Data structures 😂👌

1. **What is AVL trees**

   AVL tree is a self-balancing binary search tree. It was named after its inventor, Adelson-Velsky and Landis. It is designed to keep the height of tree always as small as possible, which makes searching, inserting, and deleting elements from the tree very efficient. The height of an AVL tree is always log(n) where n is the number of nodes in the tree. When a node is added or removed, the AVL tree automatically undergoes rotations to maintain its balance factor. The balance factor is the difference between the height of the left subtree and right subtree of any given node, and it must be between -1, 0 or 1, for the tree to be considered balanced.

2. **When do we say a tree is height balanced**
   A tree is height balanced if the difference in height between the left and right subtrees of any node in the tree is not more than one. In other words, the height difference between the left and right subtrees of any node in the tree is at most one.

3. **Write an algorithm for construction of AVL trees**
   Here is an algorithm for constructing an AVL tree:

1. Start with an empty AVL tree.

2. For each element to be added to the tree, perform the following steps:

a. Insert the element as in a binary search tree (BST).

b. Traverse from the inserted node up to the root of the tree, checking for balance.

c. If any node is found to be unbalanced, perform a rotation to balance the tree.

3. Return the AVL tree.

Here are the steps for checking balance and performing a rotation:

1. Calculate the balance factor of the node: difference in height between left and right subtrees.

2. If the balance factor is greater than 1 or less than -1, the node is unbalanced.

3. Determine the type of rotation needed based on the balance factor and the relative heights of the subtrees.

4. Perform the necessary rotation(s) to balance the tree.

Here are the types of rotations that can be performed:

Data structures 😂👌

1. Left rotation: when the balance factor is greater than 1 and the left subtree is taller than the right subtree.

2. Right rotation: when the balance factor is less than -1 and the right subtree is taller than the left subtree.

3. Left-right rotation: when the balance factor is greater than 1 and the right subtree is taller than the left subtree.

4. Right-left rotation: when the balance factor is less than -1 and the left subtree is taller than the right subtree.

To perform a left rotation, perform the following steps:

1. Set the left child of the unbalanced node as the new root.

2. Set the right child of the new root as the left child of the old root.

3. Set the old root as the left child of the new root.

4. Update the heights of the nodes.

To perform a right rotation, perform the following steps:

1. Set the right child of the unbalanced node as the new root.

2. Set the left child of the new root as the right child of the old root.

3. Set the old root as the right child of the new root.

4. Update the heights of the nodes.

To perform a left-right rotation, perform the following steps:

1. Perform a left rotation on the left child of the unbalanced node.

2. Perform a right rotation on the unbalanced node itself.

To perform a right-left rotation, perform the following steps:

1. Perform a right rotation on the right child of the unbalanced node.

2. Perform a left rotation on the unbalanced node itself.

4. **What are the rules for deciding type of notation used in AVL trees**
   There are generally two types of notations used in AVL trees - graphical and bracket notation. The decision on which notation to use depends on the preference of the developer or the software being used to represent the tree. Some developers prefer graphical representations that allow them to visualize the tree structure more easily, while others prefer bracket notation that can be easily parsed by algorithms.

   In graphical notation, an AVL tree is typically illustrated as a series of nodes with lines connecting them to form a hierarchical structure. Each node represents a value and has one or two child nodes depending on whether it is a leaf or an internal node.

   In bracket notation, an AVL tree is represented as a nested set of parentheses and values. The parentheses represent the tree structure, with each pair of parentheses enclosing a subtree, and the values are the nodes themselves. For example, a small AVL tree might be represented in bracket notation as "(3 (2 () ()) (5 () ()))".

5. **What is sorting techniques**
   Sorting techniques refer to various algorithms and methods used to arrange a collection of items in a specific order - such as ascending or descending order. These techniques are essential in computer science and programming for organizing and managing data. Some of the commonly used sorting techniques are selection sort, bubble sort, insertion sort, merge sort, quick sort, and heap sort. The choice of sorting technique depends on the size of the data, its complexity, and the implementation requirements.

6. **What are some of the factors considered in choosing sorting techniques**

   Some of the factors considered in choosing sorting techniques include:

   1. Time complexity: Sorting algorithms should be efficient and able to handle large amounts of data in a reasonable amount of time.

   2. Space complexity: Sorting algorithms should use as little memory as possible, especially in situations where memory is limited.

   3. Stability: Stable sorting algorithms preserve the relative order of equal elements in the sorted array.

4. Adaptivity: Sorting algorithms that can adapt to the input data and optimize their performance are often preferred.

5. Data distribution: Sorting algorithms may perform differently depending on the distribution of the data.

6. Nature of data: Some sorting algorithms perform better on certain types of data (e.g. already partially sorted data).

7. Ease of implementation: Simpler sorting algorithms may be preferred if the code needs to be optimized for readability and maintainability.

7. Describe different sorting techniques.

There are various types of sorting techniques available in computer science to arrange data in a particular order. Here are some of the commonly used sorting techniques:

a) Bubble Sort: It is one of the simplest sorting algorithms where adjacent elements are compared and swapped if they are in the wrong order.

b) Insertion Sort: It works by picking one element at a time and inserting it into the correct position in the sorted list.

c) Selection Sort: It is an in-place and comparison-based sorting algorithm in which the smallest or largest element from the unsorted array is placed at its correct position in the sorted array.

d) Merge Sort: It is a divide-and-conquer algorithm that divides the unsorted list into sub-lists, sorts them independently, and then merges them to form the final sorted list.

e) Quick Sort: It is also a divide-and-conquer algorithm that works by selecting a pivot element from the array and partitioning the other elements into two sub-arrays based on whether they are less than or greater than the pivot.

f) Heap Sort: It is a comparison-based sorting algorithm that works by building a heap data structure from the array and repeatedly removing the root element to obtain the sorted list.

g) Radix Sort: It is a non-comparative sorting algorithm that sorts the elements by grouping them into buckets based on their digits or letter values.