Técnicas de Programación

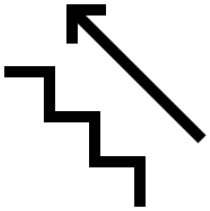
Carrera Programador full-stack

Algoritmos Básicos (Conceptos)

Algoritmos Básicos

Muchas aplicaciones requieren contar con métodos básicos para brindar funcionalidad útil y de valor:

Recorrido



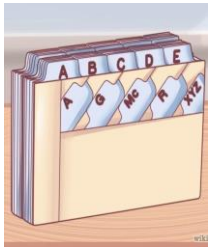
Búsqueda

Ordenamiento

Algoritmos de Ordenamiento

Objetivo y Alternativas

- Permiten dar un orden a los elementos de una estructura, por ejemplo:



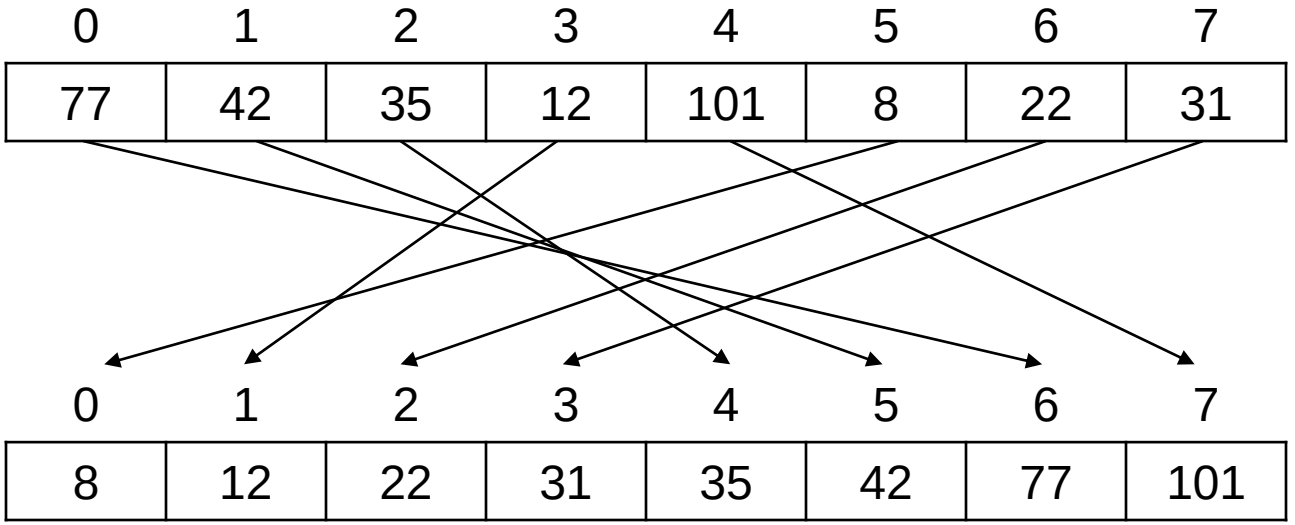
  - Orden alfabético descendente (de la Z a la A)
  - Orden numérico ascendente (0 a infinito)

- Existen diferentes variantes, que dependen de su complejidad temporal y espacial, así también de su simplicidad a la hora de programar

Algoritmos de Ordenamiento

Lineamientos del Código

- Tienen como **entrada** una **estructura** (arreglo)
- Tienen como **salida** la misma **estructura ordenada**
- Saben como **comparar** e **intercambiar** los elementos



