1️⃣ IIFE Closure Fix

We wrap the `item` in an Immediately Invoked Function Expression so each scheduled callback closes over its own copy.

Js　　Copy

```js
(function () {
  let array = ["one", "two", "three", "four", "five"];
  let max = 3;
  let started = 0;

  function recursive() {
    while (array.length > 0 && started < max) {
      started++;
      var item = array.shift();

      (function(current) {
        setTimeout(() => {
          console.log(current);
          started--;
          recursive();
        }, 1000);
      })(item);

    }
  }

  recursive();
})();
```

Why it works:

- Each loop iteration calls the IIFE with the current `item`.
- `current` is a new variable in a new scope, so the arrow function keeps the right value.

2️⃣ Pass Value as Argument to `setTimeout`

We use `setTimeout`'s extra arguments feature to pass `item` directly to the callback.

Js　　Copy

```js
(function () {
  let array = ["one", "two", "three", "four", "five"];
```

```
    let max = 3;
    let started = 0;

    function recursive() {
      while (array.length > 0 && started < max) {
        started++;
        var item = array.shift();

        setTimeout((val) => {
          console.log(val);
          started--;
          recursive();
        }, 1000, item);

      }
    }

    recursive();
  })();
```

Why it works:

- `setTimeout` calls the callback with `item` as `val`.
- Each scheduled call gets its own copy of the value, no shared `var` problem.