RECURRENCE RELATION

1) What is the distinction between a list and an array?

List	Array
Can consist of elements belonging to different data types.	Only consists of elements belonging to the same data type.
No need to explicitly import a module to use it.	Need to explicitly import a module for declaration.
Cannot directly handle arithmetic operations.	Can directly handle arithmetic operations.
Can be nested to contain different type of elements.	Must contain either all nested elements of same size.

2) What are the qualities of binary tree?

1. Binary tree does not allow duplicate values.
2. While constructing a binary, if an element is less than the value of its parent node, it is placed on the left side of it otherwise right side.
3. Binary tress is the one in which each node has maximum of two child- node.

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

At first, it looks like that inserting all the elements from tree2 to tree1 one by one is the best solution. But this solution has a time complexity of **O(n log(n))**, there is a solution that can do better than this.

We now know how to convert a sorted array to a balanced binary tree, we are going to reduce the given problem to that problem.

The idea is to store the inorder traversal of both the trees in two arrays, say, arr1 and arr2 respectively. Now we create one more array that contains all the elements of arr1 and arr2 merged together in sorted form. Two sorted arrays can be merged to form a new sorted array in linear time complexity. We then convert the sorted array to a balanced binary search tree, hence the two trees are merged.

The steps are

1. Store the in-order traversal of both the trees in two arrays, say, arr1 and arr2 respectively.

2. Merge arr1 and arr2 to form another array arr, that contains the elements of arr1 and arr2 in sorted form.

3. We now use this sorted array(arr) to build a balanced binary search tree, which is a merged version of the given trees.

4. The middle element of the array forms the root of the balanced BST and all the elements to the left of the middle element form the left sub-tree and all the elements to the right of the middle element form the right sub-tree.

5. Recursively do step 4 for the left subtree and attach it to the left of root.

6. Recursively do step 4 for the right subtree and attach it to the right of root.

7. Return root.

4) How would you describe Heap in detail?

A heap is a complete binary tree, and the binary tree is a tree in which the node can have utmost two children. Before knowing more about the heap data structure , we should know about the complete binary tree.

A complete binary tree is a binary tree in which all the levels except the last level, i.e., leaf node should be completely filled, and all the nodes should be left-justified.

There are two types of heap:

a) Min heap –
In other words, the min-heap can be defined as, for every node i, the value of node i is greater than or equal to its parent value except the root node.

b) Max heap –
In other words, the max heap can be defined as for every node i; the value of node i is less than or equal to its parent value except the root node.

The total number of comparisons required in the max heap is according to the height of the tree. The height of the complete binary tree is always logn; therefore, the time complexity would also be O(logn).

5) In terms of data structure, what is a HashMap?

In computer science, a Hash table or a Hashmap is a type of data structure that maps keys to its value pairs (implement abstract array data types). It basically makes use of a function that computes an index value that in turn holds the elements to be searched, inserted, removed, etc. This makes it easy and fast to access data. In general, hash tables store key-value pairs and the key is generated using a hash function.

Hash tables or has maps in Python are implemented through the built-in dictionary data type. The keys of a dictionary in Python are generated by a hashing function. The elements of a dictionary are not ordered and they can be changed.

An example of a dictionary can be a mapping of employee names and their employee IDs or the names of students along with their student IDs.

```
my_dict={'Dave' : '001' , 'Ava': '002' , 'Joe': '003'}
print(my_dict)
type(my_dict)
```

6) How do you explain the complexities of time and space?

Space Complexity:

Space complexity of an algorithm represents the amount of memory space needed the algorithm in its life cycle.

Space needed by an algorithm is equal to the sum of the following two components

A fixed part that is a space required to store certain data and variables (i.e. simple variables and constants, program size etc.), that are not dependent of the size of the problem.

A variable part is a space required by variables, whose size is totally dependent on the size of the problem. For example, recursion stack space, dynamic memory allocation etc.

Space complexity S(p) of any algorithm p is S(p) = A + Sp(I) Where A is treated as the fixed part and S(I) is treated as the variable part of the algorithm which depends on instance characteristic I.

Time Complexity:

Time Complexity of an algorithm is the representation of the amount of time required by the algorithm to execute to completion. Time requirements can be denoted or defined as a numerical function t(N), where t(N) can be measured as the number of steps, provided each step takes constant time.

For example, in case of addition of two n-bit integers, N steps are taken. Consequently, the total computational time is t(N) = c*n, where c is the time consumed for addition of two bits. Here, we observe that t(N) grows linearly as input size increases.

While solving some questions I have seen that while decreasing time complexity space complexity increases and vice versa just like bias and variance trade-off

7) How do you recursively sort a stack?

The idea of the solution is to hold all values in Function Call Stack until the stack becomes empty. When the stack becomes empty, insert all held items one by one in sorted order. Here sorted order is important.