Reading process:

We start with an integer array named “aArray”.(line 48) Considering that the problem statement said that A is an integer array, it will be of type “int[]”. Its maximum capacity will be 30. We will have an integer variable “sizeA”, which is first set to 0, and will represent the number of elements in aArray. (line 49)

Considering that the problem statement never mentioned anything regarding the number of elements in the array, I decided that we will read from the keyboard integers, which will be added to the array, until 0 is read. Here we use a boolean variable “ok”, whose value is “true” at first, but will become “false” when 0 is read. (lines 50-60)

Computations regarding the average:

We have to move every element of the array aArray to 2 lists, either list B or C. If list B has m elements for example, it is sure that list C will have sizeA – m elements. We also know that the average of the array aArray is:

average = sumA / sizeA, where sumA represents the sum of all elements of aArray.(line 61). To compute the sum of aArray, we use an auxiliary function named “sumArray”, which calculates the sum of the elements of an array, having as parameters the array and its size.(lines 33-38)

From above, we can write sizeA as m + (sizeA – m). List B and list C will have the sums sumB, respectively sumC. sumA can also be written as sumB + sumC. For our computations, we will write sumC as sumA – sumB.

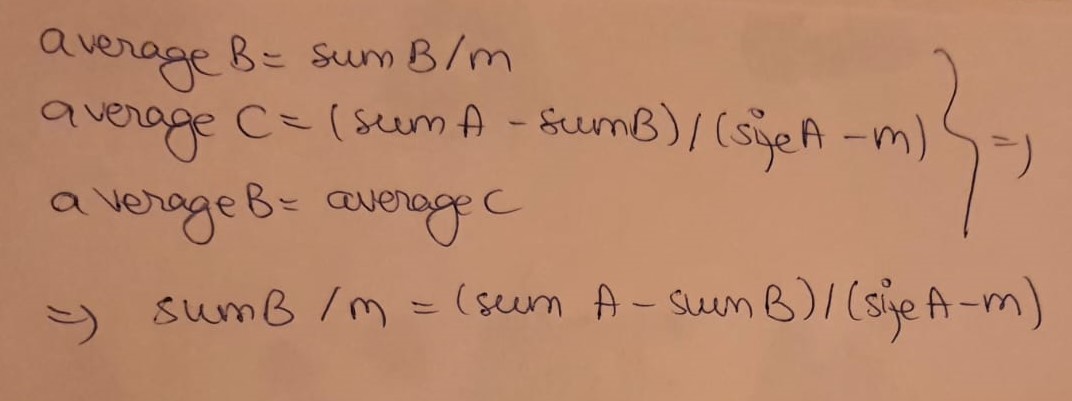
Since the problem statement asks for the averages of lists B and C to be equal, then their averages will have the formulas:

Figure 1

Since averageB = averageC, then:

