Instituto Tecnologico de Tijuana

Proyecto

Profesor:
Ray Brunett Parra Galaviz
Materia:
Estructura de datos
Nombre:
Edgar Javier Valdez Ruelas
Número de Control:
17210040

Índice

Introducción

En este documento se presenta una comparativa en tiempo de ejecución de los algoritmos de ordenamiento bubble sort, quick sort, merge sort y shellsort con un mismo arreglo, también la comparación de los algoritmos de búsqueda secuencial y binario con el mismo arreglo.

Código

```
function burbuja(array) {

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

        for (var j = 0; j < (array.length - i); j++) {
            //SI EL ARREGLO EN LA ACTUAL POSICION ES MAYOR QUE EL
SIGUIENTE
            if (array[j] > array[j + 1]) {
                k = array[j + 1]; //SE GUARDA TEMPORALMENTE EN K EL
VALOR DE LA SIGUIENTE POSICION
                array[j + 1] = array[j]; //SE EL SIGUIENTE VALOR CON EL
VALOR ACTUAL
                array[j] = k; //EL VALOR DE LA ACTUAL POSICION SE CAMBIA
POR EL TEMPORAL
            }
        }
    }
    return array;
}

function quicksort(array) {
    //se declaran 3 arreglos
    var menor = []
    var mayor = []
    var igual = []
    //mientras que el arreglo sea mayor que uno
    if (array.length > 1) {
        //escogemos el primer valor como pivote
        var pivote = array[0]
```

```
        for (var x of array) {
            //en base al pivote vamos separando los valores
            if (x < pivote)
                menor.push(x)
            if (x > pivote)
                mayor.push(x)
            if (x == pivote)
                igual.push(x)
        }
        //retornamos recursivamente la concatenacion de los 3 valores
        return quicksort(menor).concat(igual).concat(quicksort(mayor))
    } else {
        //regresamos el valor del arreglo
        return array
    }
}



function shellSort(arr) {
    //declaramos el incremento como la mitad de la longitud del arreglo
    var increment = arr.length / 2;
    //mientras que el incremento sea mayor que 0 se recorrera el arreglo
    while (increment > 0) {
        for (i = increment; i < arr.length; i++) {
            var j = i;
            var temp = arr[i];//almacenamos temporalmente el valor en la
posicion i

            while (j >= increment && arr[j - increment] > temp) {
                arr[j] = arr[j - increment];
                j = j - increment;
            }

            arr[j] = temp;
        }

        //recalculamos el valor del incremento
        if (increment == 2) {
            increment = 1;
        } else {
            increment = parseInt(increment * 5 / 11);
        }
    }
    return arr;
```

```
}


function mergeSort(array) {
    //si la longitud es 1 regresa el arreglo
    if (array.length == 1)
        return array
    //sacamos la mitad del arreglo
    var medio = Math.round(array.length / 2)
    //lo separamos en 2 arreglos izq y der
    var izquierda = array.slice(0, medio)
    var derecha = array.slice(medio)

    //retornamos la funcion mergesor2 donde llamamos recursivamente
hasta que la longitud del arreglo sea 1
    return mergeSort2(mergeSort(izquierda), mergeSort(derecha))

}

function mergeSort2(izq, der) {
    //se declaran las variables y el arreglo del resultado
    var resultado = [];
    i = 0;
    j = 0;
    //mientras que i sea menor que el arreglo izquierdo y j sea menor
que la longitud del arreglo derecho
    //se va ir anadiendo al arreglo resultado
    while (i < izq.length && j < der.length) {
        //si el elemento izq es menor que el derecho
        if (izq[i] < der[j]) {
            resultado.push(izq[i])//se agrega al resultado el elemento
izquierdo y aumenta el valor de i
            i++
        } else {
            resultado.push(der[j])//en caso contrario se agrega el valor
derecho y aumenta el valor de j
            j++
        }
    }
    //una vez que termina el ciclo juntamos los arreglos resultantes en
uno solo
    return resultado.concat(izq.slice(i)).concat(der.slice(j))
}

//---------------------------------BUSQUEDA--------------------------
```

```
//BUSQUEDA SECUENCIAL
function secuencialSearch(array, elemento) {
    //iteramos el arreglo
    for (var el in array) {
        //comparamos cada elemento del arreglo con el elemento a buscar
        if (array[el] == elemento) {
            console.log("elemento encontrado en el indice ", el)
            break;
        }
    }
}
//BUSQUEDA BINARIA
function binarySearch(arr, value) {
    // declaramos las variables iniciales
    let start = 0
    let stop = arr.length - 1
    let middle = Math.floor((start + stop) / 2)

    //Mientras el valor del medio no sea el valor que buscamos  y start
sea menor que stop
    while (arr[middle] !== value && start < stop) {
        //si el valor a buscar es menor al valor de en medio
        if (value < arr[middle]) {
            stop = middle - 1 //el valor de stop cambia por el valor de
en middle -1
        } else {
            start = middle + 1 //el valor de estart cambia por el valor
de middle +1
        }

        // se re calcula el valor de middle en cada iteracion
        middle = Math.floor((start + stop) / 2)
    }

    // Si el valor que buscamos esta en el indice middle se devuelve el
indice y en caso de no se encontrado se devuelve -1
    if (arr[middle] !== value) {
        console.log("el elemento no fue encontrado")
    } else {
        console.log("elemento encontrado en el indice ", middle)
    }
}


//funcion para generar un arreglo con valores random
```

