1. What is an Abstract Data Type (ADT)?

An abstract data type is the set of data and the functions that operate on it. These are independent of any particular implementation.

2. Explain the difference between a queue (FIFO) and a stack (LIFO)

The difference is that the way the values are added and removed from their respective data structures is different.

FIFO = First In, First Out.
As you add values to the queue, the new values are put at the end of the queue. And when you remove values from the queue, the values are taken from the front.

Index: [0, 1, 2]
Values: [oldVal, oldVal, oldVal]

enqueue()
Index: [0, 1, 2, 3]
Values: [oldVal, oldVal, oldVal, newVal]
- Added newVal at the end

dequeue()
Index: [0, 1, 2]
Values: [oldVal, oldVal, newVal]
- Removed oldVal at the start

LIFO = Last In, First Out
As you add values to the stack, the new values are put on the top of the stack. As you remove them from the stack, they are taken off the top.

Index: [0, 1, 2]
Values: [oldVal, oldVal, oldVal]

push()
Index: [0, 1, 2, 3]
Values: [newVal, oldVal, oldVal, oldVal]
- Added newVal at the start

pop()
Index: [0, 1, 2]
Values: [oldVal, oldVal, oldVal]
- Removed newVal at the start

3. Name and briefly explain three types of data structures.

Three types of data structures are:

Array: An array is a data structure that consists of a collection of elements. Each element is identified by an array index.

Linked List: A linked list is a data structure consisting of a group of nodes. For a linked list, we have access to the head node and the tail node. Each node inside the linked list has a pointer that points to the next node, but there is no node that points to the previous node. Each node also has data.

Heap: A heap is a specialized tree-based data structure that satisfies the heap property. There are two different classifications of heaps. There is max heaps and min heaps. Max heaps are where the parent nodes are always greater than or equal to those of the children, and the highest key is the root node. For min heaps, the parent nodes are less than or equal to those of the children, and the lowest key is the root node.

4. Explain what a binary tree is, what are some common operations of a binary tree?

A binary tree is a data structure where there are nodes. A node being a parent, can have 2 child nodes. The nodes under the parent on the left side is the left branch and the nodes under the parent on the right side is the right branch. The left branch is where the values are smaller than the parent node, and the right branch is where the values are bigger than the parent node.

Some common operations of a binary tree are:

Search: Search would be where it searches through the binary tree for a specific value or a specific node. It uses the fact that the smaller values are on the left, and the bigger/same values are on the right to traverse through the binary tree. If the tree reaches the point where there are no more child nodes, and the value is not found, then the value is not in the tree.

Delete: Delete will traverse the binary tree and delete the node(s) that match the node that is being deleted. When the node is deleted, the binary tree must be rebalanced and fixed.

Insert: Insert will insert a node in the place where it should be based on if it is smaller or larger than a node. If the node is smaller, it goes to the left branch, and if it is larger or equal, it will go to the right branch. If there is no child where it is checking, it is inserted in that position as the child.

Display: This will print out the binary tree.

5. Explain what a hash table (dictionary) is, what are common operations of a hash table?

A hash table is a data structure that is used to implement an associative array. It is a structure that can map keys to values. A hash table uses a hash function to compute an index into an array of buckets or slots, from which the desired value can be found.

Common operations of a hash table are:

Search: The search operation will search through the hash table using the hash keys. From there, it can traverse the chained link lists (if it uses it) or check if it's at that hash key. If it reaches the end of the chained linked list or list, then the value is not in the hash table.

Insert: The insert operation will find the correct hash key to input the data into, and will put it will insert it into the end of the chained linked list, or the end of the list.

Delete: The delete operation will move the hash key where the value will be at, and will search through the chained linked list or regular list and will delete the specific value. If the value is in the middle of the linked list, it will fixed the chaining.