Has Target Sum Lab

- Due No Due Date
- Points 1
- Submitting a website url

 (https://github.com/learn-co-curriculum/phase-1-algorithms-has-target-sum)  (https://github.com/learn-co-curriculum/phase-1-algorithms-has-target-sum/issues/new)

Learning Goals

- Practice algorithmic problem solving
- Identify the Big O of an algorithm

Introduction

Time for more practice! Fork and clone this lab, then **read the instructions before running any tests**.

Instructions

Write a function called `hasTargetSum` that receives two arguments:

- an `array` of integers
- a `target` integer

The function should return true if any pair of numbers in the array adds up to the target number.

Here are a few examples:

```
hasTargetSum([3, 8, 12, 4, 11, 7], 10);
// returns true, since 3 and 7 add up to 10

hasTargetSum([22, 19, 4, 6, 30], 25);
```

```
// returns true, since 19 and 6 add up to 25

hasTargetSum([1, 2, 5], 4);
// returns false, since no pair of numbers adds up to 4
```

This is a challenging problem, but you have the tools to come up with a solution! It's ok if your initial solution isn't optimal from a runtime perspective; it's totally fine to brute force your way to a solution and get something working before trying to optimize.

Problem Solving Approach

Use the **problem solving process (https://github.com/learn-co-curriculum/phase-1-algorithms-what-is-an-algorithm)** described in the previous lesson to come up with an approach to the problem and write your solution:

1. Rewrite the Problem in Your Own Words
2. Write Your Own Test Cases
3. Pseudocode
4. Code
5. Make It Clean and Readable
6. Optimize

Code your solution in the `index.js` file. There's space in the `index.js` file to write your pseudocode, but you're welcome to write pseudocode with pen and paper if you find that more beneficial.

You should also write some additional test cases in the `index.js` file to check your solution. After writing your test cases and coding your solution, you can view the output of your tests by running `node index.js` in the terminal.

When you're satisfied with your code, run `npm test` to run the tests and submit the lab in CodeGrade.

Once you've completed your implementation, determine what the time and space complexity of your algorithm is using Big O notation. If you came up with more than one approach, what are the tradeoffs? Which is more efficient?

Conclusion

In the next lesson, we'll review some solutions to this problem and talk through how to apply the problem solving process to this exercise. Take your time with this, and give it your best effort before reviewing the solution — it's ok if you don't end up with a working solution on the first try, so long as you make a deliberate effort at developing your problem solving process!