ALGORITHM ASSIGNMENT 1

By paul Arah

GitHub File Link:https://github.com/paularah/algo-assignment/blob/master/main.go

Problem-solving approach: To solve this problem, the first thing we need to know is how many times a sock occurs in the array. And then can determine how many pair of particular sock occurs in the array. We can create an empty map otherwise known as dictionaries or objects in some programming language and we keep track of how many times a value occurs in the socks array.

```
FUNCTION count_pairs(socks_array)

//initialize an empty map
socks_map  = {}
//initialize a variable to keep track of the number of pairs
result = 0

//loop through the socks array
FOR socks_array[index 0] to socks_array[index_end]
     //check if array value is a key in socks map
     IF array_value present as a key in socks_map

          //if they key exists increment the value by 1
          socks_map[array value] = socks_map[array value] + 1
     ELSE

     //else initialise the value to 1
     Socks_map[array_value] = 0
END FOR

//this essentially creates a count of every element in the array
and their occurrences

//loop through the sock map that now has all occurrences of the
element in the array
FOR key_value_pair in socks_map:
//ignore the key, we're interested in the values

IF value is greater than or equals to 2
```

```
     //increment result by the quotient of the division of the
value by 2,
     result = result + quotient of value divided by 2
END FOR

return result

END FUNCTION
```

Complexity Analysis - *O(n)*

The major operations here with a huge impact on the time complexity is the first loop through the array which is *O(n)* is the best and worst-case scenario. The second loop over the values in the map is varies based on the uniqueness of the values in the original array of socks and in the worst-case scenario where every value in the original socks array is unique, the time complexity is *0(n)*. Looking up a key in an associative array like a map in our case, or objects, dictionaries, etc is *O(1)* time. So in total, the time complexity of the entire solution is ***O(n).*** The naive approach to solving this problem would involve using a nested for loop and counting the pairs. The decision to use a map to keep track of the occurrences of elements in the array reduces the time complexity from *0(n^2)* to O(n) but this increases the space complexity and more memory is used to store and keep track of the maps and variables.