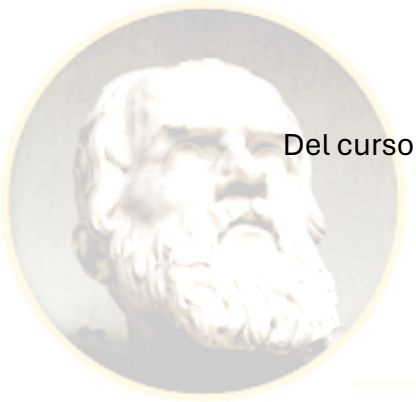
Universidad Galileo
Facultad de ingeniería de Sistemas,
informática y Ciencias de la computación
Diplomado en Desarrollo Fullstack



Tarea 7

Del curso de Introducción a la Programación

Luis Diego Orozco López
24011289
Sección T

Guatemala, 11 de junio de 2024

Tabla de Contenido

Introducción

Desde los comienzos de la computación, el problema del ordenamiento ha atraído gran cantidad de investigación debido a que cumple un rol muy importante, ya sea como un fin en sí o como parte de otros procedimientos más complejos.

Es por ello que con el tiempo se han creado distintos algoritmos de ordenación, con la meta de volver dichos procesos más rápidos y eficientes.

Los algoritmos de ordenación toman un arreglo o lista como entrada y organizan los elementos en un orden particular, ya sea numérico o alfabético. Cada uno de ellos posee características específicas, y con ventajas y desventajas sobre los demás.

A continuación, se compararán los algoritmos Bubble Sort, Selection Sort, Insertion Sort, Merge Sort y Quick Sort con el uso del lenguaje Javascript.

Algoritmos de ordenamiento

Bubble Sort

Funciona revisando cada elemento de la lista que va a ser ordenada con el siguiente, intercambiándolos de posición si están en el orden equivocado. Es necesario revisar varias veces toda la lista hasta que no se necesiten más intercambios, lo cual significa que la lista está ordenada.

Código

```
function generarArray(){
    let a = [];
    for(let i = 1; i<=10000; i++){
        a.push(Math.floor(Math.random() * 10001));
    }
    return a;
}

function bubbleSort(arr)
{
    var i, j, pivote;
    var cambio;
    for (i = 0; i < array.length - 1; i++)
    {
        cambio = false;
        for (j = 0; j < array.length - i - 1; j++)
        {
            if (arr[j] > arr[j + 1])
            {
                pivote = arr[j];
                arr[j] = arr[j + 1];
                arr[j + 1] = pivote;
                cambio = true;
            }
        }
        if (cambio == false)
        break;
    }
}

var array = generarArray();
bubbleSort(array);
console.log(array);
```

Salida

```
[Running] node "c:\Users\do30\OneDrive\Documentos\Galileo\Estructura de datos\Algoritmos de ordenamiento\bubble sort.js"
[
  1,  2,  3,  3,  5,  7,  7, 10, 10, 10, 13, 13,
 14, 15, 16, 17, 17, 19, 19, 19, 21, 21, 24, 25,
 25, 25, 26, 27, 28, 30, 33, 33, 34, 35, 35, 35,
 38, 38, 40, 40, 42, 42, 43, 44, 44, 44, 45, 46,
 47, 52, 52, 55, 55, 56, 59, 60, 61, 62, 63, 63,
 64, 64, 64, 64, 64, 66, 67, 68, 68, 69, 70, 71,
 71, 71, 71, 73, 73, 73, 73, 74, 75, 75, 76, 76,
 76, 77, 81, 82, 83, 84, 85, 86, 88, 89, 89, 89,
 90, 91, 92, 92,
 ... 9900 more items
]

[Done] exited with code=0 in 0.418 seconds
```

Selection Sort

Es uno de los mas simples y funciona de la siguiente manera:

- Encuentra el elemento más pequeño en el arreglo y lo intercambia con el primer elemento.
- Encuentra el segundo elemento más pequeño y lo intercambia con el segundo elemento del arreglo.
- Encuentra el tercer elemento más pequeño y lo intercambia con el tercer elemento del arreglo.
- Repite el proceso de encontrar el siguiente elemento más pequeño y cambiarlo a la posición correcta hasta que se ordene todo el arreglo.

