Problem 2776: Convert Callback Based Function to Promise Based Function

Problem Information

Difficulty: Medium
Acceptance Rate: 0.00%
Paid Only: No

Problem Description

Write a function that accepts another function

fn

and converts the callback-based function into a promise-based function.

The function

fn

takes a callback as its first argument, along with any additional arguments

args

passed as separate inputs.

The

promisify

function returns a new function that should return a promise. The promise should resolve with the argument passed as the first parameter of the callback when the callback is invoked without error, and reject with the error when the callback is called with an error as the second argument.

The following is an example of a function that could be passed into

promisify

.

```
function sum(callback, a, b) { if (a < 0 || b < 0) {   const err = Error('a and b must be positive');
callback(undefined, err);   } else { callback(a + b);   } }
```

This is the equivalent code based on promises:

```
async function sum(a, b) { if (a < 0 || b < 0) { throw Error('a and b must be positive');   } else {
return a + b;   } }
```

Example 1:

Input:

```
fn = (callback, a, b, c) => { callback(a * b * c); } args = [1, 2, 3]
```

Output:

```
{"resolved": 6}
```

Explanation:

```
const asyncFunc = promisify(fn); asyncFunc(1, 2, 3).then(console.log); // 6
```

fn is called with a callback as the first argument and args as the rest. The promise based version of fn resolves a value of 6 when called with (1, 2, 3).

Example 2:

Input:

```
fn = (callback, a, b, c) => { callback(a * b * c, "Promise Rejected"); } args = [4, 5, 6]
```

Output:

{"rejected": "Promise Rejected"}

Explanation:

const asyncFunc = promisify(fn); asyncFunc(4, 5, 6).catch(console.log); // "Promise Rejected"

fn is called with a callback as the first argument and args as the rest. As the second argument, the callback accepts an error message, so when fn is called, the promise is rejected with a error message provided in the callback. Note that it did not matter what was passed as the first argument into the callback.

Constraints:

1 <= args.length <= 100

0 <= args[i] <= 10

4

Code Snippets

JavaScript:

```
/**
* @param {Function} fn
* @return {Function<Promise<number>>}
*/
var promisify = function(fn) {

return async function(...args) {

}
};

/**
* const asyncFunc = promisify(callback => callback(42));
* asyncFunc().then(console.log); // 42
*/
```

TypeScript:

```
type CallbackFn = (
next: (data: number, error: string) => void,
...args: number[]
) => void
type Promisified = (...args: number[]) => Promise<number>

function promisify(fn: CallbackFn): Promisified {

return async function(...args) {

};
};

/**
* const asyncFunc = promisify(callback => callback(42));
* asyncFunc().then(console.log); // 42
*/
```

Solutions

JavaScript Solution:

```
/**
* Problem: Convert Callback Based Function to Promise Based Function
* Difficulty: Medium
* Tags: general
*
* Approach: Optimized algorithm based on problem constraints
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(1) to O(n) depending on approach
*/

/**
* @param {Function} fn
* @return {Function<Promise<number>>}
*/
var promisify = function(fn) {

return async function(...args) {

}
};

/**
* const asyncFunc = promisify(callback => callback(42));
* asyncFunc().then(console.log); // 42
*/
```

TypeScript Solution:

```
/**
* Problem: Convert Callback Based Function to Promise Based Function
* Difficulty: Medium
* Tags: general
*
* Approach: Optimized algorithm based on problem constraints
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(1) to O(n) depending on approach
*/

type CallbackFn = (
next: (data: number, error: string) => void,
...args: number[]
) => void
type Promisified = (...args: number[]) => Promise<number>

function promisify(fn: CallbackFn): Promisified {

return async function(...args) {

};
};

/**
* const asyncFunc = promisify(callback => callback(42));
* asyncFunc().then(console.log); // 42
*/
```