

## Lesson 04 Demo 05

### Implementing the Quick Sort Algorithm

**Objective:** To sort data using the quick sort algorithm in JavaScript for optimizing tasks like processing search results or product listings

**Tools required:** Visual Studio Code and Node.js

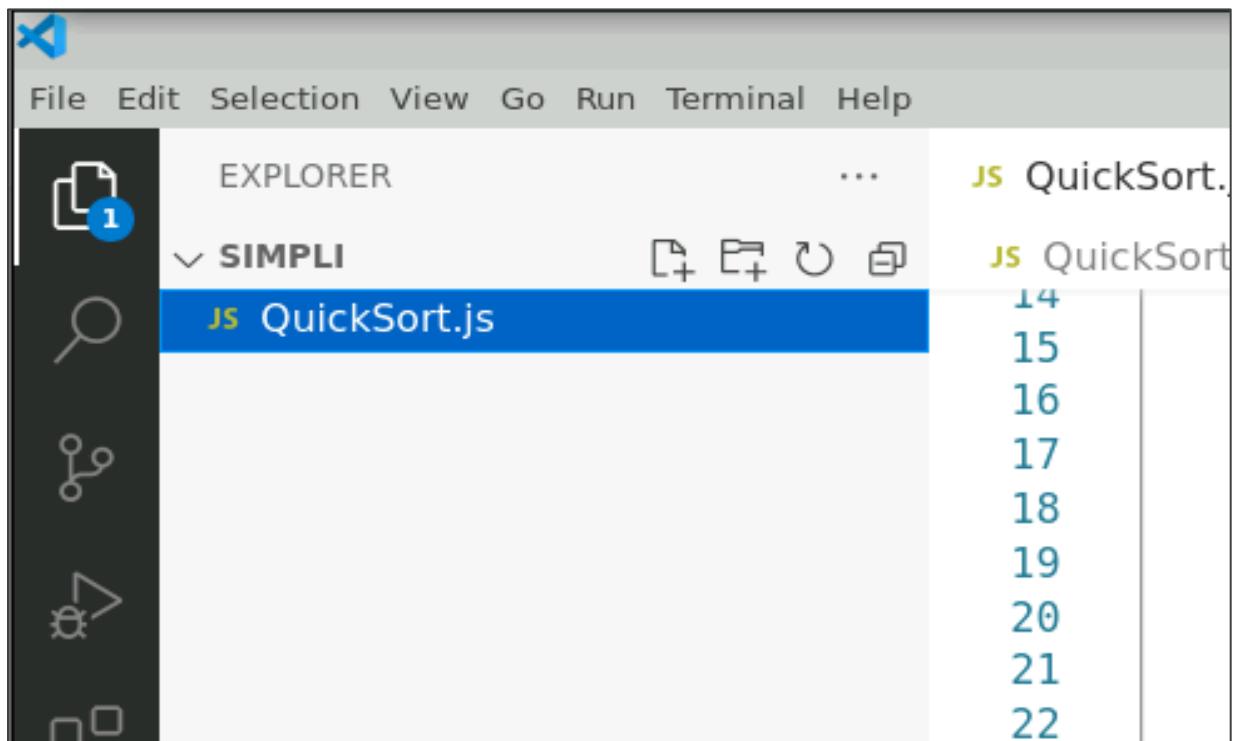
**Prerequisites:** A basic understanding of arrays and loops in JavaScript

Steps to be followed:

1. Create a JavaScript file and execute it

#### Step 1: Create a JavaScript file and execute it

1.1 Open the Visual Studio Code editor and create a JavaScript file named **QuickSort.js**



1.2 Add the following code to the file:

```
function quickSort(array, low, high) {
    if (low < high) {
        const pivotIndex = partition(array, low, high);
        // Recursive call for the left part of the array
        quickSort(array, low, pivotIndex - 1);
        // Recursive call for the right part of the array
        quickSort(array, pivotIndex + 1, high);
    }
}
// Time Complexity: Average and Best - O(n log n), Worst - O(n^2)
// Space Complexity: O(log n)

function partition(array, low, high) {
    const pivot = array[high];
    let i = low - 1;

    for (let j = low; j < high; j++) {
        if (array[j] < pivot) {
            i++;
            [array[i], array[j]] = [array[j], array[i]]; // Swap elements
        }
    }

    [array[i + 1], array[high]] = [array[high], array[i + 1]]; // Swap pivot
    return i + 1; // Return the pivot index
}

const unsortedArray = [5, 2, 4, 1, 3];
console.time("quickSort");
quickSort(unsortedArray, 0, unsortedArray.length - 1);
console.timeEnd("quickSort"); // Measures and logs the time taken for sorting
console.log(unsortedArray);
```

```

1  function quickSort(array, low, high) {
2    if (low < high) {
3      const pivotIndex = partition(array, low, high);
4      // Recursive call for the left part of the array
5      quickSort(array, low, pivotIndex - 1);
6      // Recursive call for the right part of the array
7      quickSort(array, pivotIndex + 1, high);
8    }
9  }
10 // Time Complexity: Average and Best - O(n log n), Worst - O(n^2)
11 // Space Complexity: O(log n)
12
13 function partition(array, low, high) {
14   const pivot = array[high];
15   let i = low - 1;
16
17   for (let j = low; j < high; j++) {
18     if (array[j] < pivot) {
19       i++;
20       [array[i], array[j]] = [array[j], array[i]]; // Swap elements
21     }
22   }
23
24   [array[i + 1], array[high]] = [array[high], array[i + 1]]; // Swap pivot
25   return i + 1; // Return the pivot index
26
27

```

```

27
28 const unsortedArray = [5, 2, 4, 1, 3];
29 console.time("quickSort");
30 quickSort(unsortedArray, 0, unsortedArray.length - 1);
31 console.timeEnd("quickSort"); // Measures and logs the time taken for sorting
32 console.log(unsortedArray);

```

PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS

+ × ⋮ ^ X

1.3 Press **Ctrl + S** to save the file and then execute it in the **TERMINAL** using the following commands:

```
ls  
node QuickSort.js
```

The screenshot shows a terminal window with several tabs at the top: PROBLEMS, OUTPUT, DEBUG CONSOLE, and TERMINAL. The TERMINAL tab is selected and highlighted with an orange border. In the terminal, the user has run the command `node QuickSort.js`. The output shows the quicksort function being called with an array [1, 2, 3, 4, 5], and the sorted result [1, 2, 3, 4, 5] is printed to the console. The terminal also displays the user's name and IP address.

```
6 |     // Recursive call for the right part of the array
7 |     quickSort(array, pivotIndex + 1, high);
8 |
9 |
10| // Time Complexity: Average and Best - O(n log n), Worst - O(n^2)
11| // Space Complexity: O(log n)
PROBLEMS   OUTPUT   DEBUG CONSOLE   TERMINAL
priyanshurajsim@ip-172-31-40-74:~/Downloads/Simpli$ ls
QuickSort.js
priyanshurajsim@ip-172-31-40-74:~/Downloads/Simpli$ node QuickSort.js
quicksort: 0.127ms
[ 1, 2, 3, 4, 5 ]
priyanshurajsim@ip-172-31-40-74:~/Downloads/Simpli$
```

By following these steps, you have successfully used the quick sort algorithm in JavaScript to organize data such as search results or product listings efficiently, and learned that it has a worst-case time complexity of  $O(n^2)$  and space complexity of  $O(\log n)$ .