Emma Blair
Day8 Theory Assignment
12/6/17

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

- What are the restrictions for a queue?
    - A queue is first in first out, so only the elements are only accessible in the order they are entered in. This means that only the first element can be accessed.
- What are the restrictions for a stack?
    - A stack is first in last out, so only the elements are only accessible in the opposite order they are entered in. Only the last element can be accessed.

2: Under what circumstances might we prefer to use a list backed by links rather than an array? (Use asymptotic complexity for argument).

It would be preferential to use a list backed by links rather than an array if a program removes and inserts frequently, meaning that the size is not finite. This is because when an array changes in size, it must be recreated each time, whereas for a link, the pointers for only the nodes connected to the node being removed or added are changed. This means that an array has an O(n) or linear asymptotic complexity and a link has an O(1) or constant asymptotic complexity.

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.

- Appending a new value to the end of the list.
    - The asymptotic complexity for adding a new value to the end of the list is O(n) or linear because the array must be recreated at a new size.
- Removing a value from the middle of the list.
    - The asymptotic complexity for removing a value from the middle of a list is O(n) or linear because the array must be recreated at a new size each time a new value is added to the array.
- Fetching a value by list index.
    - The asymptotic complexity for fetching a value from the middle of a list is O(1) or constant because it fetches a value at a given index and the size of the array is known.

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.

- Appending a new value to the end of the list.
    - The asymptotic complexity for adding a new value to the end of the list is O(1) or constant when linked lists have a tail pointer, so it is a constant time to set the current node to the tail node and then add a value.
- Removing the value last fetched from the list
    - The asymptotic complexity for adding a new value to the end of the list is O(1) or constant because the current node is set at the node where the value was just fetched, so the location of the node is already known.xs
- Fetching a value by list index

- The asymptotic complexity for fetching a value by list index is O(n), or linear, because you must go through the list until the index is found, which depends on the number of nodes in the list.

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 specific value.

- Given an unsorted populated array list and a value, what is the time complexity to determine if the value is in the list? Explain.
  - The time complexity to determine if the value in the list is O(1), or constant, because it is retrieving values, which is not assigning or moving values, which would cause the list to mutate and then have a linear time complexity.
- Is the time complexity different for a linked list? Explain.
  - Yes, the time complexity is different because you must iterate through each node separately and retrieve the value, which means that the time complexity is O(n), or linear.
- Given a populated binary search tree, what is the time complexity to determine if the value is in the tree? Explain with upper and lower bound.
  - Upper bound: O(n) or linear, where n equals the depth of the tree. If the tree is in order, you would have to go all the way through each child and parent node of the tree in a way similar to a singly linked list, which is equal to the number of node. The depth of the tree is described by or 2^n - 1.
  - Lower bound: $\Omega(1)$ or constant, because if value is in the root, then it is constant time to check the value in the root, which could be done with a get root method, since it is the first node and its position does not change.
- If the binary search tree is guaranteed to be complete, does the upper bound change? Explain.
  - If a binary tree is guaranteed to be complete, the upper bound changes to O(log n), because if the tree is balanced, as it is in a complete tree, then the time to find a value can be halved every time it branches from child to parent.

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.

Implementing a dictionary using a binary search tree would give a better performance than a list. For a binary tree, the time for searching can be repeatedly halved because of the subtrees, each with one or two children. The length of path between nodes is described by k - 1, where k represents the number of nodes. The time depends on the depth of the node within the tree, meaning how far away the node is away from the root of the node containing the value. For a list, the time depends on how long the list is and how far down the list the node with the value is being stored since the time complexity is linear. For a binary search tree, the time complexity is O(log n) for retrieval.