Assignment 3 - Asynchronous Execution Order in Node.js

Q1: Predict the order of asynchronous operations in the given Node.js script

```
const fs = require("fs");

console.log("🔵 1. Start of script");

// Timer: setTimeout

setTimeout(() => {

  console.log("🟢 4. setTimeout 0ms");

  process.nextTick(() => {

    console.log("🟡 5. nextTick inside setTimeout");

  });

  Promise.resolve().then(() => {

    console.log("🟡 6. Promise inside setTimeout");

  });

}, 0);

// Timer: setInterval (runs repeatedly, shown once here)

const interval = setInterval(() => {

  console.log("🟢 7. setInterval");

  clearInterval(interval); // only show once for demo

}, 0);

// I/O Async (will go into the poll phase)

fs.readFile(__filename, () => {

  console.log("🟣 10. fs.readFile (I/O)");

  setImmediate(() => {

    console.log("🟣 11. setImmediate inside fs.readFile");

  });

  setTimeout(() => {
    console.log("🟣 12. setTimeout inside fs.readFile");
  }, 0);
});
// Custom async-style callback
function customAsyncCallback(cb) {
  setTimeout(() => {
    cb("🔴 13. Custom callback after async work");
  }, 10);
}
customAsyncCallback((message) => {
  console.log(message);
});
console.log("🔵 14. End of script");
// Microtask: process.nextTick
process.nextTick(() => {
  console.log("🟡 2. process.nextTick");
});
// Microtask: Promise
Promise.resolve().then(() => {
  console.log("🟡 3. Promise.then");
});
// Check phase: setImmediate
setImmediate(() => {
  console.log("🟢 8. setImmediate");
  process.nextTick(() => {
    console.log("🟡 9. nextTick inside setImmediate");
  });
```

```
});
```

→

Execution Order:

1. Start of script - First synchronous operation, executed immediately.

14 End of script - Last synchronous operation, executed right after the first.

2 process.nextTick - Executes before Promises and any asynchronous callbacks due to its higher priority in the Microtask queue.

3 Promise.then - Executes after process.nextTick(), as Promises are also microtasks but with lower priority.

4 setTimeout 0ms - Executes in the Timers phase after the first event loop cycle.

5 nextTick inside setTimeout - Executes immediately after the setTimeout() callback, as process.nextTick() has higher priority than other asynchronous operations.

6 Promise inside setTimeout - Executes right after process.nextTick() in the Microtask queue.

7 setInterval - Executes in the Timers phase after setTimeout() since it is also a timer, and it runs only once due to clearInterval().

8 setImmediate - Executes in the Check phase after all Timers phase callbacks are completed.

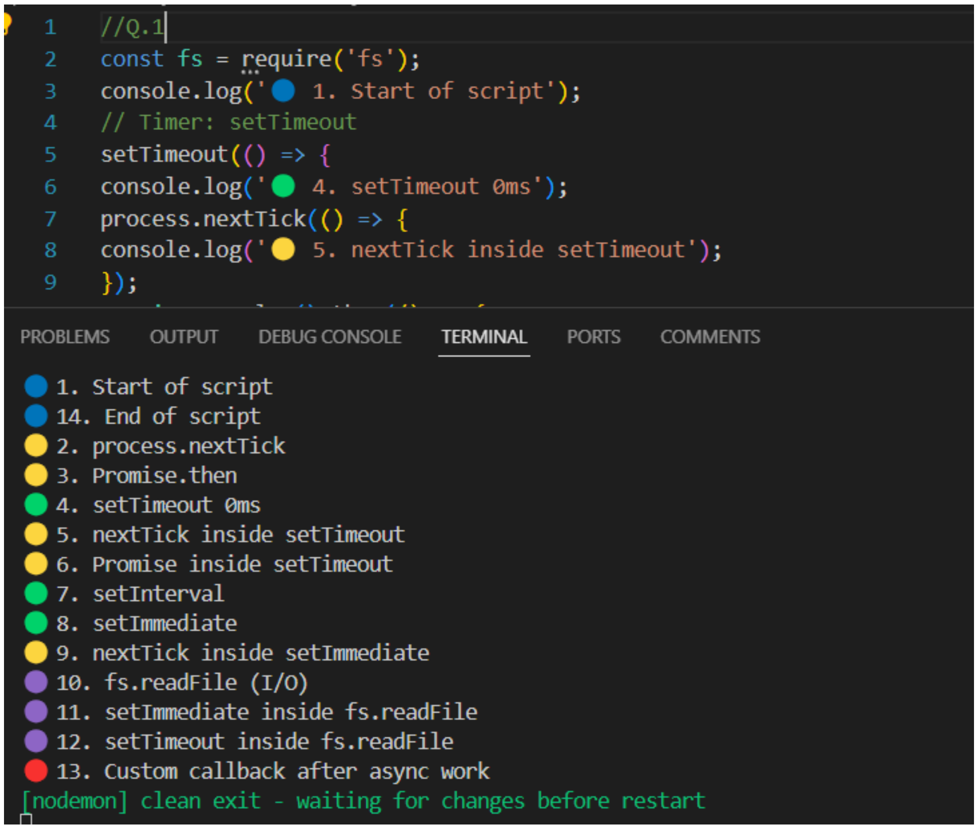
9 nextTick inside setImmediate - Executes immediately after setImmediate(), since process.nextTick() always executes before moving to the next event loop phase.

