Kathmandu University

Department of Computer Science and Engineering

Dhulikhel, Kavre

A Report on '**Lab Work 2'** [COMP 314]

Submitted by:

Chandan Kumar Mahato (31)

III-year, II semester

Submitted to:

Dr. Rajani Chulyadyo

Department of Computer Science and Engineering

Submission Date: Apr 16, 2023

Qlab1

Implementation, testing and performance measurement of sorting algorithms.

Solution:

Implementing Insertion Sort and Merge Sort algrithm using JavaScript (node.js) and writing test using (jest) a JavaScript testing framework. Plotting graph execution-time vs input-size using nodeplotlib.

Source Code:

https://github.com/ChandankMahato/DSA_Lab_6th_Sem

Scripts:

npm start
npm run test
npm run time
npm run graph

1. **Insertion Sort Implementation:**

```
function insertionSort(A) {
  for (let j = 0; j < A.length; j++) {
    let key = A[j];
    let i = j - 1;
    while (i > -1 && A[i] > key) {
      A[i + 1] = A[i];
      i = i - 1;
    }
    A[i + 1] = key;
  }
  return A;
}
module.exports = {
  insertionSort,
};
```

2. Merge Sort Implementation:

```
function mergeSort(A, p, r) {
  if (p < r) {
    const q = Math.floor((p + r) / 2);
    mergeSort(A, p, q);
    mergeSort(A, q + 1, r);
    Merge(A, p, q, r);
  }
  return A;
}

function Merge(A, p, q, r) {
  const n1 = q - p + 1;
  const n2 = r - q;
  let L = new Array(n1);
  let R = new Array(n2);
  for (let i = 0; i < n1; i++) {
    L[i] = A[p + i];
  }
  for (let j = 0; j < n2; j++) {
    R[j] = A[q + j + 1];
  }
  L[n1] = Infinity;
  R[n2] = Infinity;
  let i = 0;
  let j = 0;
  for (let k = p; k <= r; k++) {
    if (L[i] <= R[j]) {
      A[k] = L[i];
      i++;
    } else {
      A[k] = R[j];
      j++;
    }
  }
}
module.exports = {
  mergeSort
};
```

Test Implementation:

```
const { insertionSort } = require("../../Sort/insertionSort");
const { mergeSort } = require("../../Sort/mergeSort");

describe("Sort Algo Test", () => {
  it("should compare give array with sorted array from Insertion Sort", () => {
    const result = insertionSort([2, 4, 1, 3, 7, 0]);
    expect(result).toEqual([0, 1, 2, 3, 4, 7]);
  });
  it("should compare given array with sorted array from Merge Sort", () => {
    const result = mergeSort([2, 3, 4, 10, 40], 0, 5);
    expect(result).toEqual([undefined, 2, 3, 4, 10, 40]);
  });
});
```

Performance Implementation:

```
const {
  generateArray,
} = require("../../commonFiles/Performance/dataGenerator");
const { drawGraph } = require("../../commonFiles/Performance/drawGraph");
const {
  measureExecutionTime,
} = require("../../commonFiles/Performance/executionTime");
const { insertionSort } = require("../Sort/insertionSort");
const { mergeSort } = require("../Sort/mergeSort");

const size = [];
const insertionY = [];
const mergeY = [];
function collectData(length) {
  const A = generateArray(length);
  size.push(length);
  insertionY.push(measureExecutionTime(insertionSort(A)).time);
  mergeY.push(measureExecutionTime(mergeSort(A, 0, A.length - 1)).time);
}
for (let i = 10; i <= 10000000; i *= 10) {
  collectData(i);
}
drawGraph(size, insertionY, "Insertion Sort");
drawGraph(size, mergeY, "Merge Sort");
```

Observation:

Insertion Sort and Merge Sort are two common algorithms used to sort an array. Insertion Sort is a simple algorithm that iterates through each element of the array and inserts it into the correct position in a sorted subarray. Although easy to implement, it has a time complexity of O(n^2), where n is the number of elements in the array. In contrast, Merge Sort is a more efficient algorithm that divides the array into halves and recursively sorts each half before merging them back together. It has a time complexity of O(n*logn), where n is the number of elements in the array. However, Merge Sort requires additional memory for the merging process, whereas Insertion Sort sorts the array in place. Insertion Sort is useful for small or nearly sorted arrays, while Merge Sort is more efficient for larger and unsorted arrays.

