**Algo Expert**:

Dividing 2 big numbers.   
There are many libraries in Java. Try to think of multiple solutions to the same problem.

Main focus is to solve problems.

int index = Arrays.binarySearch(sortedArray, key);

// works for arrays which can be of primitive data type also.

int index = Collections.binarySearch(sortedList, key);

// works for Collections like ArrayList and LinkedList.

If not found then index will be negative.

Arrays.copyOfRange(array, j, array.length+1)

If a subarray is being returned, be careful with its indices and the original array’s indices.

Subarray’s 0th index could be kth index of bigger array.

**int** answer[][] = **new** **int**[m][n] ;

**for** (**int**[] row : answer)

Arrays.*fill*(row, -1);

**int** x[] = **new** **int**[2];

Arrays.*fill*(x, -1);

Way to initialize a 1d and 2d array.

To clear a list, use the clear() method.

Printing a list displays the value while printing an array prints a hash code.

Make a diagram for thinking.

2 number sum, 3 number sum

Can solve these using a Hash Table.

Sorting and then O(n^2).

Better to use iterators rather than for loops.

In place algorithm: algorithm which transforms input using no auxiliary data structure.

int[] array

arr.length

List<Integer> array

array.size()

To convert an array to a List, we use Arrays.asList().

Move Element to End:

We can do this in O(n) time instead of O(nlogn) time.

For O(n) time, we use two pointers, one for the starting index and another for last index.

We use a pointer to the last index because we want to move the required number towards the end of the array.

**BST Construction**:

Insertion:

Average: O(log(n)) time, O(log(n)) space

Worst: O(n) time, O(n) space.

Search:

Average: O(log(n)) time, O(log(n)) space

Worst: O(n) time, O(n) space.

Deletion:

Average: O(log(n) time, O(log(n)) space

Worst: O(n) time, O(n) space

**IMP**: If we want to make a function that returns something but not able to think about the return logic at some places, then what we can do is create another function which is called inside the current function and this function might not return anything.

Can also implement the above 3 in O(1) space.

In a binary search, what if we make the root to be any random node and start the operations from there?

In deletion, if we can’t find the element then we don’t do anything.

For deleting node with 2 children, grab the left most node in the right subtree of that node or grab the right most node in the left subtree.

Replace the left most node value with the node to be deleted. And then call delete on this leftmost node (deletion for with 0 or 1 children).

If we run the algorithms recursively, then space complexity will be O(log(n)) in average and O(n) in worst case.

Recursion uses frames on the call stack.

See the deletion code using both recursion and iteration.

**Validate BST**:

Base case would be on leaf nodes. Leaf nodes are always binary search trees.

Can use divide and conquer.

Validate all the possible subtrees.

Create a helper method.

If we want something to be true and want to continue in our code when that condition is true then create an if when that condition is false. And then write the logic for continuation.

**BST traversal**:

In order: Left, root, right

Pre order: Root, left, right

Post Order: Left, Right, Root

Using Recursion:

O(n) time, O(n) space.

Be careful with returns in Recursion.

Arrays.copyOfRange(x, 1, arr.length);

// returns an array from [ 1, arr.length-1 ]

**Maximum Subset Sum No Adjacent**: