Zachary Christmas

CSE 2300

Term Project

11/28/2020

Part 1 (Bubble Sort):

function bubbleSort1() {

*//prompt the user for a number*

let input = prompt("Please enter a number");

*//check for input for integer type*

if (input != null || Number.isInteger(prompt)) {

*//use input to generate array of random integers*

let unsortedArray = [...Array(Number.parseInt(input))].map(() => Math.floor(Math.random() \* (Number.parseInt(input) \* 10)));

*//display unsorted array to user*

alert(`Unsorted Array: ${unsortedArray}`);

*//init variables*

let swapp = false;

*//set variable for iteration count*

let n = unsortedArray.length - 1;

*//declare sorted array variable*

let sortedArray = unsortedArray;

*//loop until swapp is switched to false*

do {

*//set swapp to false once we are out of the inner loop*

swapp = false;

*//begin inner loop*

for (let i = 0; i < n; i++) {

*//check if the first element in the selected index is greater than the next*

if (sortedArray[i] > sortedArray[i + 1]) {

*//delcare variable to store original value for selected index*

let arrayIndex = sortedArray[i];

*//set the current selected index equal to the value of the next one*

sortedArray[i] = sortedArray[i + 1];

*//place the original value that was stored in the next selected index (completes the swap)*

sortedArray[i + 1] = arrayIndex;

*//set swap to true to continue the outer loop*

swapp = true;

}

}

*//decrement iteration count*

n--;

} while (swapp);

alert(`Sorted Array: ${sortedArray}`);

}

else {

alert('User input error!');

}

}

Part 2 (Selection Sort):

function selectionSort1() {

*//prompt the user for a number*

let input = prompt("Please enter a number");

*//check for input for integer type*

if (input != null || Number.isInteger(prompt)) {

*//use input to generate array of random integers*

let unsortedArray = [...Array(Number.parseInt(input))].map(() => Math.floor(Math.random() \* (Number.parseInt(input) \* 10)));

*//display unsorted array to user*

alert(`Unsorted Array: ${unsortedArray}`);

*//define compare function for selection sort*

function compare(*a*, *b*) {

return *a* - *b*;

}

*//declare and set starting variables*

let min = 0;

let index = 0;

let temp = 0;

let sortedArray = unsortedArray;

*//outer loop iterates over array length*

for (let i = 0; i < sortedArray.length; i += 1) {

*//store index variable*

index = i;

*//assume min value of array equal to current index*

min = sortedArray[i];

*//inner loop iterates over other elements in array (not the values we have already looped over)*

for (let j = i + 1; j < sortedArray.length; j += 1) {

*//compare current index for next lowest value*

if (compare(min, sortedArray[j]) > 0) {

min = sortedArray[j];

index = j;

}

}

*//store the oringal index value*

temp = sortedArray[i];

*//update current index value to next lowest value found in the inner loop*

sortedArray[i] = min;

*//swap the stored value with the place of the next lowest value we found*

sortedArray[index] = temp;

}

alert(`Sorted Array: ${sortedArray}`);

}

else {

alert('User input error!');

}

}

Part 3 (Bubble Sort 2):

function bubbleSort2(*arraySize*, *numOfArrays*) {

console.log(`Array Size: ${*arraySize*} Number of arrays: ${*numOfArrays*}`);

let total\_time = 0;

*//prompt the user for a number*

let input = *arraySize*

*//check for input for integer type*

if (input != null || Number.isInteger(prompt)) {

for (let i = 0; i < *numOfArrays*; i++) {

*//use input to generate array of random integers*

let unsortedArray = [...Array(Number.parseInt(input))].map(() => Math.floor(Math.random() \* (Number.parseInt(input) \* 10)));

console.log(`Unsorted array #${i + 1}: ${unsortedArray}`);

*//init variables*

let swapp = false;

*//set variable for iteration count*

let n = unsortedArray.length - 1;

*//declare sorted array variable*

let sortedArray = unsortedArray;

*//init timer*

let running\_time = 0;

*//start timer*

let timerStart = performance.now();

*//loop until swapp is switched to false*

do {

*//set swapp to false once we are out of the inner loop*

swapp = false;

*//begin inner loop*

for (let i = 0; i < n; i++) {

*//check if the first element in the selected index is greater than the next*

if (sortedArray[i] > sortedArray[i + 1]) {

*//delcare variable to store original value for selected index*

let arrayIndex = sortedArray[i];

*//set the current selected index equal to the value of the next one*

sortedArray[i] = sortedArray[i + 1];

*//place the original value that was stored in the next selected index (completes the swap)*

sortedArray[i + 1] = arrayIndex;

*//set swap to true to continue the outer loop*

swapp = true;

}

}

*//decrement iteration count*

n--;

} while (swapp);

console.log(`Sorted array #${i + 1}: ${sortedArray}`);

*//end timer and add it to total*

let timerEnd = performance.now();

running\_time = timerEnd - timerStart;

total\_time += running\_time;

}

console.log(`Bubble Sorted ${*numOfArrays*} arrays containing ${*arraySize*} elements each in an average of ${total\_time/*numOfArrays*} ms`);