Conclusion:

Insertion Sort and Merge Sort were implemented in the JavaScript. The written code was benchmarked, a graph was plotted (execution-time vs input-size), and test cases were written using jest.

Output and Screenshots

```
default@LAPTOP-FLU6LLN1 MINGW64 /a/Chandan Semester Work/6th sem/Algorithm and complexity/DSA_Lab_6th_Sem/Lab2 (master)
$ npm start

> lab2@1.0.0 start
> node index.js

Insertion Sort
Output: 1,2,3,4,5

Merge Sort
Output: ,1,2,3,4,4,4,5,6,9,10

default@LAPTOP-FLU6LLN1 MINGW64 /a/Chandan Semester Work/6th sem/Algorithm and complexity/DSA_Lab_6th_Sem/Lab2 (master)
$ npm run time

> lab2@1.0.0 time
> node Performance/time.js

Inseration Sort
Execution Time: 0.009800001978874207
Merge Sort
Execution Time: 0.027000002562999725

default@LAPTOP-FLU6LLN1 MINGW64 /a/Chandan Semester Work/6th sem/Algorithm and complexity/DSA_Lab_6th_Sem/Lab2 (master)
$ npm run test

> lab2@1.0.0 test
> jest --watchAll --verbose --coverage --detectOpenHandles

PASS test/unit/sort.test.js
  Sort Algo Test
    √ should compare give array with sorted array from Insertion Sort (6 ms)
    √ should compare given array with sorted array from Merge Sort (2 ms)

------------------|---------|----------|---------|---------|-------------------
File              | % Stmts | % Branch | % Funcs | % Lines | Uncovered Line #s
------------------|---------|----------|---------|---------|-------------------
All files         |     100 |      100 |     100 |     100 |
 insertionSort.js |     100 |      100 |     100 |     100 |
 mergeSort.js     |     100 |      100 |     100 |     100 |
------------------|---------|----------|---------|---------|-------------------
Test Suites: 1 passed, 1 total
Tests:       2 passed, 2 total
Snapshots:   0 total
Time:        0.85 s, estimated 1 s
Ran all test suites.
```

