Lesson 04 Demo 10

Implementing the Binary Search Algorithm

Objective: To demonstrate the binary search algorithm and explain its time and space complexity using JavaScript

Tools required: Visual Studio Code and Node.js

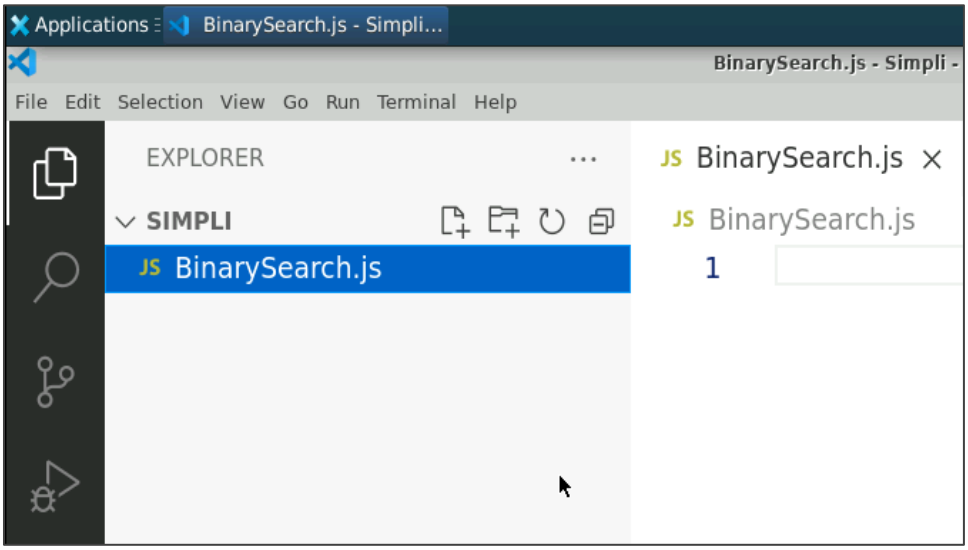
Prerequisites: Basic understanding of arrays and loops in JavaScript

Steps to be followed:

1. Create and execute the JS file

Step 1: Create and execute the JS file

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



1.2 Write the code given below in the **BinarySearch.js** file:

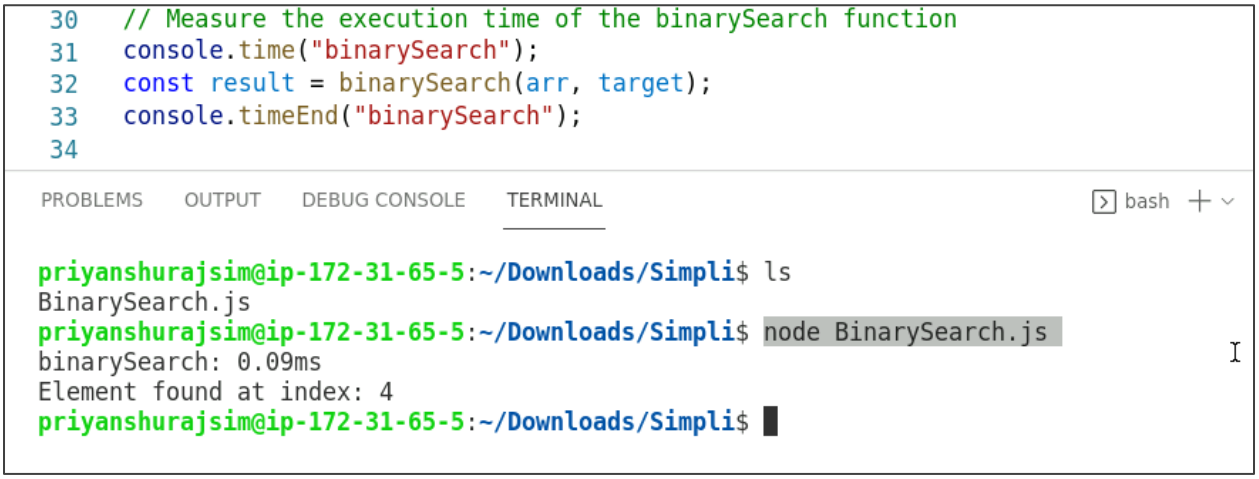
```
// Binary search function to find the index of the target in a sorted array
function binarySearch(arr, target) {
  let left = 0;
  let right = arr.length - 1;

  // Time Complexity: O(log n)
  // The while loop divides the search space in half with each iteration,
  // leading to a logarithmic time complexity when the array is sorted.
  while (left <= right) {
    const mid = Math.floor((left + right) / 2);

    if (arr[mid] === target) {
      return mid; // Element found
    }
    if (arr[mid] < target) {
      left = mid + 1; // Search in the right half
    } else {
      right = mid - 1; // Search in the left half
    }
  }
  return -1; // Element not found
}

// Example usage
const arr = [1, 3, 5, 7, 9, 11, 13, 15];
const target = 9;

// Measure the execution time of the binarySearch function
console.time("binarySearch");
const result = binarySearch(arr, target);
console.timeEnd("binarySearch");

if (result !== -1) {
  console.log(`Element found at index: ${result}`);
} else {
  console.log('Element not found in the array');
}
```

JS BinarySearch.js > ...

```javascript
// Binary search function to find the index of the target in a sorted array
function binarySearch(arr, target) {
    let left = 0;
    let right = arr.length - 1;

    // Time Complexity: O(log n)
    // The while loop divides the search space in half with each iteration,
    // leading to a logarithmic time complexity when the array is sorted.
    while (left <= right) {
        const mid = Math.floor((left + right) / 2);

        if (arr[mid] === target) {
            return mid; // Element found
        }

        if (arr[mid] < target) {
            left = mid + 1; // Search in the right half
        } else {
            right = mid - 1; // Search in the left half
        }
    }

    return -1; // Element not found
}
```

```javascript
// Example usage
const arr = [1, 3, 5, 7, 9, 11, 13, 15];
const target = 9;

// Measure the execution time of the binarySearch function
console.time("binarySearch");
const result = binarySearch(arr, target);
console.timeEnd("binarySearch");

if (result !== -1) {
    console.log(`Element found at index: ${result}`);
} else {
    console.log('Element not found in the array');
}
```

1.3 Save the file and execute it in the terminal using the command given below:
node BinarySearch.js



In our example, we used the binary search algorithm in JavaScript to find the items in an array. It has a time complexity of O(log n) and a space complexity of O(1).

By following these steps, you have successfully implemented and executed a binary search in JavaScript, including measuring its execution time.