10 fs.readFile (I/O) - Executes in the Poll phase after I/O operations are completed.

11 setImmediate inside fs.readFile - Executes in the Check phase after the Poll phase has finished processing the I/O callback.

12 setTimeout inside fs.readFile - Executes in the next Timers phase, as it was scheduled inside the I/O callback.

13 Custom callback after async work - Executes in a later Timers phase after 10ms, as it was scheduled with setTimeout().



```
//Q.1
const fs = require('fs');
console.log('🔵 1. Start of script');
// Timer: setTimeout
setTimeout(() => {
console.log('🟢 4. setTimeout 0ms');
process.nextTick(() => {
console.log('🟡 5. nextTick inside setTimeout');
});
```

```
🔵 1. Start of script
🔵 14. End of script
🟡 2. process.nextTick
🟡 3. Promise.then
🟢 4. setTimeout 0ms
🟡 5. nextTick inside setTimeout
🟡 6. Promise inside setTimeout
🟢 7. setInterval
🟢 8. setImmediate
🟡 9. nextTick inside setImmediate
🟣 10. fs.readFile (I/O)
🟣 11. setImmediate inside fs.readFile
🟣 12. setTimeout inside fs.readFile
🔴 13. Custom callback after async work
[nodemon] clean exit - waiting for changes before restart
```

Q2: Predict the execution order in the second Node.js script

```
const fs = require("fs");

console.log("🔵 1. Start of script");

// Microtask queue

process.nextTick(() => {

  console.log("🟡 2. process.nextTick (microtask)");

});

// setImmediate

setImmediate(() => {

  console.log("🟢 10. setImmediate");

});

Promise.resolve().then(() => {

  console.log("🟡 3. Promise.then (microtask)");
});
setTimeout(() => {
  console.log("🟢 6. setTimeout 0ms");
  // Nested async inside timeout
  async function nestedAsync() {
    console.log("🔵 7. Nested async before await (in setTimeout)");
    await Promise.resolve();
    console.log("🟡 8. Nested async after await (in setTimeout)");
  }
  nestedAsync();
  // Recursive timer (like interval)
  let count = 0;
  function recursiveTimeout() {
    if (count < 2) {
      console.log(`🟢 9. Recursive timeout ${count + 1}`);
      count++;
      setTimeout(recursiveTimeout, 0);
    }
  }
  recursiveTimeout();
}, 0);
// Async/await
async function asyncExample() {
  console.log("🔵 4. Inside async function (before await)");
  await Promise.resolve();
  console.log("🟡 5. After await inside async function (microtask)");
}
```

```
asyncExample();

// I/O operation

fs.readFile(__filename, () => {

  console.log("🟣 11. fs.readFile callback");

  process.nextTick(() => {

    console.log("🟡 12. nextTick inside fs.readFile");

  });

  Promise.resolve().then(() => {

    console.log("🟡 13. Promise inside fs.readFile");

  });

  setImmediate(() => {

    console.log("🟣 14. setImmediate inside fs.readFile");

  });

});

console.log("🔵 15. End of script");
```

→

Execution Order:
1. Start of script - First synchronous operation, executed immediately.

4.Inside async function (before await) - The async function starts executing, but await pauses execution at this point.

15. End of script - Last synchronous statement in the script, executed before any asynchronous tasks.

2. process.nextTick (microtask) - Executes before Promises and asynchronous callbacks due to its higher priority in the Microtask queue.

3. Promise.then (microtask) - Executes after process.nextTick(), as Promises are microtasks but with lower priority.

5. After await inside async function (microtask) - The await inside asyncExample() resolves, and the remaining code executes in the microtask queue.

6. setTimeout 0ms - Executes in the Timers phase after the first event loop cycle.

7. Nested async before await (in setTimeout) - The nested async function runs immediately before the await.

9. Recursive timeout 1 - The first recursive setTimeout() executes in the next Timers phase.

8. Nested async after await (in setTimeout) - The await inside the nested async function resolves and executes its remaining code in the microtask queue.

10. setImmediate - Executes in the Check phase after the Timers phase completes.