Fig 1: execution time and algorithm test case

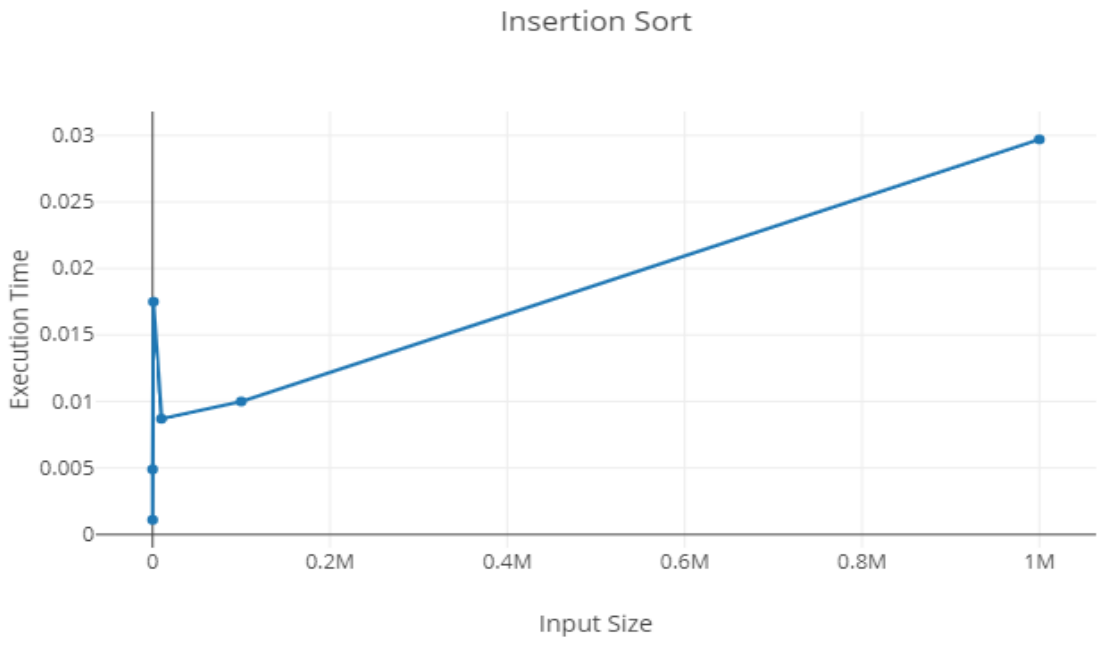

Fig 2: Insertion Sort Graph (Input Size Vs Execution Time)

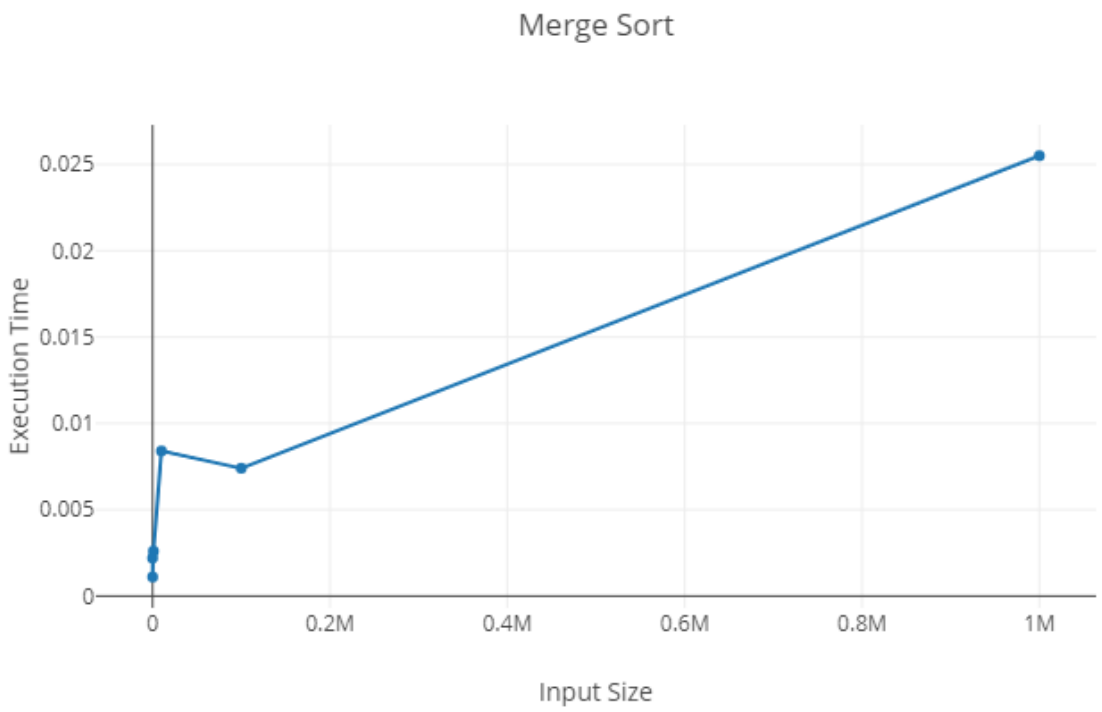

Fig 2: Merge Sort Graph (Input Size Vs Execution Time)