.

Lesson 04 Demo 11

Implementing the Jump Search Algorithm

Objective: To use jump search in JavaScript for quickly finding values in sorted data like library indexes

Tools required: Visual Studio Code and Node.js

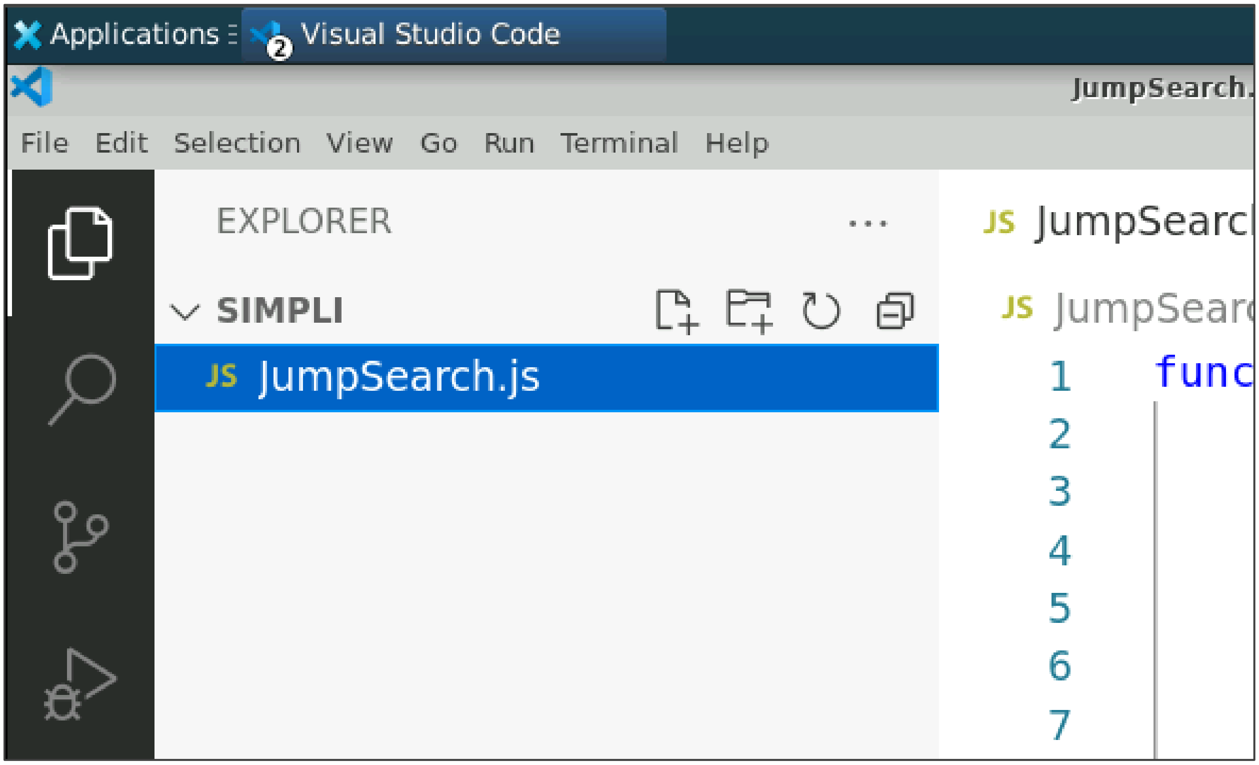
Prerequisites: A basic understanding of arrays and loops in JavaScript

Steps to be followed:

