Recap

List

1] A list has insert(push) at front and back at O(1), remove(pop) at O(1), get at O(1) in front or back. Get at certain element is O(n), insert middle is O(n), and search at O(n).

2] The data is stored as a series of nodes that does not have to be adjacent in memory.

3] The size of the list does not have to be predetermined and can be grown without having to reserve space prior.

4] New data gets added in front or back by creating a new node and setting the pointers to point to the first or last previous node on one end and to the head or tail on the other end. To add in middle, make new node point to prev node and point to next node and next node and prev node point to new node.

Vector

1] A vector has insert at O(n), delete at O(n), random access(get/at) at O(1), capacity, and reserve.

2] The data is stored in an array with memory previously reserved.

3] Allows for access to any internal element directly(O(1)) to save time instead of having O(n). Also allow for growth.

4] New data gets added by shifting everything down one spot after the insertion point.

Stack

1] A stack has push at O(1), pop at O(1), top at O(1), and size.

2] The data can be stored as a series of nodes in a linked list or through arrays in a vector.

3] To be able to use the data in the same order without having to make a copy of it.

4] New data get added by performing a push by inserting at the front/top of the list.

Queue

1] A queue has push at O(1), pop at O(1), front at O(1), and size.

2] The data can be stored as a series of nodes in a list.

3] To be able to keep track of the elements currently in queue. It also keeps the data in order since the first data in will be expected to be the first one out/accessible.

4] New data gets added by performing a push on one end of the list and pushing the rest down.

BST

1] A tree has insert at O(log n), delete at O(log n), and contains at O(log n).

2] The data is stored as a collection of nodes linked to each child of a node. Each node has its own data.

3] A tree is highly effective in searches and updates to the data. Even as more data is added, the amount of time to do those functions barely increases.

4] Make a recursive call and proceed down the tree. If data is found, do nothing, if not, add data to the last spot on the traversed path.

HashMap

1] A hashmap has set at O(1), get at O(1), remove at O(1), and contains at O(1).

2] The data is stored in a set location in an array.

3] Hash allows for quick finds. It also allows for finding keys that are similar to each other.

4] Have multiple values for each key and add to a list.

Map

1] A map has set at O(log n), get at O(log n), begin at O(1), end at O(1), size, and empty.

2] The data is stored as a collection of ordered entries that has keys and the values associated with it. The keys have to be unique whereas the values does not.

3] Allows for sorting besides using the actual data.

4] To add data, a pair of key and value type object must be provided. Key is the one that is added to the map while value is that data associated with the key.

AVL Bonus

A] Because an AVL can at most only have an one level difference.

B] Because adding a node could violate the AVL balance condition. By having it on the path that the add was performed on, a rotation can be done to modify the tree balancing before the insert step is complete.

C] Because using recursion will take up a huge amount of time. A tree with multiple branches and leaves will eat up a lot of time and memory.