Pseudocode for menu:

Include time.h

Set choice equal to 0

Create variable for timer

Define a structure to hold the course list

While (choice is not 9) {

Print “Menu”

Print “1. Load courses into the system”

Print ”2. Print full course list”

Print “3.Print specific course”

Print “9. Exit”

If (choice is equal to 1) {

Ticks= clock()

Complete method call to load courses

Define the timer variable

Print number of ticks

Print number of seconds

}

Else If (choice is equal to 2) {

Call to print function

}

Else If (choice is equal to 3) {

Calling the timer variable

Setting course equal to the search function to get the course

}

Else if (choice is equal to 9) {

Exit program

}

Else {

Print sorry that is not a valid function choice. Please try again

}

}

Pseudocode for writing a print statement

InOrder(node) {

If (node isn’t empty) {

Recursively call inOrder function to print left side of the tree

Print course number and name

Recursively call inOrder function to print right side of the tree

}

}

InOrder() {

Send to secondary InOrder function with parameter root

}

BST->inOrder print()

Binary Search:

Reading the file

|  |  |  |  |
| --- | --- | --- | --- |
| Code | Line cost | #times executes | Total cost |
| Open file | 1 | Log n | Log n |
| File doesn’topen | 1 | Log n | Log n |
| Print file didn’t open | 1 | 1 | 1 |
| Runtime cost |  |  | 2 Log n+1 |

Creating new course objects

|  |  |  |  |
| --- | --- | --- | --- |
| Code | Line cost | #times executed | Total cost |
| Extra code | 1 | 1 | 1 |
| Prerequisites | 5 | Log n | 5\* Log n |
| Check to see if in file | 2 | Log n | 2\* Log n |
| First course | 3 | Log n | 3\* Log n |
| Search for spot | 8 | Log n | 8\* Log n |
| Move down left side | 4 | Log n | 4\* Log n |
| Move down right side | 4 | Log n | 4\* Log n |

|  |
| --- |
| 26 Log n +1 |
| 28 Log n +2 |

|  |
| --- |
| Runtime cost |
| Total cost |

Hash Table:

Reading the file

|  |  |  |  |
| --- | --- | --- | --- |
| Code | Line cost | #times executed | Total cost |
| Open file | 1 | 1 | 1 |
| Check to see if file opens | 4 | n | 4\*n |

Creating new course objects

|  |  |  |  |
| --- | --- | --- | --- |
| Code | Line cost | # times executed | Total cost |
| Number prerequisites | 18 | N | 18 \* n |
| Placing entries | 22 | N | 22 \* n |
| Total combined cost |  |  | 44\* n +1 |
|  |  |  |  |

Linked List

Reading the file

|  |  |  |  |
| --- | --- | --- | --- |
| Code | Line cost | # times executed | Total cost |
| Open the file | 1 | N^2 | N^2 |
| File open check | 4 | N^2 | 4\*n^2 |

Creating new course objects

|  |  |  |  |
| --- | --- | --- | --- |
| Code | Line cost | # times executed | Total cost |
| If it has prerequisites | 15 | N^2 | 15\*n^2 |
| Get course info | 3 | 1 | 3 |

Total cost: 20\*n^2+2

Explanation and recommendation:

Binary search tree:

Advantages:

The runtime of the worst-case scenario which is the fastest of the three types of data structures that we have analyzed for this class. It is also sorted in such a way that would be good for us to be able to find the class numbers. If we are using our information for the courses, then there are 8 courses there would be 1/2/4/1 structure which means that there would only have to be 4 searches in the worst-case scenario. This is less than the others and would make for a faster time or running the program.

Disadvantages:

They tree must have a balanced structure for the program which could devolve into an array. This would cause the runtime to be much more convoluted and it can be a tough problem to overcome.

Hash Table:

Advantages:

There is a key structure that helps to be able to search for specific keys within the hash table that can allow for the ease of finding the course number relatively quickly. They are also sorted by buckets which could allow for a much easier time when it comes to looking through each with the 100,200, 300, and 400. This would lower search time tremendously

Disadvantage:

The problem lies when a course gets placed at the end of the list and there is further courses added. There could be multiple courses with the same number but through different section like a math 400 and a CS 400 might come up as the same one. There could be much more collisions that would slow down the runtime and could make the program run slower than other data structures.

Vector/LinkedList:

Advantage:

The vector data structure can grow and shrink as the program runs. There is no memory waste, and the space is created when it is needed. Implementation can be considered very easy for our data and the insert and delete functions are east to understand as compared to the others.

Disadvantages:

There is a lot of memory requirements. There is much more than required for the other two. Despite there being memory created and destroyed there still must be memory left available in the system. Traversal is also slower than with the other two. Each spot must be checked one at a time and with a worst-case scenario this could be a large list.

Recommendation:

I suggest that for this program we use a Binary Search Tree Data structure. It has the best runtime for worst case scenario and the others are a longer time to do the search function that we will require. There is less memory than the vector structure and a lower amount of check to complete than the Hash Table.