1. Create a JavaScript file and execute it

Step 1: Create a JavaScript file and execute it

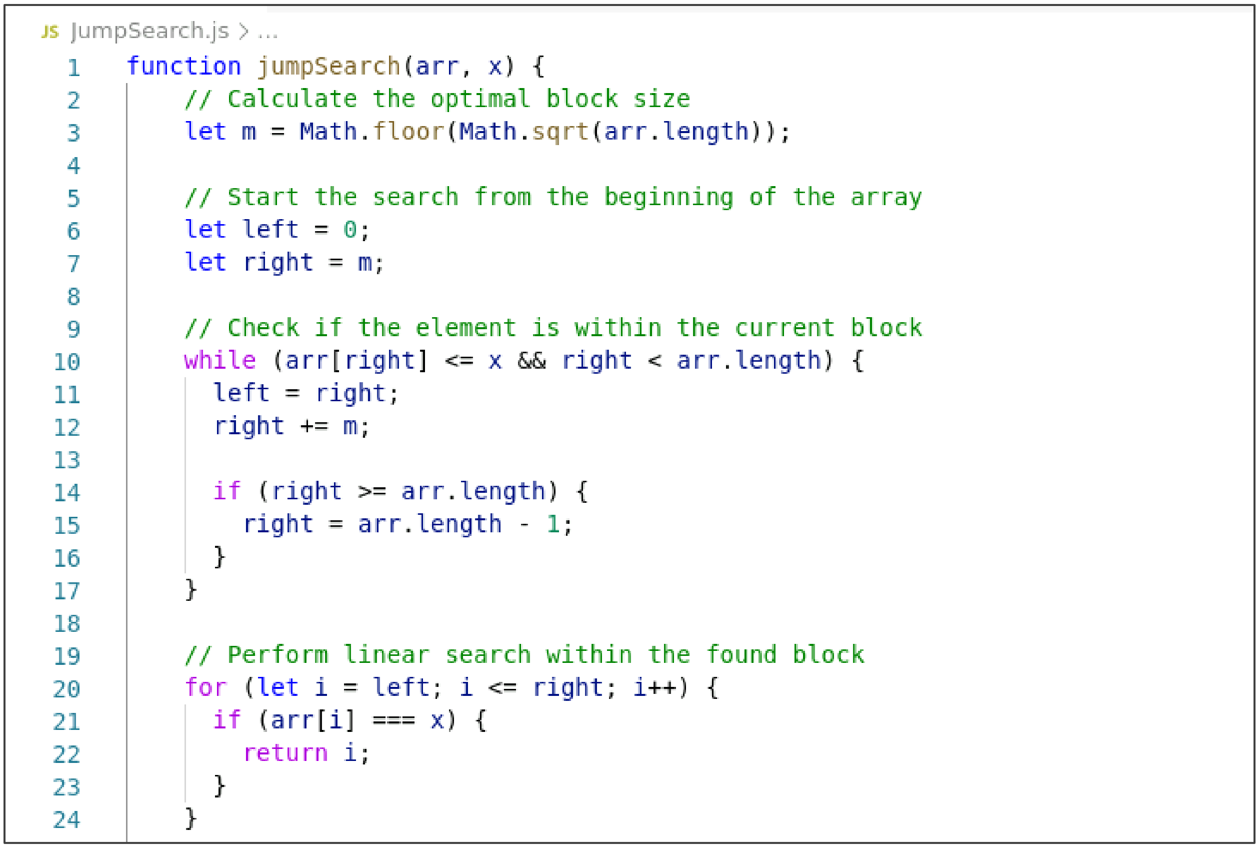
1.1 Open the Visual Studio Code editor and create a JavaScript file named **JumpSearch.js**



.

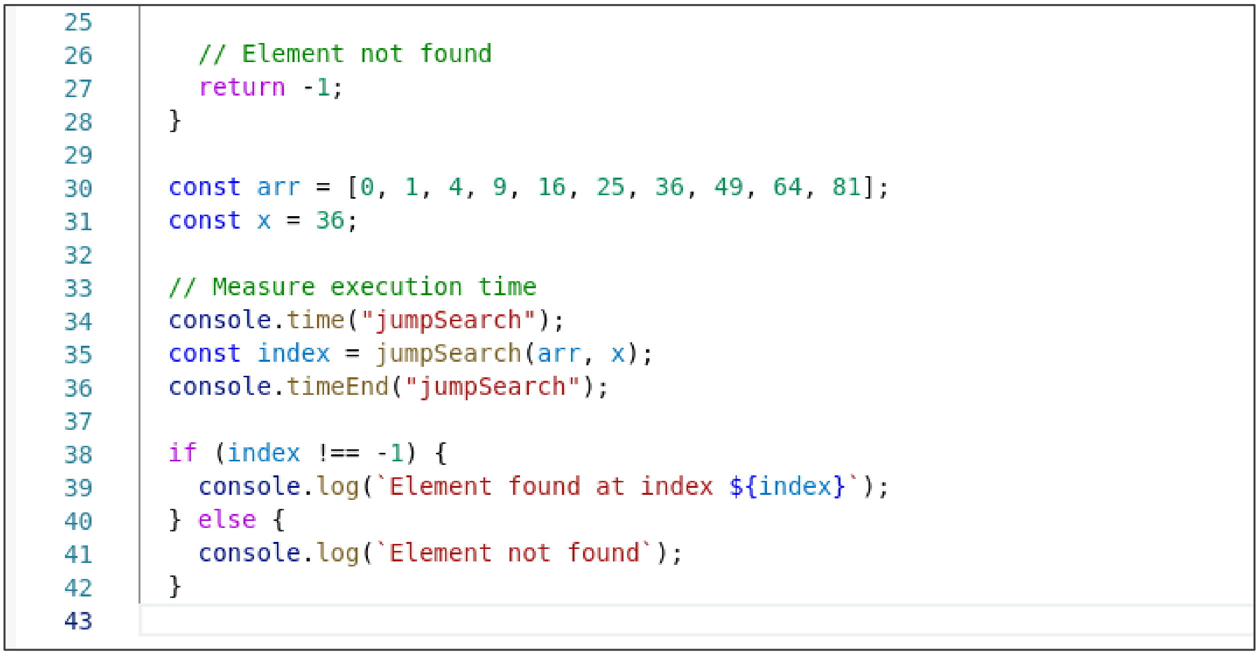
1.2 Add the following code to the file:

```
function jumpSearch(arr, x) {
  // Calculate the optimal block size
  let m = Math.floor(Math.sqrt(arr.length));

  // Start the search from the beginning of the array
  let left = 0;
  let right = m;

  // Check if the element is within the current block
  while (arr[right] <= x && right < arr.length) {
   left = right;
   right += m;
   if (right >= arr.length) {
    right = arr.length - 1;
   }
  }

  // Perform linear search within the found block
  for (let i = left; i <= right; i++) {
   if (arr[i] === x) {
    return i;
   }
  }

  // Element not found
  return -1;
 }

 const arr = [0, 1, 4, 9, 16, 25, 36, 49, 64, 81];
 const x = 36;

 // Measure execution time
 console.time("jumpSearch");
 const index = jumpSearch(arr, x);
 console.timeEnd("jumpSearch");

 if (index !== -1) {
  console.log(`Element found at index ${index}`);
 } else {
  console.log(`Element not found`)
}
```



