Count the number of possible triangles

Given an unsorted array of positive integers. Find the number of triangles that can be formed with three different array elements as three sides of triangles. For a triangle to be possible from 3 values, the sum of any two values (or sides) must be greater than the third value (or third side).

For example, if the input array is {4, 6, 3, 7}, the output should be 3. There are three triangles possible {3, 4, 6}, {4, 6, 7} and {3, 6, 7}. Note that {3, 4, 7} is not a possible triangle.

As another example, consider the array {10, 21, 22, 100, 101, 200, 300}. There can be 6 possible triangles: {10, 21, 22}, {21, 100, 101}, {22, 100, 101}, {10, 100, 101}, {100, 101, 200} and {101, 200, 300}

Method 1 (Brute force)

The brute force method is to run three loops and keep track of the number of triangles possible so far. The three loops select three different values from array, the innermost loop checks for the triangle property (the sum of any two sides must be greater than the value of third side).

Time Complexity: O(N^3) where N is the size of input array.

Method 2 (Tricky and Efficient)

Let a, b and c be three sides. The below condition must hold for a triangle (Sum of two sides is greater than the third side)

i) a + b > c

ii) b + c > a

iii) a + c > b

Following are steps to count triangle.

1. Sort the array in non-decreasing order.

2. Initialize two pointers ‘i’ and ‘j’ to first and second elements respectively, and initialize count of triangles as 0.

3. Fix ‘i’ and ‘j’ and find the rightmost index ‘k’ (or largest ‘arr[k]’) such that ‘arr[i] + arr[j] > arr[k]’. The number of triangles that can be formed with ‘arr[i]’ and ‘arr[j]’ as two sides is ‘k – j’. Add ‘k – j’ to count of triangles.

Let us consider ‘arr[i]’ as ‘a’, ‘arr[j]’ as b and all elements between ‘arr[j+1]’ and ‘arr[k]’ as ‘c’. The above mentioned conditions (ii) and (iii) are satisfied because ‘arr[i] < arr[j] < arr[k]'. And we check for condition (i)

when we pick 'k' **4.** Increment ‘j’ to fix the second element again.

Note that in step 3, we can use the previous value of ‘k’. The reason is simple, if we know that the value of ‘arr[i] + arr[j-1]’ is greater than ‘arr[k]’, then we can say ‘arr[i] + arr[j]’ will also be greater than ‘arr[k]’, because the array is sorted in increasing order.

5. If ‘j’ has reached end, then increment ‘i’. Initialize ‘j’ as ‘i + 1′, ‘k’ as ‘i+2′ and repeat the steps 3 and 4.

Following is implementation of the above approach.

C/C++

```
// C/C++ program to count number of triangles that can be
// formed from given array
#include <stdio.h>
#include <stdlib.h>

/* Following function is needed for library function
   qsort(). Refer
  http://www.cplusplus.com/reference/clibrary/cstdlib/qsort/ */
int comp(const void* a, const void* b)
{  return *(int*)a > *(int*)b ; }

// Function to count all possible triangles with arr[]
// elements
int findNumberOfTriangles(int arr[], int n)
{
    // Sort the array elements in non-decreasing order
    qsort(arr, n, sizeof( arr[0] ), comp);

    // Initialize count of triangles
    int count = 0;

    // Fix the first element.  We need to run till n-3
    // as the other two elements are selected from
    // arr[i+1...n-1]
    for (int i = 0; i < n-2; ++i)
    {
        // Initialize index of the rightmost third
        // element
        int k = i+2;

        // Fix the second element
        for (int j = i+1; j < n; ++j)
        {
            // Find the rightmost element which is
            // smaller than the sum of two fixed elements
            // The important thing to note here is, we
            // use the previous  value of k. If value of
            // arr[i] + arr[j-1] was greater than arr[k],
            // then arr[i] + arr[j] must be greater than k,
            // because the array is sorted.
            while (k < n && arr[i] + arr[j] > arr[k])
               ++k;

            // Total number of possible triangles that can
            // be formed with the two fixed elements is
            //  k - j - 1.  The two fixed elements are arr[i]
            // and arr[j].  All elements between arr[j+1]/ to
            // arr[k-1] can form a triangle with arr[i] and arr[j].
            // One is subtracted from k because k is incremented
```

```
            // one extra in above while loop.
            // k will always be greater than j. If j becomes equal
            // to k, then above loop will increment k, because arr[k]
            //  + arr[i] is always greater than arr[k]
            count += k - j - 1;
        }
    }

    return count;
}

// Driver program to test above functionarr[j+1]
int main()
{
    int arr[] =   {10, 21, 22, 100, 101, 200, 300};
    int size = sizeof( arr ) / sizeof( arr[0] );

    printf("Total number of triangles possible is %d ",
           findNumberOfTriangles( arr, size ) );

    return 0;
}
```

