**Project One**

**Pseudocode:**

***//Vector***

# Print the list of courses in alphanumeric order

def print\_course\_list(vector):

# Sort the vector in alphanumeric order

vector.sort()

# Print the sorted list

for course in vector:

print(course.course\_number, course.title)

# Print the course title and prerequisites for a given course

def print\_course(vector, course\_number):

# Find the course object in the vector

course = None

for c in vector:

if c.course\_number == course\_number:

course = c

break

# If the course object was not found, print an error message

if course is None:

print("Course not found")

else:

print(course.title)

print("Prerequisites:")

for prerequisite in course.prerequisites:

print(prerequisite)

***//Hash Table***

# Print the list of courses in alphanumeric order

def print\_course\_list(hash\_table):

# Create a list to store the sorted course numbers

sorted\_course\_numbers = []

for course\_number in hash\_table.keys():

sorted\_course\_numbers.append(course\_number)

# Sort the list in alphanumeric order

sorted\_course\_numbers.sort()

# Print the sorted list

for course\_number in sorted\_course\_numbers:

course = hash\_table[course\_number]

print(course.course\_number, course.title)

# Print the course title and prerequisites for a given course

def print\_course(hash\_table, course\_number):

# Find the course object in the hash table

course = hash\_table.get(course\_number)

# If the course object was not found, print an error message

if course is None:

print("Course not found")

else:

print(course.title)

print("Prerequisites:")

for prerequisite in course.prerequisites:

print(prerequisite)

***//Tree***

# Print the list of courses in alphanumeric order

def print\_course\_list(tree):

# Create an in-order iterator for the tree

iterator = tree.in\_order\_iterator()

# Print the course title for each node in the tree

for node in iterator:

course = node.data

print(course.course\_number, course.title)

# Print the course title and prerequisites for a given course

def print\_course(tree, course\_number):

# Find the node in the tree that contains the course object

node = tree.find(course\_number)

# If the node was not found, print an error message

if node is None:

print("Course not found")

else:

course = node.data

print(course.title)

print("Prerequisites:")

for prerequisite in course.prerequisites:

print(prerequisite)

|  |  |  |  |
| --- | --- | --- | --- |
| **Operation** | **Vector** | **Hash Table** | **Tree** |
| Create Course Object | O(1) | O(1) | O(1) |
| Add Course to Data Structure | O(n) | O(1) | O(log n) |
| Read File and Create Course Objects | O(n^2) | O(n) | O(n log n) |

**Vector**

*Advantages:*

* Simple to implement.
* Efficient for sequential access

*Disadvantages:*

* It is not efficient for searching or inserting elements in arbitrary positions.

**Hash table**

*Advantages:*

* Efficient for searching and inserting elements.

*Disadvantages:*

* Can be inefficient for sequential access.
* It can be more complex to implement.

**Tree**

*Advantages:*

* Efficient for searching and inserting elements.
* Efficient for sequential access

*Disadvantages:*

* It can be more complex to implement.

**Recommendation**

I recommend using a hash table, the main advantage of using a hash table over a vector or tree is that hash tables are more efficient for searching and inserting elements. This is because hash tables use a hash function to directly map each course number to a specific location in the data structure. This allows us to find or insert a course object in constant time, on average. Vectors and trees, on the other hand, require us to traverse the data structure to find or insert an element. This can be inefficient, especially if the data structure is large. Another advantage of using a hash table is that they are relatively simple to implement. There are many well-known algorithms for implementing hash tables, and there are also many libraries that provide pre-implemented hash table data structures. Finally, hash tables have good memory performance, they typically use less memory than vectors or trees, because they do not need to store any additional information about the order of the elements in the data structure. Ultimately, I recommend using a hash table for the course object data structure. Hash tables are efficient, simple to implement, and have good memory performance.