NUMBER OF POSSIBLE TRIANGLES FROM GIVEN ARRAY OF NUMBERS

Given an array of numbers, write an algorithm to count how many triangles are possible with three numbers from array as their side.

If input array is: {7, 3, 6, 4}

Then the output should be 3 because there are 3 possible triangles, viz.: {3, 4, 6}, {3, 6, 7} and {4, 6, 7}.

In a triangle, the sum of two sides is always greater than the third side. i.e a, b and c are sides of a triangle if all the three conditions are true

a + b > c

b + c > a

c + a > b

In the above case {3, 4, 7} cannot be the sides of a triangle because 3+4 is not greater than 7 (should be strictly greater than the third side, and not equal).

In the above question, we are only interested in the count of triangles. Hence the signature of the function is

```
1    int countTriangles(int* arr, int n)
```

Bruite force method is to have three loops and consider all possible combinations of i,j,k. This method will take $O(n^3)$ time.

O(n^2) Solution

There is another $O(n^2)$ time algorithm which sort the array and then does not traverse the entire array in all the three loops.

Algorithm:

1. Sort the array in increasing order.

2. Initialize

   i = 0; // First Element

   j = 1; // Second Element

   count = 0; // Number of triangles

3. WHILE (i<n-2)

   k = j+1;

   move k forward until arr[k] becomes > (arr[i] + arr[j]) or k moves out of bound

   count = count + (k-j-1);

   increment j

   if j is at end of array

increment i and set j=i+1

4. Print count.

Note that in this case pointer ‘k‘ is not reset every time j is incremented, because since j is increasing, we need a higher value of ‘k‘ which will be on the right side. Hence we just need to move k forward.

k is reset only when i and j both are reset, because in that case k has already reached the upper bound of array.

Code:

```
1    int countTriangles(int* arr, int n)
2    {
3        // atleast 3 numbers are required for a triangle.
4        if(n<3) return 0;
5
6        quickSort(arr, 0, n-1);
7
8        int count = 0;
9        int i = 0;
10       int j = i+1;
11
12       while(i<n-2)
```

```
13        {
14            int k = j+1;
15
16            while(k<n && arr[k] < arr[i] + arr[j])
17                k++;
18
19            count += k-j-1;
20
21            j++;
22            // If j has reached the end. then reset both i & j.
23            if(j>=n)
24            {
25                i++;
26                j = i+1;
27            }
28        }
29        return count;
30    }
```

Time Complexity: $O(n^2)$

Extra Space Used: $O(1)$

Source: http://www.ritambhara.in/number-of-possible-triangles-from-given-array-of-numbers/