Run on IDE

Java

```
// Java program to count number of triangles that can be
// formed from given array
import java.io.*;
import java.util.*;

class CountTriangles
{
    // Function to count all possible triangles with arr[]
    // elements
    static int findNumberOfTriangles(int arr[])
    {
        int n = arr.length;
        // Sort the array elements in non-decreasing order
        Arrays.sort(arr);

        // Initialize count of triangles
        int count = 0;

        // Fix the first element.  We need to run till n-3 as
        // the other two elements are selected from arr[i+1...n-1]
        for (int i = 0; i < n-2; ++i)
        {
            // Initialize index of the rightmost third element
            int k = i + 2;

            // Fix the second element
            for (int j = i+1; j < n; ++j)
            {
                /* Find the rightmost element which is smaller
                   than the sum of two fixed elements
                   The important thing to note here is, we use
                   the previous value of k. If value of arr[i] +
                   arr[j-1] was greater than arr[k], then arr[i] +
                   arr[j] must be greater than k, because the
                   array is sorted. */
```

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

                /* Total number of possible triangles that can be
                   formed with the two fixed elements is k - j - 1.
                   The two fixed elements are arr[i] and arr[j].  All
                   elements between arr[j+1] to arr[k-1] can form a
                   triangle with arr[i] and arr[j]. One is subtracted
                   from k because k is incremented one extra in above
                   while loop. k will always be greater than j. If j
                   becomes equal to k, then above loop will increment
                   k, because arr[k] + arr[i] is always/ greater than
                   arr[k] */
                count += k - j - 1;
            }
        }
        return count;
    }

    public static void main (String[] args)
    {
        int arr[] = {10, 21, 22, 100, 101, 200, 300};
        System.out.println("Total number of triangles is " +
                           findNumberOfTriangles(arr));
    }
}
/*This code is contributed by Devesh Agrawal*/
```

Run on IDE

Python

```
# Python function to count all possible triangles with arr[]
# elements

def findnumberofTriangles(arr):

    # Sort array and initialize count as 0
    n = len(arr)
    arr.sort()
    count = 0

    # Fix the first element.  We need to run till n-3 as
    # the other two elements are selected from arr[i+1...n-1]
    for i in range(0,n-2):

        # Initialize index of the rightmost third element
        k = i + 2

        # Fix the second element
        for j in range(i+1,n):

            # Find the rightmost element which is smaller
            # than the sum of two fixed elements
            # The important thing to note here is, we use
            # the previous value of k. If value of arr[i] +
            # arr[j-1] was greater than arr[k], then arr[i] +
            # arr[j] must be greater than k, because the array
            # is sorted.
            while (k < n and arr[i] + arr[j] > arr[k]):
                k += 1
```

```
            # Total number of possible triangles that can be
            # formed with the two fixed elements is k - j - 1.
            # The two fixed elements are arr[i] and arr[j]. All
            # elements between arr[j+1] to arr[k-1] can form a
            # triangle with arr[i] and arr[j]. One is subtracted
            # from k because k is incremented one extra in above
            # while loop. k will always be greater than j. If j
            # becomes equal to k, then above loop will increment k,
            #  because arr[k] + arr[i] is always greater than arr[k]
            count += k - j - 1

    return count

# Driver function to test above function
arr = [10, 21, 22, 100, 101, 200, 300]
print "Number of Triangles:",findnumberofTriangles(arr)

# This code is contributed by Devesh Agrawal
```

Run on IDE

Output:

```
Total number of triangles possible is 6
```

Time Complexity: O(n^2). The time complexity looks more because of 3 nested loops. If we take a closer look at the algorithm, we observe that k is initialized only once in the outermost loop. The innermost loop executes at most O(n) time for every iteration of outer most loop, because k starts from i+2 and goes upto n for all values of j. Therefore, the time complexity is O(n^2).

Source: http://stackoverflow.com/questions/8110538/total-number-of-possible-triangles-from-n-numbers

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.

94 Comments Category: Arrays

Related Posts:

- Longest Span with same Sum in two Binary arrays
- Count Inversions of size three in a give array
- Find the subarray with least average
- Count triplets with sum smaller than a given value
- Find zeroes to be flipped so that number of consecutive 1's is maximized
- Reorder an array according to given indexes
- Find maximum value of Sum(i*arr[i]) with only rotations on given array allowed
- Find maximum average subarray of k length

(Login to Rate and Mark)

2.5 Average Difficulty : **2.5/5.0**
Based on **4** vote(s)

Add to TODO List
Mark as DONE

Writing code in comment? Please use code.geeksforgeeks.org, generate link and share the link here.