Valid Triangle Number

Given an array consists of non-negative integers, your task is to count the number of triplets chosen from the array that can make triangles if we take them as side lengths of a triangle.

Example 1:

```
Input: [2,2,3,4]
Output: 3
Explanation:
Valid combinations are:
2,3,4 (using the first 2)
2,3,4 (using the second 2)
2,2,3
```

Note:

1. The length of the given array won't exceed 1000.
2. The integers in the given array are in the range of [0, 1000].

Solution 1

```
public static int triangleNumber(int[] A) {
    Arrays.sort(A);
    int count = 0, n = A.length;
    for (int i=n-1;i>=2;i--) {
        int l = 0, r = i-1;
        while (l < r) {
            if (A[l] + A[r] > A[i]) {
                count += r-l;
                r--;
            }
            else l++;
        }
    }
    return count;
}
```

written by compton_scatter original link here

Solution 2

O(n^2) python solution got TLE.
I assume no faster solutions exist, right?

written by iynaur87 original link here

Solution 3

Sort the array. For every pair of sticks u, v with stick u occuring before v (u <= v), we want to know how many w occuring after v have w < u + v.

For every middle stick B[j] = v, we can use two pointers: one pointer i going down from j to 0, and one pointer k going from the end to j. This is because if we have all w such that w < u + v, then decreasing u cannot make this set larger.

Let's look at an extension where our sorted array is grouped into counts of it's values. For example, instead of dealing with A = [2,2,2,2,3,3,3,3,3,4,4,4], we should deal with only B = [2, 3, 4] and keep a sidecount of C[2] = 4, C[3] = 5, C[4] = 3. We'll also keep a prefix sum P[k] = C[B[0]] + C[B[1]] + ... + C[B[k-1]] (and P[0] = 0.)

When we are done setting our pointers and want to add the result, we need to add the result taking into account multiplicities (how many times each kind of triangle occurs.) When i == j or j == k, this is a little tricky, so let's break it down case by case.

- When i < j, we have C[B[i]] * C[B[j]] * (P[k+1] - P[j+1]) triangles where the last stick has a value > B[j]. Then, we have another C[B[i]] * (C[B[j]] choose 2) triangles where the last stick has value B[j].
- When i == j, we have (C[B[i]] choose 2) * (P[k+1] - P[j+1]) triangles where the last stick has value > B[j]. Then, we have another (C[B[i]] choose 3) triangles where the last stick has value B[j].

```
def triangleNumber(self, A):
    C = collections.Counter(A)
    C.pop(0, None)
    B = sorted(C.keys())
    P = [0]
    for x in B:
        P.append(P[-1] + C[x])

    ans = 0
    for j, v in enumerate(B):
        k = len(B) - 1
        i = j
        while 0 <= i <= j <= k:
            while k > j and B[i] + B[j] <= B[k]:
                k -= 1
            if i < j:
                ans += C[B[i]] * C[B[j]] * (P[k+1] - P[j+1])
                ans += C[B[i]] * C[B[j]] * (C[B[j]] - 1) / 2
            else:
                ans += C[B[i]] * (C[B[i]] - 1) / 2 * (P[k+1] - P[j+1])
                ans += C[B[i]] * (C[B[i]] - 1) * (C[B[i]] - 2) / 6
            i -= 1
    return ans
```

written by awice original link here