After analyzing the pseudocode I created for the Vector, HashTable, and Binary Search Tree assignments, I have deducted the worst case running time of each to be: 13n + 7 for Vectors, 12n + 8 for HashTables, and (3n! \* 6n) + 3n + 6 for Binary Search Trees, where n is the number of rows/courses in the csv file. According to my calculations, the most optimal choice for storing these courses in a specific container would be to choose a vector. I chose the vector data type because using it requires the least running time. Not only is it incredibly easy to store data inside of a vector, but there is also an easy solution to alphanumeric sorting when using vectors.

There are, however, upsides and downsides to each data type, including Vectors. To start off with vectors, as mentioned prior they are incredibly easy to use and can easily sort and take in information. The downside to vectors is that they only provide access to linear searching for their contents, requiring the user to iterate over each of a vector’s innards until finding what was necessary. This can slow down the program considerably, especially when searching through large loads of data.

The second data type that I studied was the HashTable. The HashTable is superior to the Vector data type in terms of sorting information and searching for information. Information is automatically sorted upon insertion, which narrows down the location of a target you search for. The *disadvantage* of HashTables is that sometimes information with identical key values can collide, causing the program to misbehave. This can be avoided by ensuring that all data that will represent key values is kept unique, but accidents can still occur.

The third data type that I studied were Tree Data Structures/Binary Search Trees. The advantage of tree data structures was similar to HashTables. Tree Data Structures allowed for very quick searching of information, allowing for incredibly fast searching even amongst very large quantities of data. Unlike HashTables where the quality of the sorting depends on the amount of buckets the HashTable has, the Tree Data Structure starts off with a “root,” which is the first value it receives. It then sorts values larger than it to the right and smaller than it to the left. This is incredibly advantageous, especially when creating trees from randomized values and then searching for one of them. Whereas a Vector or HashTable’s bucket would require them to iterate over each value in search of a target, in good situations a Tree Data Structure can find a target very quickly. The main disadvantage to this data type, however, is that if a tree is created from pre-sorted values, it becomes no different from a vector. In that case the root will become either the largest value and all other values will span to the left in descending order, or the root will be the smallest value and all others will span to the right in ascending order. This will nullify all advantages of the Tree Data Structure.

With all this in mind, I recommend using the Vector Data Structure, as it has the fastest running time and is the easiest to use, with minimal disadvantages.