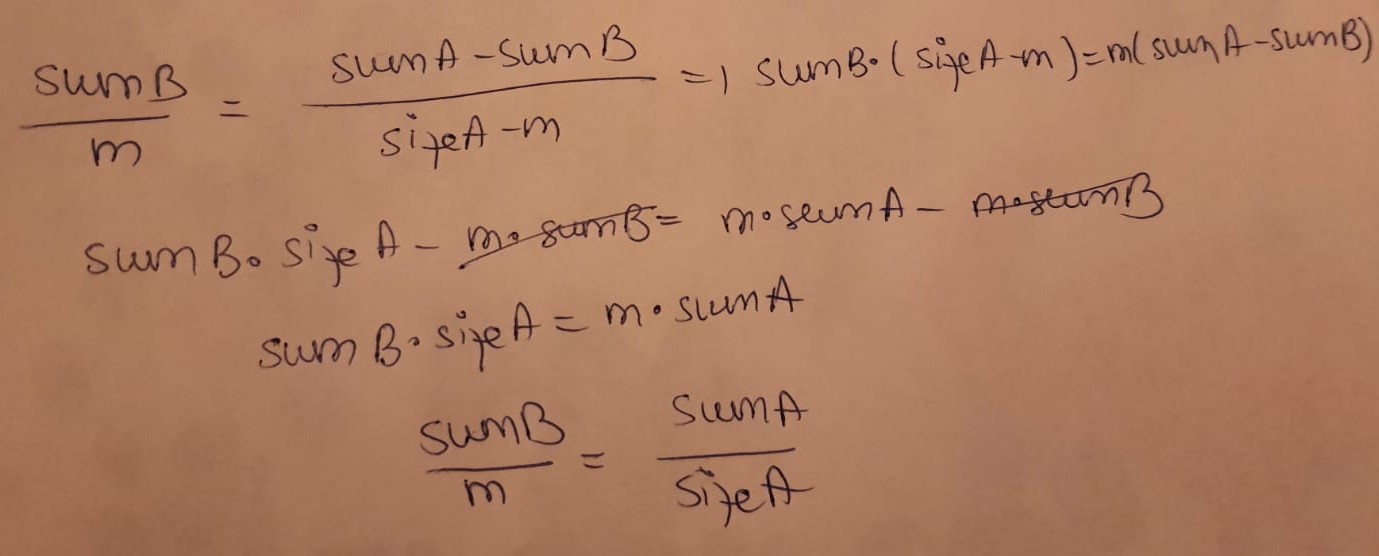


Figure 2

From the computations, we realize that the average of list B is equal to the average of aArray, which means that the averages of lists B and C will be equal only if they match the average of aArray.

Computing the lists:

Considering that the problem statement said that B and C are lists, we start with 2 lists, “bList” and “cList” of type ArrayList<Integer> (lines 63-64).

To find a combination of numbers with the average equal to the one of the array, we do the following trick: as the average is written as sum/m number of elements, then we have to find a combination of m number of elements which have a certain sum, so that we get the right average. For this, we use an auxiliary function called “findElemsForAvg”, having as parameters the array, its size, the needed sum and the numbers of elements the combination should have, so that it has the right average (lines 7-31). We reuse the boolean variable “ok” and mark its value as false, meaning that we haven’t found a good combination yet (line 65).

Function for computing the combinations:

The function “findElemsForAvg” is a recursive function. We decrease the sum and the variable “shouldHave”(which represents the number of needed elements) when we find an element that can be part of the combination. It treats the following cases:

* if the sum is 0 and the number of needed elements is 0, it means we found a combination with the right average, and we return it (lines 10-11)
* if the size of the array gets to 0, then we haven’t found a combination which matches the requirements (lines 12-13)
* by starting from the last element of the array, we verify if it is bigger than the sum. If it is, then we call the function, with the size decremented(so that we continue searching in the other elements) (lines 15-16)
* if the element is smaller than the sum, it means it can be part of the combination. we treat 2 cases: one in which the element is included in the combination, and one in which it is excluded
  + we try to find a combination without the current element (line 19). if we can’t find any combination, then we try to find a combination which includes the current element. if we find one, we return the new array (lines 29-30)
  + by including the current element, not only the size will be decremented, but the sum will also decrease(it will become sum – current element). by calling the function, we try to find a combination with the current element. if there’s none, we return null. otherwise, when coming back from the recursion, the current element is added to the combination and we return the new array(lines 20-28)

Final search:

In a for-loop from 1 to sizeA, we try to find if there is a combination with i elements that has the right average. From figure2, we calculate the sum that the combination should have(line 68). bList will have the value of the function findElemsForAvg (line 69). If we found a combination, we create the other list, which has the other elements of aArray (lines 70-74). Every time we create cList, we should make sure it’s empty at first(line 67). Using a function called “sumList”(lines 40-45), we calculate the sums of the elements from bList and cList. Since aArray and bList have different structures, we call sum functions that respect their type.

We calculate the average of each list(lines 77-78). bList has the size I, while cList has the size sizeA-i (the difference). Because we might have division by 0 (i or sizeA-i can be 0 if there is no combination possible), we verify if the averages are equal in a try-catch block(lines 79-87). If they are equal, the variable ok becomes true and we break the loop.

Since we have to print a boolean value as a result, we will print the value of variable ok at the end (line 88).