Codigo

```
function generarArray(){
    let a = [];
    for(let i = 1; i<=10000; i++){
        a.push(Math.floor(Math.random() * 10001));
    }
    return a;
}

function swap(arr,xp, yp)
{
    var temp = arr[xp];
    arr[xp] = arr[yp];
    arr[yp] = temp;
}

function selectionSort(arr,  n)
{
    var i, j, min_idx;

    for (i = 0; i < n-1; i++)
    {
        min_idx = i;
        for (j = i + 1; j < n; j++)
        if (arr[j] < arr[min_idx])
            min_idx = j;

        swap(arr,min_idx, i);
    }
}


var array = generarArray();
var n = 5;
selectionSort(array, array.length);
console.log(array);
```

Salida

```
[Running] node "c:\Users\do30\OneDrive\Documentos\Galileo\Estructura de datos\Algoritmos de ordenamiento\selection sort.js"
[
  0,  1,  2,  2,  3,  4,  5,  7,  7,  8,  9,  9,
 10, 11, 12, 14, 14, 15, 16, 16, 16, 20, 20, 20,
 22, 24, 25, 25, 26, 26, 26, 27, 28, 28, 29, 30,
 30, 31, 33, 33, 34, 34, 37, 38, 39, 39, 40, 40,
 42, 42, 42, 43, 46, 46, 48, 48, 49, 49, 51, 52,
 52, 53, 53, 54, 54, 56, 57, 57, 59, 60, 62, 63,
 64, 65, 66, 69, 70, 71, 71, 73, 74, 75, 75, 76,
 77, 80, 81, 84, 84, 85, 85, 86, 86, 89, 89, 89,
 91, 91, 92, 92,
 ... 9900 more items
]

[Done] exited with code=0 in 0.315 seconds
```

Insertion sort

El algoritmo recibe un arreglo o lista de objetos comparables y retorna el mismo ordenado, para esto realiza un ciclo comenzando desde la segunda posición y terminando en N (última posición), entonces toma cada elemento del y lo inserta en su lugar correspondiente de la secuencia previamente ordenada arreglo. Y de esta manera elemento a elemento vamos ordenando el arreglo.

Codigo

```
function generarArray(){
    let a = [];
    for(let i = 1; i<=10000; i++){
        a.push(Math.floor(Math.random() * 10001));
    }
    return a;
}

function insertionSort(arr, n)
{
    let i, key, j;
    for (i = 1; i < n; i++)
    {
        key = arr[i];
        j = i - 1;


        while (j >= 0 && arr[j] > key)
        {
            arr[j + 1] = arr[j];
            j = j - 1;
        }
        arr[j + 1] = key;
    }
}


let arr = generarArray();
let n = arr.length;
insertionSort(arr, n);
console.log(arr);
```

