华为OD机试 - 生日礼物（Java & JS & Python）

题目描述

小牛的孩子生日快要到了，他打算给孩子买蛋糕和小礼物，蛋糕和小礼物各买一个，他的预算不超过x元。蛋糕cake和小礼物gift都有多种价位的可供选择。

请返回小牛共有多少种购买方案。

输入描述

第一行表示cake的单价，以逗号分隔

第二行表示gift的单价，以逗号分隔

第三行表示x预算

输出描述

输出数字表示购买方案的总数

备注

- 1 ≤ cake.length ≤ 10^5
- 1 ≤ gift.length ≤10^5
- 1 ≤ cake[i]，gift[i] ≤ 10^5
- 1 ≤ X ≤ 2*10^5

用例

输入	10,20,5 5,5,2 15
输出	6
说明	解释: 小牛有6种购买方案，所选蛋糕与所选礼物在数组中对应的下标分别是: 第1种方案: cake [0] + gift [0] = 10 + 5 = 15; 第2种方案: cake [0] + gift [1]= 10 + 5 = 15; 第3种方案: cake [0] + gift [2] = 10 + 2 = 12; 第4种方案: cake [2] + gift [0] = 5 + 5 = 10; 第5种方案: cake [2] + gift [1]= 5 + 5 = 10; 第6种方案: cake [2] + gift [2] = 5 + 2 = 7。

题目解析

本题可以使用 二分查找 解决。

我的解题思路如下:

由于蛋糕和小礼物各买一个，且总预算为x。

因此，假设我们先买了蛋糕花了cake元，那么能用于买到的小礼物的最高价格就已经确定了，为x - cake元。因此只要<=x - cake元的小礼物，都可以可以cake元的蛋糕组合。

为了避免花费O(n)时间在gifts中找<=x - cake元的小礼物，我们可以将gifts进行升序，然后通过二分查找x-cake元在升序后的gitfs中的位置

二分查找目标值target在 有序数组 nums中的位置，有两种情况:

- nums中存在target，则二分查找最终会返回target在nums中的位置
- nums中不存在target，则二分查找会返回target在nums中的有序插入位置

关于这两个位置的实现，可以看下面博客中二分查找部分:

算法设计 - 二分法和三分法，洛谷P3382_三分法和二分法_伏城之外的博客-CSDN博客

如果gifts进行升序后，二分查找x-cake元的位置 i :

- i 是目标位置，则可以产生 i + 1 种组合
- i 是有序插入位置，则 i = -i -1，即需要先变为有序插入位置，而有序插入位置的必然 gitfs[i] > x - cake，因此只能产生 i 种组合

2023.06.12

本题中如果存在多个相同的价格的蛋糕或者礼物，

比如gifts数组升序后为：1,2,3,3,3,3,3,4

而现在选择的蛋糕价格是3，总预算是6，那么gitfs最高价格可选3。

因此，我们期望二分查找返回gifts中价格3的位置是6，即最后一个价格3的位置。

而普通的二分查找，无法找最后一个目标值的位置，此时的解决思路是:

LeetCode - 34 在排序数组中查找元素的第一个和最后一个位置_伏城之外的博客-CSDN博客

另外，关于最终的购买方案，可能会存在价位重复的组合，那么是否需要去重?

可以看下用例1中，第4种和第5种方案，是不需要去重的。

```java
import java.util.Arrays;
import java.util.Scanner;

public class Main {
  public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);

    int[] cakes = Arrays.stream(sc.nextLine().split(",")).mapToInt(Integer::parseInt).toArray();
    int[] gifts = Arrays.stream(sc.nextLine().split(",")).mapToInt(Integer::parseInt).toArray();
    int x = Integer.parseInt(sc.nextLine());

    System.out.println(getResult(cakes, gifts, x));
  }

  public static long getResult(int[] cakes, int[] gifts, int x) {
    Arrays.sort(cakes);

    long ans = 0;
    for (int gift : gifts) {
      if (x <= gift) continue;

      int maxCake = x - gift;
      int i = searchLast(cakes, maxCake);

      if (i >= 0) {
        ans += i + 1;
      } else {
        i = -i - 1;
        ans += i;
      }
    }

    return ans;
  }

  public static int searchLast(int[] arr, int target) {
    int low = 0;
    int high = arr.length - 1;
```

```
39
40      while (low <= high) {
41        int mid = (low + high) >> 1;
42        int midVal = arr[mid];
43
44        if (midVal > target) {
45          high = mid - 1;
46        } else if (midVal < target) {
47          low = mid + 1;
48        } else {
49          // 向右延伸判断，mid是否为target数域的右边界，即最后一次出现的位置
50          if (mid == arr.length - 1 || arr[mid] != arr[mid + 1]) {
51            return mid;
52          } else {
53            low = mid + 1;
54          }
55        }
56      }
57
58      return -low - 1; // 找不到则返回插入位置
59    }
60  }
```