Here we will evaluate the run time and explain the program parameters in detail to understand the pseudocode. We will touch on all aspects of the sorting and the menu as well. Throughout this we have been working with the different sorting types and created pseudocode for each of them as they would pertain to the courses for the computer science program at ABCU. We will also look at the runtime analysis of the program and the runtime algorithm of the big o which in turn would be O(n).

Let’s start with the Pseudocodes we worked on for each type of sorting. In this we used Vector sorting, Hash Tables, Tree Branch, and Linked List. Each sorting feature has their own special way of sorting the information. Depending on how the preference of the user they will get to choose how the data is sorted and how they view it. When presented with the menu, which we will discuss later, the user will get the option to choose the sorting feature they want. In each sorting concept they will be presented with the same data, only the view will be different. Each sorting option will start with opening and reading the files. Which means no matter what option in the menu the user chooses the program will open the files and read them first. We use the call to open files and read files to have the program open the files at the start of the ort and read them. As it reads them it will have a code design to read each one and check for any errors.

After the files are read in the program you will have the sorting begin to print out the course and the course information depending on what is chosen. While it is sorting you will have courses removed from the order and put back in once things are in the order they need to be. It will bring out courses with prerequisites and more of them to the side in a separate class and then move them to the list when it is their turn. The design will move the prerequisite courses out to place in the courses that do not have prerequisites and then take the courses with prerequisites put into the sorted information.

When looking at each Pseudocode for each sorting we have to take into account the runtime analysis to see which would be best- and worst-case scenarios for the sorting systems we are using. We will assume that each line costs 1 for executing, and then if they must call a function, we will assume total cost 1+n. For the worst-case scenario, we would look at it as 5n+1, since most of the executions and runtimes will cap out at total costs of 4n+1, based on our example provided in our material. For the runtime itself it will be based on the O(n) algorithms.

Let’s take a look at each sorting option now and dive deeper with them. We will start with Vector sorting. In this it will take a look at all of the course and return those that have prerequisite classes and continue through it until it gets through all of them. As it goes it will continue to dd those classes to the lists and display them in order from those with no prerequisites to those with prerequisites. This can be helpful because the classes are looked at one by one and put into the order they are wanted and can lead to less errors. But also, in the same way this can be a disadvantage because it does go through tings one by one and can take time which can be a killer in the long run especially if an error does come to play. The next we have is the HashTable. This breaks everything up for us and gives us key pairs. This will allow us to see the course number and course name at the same time in a clear to read table for the user. In this it will start with classes that do not require prerequisites and place them at the top of the table and work its way down to those that do require prerequisites. During the run of this, it will check the courses along the way and as non-prerequisite classes are found it will place them into the list at the top in alphanumeric order. The last section we worked was the Binary tree. In the Binary tree you essentially have branches going to the left and the right. We start this with a null node and start going through the courses. As it finds a course that requires no prerequisite it places it at the top of the tree, as it travels along non-prerequisite classes will go to the left branch and those that do have prerequisites will go to the right branch. As it is going through the sorting it will give us a full tree with the needs of the courses in alphanumeric order. This like the Has Table is in an easy-to-read tabled format and depending on what direction the course goes the branch will go that route. The disadvantage is while putting in alphanumeric order it could take longer, or the branch could not go out like you expect.

Based on all of the options we have and the runtime, I would recommend using the has table when running the program and sorting everything. Besides it runs the fastest because it keeps things in the table and just adds as needed, it will also be the easiest to read format. The table will be in a nice format that will give any student or advisor peace of mind when they are given the results in a nice format. Also, because it runs the program and not as many functions it also has the best-case scenario when you run O(n). While all the sorting options will be placed into the program for the client and the user to choose. However, the user will be advised that the Hash Table will be the recommended option.