1a. A queue is restricted in the manner that you can access data. When using a Queue one can only access information that was added first and therefore cannot access information that was recently added to the Queue which is a major restriction. Otherwise known as first in first out (fifo).

1b. The restrictions for a stack are that data can only be accessed that was recently put into the stack and is therefore at the top of the stack. Data that is at the bottom of the stack is inaccessible which is a major restriction. Otherwise known as first in last out (filo).

2. A double linked list would have faster asymptotic complexity, of around linear complexity, if the primary operations being performed on that list were insertions or appends because with a linked list one only has to reach the correct index in the list, create a new node, and then redirect the adjacent pointers, as well as the newly created node, to the correct other nodes. Inserting and appending in an array backed list yields greater linear asymptotic complexity as it requires the generation of slightly longer list and the copying of all of the elements from the old list into the new list plus the new index.

3a. This would yield Linear time complexity with respect to the length of the list because to add a index one has to create a new list that is one size smaller and copy through everything from the old into the new with the addition of the new index.

3b.This would yield Linear time complexity with respect to the length of the list because to add a index one has to create a new list that is one size larger and copy through everything from the old into the new without the removed index.

3c. Constant time is required for fetching a value from an index in a list because the list is already stored in memory and all the fetch does is go to the list and go down it until it finds the correct index.

4a. This operation requires constant asymptotic complexity because all the double linked list does is go to the tail pointer, create a new node and rearrange the pointers so the new node is the tail. This does not require going through the whole list.

4b.this would take constant asymptotic complexity because fetching would bring the current value to that index and then one only has to remove at the current index which is linear.

4c. This would require linear time because we would have to start at the head of the list and go through each node checking for the correct value until we do arrive at the correct value.

5a.this would require linear time because the list is unsorted which requires us to go through the whole list from the beginning and check each index to see if it contains the correct value.

5b. The complexity would be the same because we still have to go through the whole list checking for the value starting from the beginning because we don't have any notion of where the desired value is.

5c. The upper bound would be linear time complexity in the case that there is only children running down the left branches or just the right branches in which case to find a value one would have to go through the whole line of keys. The lower bound would be log time in the case that the tree is complete and full in which each branch decision done when trying to the value halves the amount of options still present which cuts the time in half when going down a branch which resolves to log time.

5d.The upper bound is guaranteed to change because the amount of time required to find a value in a complete tree is halved after each branch choice going down the tree.

6.We would get better retrieval performance from dictionary backed by a binary search tree because the lower bound for a list is linear and the upper bound for bst is linear meaning that in all cases a bst would be better for retrieval especially considering that the lower bound for a bst is log(n).