

## Part 5: Communication and English

At talent.io, the language we work with is English so we are looking for teammates able to communicate in that language.

Can you answer the following in about 100-150 words:

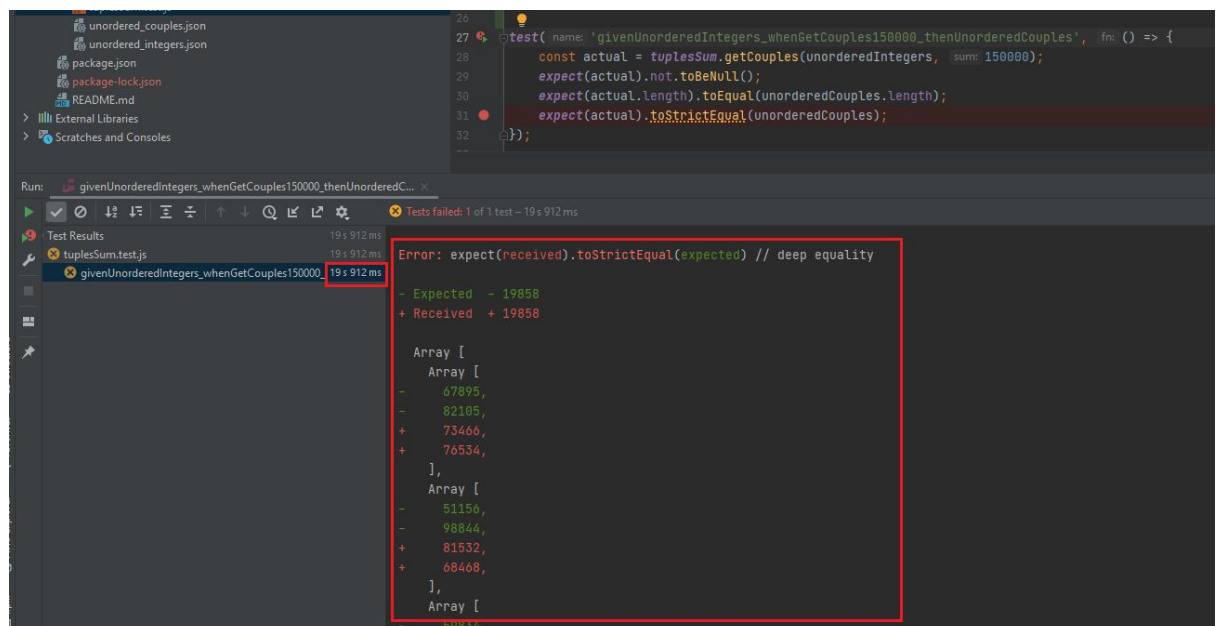
Talk about the last bug that you had to solve.

In the first part “*tuples\_sum*” of this software assignment, following the first implementation of my hash solution and given the provided “*ordered\_positive\_couples.json*” file, my function actually provided **more than the expected** number of couples. This was according to my unit testing tool Jest. So, my programmed **successfully computed both** entities : expected and actual, or actually the provided file and and the result of my algorithm.

The obvious reason of the **size difference** was caused by the “multiple instance of the same couple”, so I managed to mark the already added **number** by setting the corresponding *hash* object value to *false*. This way, if an hash value was :

- true : the number has to be added, along with its diff
- undefined : the number has not been found yet, it can be set to true for later
- strictly false: the number has already been added. Skip. Which looked like:
  - `if (hash[n] === false) { // avoid repetitions`
  - `continue;`
  - `}`

But while this was supposed to make the program run faster by skipping already added numbers, it then felt **slow**, going up to 19 seconds:



```
26
27 test( name: 'givenUnorderedIntegers_whenGetCouples150000_thenUnorderedCouples', fn () => {
28   const actual = tuplesSum.getCouples(unorderedIntegers, {sum: 150000});
29   expect(actual).not.toBeNull();
30   expect(actual.length).toEqual(unorderedCouples.length);
31   expect(actual).toEqual(unorderedCouples);
32 });
```

