### Pseudocode for Vector

**1. Open, Read, Parse File, and Check Formatting Errors**

function loadCoursesFromFile(filename: String) -> Vector<Course>:

open file with name filename

initialize Vector<Course> courses

for each line in file:

parse line to extract courseNumber, courseTitle, prerequisites

if line format is incorrect:

print "Formatting error on line: " + line

else:

create Course object with courseNumber, courseTitle, prerequisites

add Course to courses

close file

return courses

**2. Create Course Object**

class Course:

String courseNumber

String courseTitle

Vector<String> prerequisites

constructor(courseNumber: String, courseTitle: String, prerequisites: Vector<String>):

this.courseNumber = courseNumber

this.courseTitle = courseTitle

this.prerequisites = prerequisites

**3. Print Course Information and Prerequisites**

function searchCourse(courses: Vector<Course>, courseNumber: String):

for each course in courses:

if course.courseNumber == courseNumber:

print "Course: " + course.courseNumber + " - " + course.courseTitle

print "Prerequisites: " + course.prerequisites

return

print "Course not found"

**4. Menu**

function displayMenu():

print "1. Load Data"

print "2. Print Course List"

print "3. Print Course Information"

print "9. Exit"

choice = get user input

return choice

function main():

Vector<Course> courses

while true:

choice = displayMenu()

if choice == 1:

filename = get user input for filename

courses = loadCoursesFromFile(filename)

elif choice == 2:

printCourseList(courses)

elif choice == 3:

courseNumber = get user input for course number

searchCourse(courses, courseNumber)

elif choice == 9:

break

else:

print "Invalid choice"

**5. Print Sorted Course List**

function printCourseList(courses: Vector<Course>):

sort courses by courseNumber

for each course in courses:

print course.courseNumber + ": " + course.courseTitle

### Pseudocode for Hash Table

**1. Open, Read, Parse File, and Check Formatting Errors**

function loadCoursesFromFile(filename: String) -> HashTable<String, Course>:

open file with name filename

initialize HashTable<String, Course> courses

for each line in file:

parse line to extract courseNumber, courseTitle, prerequisites

if line format is incorrect:

print "Formatting error on line: " + line

else:

create Course object with courseNumber, courseTitle, prerequisites

add course to courses with courseNumber as key

close file

return courses

**2. Create Course Object**

class Course:

String courseNumber

String courseTitle

Vector<String> prerequisites

constructor(courseNumber: String, courseTitle: String, prerequisites: Vector<String>):

this.courseNumber = courseNumber

this.courseTitle = courseTitle

this.prerequisites = prerequisites

**3. Print Course Information and Prerequisites**

function searchCourse(courses: HashTable<String, Course>, courseNumber: String):

if courses.containsKey(courseNumber):

course = courses.get(courseNumber)

print "Course: " + course.courseNumber + " - " + course.courseTitle

print "Prerequisites: " + course.prerequisites

else:

print "Course not found"

**4. Menu**

function displayMenu():

print "1. Load Data"

print "2. Print Course List"

print "3. Print Course Information"

print "9. Exit"

choice = get user input

return choice

function main():

HashTable<String, Course> courses

while true:

choice = displayMenu()

if choice == 1:

filename = get user input for filename

courses = loadCoursesFromFile(filename)

elif choice == 2:

printCourseList(courses)

elif choice == 3:

courseNumber = get user input for course number

searchCourse(courses, courseNumber)

elif choice == 9:

break

else:

print "Invalid choice"

**5. Print Sorted Course List**

function printCourseList(courses: HashTable<String, Course>):

keys = courses.keys()

sort keys

for each key in keys:

course = courses.get(key)

print course.courseNumber + ": " + course.courseTitle

### Pseudocode for Binary Search Tree

**1. Open, Read, Parse File, and Check Formatting Errors**

function loadCoursesFromFile(filename: String) -> Tree<Course>:

open file with name filename

initialize Tree<Course> courses

for each line in file:

parse line to extract courseNumber, courseTitle, prerequisites

if line format is incorrect:

print "Formatting error on line: " + line

else:

create Course object with courseNumber, courseTitle, prerequisites

add Course to tree using courseNumber as key

close file

return courses

**2. Create Course Object**

class Course:

