**Count possible triangles**

Show Topic Tags   

[Amazon](http://practice.geeksforgeeks.org/company/Amazon/)

Given an unsorted array of positive integers. Find the number of triangles that can be formed with three different array elements as lengths of three sides of triangles.

**Input:**  
The first line of the input contains T denoting the number of testcases. First line of test case is the length of array N and second line of test case are its elements.  
  
**Output:**  
Number of possible triangles are displayed to the user.  
  
**Constraints:**

1 <=T<= 100  
3 <=N<= 100  
1 <=arr[i]<= 1000

**Example:**

**Input:**  
2  
3  
3 5 4  
5  
6 4 9 7 8

**Output:**  
1  
10

\*\*For More Examples Use Expected Output\*\*

<http://practice.geeksforgeeks.org/problems/count-possible-triangles/0>

/\*

\* To change this template, choose Tools | Templates

\* and open the template in the editor.

\*/

package javaapplication241;

import java.io.BufferedReader;

import java.io.IOException;

import java.io.InputStreamReader;

import java.util.ArrayList;

import java.util.Arrays;

import java.util.HashSet;

import java.util.Iterator;

/\*\*

\*

\* @author Administrador

\*/

public class JavaApplication241 {

/\*\*

\* @param args the command line arguments

\*/

// Function to count all possible triangles with arr[]

// elements

static int findNumberOfTriangles(int arr[], int n)

{

// Sort the array elements in non-decreasing order

// qsort(arr, n, sizeof( arr[0] ), comp);

Arrays.sort(arr);

// Initialize count of triangles

int count = 0;

/\*

Corrige el primer elemento. Necesitamos correr hasta n-3

             como los otros dos elementos se seleccionan de

             arr [i + 1 ... n-1]

\*/

for (int i = 0; i < n-2; ++i)

{

/\*

Inicializar el índice del tercero más a la derecha

                 elemento\*/

int k = i+2;

// Fix the second element

for (int j = i+1; j < n; ++j)

{

/\*

Encuentra el elemento más a la derecha que es

                     menor que la suma de dos elementos fijos

                     Lo importante a destacar aquí es que

                     utiliza el valor anterior de k. Si el valor de

                     arr [i] + arr [j-1] fue mayor que arr [k],

                     entonces arr [i] + arr [j] debe ser mayor que k,

                     porque la matriz está ordenada.

\*/

while (k < n && arr[i] + arr[j] > arr[k])

++k;

/\*

Número total de triángulos posibles que pueden

                     se forma con los dos elementos fijos es

                     k - j - 1. Los dos elementos fijos son arr [i]

                     y arr [j]. Todos los elementos entre arr [j + 1] / a

                     arr [k-1] puede formar un triángulo con arr [i] y arr [j].

                     Uno se resta de k porque k se incrementa

                     un extra en el bucle anterior.

                     k será siempre mayor que j. Si j se convierte en igual

                     a k, entonces el bucle superior incrementará k, porque arr [k]

                     + arr [i] es siempre mayor que arr [k]

\*/

count += k - j - 1;

}

}

return count;

}

public static void main(String[] args) throws IOException {

// TODO code application logic here

BufferedReader br = new BufferedReader(new InputStreamReader(System.in));

int t = Integer.parseInt(br.readLine());

while(t-- > 0) {

int n = Integer.parseInt(br.readLine());

String[] input = br.readLine().trim().split(" ");

int[] arr = new int[n];

for(int i =0; i<n; i++) {

arr[i] = Integer.parseInt(input[i]);

}

System.out.println( findNumberOfTriangles(arr, arr.length));

}

}

}