1. Two Sum

Given an array of integers `nums` and an integer `target`, return *indices of the two numbers such that they add up to* `target`.

You may assume that each input would have ***exactly* one solution**, and you may not use the *same* element twice.

You can return the answer in any order.

```
1  var twoSum = function (nums, target) {
2    const seen = new Map([[nums[0], 0]]);
3    for (let i = 1; i < nums.length; i++) {
4
5      if (seen.has(target - nums[i])) {
6        return [seen.get(target - nums[i]), i];
7      } else {
8          seen.set(nums[i], i);
10     }
11   }
12 };
13 twoSum([2,3,5,6],9)
```

The *Map* object holds key-value pairs and remembers the original insertion order of the keys. Any value (both objects and primitive values) may be used as either a key or a value.

The `has()` method returns a boolean indicating whether an element with the specified key exists or not.

The `get()` method returns a specified element from a `Map` object. If the value that is associated to the provided key is an object, then you will get a reference to that object and any change made to that object will effectively modify it inside the *Map* object.

The `set()` method adds or updates an element with a specified key and a value to a `Map` object.

LeetCode

2 `twoSum([2,3,5,6],9)`

nums	
target	9
seen	undefined

array: 2 3 5 6

3 `const seen = new Map([[nums[0], 0]]);`

map: 2 0

loop (from Map): i und

4 `for (let i = 1; i < nums.length; i++)`

loop (from Map): i 1

for loop: i 1

5 `if (seen.has(target - nums[i])) {`

6
```
} else {
    seen.set(nums[i], i);
```

map: 2 0 / 3 1

7 `for (let i = 1; i < nums.length; i++)`

for loop: i 2

8 `if (seen.has(target - nums[i])) {`

9
```
} else {
    seen.set(nums[i], i);
```

map: 2 0 / 3 1 / 5 2

10 `for (let i = 1; i < nums.length; i++)`

for loop: i 2

11 `if (seen.has(target - nums[i])) {`

12 `return [seen.get(target - nums[i]), i];`

RETURN VALUE: 1 3

jonchristie.net

1. Two Sum

Given an array of integers `nums` and an integer `target`, return *indices of the two numbers such that they add up to* `target`.

You may assume that each input would have ***exactly* one solution**, and you may not use the *same* element twice.

You can return the answer in any order.

```
1  var twoSum = function (nums, target) {
2    const seen = new Map([[nums[0], 0]]);
3    for (let i = 1; i < nums.length; i++) {
4
5      if (seen.has(target - nums[i])) {
6        return [seen.get(target - nums[i]), i];
7      } else {
8          seen.set(nums[i], i);
10     }
11   }
12 };
13 twoSum([2,3,5,6],9)
```

The `Map` object holds key-value pairs and remembers the original insertion order of the keys. Any value (both objects and primitive values) may be used as either a key or a value.

The `has()` method returns a boolean indicating whether an element with the specified key exists or not.

The `get()` method returns a specified element from a `Map` object. If the value that is associated to the provided key is an object, then you will get a reference to that object and any change made to that object will effectively modify it inside the `Map` object.

The `set()` method adds or updates an element with a specified key and a value to a `Map` object.

LeetCode

2 `twoSum([2,3,5,6],9)`

nums	
target	9
seen	undefined

array: 2 3 5 6

3 `const seen = new Map([[nums[0], 0]]);`

map: 2 0

loop (from Map): i und

4 `for (let i = 1; i < nums.length; i++)`

loop (from Map): i 1

for loop: i 1

5 `if (seen.has(target - nums[i])) {`

6
```
} else {
    seen.set(nums[i], i);
```

map: 2 0 / 3 1

7 `for (let i = 1; i < nums.length; i++)`

for loop: i 2

8 `if (seen.has(target - nums[i])) {`

9
```
} else {
    seen.set(nums[i], i);
```

map: 2 0 / 3 1 / 5 2

10 `for (let i = 1; i < nums.length; i++)`

for loop: i 2

11 `if (seen.has(target - nums[i])) {`

12 `return [seen.get(target - nums[i]), i];`

RETURN VALUE: 1 3

jonchristie.io

4. Median of Two Sorted

Problem

Given two sorted arrays nums1 and nums2 of size m and n respectively, return the median of the two sorted arrays. The overall run time complexity should be O(log (m+n)).

Solution

Psuedocode:

// combine two arrays and sort them in increasing order

// use length to determine how to calculate median....

// if EVEN take average of two middle elements,

// if ODD take value of middle item by dividing length by 2 and rounding down to nearest integer

Walkthrough

1. `findMedianSortedArrays([1,3,5],[4,7,9]);`

```
nums1 = [1,3,5]
nums2 = [4,7,9]
```

2. `const sortedArray0 = nums1.concat(nums2);`

```
sortedArray0 = [ 1, 3, 5, 4, 7, 9 ]
```

`nums1.concat(nums2);`

3. `const sortedArray = sortedArray0.sort((a,b) => a - b);`

```
a = 1
b = 3
return = -2, so already in order, move to next item, '5':

a = 3
b = 5
return = -2, so already in order, move to next item, '4':

a = 5
b = 4
return = 1, + so switch positions, so '4' is before '5', and compare 4 with previous item, '3':

a = 3
b = 4
return = -1, so already in order, and 4 is currently in position, so compare current top in sortedArray, '5', with next item, '7':

a = 5
b = 7
return = -2, so already in order, move to next and last item, '9':

a = 7
b = 9
return = -2, so already in order, and we have our sortedArray:
```

```
sortedArray = [1, 3, 4, 5, 7, 9]
```

`sort((a,b) => a - b)`

4.

```
const len = sortedArray.length;
const avg = len / 2;
```

```
len = 6
avg = 3
```

5.

```
return len % 2 === 0
?
(sortedArray[avg] + sortedArray[avg - 1]) / 2
:
sortedArray[Math.floor(avg)];
```

```
since len = 6 and 6 % 2 === 0

then
answer
= (sortedArray[avg] + sortedArray[avg - 1]) / 2
= (sortedArray[3] + sortedArray[3 - 1]) / 2
= (5 + 4)/2
= 9/2
= 4.5
```

`return ternary operator`

4.5

```
function findMedianSortedArrays(nums1, nums2) {
const sortedArray0 = nums1.concat(nums2);
const sortedArray = sortedArray0.sort((a,b) => a - b);
const len = sortedArray.length;
const avg = len / 2;
return len % 2 === 0 ? (sortedArray[avg] + sortedArray[avg -
1]) / 2 : sortedArray[Math.floor(avg)];
};

findMedianSortedArrays([1,3,5],[4,7,9]);
```

jonchristie.io