**Dynamic Array Program Analysis (Sorted List)**

*Description for Program Runtime:*

Program starts with data initialization and inserts all data read from the file into the data structure. The first and second inserts occur in the constant time. The third insert requires a shift of 2 elements over. The next n-3 times requires n-3 comparisons plus a constant shift of 2 elements to move over. Increase capacity method is also called 9 times total throughout the duration of the initialization which requires copies of the array be made 9 times. The copies made in total: 10, 20, 40, 80,160, 320,640, 1280,2560, respectively, constant cost. Then the program creates a new element then inserts after 9 comparisons and n-9 shifts of elements. Deleting this element costs 9 comparisons and again n-9 shifts of elements. Program sorts the list which is occurs n-times (bubble sort), the best case n-times, because the list is already sorted.

*Program Runtime:* T(n) = c0+ (n-3)\*(n-3)c1 + c2 + (n-9)c3 + (n-9)c4 + nc5

= c0+ (n2 –6n+9) c1 + c2 + (n-9)c3 + (n-9)c4 + nc5

= c0 + n2c1 -6nc1 +9c1 + c2 + nc3-9c3+nc4-9c4+nc5

*Asymptotic Tightbound:* O(n2) since n2 is the dominant term in the above function.

**Linked List Program Analysis (Singly Linked List)**

*Description for Program Runtime:*

Program starts with data initialization and inserts all data read from the file into the data structure. The inserts method itself occurs in constant time because it inserts at the beginning which is its best case but this occurs for each item in the data so this happens n-times. Next creation of a new element, which is constant, then it’s inserted at the end which is the worst case because it iterates through all the elements already in the linked list. Then it calls the sort method with a quick sort implementation that occurs nlogn times since this list is sorted in reverse, but it also uses a toArray method that runs in n-times. Next it calls the find method to find the new element, which runs in the worst case (n-times) because it’s the last element. Finally it deletes this new element, again running in the worst case.

Program Runtime : T(n) = nc1 + c2 + nc3 + nlognc4 + c5 + nc6 + nc7

Asymptotic Tightbound: O (nlogn) since nlogn is the dominant term in the above function

**BST Tree Program Analysis (AVL)**

*Description for Program Runtime:*

Program starts with data initialization and inserts all data read from the file into the data structure. The insert starts constant for the first 4 times. Then for all elements n-4 to n, because it has to account for balancing, insert has an logn cost. A sort occurs with a toArray method that has a cost 2n/2 times. A new element is created. It is inserted constant. Then a logn search occurs to find it then another logn search occurs to delete it.

Program Runtime : T(n) = c0 +(n-4)(logn)c­1+c2+(2n/2)c3+c4+lognc5+lognc6

Asymptotic Tightbound: O (nlogn) since nlogn is the dominant term in the above function

**Hashmap Analysis**

Program starts with data initialization and inserts all the data that into an AVLtree so this runtime would include the cost time for initialization from the AVL tree analysis. Then we create a new element to insert which is a constant cost. This insert has constant time since it is the only element inserted into the hashmap. It uses a get method to find the element in hashmap, calling hash then, as it is the only value in the map and at that key, it would run a constant runtime.

Program Runtime : T(n) = c0 +(n-4)(logn)c­1+c2+c3

Asymptotic Tightbound: O (nlogn) since nlogn is the dominant term in the above function

**Closed Hashtable with Double Hashing**

Program starts with data initialization and inserts all the data that it reads into an AVL tree so this runtime would include the cost time for the initialization from the AVL tree analysis. Then we create a new element to insert which is a constant cost. Then we insert another element with the same key to employ double-hashing, again another cost. Then we find these two elements in the hashmap. The first element inserted will use constant time and the second element would also use constant time but would need to employ double-hashing to index into the hashtable to find its appropriate key.

Program Runtime : T(n) = c0 +(n-4)(logn)c­1+c2+c3

Asymptotic Tightbound: O (nlogn) since nlogn is the dominant term in the above function

**Comparison Conclusion [tobeedited]**

For all three programs, using insert for the initialization of the data into the data structure is the most costly and most used operation out of all the others. For the sorted list, since it still requires (n-3)\*(n-3) shifting of elements, it is the most costly initialization. Both the AVL tree and the Singly Linked List programs however have the same asymptotic tightbound for their runtimes. The AVL tree is much more complicated and requires more method calls to remain its balancing so it would cost more than the Singly Linked List. Therefore, the singly linked list would be the more suited data structure for our data set.