Ian Sanborn - Complexity

1. Queues and stacks can be thought of as specializations of lists that restrict which elements can be accessed.

1a. What are the restrictions for a queue?

In a queue, you must add an item to the back of the queue and only remove items from the front. This means that the user only has access to the front item in the queue and can only view the front item's value.

1b. What are the restrictions for a stack?

A stack, similar to a queue, only allows access to the first, or in this case the top item in the stack. In a stack, you are only able to add and remove items from the top of the stack.

2. We have looked at lists backed by arrays and links in this class. Under what circumstances might we prefer to use a list backed by links rather than an array? (Your argument should include asymptotic complexity).

We would rather have a list backed by links when we add or remove items from the list, because when we insert or remove an item in an array list we would need to resize the list each time. Using an array list would use linear time. Using a linked list allows us to reach only constant time because we do not need to recreate the list every time. We would only need to create a need node and then reassign pointer values.

3. Give the asymptotic complexity for the following operations on an array backed list. Also provide a brief explanation for why the asymptotic complexity is correct.

3a. Appending a new value to the end of the list.

Linear time

This gives us linear time in respect to the length of the list, because we would need to create a new array and copy everything over and add in the new element.

3b. Removing a value from the middle of the list.

Linear time

Removing a value from the middle would give linear time complexity because it requires a resizing of the list and a shifting of elements down one index.

3c. Fetching a value by list index.

Constant time

Fetching a value at an index gives us constant time because you are able to read in an index and grab the value at that index.

4. Give the asymptotic complexity for the following operations on a doubly linked list. Also provide a brief explanation for why the asymptotic complexity is correct.

4a. Appending a new value to the end of the list.

Constant time

Adding a value to the end of a double linked list gives constant time because we are able to go to the tail value of the list and then reassign pointers and reassign the value for the tail.

4b. Removing the value last fetched from the list.

Constant time

Assuming that we are already at a the index in the list , since we already fetched the value, removing the value would take constant time because we only need to reassign the pointers. However, if we still need to go through the list then this function will take linear time.

4c. Fetching a value by list index.

Linear time

This function would give us linear time because we need to scan through each item in the list to get to a specific index to fetch it's value.

5. One of the operations we might like a data structure to support is an operation to check if the data structure already contains a particular value.

5a. Given an unsorted populated array list and a value, what is the time complexity to determine if the value is in the list? Please explain your answer.

This would give us constant time because all we need to do is fetch the values for each item in the array and compare it to the given value. We will not be mutating the list or allocating new memory, so it should only take constant time.

5b. Is the time complexity different for a linked list? Please explain your answer.

Yes, the time complexity for a linked list would be linear because instead of just going to a specific index in an array list, for the linked list we must go through each node and get its value.

5c. Given a populated binary search tree, what is the time complexity to determine if the value is in the tree? Please give upper and lower bound with an explanation of your answer.

In the best case, the lower bound would be constant time because if the value is the root then we know the value is in the tree. The worst case would be if we needed to check every single node in the tree. This causes the upper bound time complexity to be exponential, O(2^n). n is the depth of the tree and 2 is the amount of nodes per parent.

5d. If the binary search tree is guaranteed to be complete, does the upper bound change? Please explain your answer.

The upper bound would not change because we would still need to check every node in the tree. The depth of the tree is the only factor that changes the amount of time since n = the depth of the tree.

6. A dictionary uses arbitrary keys to retrieve values from the data structure. We might implement a dictionary using a list, but would have O(n) time complexity for retrieval. Since we expect retrieval to occur more frequently than insertion, a list seems like a poor choice. Could we get better performance implementing a dictionary using a binary search tree? Explain your answer.

We could get a better performance using a binary search tree depending on the value that is getting retrieved. However, if you end up needing to search through every element in the dictionary then you will get O(2^n).