | n | n | O(n) |
|  | Search for Course | 1 | log(n) | log(n) | O(log(n))(avg) / O(n) (worst) |

**Advantages and Disadvantages Analysis**

**Advantages:**

* Efficient Searching: BSTs provide O(log(n)) average-case time for search, insertion, and deletion operations when balanced.
* In-Order Traversal: BSTs allow for easy in-order traversal, making it straightforward to list all courses in alphanumeric order without additional sorting.
* Dynamic Structure: Unlike vectors, BSTs don't require shifting elements around when inserting or deleting elements.

**Disadvantages:**

* Balancing Required: BSTs can degrade to O(n) performance if they become unbalanced, which can occur if insertions are not random.
* Complex Implementation: Implementing self-balancing BSTs is more complex than vectors or basic hash tables.
* Pointer Management: BSTs require careful management of pointers, which can lead to increased complexity and bugs in the implementation.