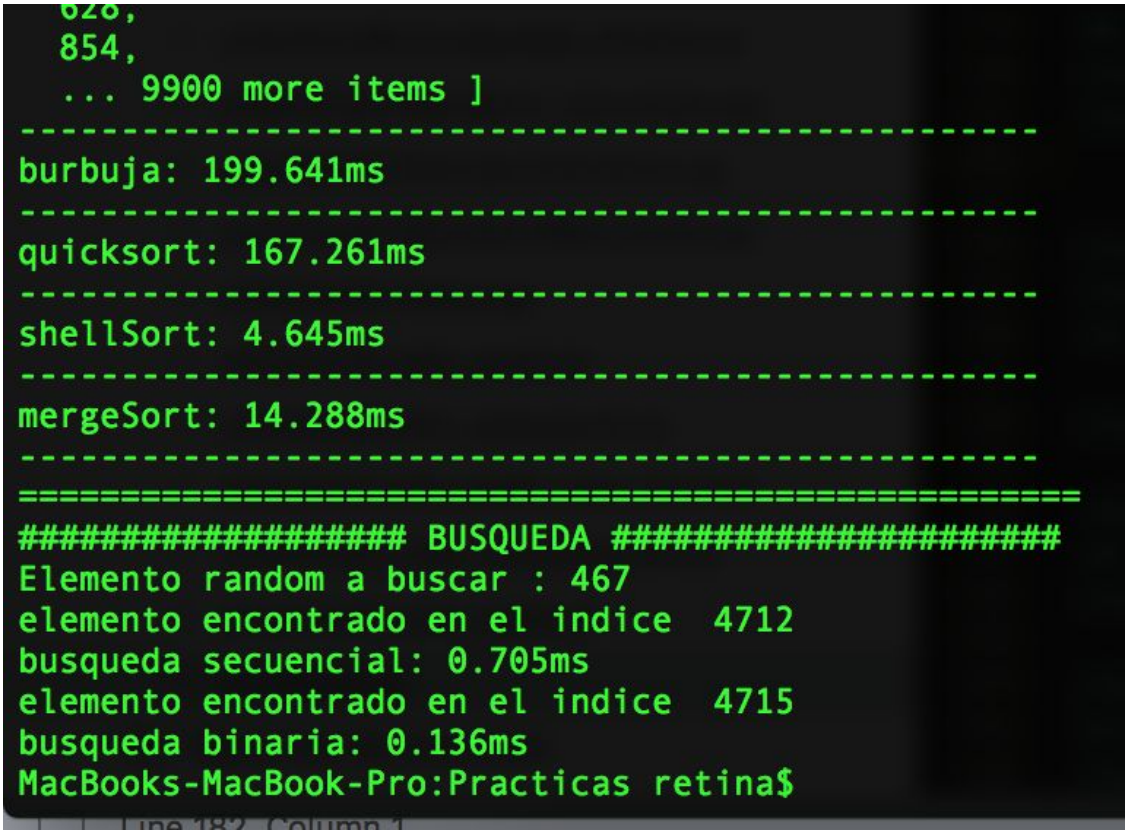
```
function generacionDeArray(long) {
    var arr = []
    for (var i = 0; i < long; i++) {
        arr.push(Math.floor(Math.random() * 1000) + 1)
    }
    return arr
}


var array = generacionDeArray(10000);

//DESPLIEGUE DE RESULTADOS

console.log(array)
console.log("-----------------------------------------------------")
console.time('burbuja')
burbuja(array)
console.timeEnd('burbuja');
console.log("-----------------------------------------------------")
console.time('quicksort')
quicksort(array)
console.timeEnd('quicksort');
console.log("-----------------------------------------------------")
console.time('shellSort')
shellSort(array)
console.timeEnd('shellSort');
console.log("-----------------------------------------------------")
console.time('mergeSort')
mergeSort(array)
console.timeEnd('mergeSort');
console.log("-----------------------------------------------------")


console.log("=====================================================")
console.log("#################### BUSQUEDA ######################")
var elementoABuscar = Math.floor(Math.random() * 1000) + 1;

console.log("Elemento random a buscar :", elementoABuscar);

console.time('busqueda secuencial')
secuencialSearch(array, elementoABuscar);
console.timeEnd('busqueda secuencial');

console.time('busqueda binaria')
binarySearch(array, elementoABuscar);
```

```
console.timeEnd('busqueda binaria');
```

