## Questions:

### Biography:

How old are you?

* 23

Have you used JavaScript before?

* Yes

Have you ever done any asynchronous programming?

* Not really

From a scale of 1-10 rate your understanding of how asynchronous code works in JavaScript?

* 2

### Reading the code:

***Now pass the code example to the user. Give them enough time to read and understand the code.***

Did you fully understand the functionality of the provided code?

* yes

How much time do you expect each async function in the code will use?

* Fetching from the website would take the longest (300-600)
* Then reading the file (5-10)

How much CPU/memory resource do you expect each async function in the code will use?

* Low cpu usage for each

### Visualization:

***Now pass the visualization example to the user. Give them enough time to understand the graphs.***

How long did it take you to understand the relation between pieces of the graph and code?

* 1 minute ish

By looking at the profiler, did you learn anything new about the code performance which you didn’t expect when inspecting the code?

* The read file is longer than my estimate

In terms of usefulness, give a score of 1-10 for each part of the visualization:

* 1. 8, I can see how long each part of the code takes
* 2. 6, Not really how this could be useful

In terms of intuitiveness, give a score of 1-10 for each part of the visualization:

* 1. 8, makes sense how to interpret this
* 2. 8, make sense how to interpret this

What other pieces of info do you think would have been useful in the visualization graph?

Finally, from a scale of 1-10 how would you overall rate the profiler?

* 7

# Code:

Provide this piece of code to the user and make sure they understand the whole code.

Code.js

This code is written by ChatGPT 3.5

| const fs = require('fs').promises; const fetch = require('node-fetch');  const API\_KEY = 'your\_api\_key'; *// Replace with your actual API key* const CITIES = ['New York', 'London', 'Tokyo']; *// List of cities to fetch weather for* const FILE\_PATH = 'example.txt'; *// Path to the file to read*  async function fetchWeather(city) {  try {  const response = await fetch(`http://api.openweathermap.org/data/2.5/weather?q=${city}&appid=${API\_KEY}`);  const data = await response.json();  return { city, weather: data.weather[0].description };  } catch (error) {  console.error(`Error fetching weather for ${city}:`, error);  throw error;  } }  async function readFile(filePath) {  try {  const data = await fs.readFile(filePath, 'utf8');  return data;  } catch (error) {  console.error('Error reading file:', error);  throw error;  } }  async function main() {  try {  console.time('totalTime');   *// Perform IO operation (reading from file) asynchronously*  const fileData = await readFile(FILE\_PATH);  console.log('File contents:', fileData);   *// Perform multiple API requests concurrently using Promise.all*  const weatherPromises = CITIES.map(city => fetchWeather(city));  const weatherData = await Promise.all(weatherPromises);   *// Log weather data for each city*  weatherData.forEach(data => {  console.log(`Current weather in ${data.city}: ${data.weather}`);  });   console.timeEnd('totalTime');  } catch (error) {  console.error('An error occurred:', error);  } }  main(); |
| --- |

## Estimation:

**Reading from file (IO operation):**

Time Estimate: Let's say the file is small, around a few kilobytes, and the disk speed is moderate. Reading such a file might take around 5-10 milliseconds.

Resource Estimate: Minimal CPU and memory usage, mainly IO-bound.

**API requests (fetching weather data):**

Time Estimate: Each API request to OpenWeatherMap typically takes around 100-200 milliseconds, depending on network latency and the response time of the API server. Since we're making multiple requests concurrently, the total time would be determined by the longest request (which is typically still within the range of 100-200 milliseconds).

Resource Estimate: Each API request consumes network bandwidth and CPU for processing the response. However, since we're using asynchronous operations, CPU usage is relatively low during waiting periods.