String courseNumber

String courseTitle

Vector<String> prerequisites

constructor(courseNumber: String, courseTitle: String, prerequisites: Vector<String>):

this.courseNumber = courseNumber

this.courseTitle = courseTitle

this.prerequisites = prerequisites

**3. Print Course Information and Prerequisites**

function searchCourse(courses: Tree<Course>, courseNumber: String):

course = courses.search(courseNumber)

if course is not null:

print "Course: " + course.courseNumber + " - " + course.courseTitle

print "Prerequisites: " + course.prerequisites

else:

print "Course not found"

**4. Menu**

function displayMenu():

print "1. Load Data"

print "2. Print Course List"

print "3. Print Course Information"

print "9. Exit"

choice = get user input

return choice

function main():

Tree<Course> courses

while true:

choice = displayMenu()

if choice == 1:

filename = get user input for filename

courses = loadCoursesFromFile(filename)

elif choice == 2:

printCourseList(courses)

elif choice == 3:

courseNumber = get user input for course number

searchCourse(courses, courseNumber)

elif choice == 9:

break

else:

print "Invalid choice"

**5. Print Sorted Course List**

function printCourseList(courses: Tree<Course>):

courseList = inOrderTraversal(courses)

for each course in courseList:

print course.courseNumber + ": " + course.courseTitle

function inOrderTraversal(tree: Tree<Course>) -> Vector<Course>:

result = empty Vector<Course>

if tree is not null:

result.addAll(inOrderTraversal(tree.left))

result.add(tree.value)

result.addAll(inOrderTraversal(tree.right))

return result

### Runtime Analysis

#### Vector

**Load Courses From File**

* Opening file: O(1)O(1)O(1)
* Parsing and loading each line: O(n)O(n)O(n)

**Search Course**

* Iterating through courses: O(n)O(n)O(n)

**Print Course List**

* Sorting courses: O(nlog⁡n)O(n \log n)O(nlogn)
* Printing courses: O(n)O(n)O(n)

**Total**: Loading: O(n)O(n)O(n), Search: O(n)O(n)O(n), Print: O(nlog⁡n)O(n \log n)O(nlogn)

#### Hash Table

**Load Courses From File**

* Opening file: O(1)O(1)O(1)
* Parsing and loading each line: O(n)O(n)O(n)

**Search Course**

* Hash lookup: O(1)O(1)O(1)

**Print Course List**

* Extracting keys: O(n)O(n)O(n)
* Sorting keys: O(nlog⁡n)O(n \log n)O(nlogn)
* Printing courses: O(n)O(n)O(n)

**Total**: Loading: O(n)O(n)O(n), Search: O(1)O(1)O(1), Print: O(nlog⁡n)O(n \log n)O(nlogn)

#### Binary Search Tree

**Load Courses From File**

* Opening file: O(1)O(1)O(1)
* Parsing and loading each line: O(nlog⁡n)O(n \log n)O(nlogn)

**Search Course**

* Tree search: O(log⁡n)O(\log n)O(logn)

**Print Course List**

* In-order traversal: O(n)O(n)O(n)
* Printing courses: O(n)O(n)O(n)

**Total**: Loading: O(nlog⁡n)O(n \log n)O(nlogn), Search: O(log⁡n)O(\log n)O(logn), Print: O(n)O(n)O(n)

### Advantages and Disadvantages

#### Vector

**Advantages**:

* Simple to implement.
* Efficient for sequential access.

**Disadvantages**:

* Inefficient for search operations (O(n)O(n)O(n)).
* Requires sorting for ordered print (O(nlog⁡n)O(n \log n)O(nlogn)).

#### Hash Table

**Advantages**:

* Fast search operations (O(1)O(1)O(1)).
* Efficient insertion and deletion (O(1)O(1)O(1)).

**Disadvantages**:

* Does not maintain order.
* Requires additional space for hash table structure.

#### Binary Search Tree

**Advantages**:

* Maintains sorted order of elements.
* Efficient search operations (O(log⁡n)O(\log n)O(logn)).

**Disadvantages**:

* More complex to implement.
* Performance degrades if the tree becomes unbalanced (O(n)O(n)O(n) in worst case).

### Recommendation

Based on the Big O analysis and the needs of the advising program, the **hash table** data structure is recommended. It offers the fastest search time, which is crucial for quickly finding course information. Although it does not maintain order, we can extract and sort the keys when needed to print the course list. This approach provides a good balance of efficiency and functionality for the advising program.

4o