**TASK 1:**

a) I import the fs module to save the generated results into a text file which I name as Output.txt for no unique checked values funtion,Output1.txt for unique values checked function,

I generate N values

1st test case: 20000 values

2nd test case: 30000 values

3rd test case: 40000 values

4th test case: 50000 values

5th test case: 60000 values

* I decide to stop at 60000 values because it will run for super long with 70000 values and above for checking unique values function, in order to compare i decide to take up to 60000 values for both 2 functions

b)

**Generate random with no unique values checking function**

const fs = require('fs');

function getRandomInt(min, max) {

min = Math.ceil(min);

max = Math.floor(max);

return Math.floor(Math.random() \* (max - min)) + min;

}

let results = [];

var foo = new Array(n);

const { performance } = require('perf\_hooks');

var t0 = performance.now();

for (var i = 0; i < foo.length; i++) {

getRandomness = getRandomInt(-35000, 33600);

results.push(getRandomness);

}

let data = results;

fs.writeFile('Output.txt', data, err => {

*// In case of a error throw err.*

if (err) throw err;

});

var t1 = performance.now();

console.log(results);

console.log('Generate n values takes ' + (t1 - t0) + ' milliseconds.');

**Generate random numbers with unique values checking function**

**CODE DEMONSTRATION**

const fs = require('fs');

function getRandomInt(min, max) {

min = Math.ceil(min);

max = Math.floor(max);

return Math.floor(Math.random() \* (max - min)) + min;

}

let results = [];

var foo = new Array(n); //change this to any number of n inputs if you want to test my code

const { performance } = require('perf\_hooks');

var t0 = performance.now();

for (var i = 0; i < foo.length; i++) {

getRandomness = getRandomInt(-35000, 33600);

*while (results.includes(getRandomness)) {*

*getRandomness = getRandomInt(-35000, 33600);*

*}*

results.push(getRandomness);

}

let data = results;

fs.writeFile('Output1.txt', data, err => {

*// In case of a error throw err.*

if (err) throw err;

});

var t1 = performance.now();

console.log(results);

console.log('Generate n values takes ' + (t1 - t0) + ' milliseconds.'); //change this to any number of n inputs if you want to test my code

c) Graph of running time comparsion

**Graph#1**

**Graph#2**

d)

|  |  |  |  |
| --- | --- | --- | --- |
| Generate random numbers | Time Performance | | |
| Worst | Average | Best |
| non-unique random values | n | n | n |
| unique random values | n^2 | n^2 | n^2 |

**Task 2**

1st test case: 10 000 values

2nd test case: 15 000 values

3rd test case: 20 000 values

4th test case: 30 000 values

5th test case: 40 000 values

6th test case: 50 000 values

7th test case: 60 000 values

**Code for insertion Sort**

function insertionSort(arr) {

for (var I = 1; I < arr.length; i++) {

var currentVal = arr[i];

for (var j = I – 1; j < I && arr[j] > currentVal; j--) {

arr[j + 1] = arr[j];

}

arr[j + 1] = currentVal;

}

return arr;

}

insertionSort(results);

let data = results;

fs.writeFile(‘Output.txt’, data, err => {

*// In case of a error throw err.*

If (err) throw err;

});

**Then I put it in my original random generating numbers program in the task 1**

const fs = require(‘fs’);

function getRandomInt(min, max) {

min = Math.ceil(min);

max = Math.floor(max);

return Math.floor(Math.random() \* (max – min)) + min;

}

let results = [];

var foo = new Array(n); //change this to any number of n inputs if you want to test my code

const {

performance

} = require(‘perf\_hooks’);

var t0 = performance.now();

for (var I = 0; I < foo.length; i++) {

getRandomness = getRandomInt(-35000, 33600);

while (results.includes(getRandomness)) {

getRandomness = getRandomInt(-35000, 33600);

}

results.push(getRandomness);

}

function insertionSort(arr) {

for (var I = 1; I < arr.length; i++) {

var currentVal = arr[i];

for (var j = I – 1; j < I && arr[j] > currentVal; j--) {

arr[j + 1] = arr[j];

}

arr[j + 1] = currentVal;

}

return arr;

}

insertionSort(results);

let data = results;

fs.writeFile(‘Output.txt’, data, err => {

*// In case of a error throw err.*

If (err) throw err;

});

var t1 = performance.now();

console.log(results);

console.log(‘Generate n values takes ‘ + (t1 – t0) + ‘ milliseconds.’); //change this to any number of n inputs if you want to test my code

**Code for Quick Sort**

function pivot(arr, start = 0, end = arr.length – 1) {

const swap = (arr, I, j) => {

[arr[i], arr[j]] = [arr[j], arr[i]]

};

var pivot = arr[start];

var swapIndex = start;

for (var I = start + 1; I <= end; i++) {

if (pivot > arr[i]) {

swapIndex++;

swap(arr, swapIndex, i);