JS JumpSearch.js > ...

```javascript
function jumpSearch(arr, x) {
    // Calculate the optimal block size
    let m = Math.floor(Math.sqrt(arr.length));

    // Start the search from the beginning of the array
    let left = 0;
    let right = m;

    // Check if the element is within the current block
    while (arr[right] <= x && right < arr.length) {
      left = right;
      right += m;

      if (right >= arr.length) {
        right = arr.length - 1;
      }
    }

    // Perform linear search within the found block
    for (let i = left; i <= right; i++) {
      if (arr[i] === x) {
        return i;
      }
    }
```



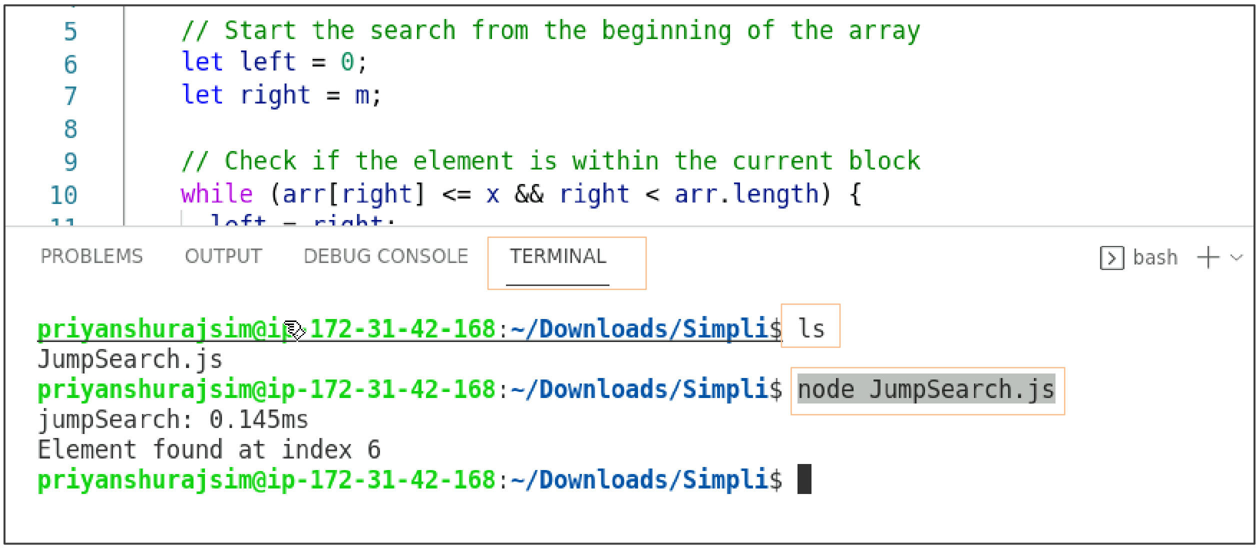
```javascript

  // Element not found
  return -1;
}

const arr = [0, 1, 4, 9, 16, 25, 36, 49, 64, 81];
const x = 36;

// Measure execution time
console.time("jumpSearch");
const index = jumpSearch(arr, x);
console.timeEnd("jumpSearch");

if (index !== -1) {
  console.log(`Element found at index ${index}`);
} else {
  console.log(`Element not found`);
}

```

.

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

ls

node JumpSearch.js



```
    // Start the search from the beginning of the array
    let left = 0;
    let right = m;

    // Check if the element is within the current block
    while (arr[right] <= x && right < arr.length) {
      left = right;
```

```
priyanshurajsim@ip-172-31-42-168:~/Downloads/Simpli$ ls
JumpSearch.js
priyanshurajsim@ip-172-31-42-168:~/Downloads/Simpli$ node JumpSearch.js
jumpSearch: 0.145ms
Element found at index 6
priyanshurajsim@ip-172-31-42-168:~/Downloads/Simpli$ 
```

By following these steps, you have successfully implemented and executed the jump search algorithm in JavaScript, efficiently searching for items in an array while analyzing its time complexity of O(√n) and space complexity of O(1).