Pseuodocode design for menu:

while not exit

print choices and get input

if input == load

do load

if input == printall

do printall IN ALPHABETICAL ORDER

if input == printcourse

do print(passed course)

if iput == exit

exit

}//while not exit, it will print choices and get input over and over

Pseudocode design for: printing all objects IN ALPHABETICAL ORDER

*vector and hash table (both implemented with vector)*

vector:

run a sorting algorithm on vector

linearly sort said vector, comparing entries and moving as needed to sort the vector then linearly print the entire list

hash table:

dont want to unmap the keys and indexes, so copy said vector, then sort as stated above. and print from sorted vector

BST:

traverse in order, printing each node to display

Pseudocode analysis VECTOR:

vector of courses objects

each course object has class number, title, n prerequisites

To read and store the course objects into a vector:

while not eof n

each line count commas 1

if commas < 1 error 1

if commas == 1 1

while not comma and not end of line 1

put into course.number 1

skip comma 1

while not end of line 1

put into course.name 1

if commas >= 2 1

for commas – 1

put into an array of prereqs 1

put number, name, and vector of prereqs into new course object 3

put course object into main vector 1

iterate

This algorithm is based on the input size – evaluates to n.

the other lines are not in terms of n and omitted, n is the highest order, the big O for reading, creating and storing in class objects and then into the data structure of vector is O(n).

so O(n) for a linked list via vector.

Pseudo code analysis HASH TABLE:

to read and store the items into a hash table:

while not eof n

each line count commas 1

if commas < 1 error 1

if commas == 1 1

while not comma and not end of line 2

put into course.number 1

skip comma 1

while not end of line 1

put into course.name 1

if commas >= 2 1

for commas – 1

put into an array of prereqs 1

put number, name, and vector of prereqs into new course object 3

put course object into main vector via hash table by

courseId % 10 = index of main vector to place Course object into 1

if index is not empty put into next available (+1) index

iterate

evaluates to n.

the other lines within are are not in terms of n and omitted, n is the highest order, the big O for hash table via vector is O(n)

Thus the big O is O(n)

Pseudocode analysis BINARY SEARCH TREE:

to read and store the items into a BST:

while not eof n

each line count commas

if commas < 1 error 1

if commas == 1 1

while not comma and not end of line 1

put into course.number 1

skip comma 1

while not end of line 1

put into course.name 1

if commas >= 2 1

for commas – 1

put into an array of prereqs 1

put number, name, and vector of prereqs into new course object 1

put course object into binary search tree by courseId:

if tree is empty put course into binary search tree as first node 1

else

if courseId is greater and right not null make it the right node 1

if less and left not null make it the left node 1

iterate

the overall loop structure is controlled by the input size of the list of classes

the other instructions are not in terms of n, so are omitted

thus O(n)

|  |  |
| --- | --- |
| Data structure to read from file and create objects | Big O |
| Vector | O(n) |
| Hash Table | O(n) |
| Binary Search Tree | O(n) |

Upon seeing this, I don’t feel like any one decision can be made on which data structure to be used can be made. BUT if we take into space considerations, AND time in which the theoretical pseudocode of the requirement that alphabetizes and then prints all of the course objects, I think a clear structure is most advantageous. A quick extra analysis of each:

vector: space considerations: n

if you bucket sorted the n sized array the time efficiency is T(n+k)

hash table:

a common size for hash tables is 2n + 1

one of the disadvantages for sorting the hash table is that in order to keep the integrity of the mapped values, in order to put them into some sort of order you would have to copy the hash table to a vector or array that you can modify. If the whole hash table was full, this is now 4n+2 for space complexity.

The bucket sort of the copied vector/array would be the same as above.

Binary Search tree:

size complexity is n

items are added following the BST ordering rules comparing on the alphanumeric ordering of the course objects then no extra sorting algorithm is needed, and you will just traverse inorder.

Due to this extra information, I think that the clear choice to use would be the Binary Search Tree.