Codigo en linea

Array 2

countEven: The variable "x" serves a counter, the for loop goes through the array, and ask if the value on [i] divided by 2 has a reminder, if it is then being not pair, if not x increase by 1.
At last return x

bigDiff: The variables "x" and "y" stores the greater and the smaller value on the array, the loop goes through the values on the array and ask if [i] is greater than "x", if it is, then [i] will be assigned to "x". Then ask if [i] is smaller than "y", and if it is, the [i] will be assigned to "y". At last return x - y.

lucky13: The loop goes through the array and ask for every value on the index [i] if it's equal to 1 or 3, if true, return false, if the loop ends then return true.

sum28: The variable "x" serves as a counter for every 2 in the array. The for loop goes through every position on the array and ask if there's a 2, if true, increase "x" by 1. At the end multiply "x" by 2 and return if is equal to 8.

more14: The variables "x" and "y" stores the number of 1's and 4's on the array. The for loop goes through every position and ask if there's a 1 or 4 on the position, then increase by 1, "x" if it's a 1 and "y" if it's a 4. At last return if "x" is greater than "y".

Array 3

maxSpan: The algorithm consists of two "for loops" which will used for a double check of the array, and two variables, "maxSpan" and "currentSpan".
"maxSpan" will serve as champion counter and will allow the algorithm to compare the last longer span with a possible new span.
"currentSpan" is used to save a new span and then compare it with the span in the champion counter.
Both used loops are nested, and in the first loop with the variable i, initialized in 0, the variable "currentSpan" is initialized in 0 since this counter must be restarted every time we finish counting a span.
The second loop with the variable j, initialized in i, since we do not care about the numbers behind the index i because they were already compared.
In this loop we look for a pair for the position [i] in the index j of the array, since if in the index i there's the same number that is in the index j then there will be a span from the position [i] to the position [j], which is equal to the subtraction: j - (i + 1), we add 1 since the position i starts from 0.
At the end the variable "maxSpan" will be returned since it saves the length of the longest span.

Fix34 and Fix45: This algorithm consists of two nested "for loops" both initialized in 0, which will be used for a double check of the array.
The first loop will be searching for an index i where the number 3 or 4 exists (3: fix34, 4: fix45), when it is found it will pass to the next loop that will look for a 4 or 5 (4: fix34, 5: fix45) within the array, after finding it, it will verify that it is not ordered with a 3 or 4 (3: fix34, 4: fix45) in the position j - 1, previously verifying that j is not equal to 0, since we would be standing out of the array, if the 4 or 5 (4: fix34, 5: fix45) found is already ordered, the loop will continue looking for another one, on the other hand if it is not ordered, the values on the positions [j] and [i + 1] will swap.

canBalance: This algorithm consists of two "for loops" used to make a double check in the array and two variables "sum" and "temp", the first one initialized in 0 that will be used to add the values to the left of the array, the second one initialized in "sum" to subtract the right part of the array.
The first loop will divide the array into two parts (right and left), the values on the index i will be added to "sum" and the value of "sum" will be assigned to "temp".
In the second loop initialized in i + 1 since "i" represents the division of the array. The values to the right of the array will be subtracted from "temp", and if at the end "temp" is equal to 0, return true, else return false.

linearIn: This algorithm consists of two nested "for loops" used to check each of the two arrays and two variables "posInner" and "path", the first one initialized in 0, will save the position of the last comparison in the second array, and the second used as a counter for the number of pairs between both array.
The first loop will only be used to go through the "Outer" array.
For each iteration in the first loop the second one will search from "path" to the length of the second array for a pair value in the array "inner" for the value on the position [i] of the "Outer" array, and if it finds it, then "path" will increase by 1, "j" will be assigned to "PosInner", and break the loop.
At the end it is returned if the path is equal to the length of the array.