Without Debouncing

- `fetchResults` function will be called on each input in the search box, which hinders performance.

```
function fetchResults(event) {
    // Simulate an API request
    console.log(`Fetching results for ${event.target.value}`);
    // document.getElementById('results').innerText = `Results for ${event.target.value
}

document.getElementById('search').addEventListener('input', fetchResults);
```

Debouncing

Debouncing is a technique in programming that helps to avoid calling functions frequently when events (keystrokes, mouse movements, or window resizing) occur in rapid succession.

- By debouncing, the performance and efficiency of your application are improved.
- Debouncing is mostly used while implementing search features.
- A common use case for debouncing is when dealing with user input in a search box. You might want to wait until the user has finished typing before sending an API request to get search results.

```
<input type="text" id="search" placeholder="Search...">
<div id="results"></div>

<script>
function debounce(func, delay) {
    let debounceTimer;
    return function (...args) { // Capture arguments for later use
        clearTimeout(debounceTimer);
        debounceTimer = setTimeout(() => func(...args), delay);
    };
}

function fetchResults(event) {
    // Simulate an API request
    console.log(`Fetching results for ${event.target.value}`);
    document.getElementById('results').innerText = `Results for ${event.target.value}`;
}

const debouncedFetchResults = debounce(fetchResults, 1000); // Store the debounced func

document.getElementById('search').addEventListener('input', debouncedFetchResults);
</script>
```

- With debouncing, `debouncedFetchResults` function will be called after every 300 milliseconds. Even if we input multiple times, the function will not be called before the timer completes. If the user inputs again before the timer finishes, the previous timer will be cleared.

References:

- YouTube Video
- GeeksforGeeks Article

Throttling

- Throttling is a technique in which, no matter how many times the user fires the event, the attached function will be executed only once in a given time interval.
- Throttling ensures that the function executes at regular intervals.

```
function throttle(func, delay) {
    let lastCall = 0;
    return function (...args) {
        const now = new Date().getTime();
        if (now - lastCall < delay) {
            return;
        }
        lastCall = now;
        func(...args);
    };
}

function logMessage(message) {
    console.log(message);
}

const throttledLogMessage = throttle(logMessage, 1000);

// Logs 'Hello'
throttledLogMessage('Hello');

// Doesn't log anything
throttledLogMessage('World');

// Logs 'Delayed' after 2 seconds
setTimeout(() => throttledLogMessage('Delayed'), 2000);
```

References:

- GeeksforGeeks Article
- YouTube Video