**4. Evaluate run time**

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Vector** | **Hash Table** | **Binary Tree** |
| **Loading Courses** | O(n) | O(n) | O(n) |
| **Search** | O(n) | O(1) | O(log n)  *Worst case O(n)* |
| **Sort / Print** | O(n log n)  *After sorting print time would be O(n)* | O(log n)  *Worst case O(n)* | O(n) |

**5. Advantages and disadvantages of each structure**

* **Vector**
  + Unsorted vectors are fast to load using the push back method but will be unable to use binary search. The disadvantage is that you would need to do a linear search for the sorting and search functions which takes more time than the other 2 structures.
* **Hash Table**
  + Hash tables are efficient for inserting, searching, and printing with a O(1) time complexity. Hash tables use memory efficiently using dynamic resizing. Hash tables are not sorting and depending on collisions can be difficult to sort or search.
* **Binary Tree**
  + Binary trees are already mostly sorted, which makes traversal for searching, sorting or printing the same or quicker than a vector. It has the same speed as the hash table when it comes to sorting and print but depending on the balance of the tree can fall short to the hash table. Binary trees can also utilize more memory because they require pointers from parents to children.

**6. Recommendation**

Because the primary role of this program is to look up courses, I recommend using a hash table. Hash tables excel when requiring fast access to elements. Also, when using dynamic resizing, it can minimize performance issues due to collisions.