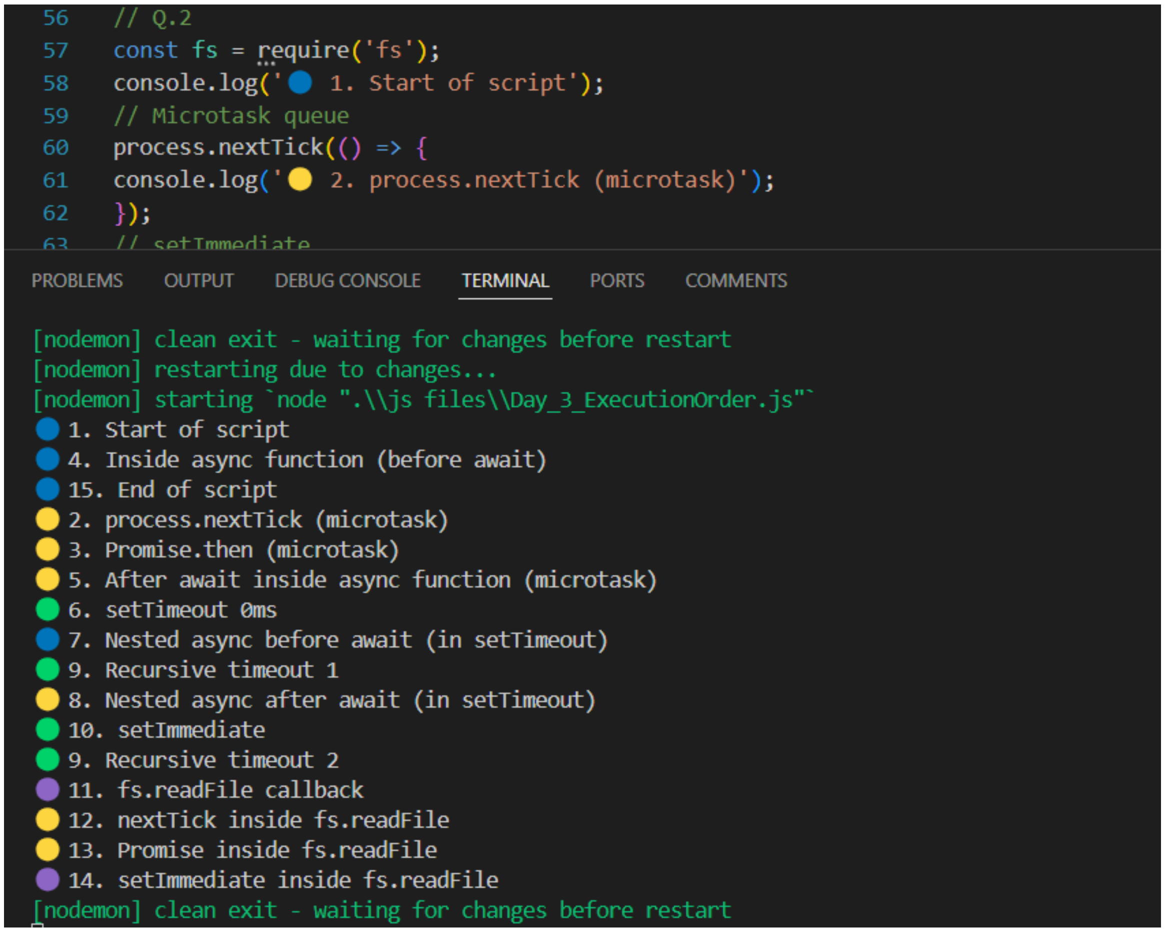
11. fs.readFile callback - The I/O operation completes and executes in the Poll phase.

12. nextTick inside fs.readFile - Executes immediately after the I/O callback, as process.nextTick() has the highest priority.

13. Promise inside fs.readFile - The Promise inside the I/O callback executes next, as microtasks run before moving to the next phase.

14. setImmediate inside fs.readFile - Executes in the Check phase after the Poll phase has finished.

9. Recursive timeout 2 - The second recursive setTimeout() executes in the next Timers phase.



```
// Q.2
const fs = require('fs');
console.log('🔵 1. Start of script');
// Microtask queue
process.nextTick(() => {
console.log('🟡 2. process.nextTick (microtask)');
});
// setImmediate
```

```
[nodemon] clean exit - waiting for changes before restart
[nodemon] restarting due to changes...
[nodemon] starting `node ".\\js files\\Day_3_ExecutionOrder.js"`
🔵 1. Start of script
🔵 4. Inside async function (before await)
🔵 15. End of script
🟡 2. process.nextTick (microtask)
🟡 3. Promise.then (microtask)
🟡 5. After await inside async function (microtask)
🟢 6. setTimeout 0ms
🔵 7. Nested async before await (in setTimeout)
🟢 9. Recursive timeout 1
🟡 8. Nested async after await (in setTimeout)
🟢 10. setImmediate
🟢 9. Recursive timeout 2
🟣 11. fs.readFile callback
🟡 12. nextTick inside fs.readFile
🟡 13. Promise inside fs.readFile
🟣 14. setImmediate inside fs.readFile
[nodemon] clean exit - waiting for changes before restart
```