Struct Course()

String courseID

String courseName

Vector courseList

Vector coursePre

Int preCount = 0

Open file()

Locate file

Open file

If (file not open)

Print “File failed to open”

Return

Close file()

If (file is not open)

Return

Close file

If (file is open)

Print “ File failed to close”

Return

Load Data()

Read through each line

While (not end of file)

Add

If (items on list < 2)

Return error

Break

courseId = first item

courseName = second item

Add Course(courseId, courseName)

if (more items)

item added to coursePre

preCount++

Add Course(courseId, courseName)

Add new node

Node.courseId = courseId

Node.courseName = courseName

Add new node to vector courseList

Alphanumeric Print()

Std::sort(courseList.begin(), courseList.end(), compareFunciton)

For (int i =0; i < courseList.length(); i++)

Print courseList[i], endl;

Print Prereq(courseId)

Print courseId + coursePre

Courses(courseId)

This->courseId = courseId

For (int i = 0; i < courseList.size(); i++)

If ( courseList.courseId == courseId)

Print Prereq(courseId)

Print Menu()

Print “Option 1: Load file data.”

Print “Option 2: Print courses alphanumerically.”

Print “ Option 3: Print the course title and its prerequisites.”

Print “Option 4: Quit”

Main()

int choice = 0

string courseSearch

while (choice != 9)

Print Menu()

Input = choice

Switch (choice)

Case 1:

Open file()

Load data()

break

Case 2:

Alphanumeric Print()

break

Case 3:

Print “Enter Course ID”

Input = courseSearch

Courses(courseSearch)

break

Default:

Print “Invalid input”

Close file()

Print “Goodbye”

Return 0

**Runtime Analysis**

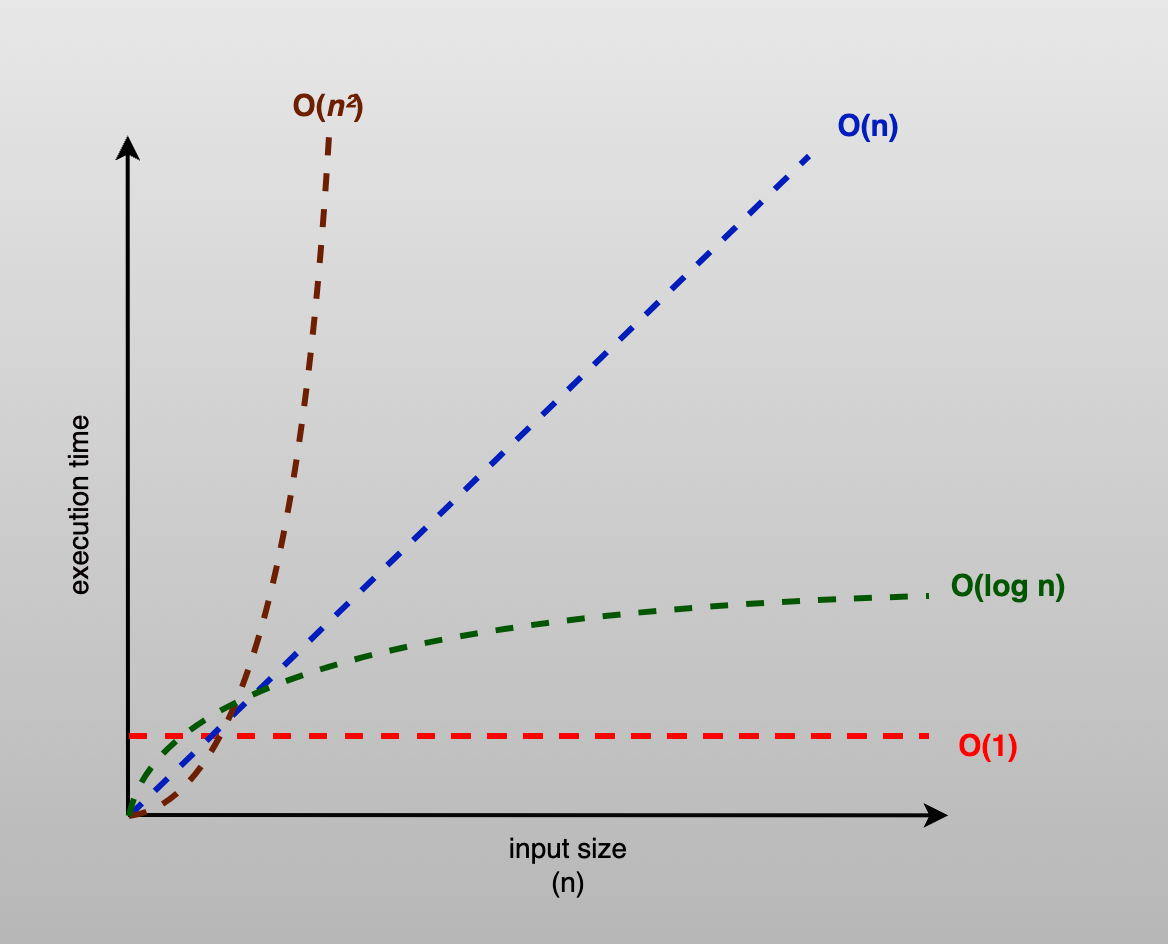


Image from Medium.com

Blue Line: Linked List O(N)

Red Line: Binary Search Tree O(log N)

Orange Line: Hash Tables O(1)

For ABC University they want to be able to load their courses into their system, recall them, and later present them as requested by the user. While there are many ways to program this system, each with differing amounts of speed, there is no prefect system in which they must choose. Rather, it’s important to understand the pros and cons for each option. Here I will go through three options and at the end give my personal recommendation.

The Linked List, represented by the blue line, is an option that allows for simple data management. It acts like a chain in which new data points are added to the end in the order received. It uses a system of pointers that tell the system where to look next for the specific information requested. Simply put, it’s a useful tool for adding information. However, the ease at which data is entered ultimately costs the system in its recall speed. Linked Lists are incapable of using indexing to locate specific information and is required to analyze each bit of information until the requested item is found.

Binary Search Trees (BST), represented by the red line, are like the linked list. It uses pointers to identify what follows. What’s different is that each node in a BST has at most two child nodes. These nodes are organized in a way that allows for organized searches that, while also lacking index searches, make searches return results fast.

Lastly, we have the Hash Table, represented by the orange line. Of all the options Hash tables can recall their information the fastest. The way it works is a vector is set up and instead of data being put in first to last, the information is put through an equation and assigned an index point. The system is designed to handle large amounts of data and recall it faster. The caviate is it requires more attention in the coding process.

It would be simple to recommend the fastest recall option for the ABCU system, however this recall speed matters little with smaller data sets. Therefore, I recommend the usage of Binary Search Trees. The set up and maintenance is going to be easier, and there is little to no sacrifice in speed.

Citation

Talreja, Priya. “Time Complexity in Datastructure-Big O Notation.” *Medium*, Medium, 3 June.2023,medium.com/@priya\_talreja/time-complexity-in-datastructure-big-o-notation-9e4db1621b45.