4. Evaluate the runt-time and memory of data structures.

1. When the program opens the file, It validates each line as it reads them. Validating each line requires checking if there is a comma and if there are less than 7 characters before that first comma else if the program completes these will always be the same result of this part of the loop otherwise the program will end early so we can assume the big 0 will always read them as a correct file. Then it will parse each line by the amount of commas which will vary by the number of commas in each line the first 7 characters being the course ID, the characters after that being the course name until either the next comma or end of the line. Every prerequisite that the course could have will be parse from either the second comma to the end or next prerequisite this will happen until parse section is the last section.
2. PseudoCode For and Costs for Opening File
   1. Open File **Cost** 1
      1. For Each Line in File **Cost** N = Number of Lines
3. Set Temp Value to Current Line **Cost** 1
4. Set Int Count Value to Beginning to End of String number of Commas **Cost** 1
5. Validate Course Function **Cost 2**
6. Int CurrComma = 0
7. For Each Comma +1 **Cost M = Number of Commas + End of Line**

IF CurrComma = 0 // **Cost 1**

String CourseID = First 7 characters **Cost 1**

CurrComma = Temp.find(“,”) **Cost 1**

Else IF CurrComma = 7 // **Cost 1**

String CourseName = CurrComma to Next Comma or End of Line **Cost 1**

CurrComma = Next Comma Location **Cost 1**

Else IF CurrComma > 7 **Cost 1**

Add to Vector<String> PreReq

CurrentComma to Next or End of Line **Cost 1**

CurrComma = Next Comma Location **Cost 1**

1. Create Object( CourseID, CourseName, PreReq) **Cost 1**.

5.

1. Vector
   1. Advantages : Easily adds the object into the vector and built in sorting methods this makes it easy to understand and code and simple to explain.
   2. DisAdvantages : Sorting the data is difficult since information can only be added at the beginning or end of the vector; a sort function will need to be run every time new data is entered or when printed to keep the information sorted and the sort time is longer than the other two but both lead methods add on more runtime. Search requires combing through the entire vector until the correct one is found leading to the largest Big O search..
2. Hash Table
   1. Advantages : Adds Objects into the structure based on a key, this key can be used to help sort the information as it enters and insert them non-linearly helping to increase the speed in which the print function occurs as the information will already be mostly sorted depending on the method of collision.Searching is also fast as it searches by the key and goes directly to close location.
   2. DisAdvantages : The hash table reserves memory space for unused data so its memory requirements are the highest. It's also the most complex to write and troubleshoot when trouble occurs
3. Binary Tree
   1. Advantages: A linked list binary tree is perhaps the middle ground between the two in searchability and adding to the structure and as it also sorts the information as it inserts it in fact more cleanly than a hash table would assuming the data we are provided is unsorted. Its search times are faster than the vector method but slower than the hash tables. While printing in order it will be faster than the vector because it’ll be sorted on insertion faster and faster than hashtable as there will be no collision chances to validate order or vectors inside the buckets to sort.
   2. DisAdvantages : The Binary tree information to be efficient, needs its information provided out of order to avoid creating a linked list instead of a binary tree, searching information will almost always be slower than a hash table. If the functions are able to find specific items often a binary tree will take more time if the numbers keep getting larger. It will also take less space than a hash table.

6. Recommendation

My Recommendation is based off the following factors, 1st that the order of the information we were provided in the course information doc is the order we’d receive it to parse in the file, second that this program would be intended to expand into a larger program that included other departments not just the computer science department.

I will first go over why I didn’t choose the binary tree. If the information provided is given in that order the information would have to be simply sorted by the course number they are already in order and would give us the deepest binary tree, the other reason I felt the binary tree would be difficult was the idea that this would be scaled into other programs at ABC university. If that happened then when courses needed to have multiple 100 courses it would become a problem as if there are 6 101 courses and we are dropping multiple layers down straight without branching and being very inefficient in the tree.

Now I will go over why I’d rather use a hash table, the hash table for an individual search will have the fastest lookup time, if you’re trying to find a single course most cases it will be the fastest, As for printing all Its difference from the binary tree will be clear but only with the smaller groups If my intended expectation is correct and this is expanded into the entire campus course then the difference will not be a clear from the binary tree.