}

}

Part 3 Results:

|  |  |
| --- | --- |
| **Number Of Items Sorted** | **Average Running Time** |
| 500 | 0.6818299996957649 ms |
| 2500 | 18.600894998962758 ms |
| 5000 | 81.03417499980424 ms |

Part 4 (Selection Sort 2):

function selectionSort2(*arraySize*, *numOfArrays*) {

console.log(`Array Size: ${*arraySize*} Number of arrays: ${*numOfArrays*}`);

let total\_time = 0;

*//prompt the user for a number*

let input = *arraySize*;

*//check for input for integer type*

if (input != null || Number.isInteger(prompt)) {

for (let i = 0; i < *numOfArrays*; i++) {

*//use input to generate array of random integers*

let unsortedArray = [...Array(Number.parseInt(input))].map(() => Math.floor(Math.random() \* (Number.parseInt(input) \* 10)));

console.log(`Unsorted array #${i + 1}: ${unsortedArray}`);

*//define compare function for selection sort*

function compare(*a*, *b*) {

return *a* - *b*;

}

*//declare and set starting variables*

let min = 0;

let index = 0;

let temp = 0;

let sortedArray = unsortedArray;

*//init timer*

let running\_time = 0;

*//start timer*

let timerStart = performance.now();

*//outer loop iterates over array length*

for (let i = 0; i < sortedArray.length; i += 1) {

*//store index variable*

index = i;

*//assume min value of array equal to current index*

min = sortedArray[i];

*//inner loop iterates over other elements in array (not the values we have already looped over)*

for (let j = i + 1; j < sortedArray.length; j += 1) {

*//compare current index for next lowest value*

if (compare(min, sortedArray[j]) > 0) {

min = sortedArray[j];

index = j;

}

}

*//store the oringal index value*

temp = sortedArray[i];

*//update current index value to next lowest value found in the inner loop*

sortedArray[i] = min;

*//swap the stored value with the place of the next lowest value we found*

sortedArray[index] = temp;

}

*//end timer and add it to total*

let timerEnd = performance.now();

console.log(`Sorted array #${i + 1}: ${sortedArray}`);

running\_time = timerEnd - timerStart;

total\_time += running\_time;

}

console.log(`Selection Sorted ${*numOfArrays*} arrays containing ${*arraySize*} elements each in an average of ${total\_time/*numOfArrays*} ms`);

}

}

Part 4 Results:

|  |  |
| --- | --- |
| **Number Of Items Sorted** | **Average Running Time** |
| 500 | 0.28921499953139573 ms |
| 2500 | 5.522815000382252 ms |
| 5000 | 21.403805000329157 ms |

Part 5 Spreadsheet:

|  |  |  |
| --- | --- | --- |
| **Number Of Items Sorted** | **Bubble Sort Avg Time** | **Selection Sort Avg Time** |
| 500 | 0.6818299996957649 ms | 0.28921499953139573 ms |
| 2500 | 18.600894998962758 ms | 5.522815000382252 ms |
| 5000 | 81.03417499980424 ms | 21.403805000329157 ms |

Part 5 Graph

Chart, bubble chart

Description automatically generated

Part 6 (Observation Report):

Machine Specifications:

* **Processor**: 2.3 GHz Intel Core i7
* **Memory**: 16 GB 1600 MHz DDR3
* **Graphics**: Intel HD Graphics 4000 1536 MB
* **Browser**: Google Chrome - Version 87.0.4280.67
* **JS Engine**:[V8](https://opensource.google/projects/v8)

Observations:

I chose JavaScript as the language to use for this experiment for a few reasons. The primary reason is that most modern browsers support JS, and most computers have a modern browser - so anyone should be able to run this demo on their own computer or phone. Another reason is that it is easy to write code and debug directly in the console. No need to install any SDKs, runtimes, etc. The final reason is that it is a single-threaded language, so we can accurately measure the performance of these algorithms without worrying about any type of concurrency happening.

Looking at the results of the experiment, I have concluded that between the selection sort and bubble sort, the selection sort is the most performant. Knowing that both the bubble sort and selection sort algorithms have an O(n^2) complexity, the difference can be found using the knowledge of complexity analysis this course has taught us.

Both algorithms have a worst-case scenario of O(n^2), but what I found was interesting is that the bubble sort actually has a better best-case scenario of O(n). We can't only look at the best- and worst-case scenarios to determine what algorithm will perform the best, but we must also take into account the average number of comparisons between both of these different sorting algorithms.

Between the bubble sort and the selection sort, the selection sort greatly reduces the number of comparisons made between elements in a collection. This is the reason that selection sort is favored over bubble sort. When using a selection sort, the algorithm will swap *n* times at most. Bubble sort almost swaps *n\*(n-1)* times. Swap times in these algorithms have the biggest impact on memory and performance, so the less swaps, the better.

You can take my word for the performance of these two algorithms, but it can also be backed by data. For an array of 500 integers, the selection sort performed a little more than twice as fast as the bubble sort. When we scale that up x5 to 2500 integers, the selection sort performed more than three times the speed of the bubble sort. Finally at 5000 integers the selection sort performed four times faster than the bubble sort. Keeping this in mind, we can determine that as the size of the collection increases, the difference between the performance of the selection sort and bubble sort increases almost exponentially!