Salida

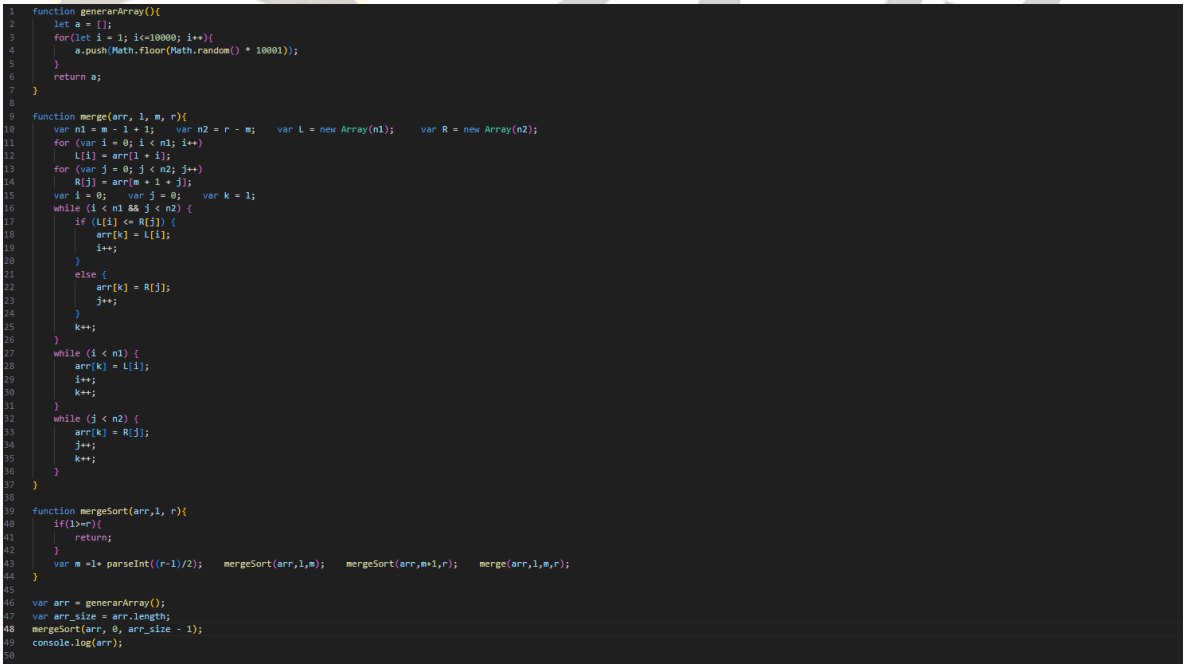
```
[Running] node "c:\Users\do30\OneDrive\Documentos\Galileo\Estructura de datos\Algoritmos de ordenamiento\insertion sort.js"
[
   1,   2,   2,   6,  14,  15,  15,  15,  17,  18,  20,  21,
  24,  24,  26,  26,  28,  28,  29,  29,  30,  31,  32,  33,
  35,  35,  38,  38,  39,  39,  40,  41,  42,  46,  47,  48,
  50,  52,  53,  55,  57,  57,  59,  61,  65,  66,  67,  68,
  68,  69,  69,  69,  70,  70,  70,  70,  72,  76,  76,  77,
  77,  77,  77,  78,  79,  80,  82,  82,  85,  85,  87,  88,
  88,  89,  89,  91,  91,  95,  99, 101, 101, 105, 105, 109,
 110, 111, 112, 114, 116, 116, 118, 118, 119, 121, 121, 123,
 125, 126, 127, 127,
 ... 9900 more items
]

[Done] exited with code=0 in 0.307 seconds
```

Merge Sort

Merge Sort es un algoritmo Divide y vencerás. Divide el arreglo de entrada en dos mitades, se llama a sí mismo para las dos mitades y luego fusiona las dos mitades ordenadas. La mayor parte del algoritmo tiene dos matrices ordenadas, y tenemos que fusionarlas en un único arreglo ordenado. Todo el proceso de ordenar un arreglo de N enteros se puede resumir en tres pasos:

- Divide el arreglo en dos mitades.
- Ordena la mitad izquierda y la mitad derecha usando el mismo algoritmo recurrente.
- Combina las mitades ordenadas.

Código



```
function generarArray(){
    let a = [];
    for(let i = 1; i<=10000; i++){
        a.push(Math.floor(Math.random() * 10001));
    }
    return a;
}

function merge(arr, l, m, r){
    var n1 = m - l + 1;    var n2 = r - m;    var L = new Array(n1);    var R = new Array(n2);
    for (var i = 0; i < n1; i++)
        L[i] = arr[l + i];
    for (var j = 0; j < n2; j++)
        R[j] = arr[m + 1 + j];
    var i = 0;    var j = 0;    var k = l;
    while (i < n1 && j < n2) {
        if (L[i] <= R[j]) {
            arr[k] = L[i];
            i++;
        }
        else {
            arr[k] = R[j];
            j++;
        }
        k++;
    }
    while (i < n1) {
        arr[k] = L[i];
        i++;
        k++;
    }
    while (j < n2) {
        arr[k] = R[j];
        j++;
        k++;
    }
}

function mergeSort(arr,l, r){
    if(l>=r){
        return;
    }
    var m =l+ parseInt((r-l)/2);    mergeSort(arr,l,m);    mergeSort(arr,m+1,r);    merge(arr,l,m,r);
}

var arr = generarArray();
var arr_size = arr.length;
mergeSort(arr, 0, arr_size - 1);
console.log(arr);
```