}

}

swap(arr, start, swapIndex);

*// console.log(swapIdx)*

return swapIndex;

}

function uicksort(arr, left = 0, right = arr.length – 1) {

if (left < right) {

let pivotIndex = pivot(arr, left, right)

*//on the left side*

uicksort(arr, left, pivotIndex – 1)

*//on the right side*

uicksort(arr, pivotIndex + 1, right)

}

return arr;

}

**Then I put it in my original random generating numbers program in the task 1**

const fs = require(‘fs’);

function getRandomInt(min, max) {

min = Math.ceil(min);

max = Math.floor(max);

return Math.floor(Math.random() \* (max – min)) + min;

}

let results = [];

var foo = new Array(n); //change this to any number of n inputs if you want to test my code

const {

performance

} = require(‘perf\_hooks’);

var t0 = performance.now();

for (var I = 0; I < foo.length; i++) {

getRandomness = getRandomInt(-35000, 33600);

while (results.includes(getRandomness)) {

getRandomness = getRandomInt(-35000, 33600);

}

results.push(getRandomness);

}

*// Sorting function starts*

function pivot(arr, start = 0, end = arr.length – 1) {

const swap = (arr, I, j) => {

[arr[i], arr[j]] = [arr[j], arr[i]]

};

var pivot = arr[start];

var swapIndex = start;

for (var I = start + 1; I <= end; i++) {

if (pivot > arr[i]) {

swapIndex++;

swap(arr, swapIndex, i);

}

}

swap(arr, start, swapIndex);

*// console.log(swapIdx)*

return swapIndex;

}

function uicksort(arr, left = 0, right = arr.length – 1) {

if (left < right) {

let pivotIndex = pivot(arr, left, right)

*//on the left side*

uicksort(arr, left, pivotIndex – 1)

*//on the right side*

uicksort(arr, pivotIndex + 1, right)

}

*// console.log(arr);*

return arr;

}

uicksort(results)

*// writing data*

let data = results;

fs.writeFile(‘Output1.txt’, data, err => {

*// In case of a error throw err.*

If (err) throw err;

});

var t1 = performance.now();

console.log(results);

console.log(‘Generate n values takes ‘ + (t1 – t0) + ‘ milliseconds.’); //change this to any number of n inputs if you want to test my code

e) Graph comparison for Quick Sort and Insertion Sort

**Insertion Sort Graph#1**



**Quick Sort Graph#2**

**Summary Graph**



* Quick Sort performs much better than Insertion Sort because the blue line which represents for Quick Sort is lower than Insertion Sort’s line.

g)

|  |  |  |  |
| --- | --- | --- | --- |
| Sort Algorithm | Sorting Time | | |
| Worst | Average | Best |
| Insertion Sort | O(n^2) | O(n^2) | O(n) |
| Quick Sort | O(n^2) | O(n log n) | O(n log n) |

Task 3:



Graph for Time Complexity Comparison



Specs of the Laptop running Window vs Laptop running macOS

|  |  |
| --- | --- |
| Window Laptop | Macbook |
| Processor: Intel(R) Core(TM) i5-7200 CPU @ 2.5GHz 2.7 GHz  Installed memory (RAM): 12.0 GB  System type: 64 bit Operating System, x64-based processor | Processor 2.4 GHz Quad-Core Intel Core i5  Memory 16GB 2133 MHz LPDDR3 |

I have successfully tested my code for task 1, task 2, task 3 on two laptop, one running Window, the other running macOS. The results shown above shows that the specs of computer, RAM or Operating systems do not make much difference in algorithm performance but rather the code itself. The Macbook has more RAM than the Window laptop does which makes the Macbook more powerful than the Window laptop. Although the Macbook is much powerful compared to the chosen Window laptop, it does make much difference on the algorithm performances in task 1, task 2, task 3.

**Explanation:**

An increase in hardware performance will give you a constant C times the running time of your algorithm. Meaning if you have computer A which is overall 2 times slower than computer B. Than your algorithm will be twice as fast on computer B. Twice as fast though really makes hardly no difference when you consider big input values to an algorithm though.

In big O notation that is to say you will have something like O(n) compared to CO(n) = O(cn) = O(n). The complexity of the algorithm and general running time for large values will be about the same on both Computer A and Computer B.

Analyze an algorithm's running time using something like big O notation, computer performance won't give you any kind of advantage when you are comparing an algorithm that is O(logn) compared to O(n^2).

1. Hence, the no. of CPUs in a machine significantly doesn’t affect much

the performance of the unknown algorithms (in Part 1), sorting algorithms (in Part 2) and data structures (in Part 3)

1. Having greater RAM does not help improve performance for all input sizes
2. the kind of Operating System running my programs only play a small part in maximizing the algorithm performance because it does not cause much difference in the running time when I tested my code on two different operating systems, one on Window, the other on macOS.