Ejecuciones

Se utilizó un arreglo de 10,000 elementos completamente aleatorios

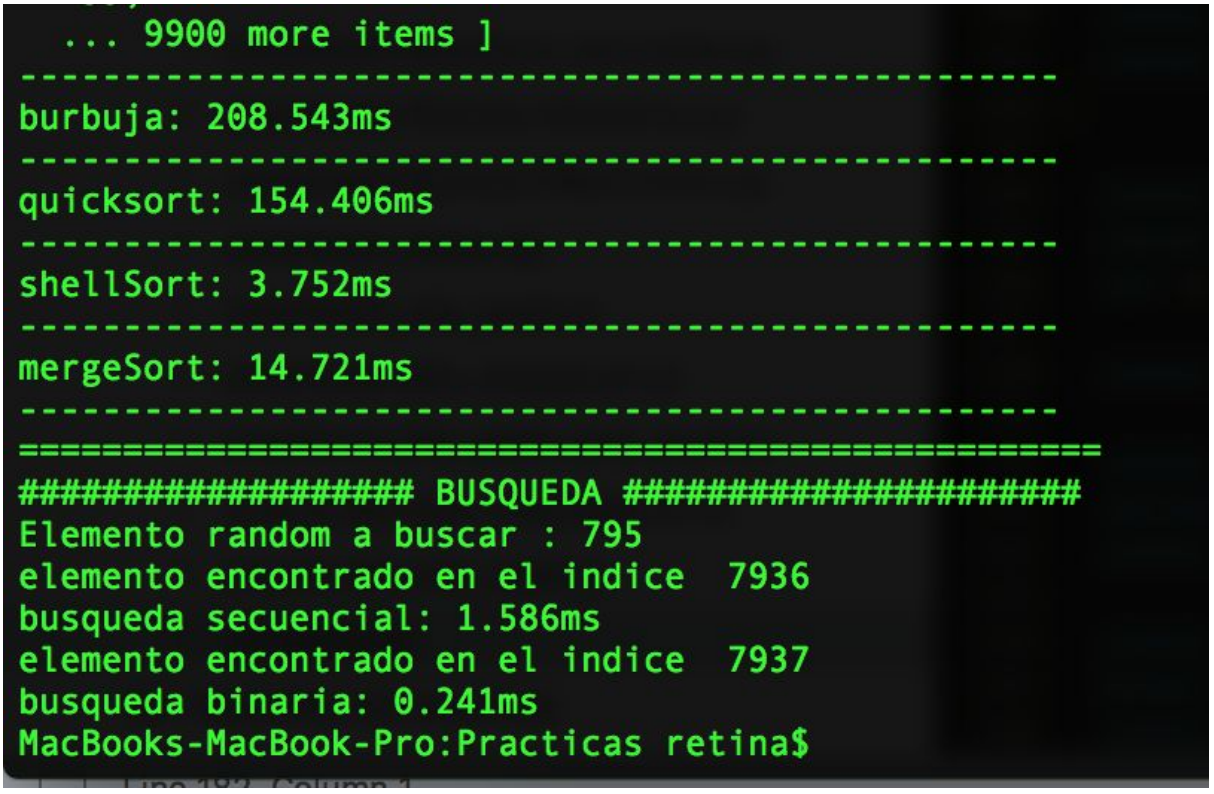
Ejecución 1



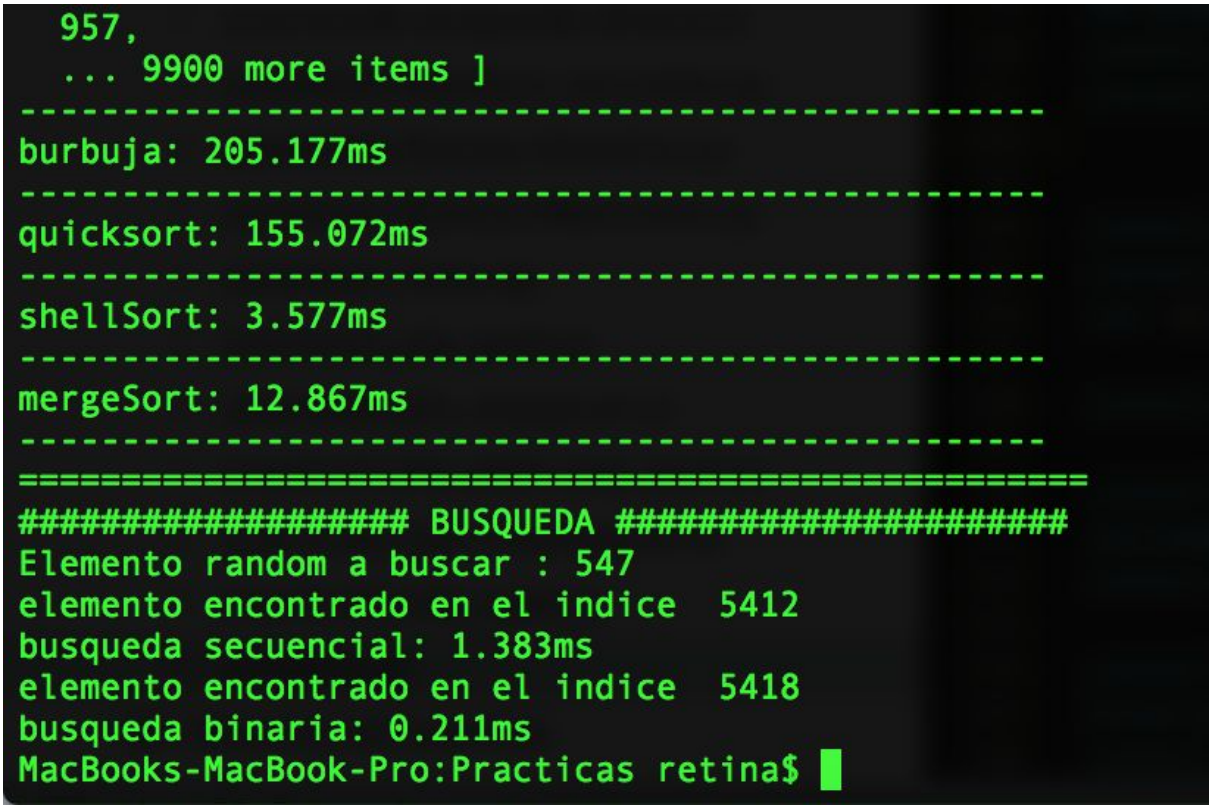
Ejecucion 2


373,
... 9900 more items]
burbuja: 202.581ms
quicksort: 192.714ms
shellSort: 4.165ms
mergeSort: 16.469ms
BUSQUEDA
Elemento random a buscar : 877
elemento encontrado en el indice 8752
busqueda secuencial: 1.481ms
elemento encontrado en el indice 8758
busqueda binaria: 0.219ms
MacBooks-MacBook-Pro:Practicas retina$

Ejecución 3


... 9900 more items]
burbuja: 208.543ms
quicksort: 154.406ms
shellSort: 3.752ms
mergeSort: 14.721ms
BUSQUEDA
Elemento random a buscar : 795
elemento encontrado en el indice 7936
busqueda secuencial: 1.586ms
elemento encontrado en el indice 7937
busqueda binaria: 0.241ms
MacBooks-MacBook-Pro:Practicas retina$

Ejecucion 4



Comparación de resultados

Metodos de ordenamiento.

	Bubble Sort	Quick Sort	Shell Sort	Merge Sort
Ejecución 1	199.641ms	167.261ms	4.645ms	14.288ms
Ejecución 2	202.581ms	192.714ms	4.165ms	16.469ms
Ejecución 3	208.543ms	154.406ms	3.752ms	14.721ms
Ejecución 4	205.177ms	155.072ms	3.577ms	12.867ms

Metodos de busqueda.

	Secuencial	Binaria
Ejecución 1	0.705ms	0.136ms
Ejecución 2	1.481ms	0.219ms
Ejecución 3	1.586ms	0.241ms
Ejecución 4	1.383ms	0.211ms

Conclusión

En cuanto a métodos de ordenamiento shell sort superó por mucho a los otros 3 métodos en cada una de las ejecuciones, le siguien merge,quick y bubble siendo este ultimo el mas lento de los 4. Y en cuanto a los metodos de busqueda esta claro que el mas rapido fue el metodo de busqueda binaria por bastante.