Salida

```
[Running] node "c:\Users\do30\OneDrive\Documentos\Galileo\Estructura de datos\Algoritmos de ordenamiento\merge sort.js"
[
   1,  1,  3,  4,  5,  6,  6,  6,  9, 11, 12, 13,
  15, 15, 15, 15, 16, 16, 16, 18, 19, 20, 22, 23,
  23, 24, 24, 25, 25, 26, 26, 26, 30, 30, 30, 32,
  32, 33, 33, 35, 37, 40, 42, 43, 43, 45, 47, 49,
  50, 52, 53, 55, 55, 55, 56, 57, 57, 59, 60, 60,
  61, 61, 61, 65, 66, 66, 68, 69, 71, 71, 71, 71,
  72, 72, 73, 73, 73, 74, 74, 75, 77, 77, 78, 78,
  78, 78, 78, 79, 79, 81, 81, 82, 83, 84, 85, 85,
  86, 86, 87, 87,
  ... 9900 more items
]

[Done] exited with code=0 in 0.243 seconds
```

Quick Sort

El algoritmo QuickSort se basa en la técnica de "divide y vencerás" por la que en cada recursión, el problema se divide en subproblemas de menor tamaño y se resuelven por separado (aplicando la misma técnica) para ser unidos de nuevo una vez resueltos. Es el algoritmo de ordenacion mas rapido conocido.

Codigo

```javascript
function generarArray(){
    let a = [];
    for(let i = 1; i<=10000; i++){
        a.push(Math.floor(Math.random() * 10001));
    }
    return a;
}

function partition(arr, low, high) {
    let pivot = arr[high];
    let i = low - 1;

    for (let j = low; j <= high - 1; j++) {
        if (arr[j] < pivot) {
            i++;
            [arr[i], arr[j]] = [arr[j], arr[i]];
        }
    }

    [arr[i + 1], arr[high]] = [arr[high], arr[i + 1]];
    return i + 1;
}

function quickSort(arr, low, high) {
    if (low < high) {
        let pi = partition(arr, low, high);
        quickSort(arr, low, pi - 1);
        quickSort(arr, pi + 1, high);
    }
}

let arr = generarArray();
let N = arr.length;

quickSort(arr, 0, N - 1);
console.log(arr);
```

Salida

```
[Running] node "c:\Users\do30\OneDrive\Documentos\Galileo\Estructura de datos\Algoritmos de ordenamiento\quick sort.js"
[
  0,  0,  1,  2,  2,  3,  3,  4,  5,  5,  5,  5,
  6,  7,  8,  8,  9, 12, 16, 17, 18, 19, 19, 20,
 20, 20, 22, 23, 23, 24, 24, 25, 25, 27, 27, 28,
 29, 31, 31, 32, 32, 34, 35, 36, 37, 40, 42, 43,
 44, 44, 45, 45, 45, 46, 46, 47, 47, 47, 49, 50,
 52, 53, 55, 58, 58, 60, 62, 62, 62, 64, 65, 65,
 65, 66, 66, 67, 67, 67, 71, 71, 73, 76, 76, 77,
 78, 81, 81, 81, 84, 84, 85, 86, 86, 86, 88, 89,
 92, 96, 96, 97,
 ... 9900 more items
]

[Done] exited with code=0 in 0.23 seconds
```

Conclusión

A continuación, se enlistan los algoritmos de ordenación desde el mas lento hasta el más rápido:

1. Bubble Sort
2. Selection Sort
3. Insertion Sort
4. Merge Sort
5. Quick Sort

Si bien la diferencia de tiempo es de milisegundos a la hora de ordenar 10,000 datos, puede llegar a ser más evidente a la hora de ordenar millones de datos.

Con esta prueba también se pudo determinar que si bien el código del algoritmo Quick Sort es el segundo mas largo (ya que es el segundo con más líneas de código) es el más rápido en ordenar, así como Merge Sort, que es el algoritmo con más líneas de código pero el segundo más rápido en ordenar.

Al final del día es importante considerar lo que se requiere en un proyecto o lo que es más importante. Ya que, si bien en este caso el más rápido es Quick Sort, cualquiera podría usar otro algoritmo en un proyecto donde la rapidez no es lo mas importante o si la base de datos no es demasiado grande, ya que mientras menos datos se ordenan, menos diferencia existe entre los algoritmos.