**Whole Minute Dilemma**

QUESTION DESCRIPTION

A music player allows users to choose songs to play, but only in pairs and only pairs of songs with

durations that add up to a multiple of 60 seconds (e.g., 60, 120, 180). Given a list of song durations,

calculate the total number of different song pairs that can be chosen.

**Example**

n = 3

songs = [40, 20, 60]

One pair of songs can be chosen whose combined duration is a multiple of a whole minute (40 + 20 = 60) and the return value would be 1. While the third song is a single minute long, songs must be chosen in pairs.

**Function Description**

Complete the function playlist in the editor below.

playlist has the following parameter(s):

int songs[n]: array of integers representing song durations in seconds

**Returns:**

int: the number of songs pairs that add up to a multiple of a minute

**Constraints**

1 ≤ n ≤ 10

1 ≤ songs[i] ≤ 1000, where 0 ≤ i < n

**Input Format For Custom Testing**

Input from stdin will be processed as follows and passed to the function.

The first line contains an integer, n, that denotes the number of elements in songs.

The next n lines each contain an integer that describes songs[i] and denotes the duration of the i song in seconds.

**Sample Case 0**

**Sample Input For Custom Testing**

STDIN Function

4 → songs[] size n = 4

10 → songs = [10, 50, 90, 30]

50

90

30

**Sample Output**

2

**Explanation**

The first and second songs pair to 60 seconds. The third and fourth songs pair to 120 seconds. No

other pairs will satisfy the requirement.

**Sample Case 1**

Sample Input For Custom Testing

STDIN Function

5 → songs[] size n = 5

30 → songs = [30, 20, 150, 100, 40]

20

150

100

40

**Sample Output**

3

**Explanation**

There are three pairs of songs whose whole duration is a multiple of a whole minute. They are 20 + 100, 30 + 150, and 20 + 40 corresponding to (1, 3), (0, 2) and (1, 4).

**Sample Case 2**

Sample Input For Custom Testing

STDIN Function

3 → songs[] size n = 3

60 → songs = [60, 60, 60]

60

60

**Sample Output**

3

**Explanation**

There are three pairs of songs that end in whole minutes. They are (0, 1), (1, 2) and (0, 2).

**Hint 1**

Notice that for two songs to have a combined duration that is a multiple of 60, the sum of their values modulo 60 must be 0.

**Hint 2**

For a song with duration modulo 60 = x, we can pair it up with a song of duration modulo 60 = 60-x.Maintaining a frequency table might help.

**Solution**

Concepts covered: Hash Tables, Modulo Arithmetic

**Optimal Solution**:

Let's maintain a frequency table for each i (0 ≤ i ≤ 60) denoting the number of songs with duration modulo 60 = i. Pair each value at freq[i] with the value at freq[60-i] where 1 ≤ i ≤ 29. The number permutations is freq[i]\*freq[60-i]. Now, for songs have a length modulo 60 of 0 or 30, indices 0 and 30, they pair with others at the same index. The numbers of pairs for these indices are ⌊freq[0]\*(freq[0] -1)/2⌋ and ⌊freq[30]\*(freq[30]-1)/2⌋.

def playlist(songs):

freq = {}

for song in songs:

freq[song % 60] = freq.get(song % 60, 0) + 1

ans = 0

for i in range(1, 30):

ans += freq.get(i, 0) \* freq.get(60 - i, 0)

ans += freq.get(0, 0) \* (freq.get(0, 0) - 1) // 2

ans += freq.get(30, 0) \* (freq.get(30, 0) - 1) // 2

return ans

Brute Force Approach: Passes 9 out of 15 test cases

def playlist(songs):

ans = 0

for i in range(len(songs)):

for j in range(i+1, len(songs)):

if (songs[i] + songs[j]) % 60 == 0:

ans += 1

return ans

**Error Handling**: There are two edge cases in the problem: Pairing songs with length modulo 60 as 0

and 30. Candidates, may attempt to pair a song with itself in such cases.

**CANDIDATE ANSWER**

Language used: Java 8

class Result {

/\*

\* Complete the 'playlist' function below.

\*

\* The function is expected to return a LONG\_INTEGER.

\* The function accepts INTEGER\_ARRAY songs as parameter.

\*/

public static long playlist(List<Integer> songs) {

}