Dan Story  
Project One Run Time Analysis  
13APR2025

The three structures that we are examining are Vectors, Hash Tables, and Binary Search Trees. I will explore the weaknesses and strengths of each.  
  
**Vectors** A vector is an ordered list of a single data type. There is no fixed size, so a vector’s size can be variable which means it can be expanded or shrunk as needed. It’s very easy to remove things from the end of a Vector. You access the data you need via an index, with [0] being the lowest index and [n-1] being the highest index. It can be cumbersome to move data from the beginning or middle of a vector, as data has to “shift” into the now empty spot in the index. Since Data is stored and accessed in a sequential manner, that means the only way to search a vector is sequentially. The best case for a vector will be on the ends, as well as the worst case being on the extreme end. For example, best case for insertion will be if it’s inserted at the end. It would go right on the tail. If you have to insert data at the beginning of the vector, now you have to iterate through the entire list.   
Loading data : O(1)  
Search : O(n)

Sort (quick sort) : O (N log N)  
  
**Hash Tables** A hash table takes unsorted data, and we sort them using data into a specific index. The word Hash means to “chop and mix”, and the hash function does the same thing. It results in a fixed string and way to point to an index in the array reliably. This means it’s very easy to find data as long as you have the hash “key”. It’s also easier than a vector to insert and remove things, as you don’t have to move the entire list around in the array if it’s not the tail. The real problem is that not everything is easy to “hash”. Let’s take our courses for example. If we hash using the letters in the name of the course, like “CS-300”, what happens with “CS-230”? This causes a collision. We essentially make “buckets” of lists to hold these hash values. All the CS courses are grouped together. Suddenly we have the problems of a list within a list. Another problem is that hash tables have a fixed size. Hash tables really excel with unique data you need to search, access, and modify, but falls short for variable list sizes or if you have too many collisions.  
  
Loading data: O(1) – O(N) depending on collisions  
Search : O(1) – O(N) depending on collisions  
Sort : O(N) if the table is created in order  
  
**Binary Search Trees** A binary search tree organizes an array like a, well, tree. We start at the root. The next data added goes to the left if it’s smaller than the root and right if it’s larger than the root. This node becomes a “leaf”. If it gains leaves of it’s own, it becomes a “branch”. Each node has a parent child relationship similar to the root. If a child is larger than it’s parent it’s to the right of it, and if it’s smaller it’s to the left of it. The largest advantage of a Binary Search Tree is it’s extremely easy to search especially when the tree is “balanced”. This means that there are an equal number of left and right nodes for every parent and child. Once a tree becomes unbalanced, say a majority right or left nodes, the tree becomes something like a linked list. Binary Search Trees also have a pretty easy time inserting and removing data as there is a simple method of replacing the parent with the child node. Binary Search Trees are also fixed in size.   
  
Loading data : O(log N), but O(n) in a worst case  
Search : O(log N), but O(n) in a worst case  
Deletion : O(log N), but O(n) in a worst case  
  
 For this specific project, the number of courses are fixed. We also will have a large number of courses that will be grouped together, since they will all start with the same values (CS, MAT etc). For that reason, I’d recommend a Binary Search Tree. A Binary Search Tree comes already sorted, is easy to search for a specific key, and we don’t have to worry about the problems with collisions that are in a Hash Table. Since the university offers a wide array of courses, the tree should be relatively balanced as well, compared to the large number of collisions a Hash Table would have.