**Algo Expert Sum:**

**4 Number Sum**:

Function that takes in a non-empty array of distinct integers and an integer representing a target sum.

Find all quadruplets in the array that sum up to the target sum and return a 2d array of all these quadruplets in no particular order.

If no 4 numbers sum up to the target sum, the function should return an empty array.

To create 2D array with variable number of rows and columns, we can use List<Integer[ ]> or List<List<Integer>>

Naïve solution: 4 for loops.

O(n^4)

A quadruplet can be expressed as a pair of numbers. We can turn a quadruplet into a pair of numbers.

Can reduce to 2 number sum.

x, y, z, k

P: x, y

Q: z, k

In the hash table, we can store the number P and can map it to (x, y).

x and y are the numbers that generate P.

The number P can be obtained by summing up a bunch of pairs of numbers.

{ 6: [ [4, 2], [7, -1] ], }

We should store an array of pair of numbers for a particular number P.

When we are going to be generating our pairs of numbers and then our quadruplets. We have to prevent double counting the quadruplets or pairs.

Double counting can happen like this:

13: [7, 6], 3: [4, -1]

And 10: [6, 4], 6: [7, -1]

In 2 number sum solution, we iterated through the array in one single pass.

Here, we can have 1 for-loop and then 2 inner for loops. These will be used strategically to not double count quadruplets.

We first iterate one by one through all the numbers in the array.

The first inner for loop will simply iterate through all the numbers that come after our current number.

Second for loop will iterate through numbers before the current number.

In the first for loop,

For these numbers we will generate the sum of this number and our current number.

Suppose the current number is 4. In the 1st for loop, we go through the numbers {-1, 1, 2}

P = 4 + -1

4 + 1

4 + 2

We check whether (targetSum – P) is in the hash table or not.

If it is, then we have found the quadruplet but if it is not, then we will not add that sum to the hash table yet.

Using the 2nd for loop, we iterate through the numbers that comes before our current number.

The numbers that come before 4 are {7, 6}

Here, we try to generate Q

Q = 4 + 6

4 + 7

These sums are the ones that we store in our hash table and the ones that we eventually use. That will allow us to not duplicate our quadruplets.

**IMP NOTE**:

In the collections we always use object types and not primitive types.

If we have List<Integer[ ]>

Arrays are of fixed size. So, to iterate over arrays we should use for-each loop.

Elements have to added in the array at the time of creation.

Adding elements into a list can be done anytime.

**SubArray Sort**:

Return an array of starting and ending indices of the smallest subarray in the input array that needs to be sorted in place in order for the entire array to be sorted.

How to determine which array to be sorted?

In a case, left part and right part of the array could be sorted. In the middle, we could have unsorted numbers.

If we get number in array which is not in sorted order, we will have 2 numbers which will not be in sorted order.

IMPORTANT operation:

Create a subarray from an existing array.   
Arrays.copyOfRange()