Q1: Given an array with N+1 integers, with each int in the range from 1 to N, there is at least a number with duplicates, please find the duplicate, and try to optimize time/space complexity if possible.

Please note, the number of duplicates might be greater than 2.

**Example:**

**Input:**

[1,3,3,3,2]

**Output:** 3

/\*\*

\* @param {number[]} nums

\* @return {number}

\*/

const findDuplicated = function(nums) {

// your code here.

};

Q2: An image is represented by a 2-D array of integers, each integer representing the pixel value of the image (from 0 to 65535).

Given a coordinate (sr, sc) representing the starting pixel (row and column) of the flood fill, and a pixel value newColor, "flood fill" the image.

To perform a "flood fill", consider the starting pixel, plus any pixels connected 4-directionally to the starting pixel of the same color as the starting pixel, plus any pixels connected 4-directionally to those pixels (also with the same color as the starting pixel), and so on. Replace the color of all the pixels with the newColor.

At the end, return the modified image.

**Example 1:**

**Input:**

**image = [[1,1,1], [1,1,0], [1,0,1]]**

**sr = 1, sc = 1, newColor = 2**

**Output: [[2,2,2], [2,2,0], [2,0,1]]**

**Explanation:**

From the center of the image (with position (sr, sc) = (1, 1)), all pixels connected

by a path of the same color as the starting pixel are colored with the new color.

Note the bottom corner is not colored 2, because it is not 4-directionally connected.

to the starting pixel.

/\*\*

\* @param {number[][]} image

\* @param {number} sr

\* @param {number} sc

\* @param {number} newColor

\* @return {number[][]}

\*/

const floodFill = function(image, sr, sc, newColor) {

// your code here.

};

Q3.1: Please implement below multiply function to make it work.

multiply(2)(5); // 10

function multiply(num) {

// your code here

}

Q3.2: Similar to 3.1, but now multiply allow arbitrary number of parameters, please implement below multiply function to make it work.

multiply(2)(5); // 10

multiply(3)(4)(5); // 60

multiply(2)(3)(4)(5); // 120

function multiply(num) {

// your code here

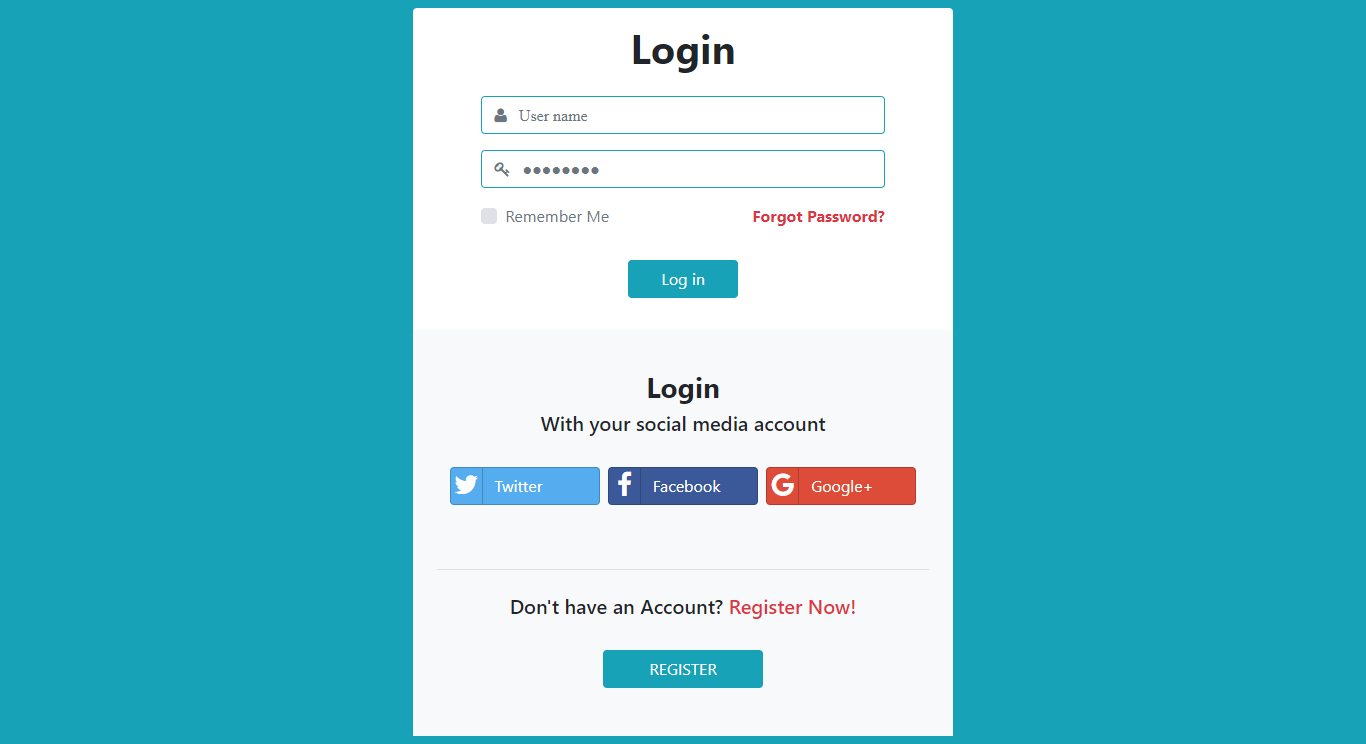
}

Q4: Q: What is event loop?

Q5: Please implement below SignIn React component as below picture shown.

Implement this component as detailed as possible, and feel free to create subcomponent if necessary.

* While clicking on “Log in”, the form content will be sent to ***./api/login*** with HTTP POST.
* While clicking on “Twitter”, “facebook”, “Google+”, user will be redirected to third-party signin page.
  + Twitter -> ***./sso/twitter***
  + Facebook -> ***./sso/facebook***
  + Google+ -> ***./sso/googleplus***
* While clicking on “REGISTER”, user will be redirected to register page ***./signin/register***
* While “Remember Me” is checked, we will store user name in locally.



Continued to next page…

class SignIn extends React.Component {

constructor(props) {

super(props);

this.state = {: ''};

this.handleChange = this.handleChange.bind(this);

this.handleSubmit = this.handleSubmit.bind(this);

}

handleChange(event) {

// put your handler logic here.

}

handleSubmit(event) {

// put your handler logic here.

}

// put more helper/handler here if necessary.

render() {

return (

<form onSubmit={this.handleSubmit}>

<h1>Login</h1>

<input type="text" value={this.state.name} onChange={this.handleChange} />

<input type="password " value={this.state.password} onChange={this.handleChange} />

<input type="submit" value="Submit" />

// add more components here

</form>

);

}

}

Q6: Please tell me about a tough software development problem you met, and how you solved it.