Lesson 04 Demo 06

Implementing Heap Sort Algorithm

Objective: To demonstrate the heap sort algorithm and explain its time and space complexity using JavaScript

Tools required: Visual Studio Code and Node.js

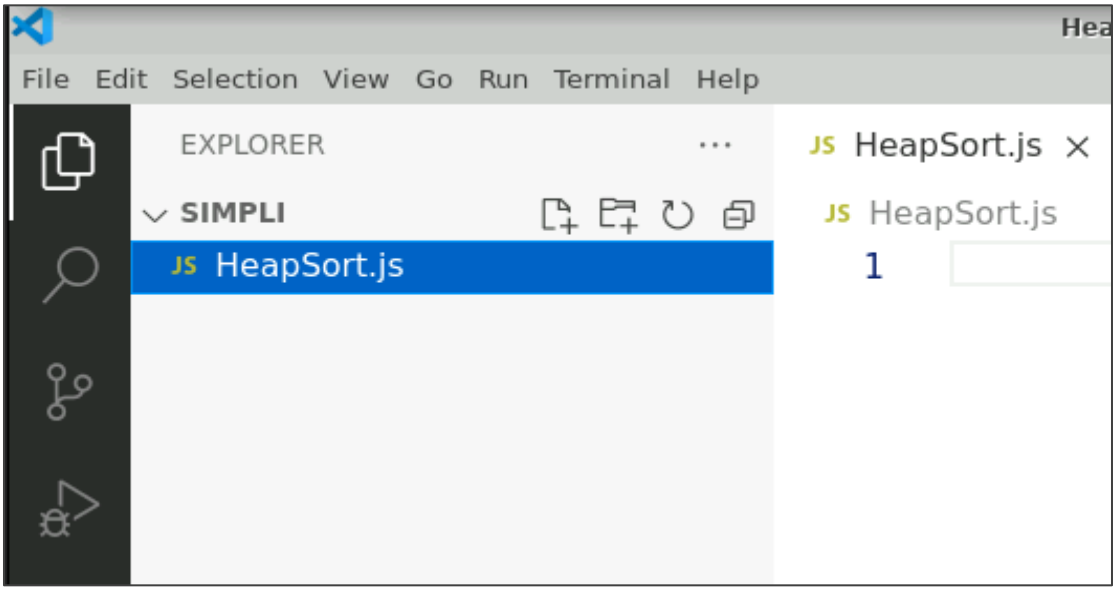
Prerequisites: Basic understanding of arrays and loops in JavaScript

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 **HeapSort.js**



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

```
// Function to swap two elements in an array
function swap(arr, i, j) {
  const temp = arr[i];
  arr[i] = arr[j];
  arr[j] = temp;
}

// Function to heapify a subtree rooted at index i
function heapify(arr, n, i) {
  const left = 2 * i + 1;
  const right = 2 * i + 2;

  let largest = i;

  // Compare with left child
  if (left < n && arr[left] > arr[largest]) {
    largest = left;
  }

  // Compare with right child
  if (right < n && arr[right] > arr[largest]) {
    largest = right;
  }

  // If the largest element is not the root, swap and heapify the affected subtree
  if (largest !== i) {
    swap(arr, i, largest);
    heapify(arr, n, largest);
  }
}

// Function to perform heap sort on the given array
function heapSort(arr) {
  const n = arr.length;

  // Build a max heap (rearrange array)
  for (let i = Math.floor(n / 2) - 1; i >= 0; i--) {
    heapify(arr, n, i);
  }
```

```
 // One by one extract elements from the heap
 for (let i = n - 1; i >= 0; i--) {
  // Move the current root to the end
  swap(arr, 0, i);

  // Call heapify on the reduced heap
  heapify(arr, i, 0);
 }
}
// Example usage
const arr = [3, 0, 2, 5, -1, 4, 1];
console.log('Unsorted array:', arr);
// Measure the execution time of heapSort
console.time('Heap Sort');
heapSort(arr);
console.timeEnd('Heap Sort');

console.log('Sorted array:', arr);
```

```
1   // Function to swap two elements in an array
2   function swap(arr, i, j) {
3       const temp = arr[i];
4       arr[i] = arr[j];
5       arr[j] = temp;
6     }
7
8     // Function to heapify a subtree rooted at index i
9     function heapify(arr, n, i) {
10      const left = 2 * i + 1;
11      const right = 2 * i + 2;
12
13      let largest = i;
14
15      // Compare with left child
16      if (left < n && arr[left] > arr[largest]) {
17        largest = left;
18      }
19
20      // Compare with right child
21      if (right < n && arr[right] > arr[largest]) {
22        largest = right;
23      }
24
25      // If the largest element is not the root, swap and heapify the affected subtree
26      if (largest !== i) {
27        swap(arr, i, largest);
28        heapify(arr, n, largest);
29      }
30    }
```

```
// Function to perform heap sort on the given array
function heapSort(arr) {
  const n = arr.length;

  // Build a max heap (rearrange array)
  for (let i = Math.floor(n / 2) - 1; i >= 0; i--) {
    heapify(arr, n, i);
  }

  // One by one extract elements from the heap
  for (let i = n - 1; i >= 0; i--) {
    // Move the current root to the end
    swap(arr, 0, i);

    // Call heapify on the reduced heap
    heapify(arr, i, 0);
  }
}

// Example usage
const arr = [3, 0, 2, 5, -1, 4, 1];
console.log('Unsorted array:', arr);

// Measure the execution time of heapSort
console.time('Heap Sort');
heapSort(arr);
console.timeEnd('Heap Sort');

console.log('Sorted array:', arr);
```

1.3 Save the file and execute it in the terminal using the following command:
node HeapSort.js

```
priyanshurajsim@ip-172-31-40-74:~/Downloads/Simpli$ ls
HeapSort.js
priyanshurajsim@ip-172-31-40-74:~/Downloads/Simpli$ node HeapSort.js
Unsorted array: [
   3, 0, 2, 5,
  -1, 4, 1
]
Heap Sort: 0.199ms
Sorted array: [
  -1, 0, 1, 2,
   3, 4, 5
]
priyanshurajsim@ip-172-31-40-74:~/Downloads/Simpli$
```

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

By following these steps, you have successfully implemented and executed the heap sort algorithm in JavaScript, including measuring its execution time.