

Lesson 04 Demo 11

Implementing the Jump Search Algorithm

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

Tools required: Visual Studio Code and Node.js

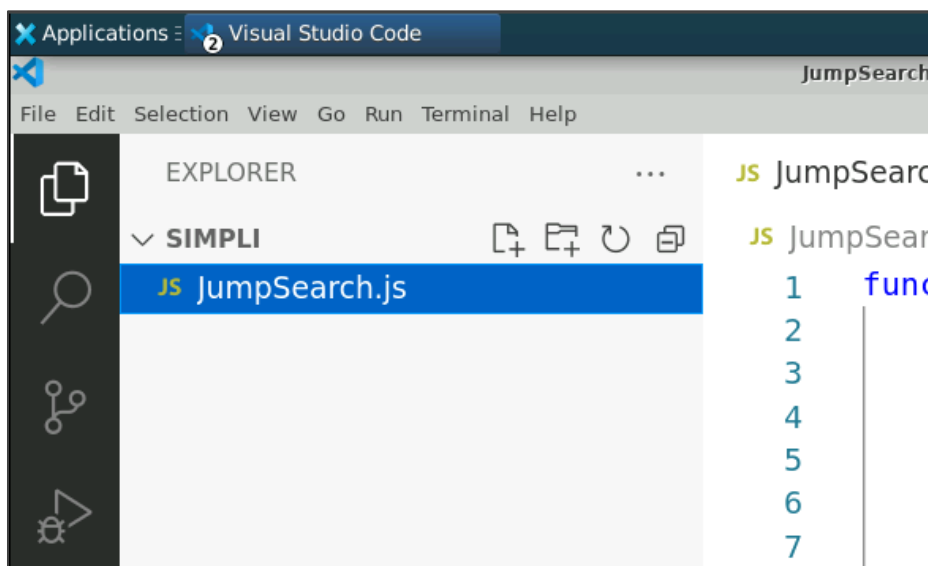
Prerequisites: Basic understanding of arrays and loops in JavaScript

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 **JumpSearch.js**



1.2 Write the code given below in the **JumpSearch.js** 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 > ...

```
1  function jumpSearch(arr, x) {
2      // Calculate the optimal block size
3      let m = Math.floor(Math.sqrt(arr.length));
4
5      // Start the search from the beginning of the array
6      let left = 0;
7      let right = m;
8
9      // Check if the element is within the current block
10     while (arr[right] <= x && right < arr.length) {
11         left = right;
12         right += m;
13
14         if (right >= arr.length) {
15             right = arr.length - 1;
16         }
17     }
18
19     // Perform linear search within the found block
20     for (let i = left; i <= right; i++) {
21         if (arr[i] === x) {
22             return i;
23         }
24     }
```

```
25
26     // Element not found
27     return -1;
28 }
29
30 const arr = [0, 1, 4, 9, 16, 25, 36, 49, 64, 81];
31 const x = 36;
32
33 // Measure execution time
34 console.time("jumpSearch");
35 const index = jumpSearch(arr, x);
36 console.timeEnd("jumpSearch");
37
38 if (index !== -1) {
39     console.log(`Element found at index ${index}`);
40 } else {
41     console.log(`Element not found`);
42 }
43
```

1.3 Save the file and execute it in the terminal using the command given below:

node JumpSearch.js

```
5 // Start the search from the beginning of the array
6 let left = 0;
7 let right = m;
8
9 // Check if the element is within the current block
10 while (arr[right] <= x && right < arr.length) {
11     left = right;
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL bash + v

```
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$
```

In our example, we used the jump search algorithm in JavaScript to find the items in an array. It has a time complexity of $O(\sqrt{n})$ and a space complexity of $O(1)$.

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