Run: givenUnorderedIntegers\_whenGetCouples150000\_thenUnorderedC... x

Test Results 19 s 912 ms

tuplesSum.test.js 19 s 912 ms

givenUnorderedIntegers\_whenGetCouples150000 19 s 912 ms

Error: expect(received).toEqual(expected) // deep equality

- Expected - 19858

+ Received + 19858

Array [

Array [

- 67895,

- 82105,

+ 73466,

+ 76534,

],

Array [

- 51156,

- 98844,

+ 81532,

+ 68468,

],

Array [

- 59834,

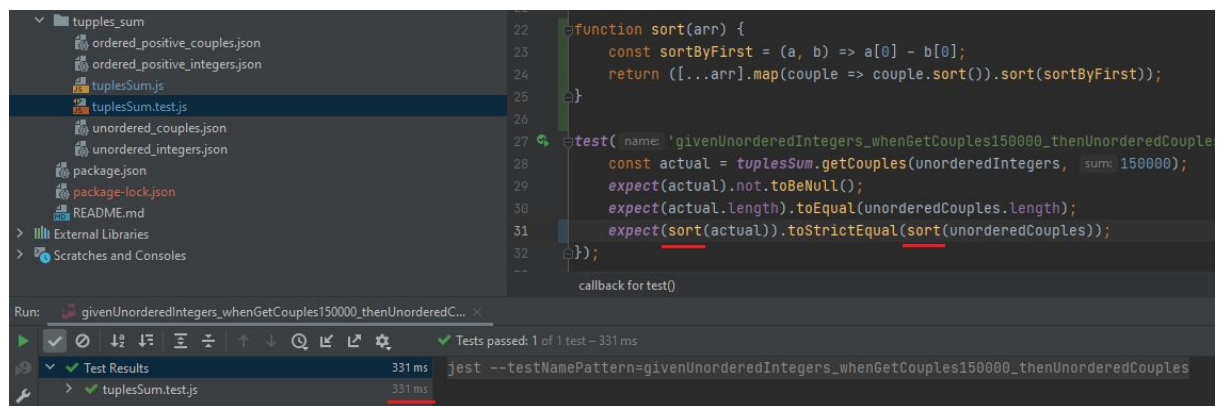
And it was still not working: the expected and actual couples were reported as different.

So I made multiple attempts checking the code, the *continue* keyword, and made sure it behaved as expected. Yet while adding this *continue* instruction, the unit tests definitely took longer.

Then it hit me: in this series of tests, I first compared the data sizes, then their actual content. Thus, without the *continue* keyword, the first test would fail and the next one was **skipped**. So my program was definitely behaving as expected.

Now then, why were the expected and actual data different? They were ordered differently. The provided result test file is **meant for a naive quadratic loop**, where first numbers are added. Whilst in the hash solution, they are added only when the second member of a couple is found while traversing the list.

I solved this by **sorting both** data arrays, and it worked, actually pretty fast : 331ms in this iteration.



The screenshot shows a VS Code editor with a file explorer on the left and a code editor on the right. The file explorer shows a project named 'tuples\_sum' with files like 'ordered\_positive\_couples.json', 'ordered\_positive\_integers.json', 'tuplesSum.js', 'tuplesSum.test.js', 'unordered\_couples.json', 'unordered\_integers.json', 'package.json', 'package-lock.json', and 'README.md'. The code editor shows the content of 'tuplesSum.test.js' with a Jest test. The test is named 'givenUnorderedIntegers\_whenGetCouples150000\_thenUnorderedCouple' and it checks that the result of 'tuplesSum.getCouples(unorderedIntegers, sum: 150000)' is not null, has the same length as 'unorderedCouples', and is strictly equal to 'sort(unorderedCouples)' after sorting both arrays. The test is marked as passed. Below the code editor, the 'Run' panel shows the test results: 'Test Results' for 'tuplesSum.test.js' with a duration of 331 ms. The command used is 'jest --testNamePattern=givenUnorderedIntegers\_whenGetCouples150000\_thenUnorderedCouples'.

```
function sort(arr) {
  const sortByFirst = (a, b) => a[0] - b[0];
  return [...arr].map(couple => couple.sort()).sort(sortByFirst);
}

test( name: 'givenUnorderedIntegers_whenGetCouples150000_thenUnorderedCouple', () => {
  const actual = tuplesSum.getCouples(unorderedIntegers, sum: 150000);
  expect(actual).not.toBeNull();
  expect(actual.length).toEqual(unorderedCouples.length);
  expect(sort(actual)).toStrictEqual(sort(unorderedCouples));
});
```

Run: givenUnorderedIntegers\_whenGetCouples150000\_thenUnorderedC...  
Tests passed: 1 of 1 test - 331 ms  
Test Results: 331 ms  
tuplesSum.test.js: 331 ms  
jest --testNamePattern=givenUnorderedIntegers\_whenGetCouples150000\_thenUnorderedCouples

Sorting requires additional compute time, so one mystery remained: why was it faster than the non working solution? My deduction is that in case of failure, Jest needs time to generate a **detailed Comparison Failure**. While it definitely helped to find bugs, the waiting time of 19s could have been nice to skip with an argument.

Note: I'm sorry, this is actually 394 words long, and I provided my solution for a whole track instead of a single bug.