* What is the distinction between a list and an array?

Array stores homogenous data values, but list can store different types of data values. For example, you can see string, integer, float even list inside list. But you can see only one of these data types inside array. Array has an advantage in performing mathematical operations, lists are less efficient from this side. Besides that, the list is built-in data type, but array needs to be imported from NumPy library.

* What are the qualities of a binary tree?

- The number of nodes n in a full binary tree is 2**(h+1) – 1. Since there are h levels, we need to add all nodes at each level.

- The number of nodes n in a complete binary tree is between 2**h (minimum) and 2**(h+1) – 1 (maximum).

- The number of leaf nodes in a full binary tree is 2**h.

- The number of NULL links (wasted pointers) in a complete binary tree of n nodes is n + 1

* What is the best way to combine two balanced binary search trees?

Merge, Sort, Reconstruct

In this set of rules, the principal concept is to transport the factors of each BSTs to an array, kind and shape the merged BST.

This set of rules has the subsequent three number one steps:

step 1: Process the BSTs and show the factors of their respective arrays.

step 2: Combine the arrays in a merged array and find the factors in them. We can use any of the famous sorting strategies which includes bubble kind, insertion kind, etc.

step 3: Copy the factors of the merged array to create merged BST with the aid of using in-order traversing.

Since it will increase with the dimensions of the dataset, the complicated it of step 1 is O(n1 + n2). The n1 and n2 are the dimensions of the BSTs which might be merged.

The time complexity of step 2 relies upon the form of the sorting set of rules used, it'll be O(n^2) if it`s bubble kind and O (n log n) if it's insertion kind.

We want to show all factors withinside the array, so the gap complexity of the step is O(n).

* How would you describe Heap in detail?

A data structure that satisfies the heap property is referred to as a heap. The heap property states that a parent node and its child nodes must have some sort of relationship. This property must be applied to the whole heap. A heap is a tree with certain unique characteristics. A heap's most basic criterion is that

a node's value must be greater than (or smaller than) the values of its children. This is referred to as heap property. Heap should produce a complete binary tree.

In a min heap, the parent-child connection is characterized by the parent's constant need to protect his or her children.

Its children must be smaller than or equal to it. As a result, the lowest element in the heap becomes the lowest element in the heap. The root node must exist.

In a max heap, on the other hand, the parent is more than or equal to its kid or its child's children. As a result, the root node is made up of the greatest value.

* In terms of data structure, what is a HashMap?

The hash map is very correlated with dictionaries. If we want to understand hash map in simple terms let's look at it in example of dictionaries. A hash map is an array optimized to store key value pairs. It is so powerful because you can retrieve, delete, insert data in constant O (1) time. As we know from dictionaries, we can access data from its key. Keys are strings and we access data through the strings which are keys in our case. It seems like we are able to use strings as if indices. But how is it possible? It is because of the hash function. The hash function takes the key and returns the hash code. If we take the modulus of the hash code to the length of array, we can use that as an index to store its value. But, sometimes the hash of two keys takes same result from modulo operation. This is called collision and to deal with it we can use linked lists. Another example of HashMap is the object in Java.

* How do you explain the complexities of time and space?

Time and space complexities have a tradeoff between each other. Time is more important for us and it is a priority since we can buy hardware and increase memory but not time. Time complexity basically is how our program will behave in very large amount of input and which analogy it will use when running. For example, we can have 2 algorithms, one running in O (N) time and the other in O(2*N) time. Both are in the group of O (N) time complexity. The only thing that matters for us is input size N not coefficient. Let's look at some time complexities and algorithms running in these time complexities.

Big O – Upper bound (as input is very large)

O (1) - constant time (as n input becomes very large running time does not change) - HashMap

O (log(N)) - Binary Search Tree (Halving)

O(N) - Linear Search (linear time)

O(N*log(N)) -MergeSort & QuickSort

O(N**2) - BubbleSort

O(2**N) - recursion, backtracking

O(n!) - Factorial

Space Complexity

Space complexity is as same as time complexity, but it does not take time but space. It can take no space just basically run in a place which is O (1) or it can take as much space as depending on the size of input. It can allocate exponential amount of memory and in some cases can take O(log(N)) space.

* How do you recursively sort a stack

To do that, we can pop all elements from stack and store popped element in some variable. After we have an empty stack, we can call “inserted_in_sorted_order()” function recursively and it will insert the first coming number at bottom of stack. Then it will add the next numbers comparing the new number to already existing numbers and after all numbers go through this process, we will get a sorted stack