Algoritmos de Ordenamiento

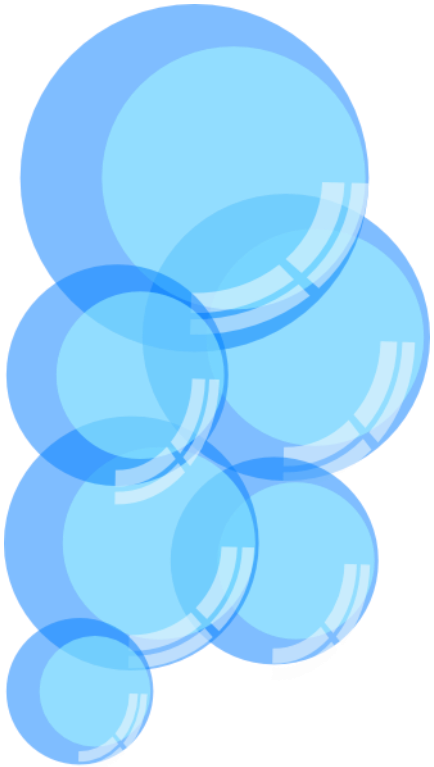
Tipos de Algoritmos

- Pueden ser iterativos o recursivos
- Pueden tardar **más o menos** según:
  - La cantidad de veces que recorren la estructura
  - La cantidad de comparaciones que hacen
  - La cantidad de veces que intercambian valores
- Clasificados por su desempeño promedio, el mejor y el peor caso
- Algoritmos:
  - Burbuja (bubble-sort)
  - Selección (selection-sort)
  - Mezclado (merge-sort)
  - Rápido (quick-sort)
  - Muchos más…

Algoritmos de Ordenamiento

Burbuja (bubble-sort)

- Se **comparan** los elementos **adyacentes** y se simula un burbujeo, donde las burbujas más grandes se cambian con las más chicas
- Se **intercambian** los elementos solamente si no están en el **orden correcto**
- Es uno de los algoritmos de ordenamiento más **simples** de programar porque solo hace **comparaciones** entre **vecinos**



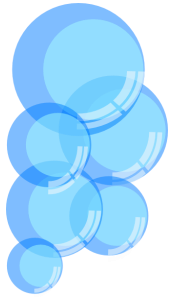
Algoritmos de Ordenamiento

Burbuja (video)

https://www.youtube.com/watch?v=lyZQPjUT5B4

Algoritmos de Ordenamiento

Burbuja (razonamiento)

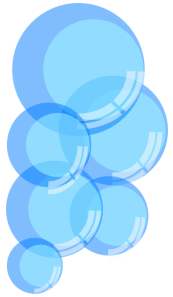


Como se codifica:

- Dos bucles (con índices i y j)
- El primero itera la cantidad de veces que tenemos que burbujear
- El segundo delimita desde donde empieza y donde termina el burbujeo
- El burbujeo consiste en comparar a[j] y a[j + 1] y darlos vuelta si corresponde
- Tener en cuenta a medida que burbujeamos los elementos al final del arreglo empiezan a estar ordenados

Algoritmos de Ordenamiento

Burbuja (código)

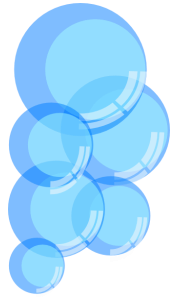


Este método permite cargar un arreglo “arreglo” de dimensión “cantidad” y llenarlo de valores generados al azar entre 0 y “numAzar” (parámetro)

```
function cargar(arreglo:number[], cantidad:number, numAzar:number)
    let i : number;
    for (i = 0 ; i<cantidad; i++ ) {
        arreglo[i] = Azar(numAzar);
    }
}
```

Algoritmos de Ordenamiento

Burbuja (código)

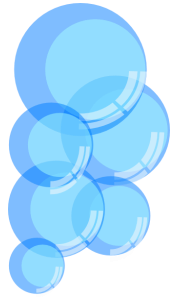


Este método permite mostrar un arreglo “arreglo” de dimensión “cantidad” en una única línea, separando los valores con un espacio

```
function escribirEnUnaLinea(arreglo:number[], cantidad:number) {
    let i:number;
    let vector:string = "" ;
    for (i = 0 ; i<cantidad; i++) {
        vector += `${arreglo[i]} `;
    }
    console.log (vector);
}
```

Algoritmos de Ordenamiento

Burbuja (código)

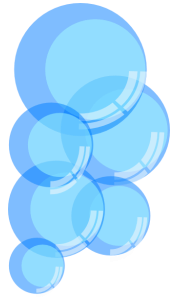


Este método permite intercambiar los valores en las posiciones “i” y “j” de un arreglo “arreglo” utilizando una variable auxiliar

```
function intercambiar(arreglo:number[], i:number, j:number) {
    let aux:number;
    aux = arreglo[i] ;
    arreglo[i] = arreglo[j] ;
    arreglo[j] = aux ;
}
```

Algoritmos de Ordenamiento

Burbuja (código)



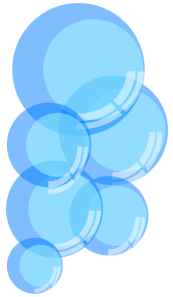
Este método permite comparar los valores en las posiciones “i” y “j” del arreglo “arreglo”

- Devuelve 0 si son iguales,
- 1 si lo que hay en “i” es mayor a lo que hay en “j”
- -1 si lo que hay en “i” es menor a lo que hay en “j”

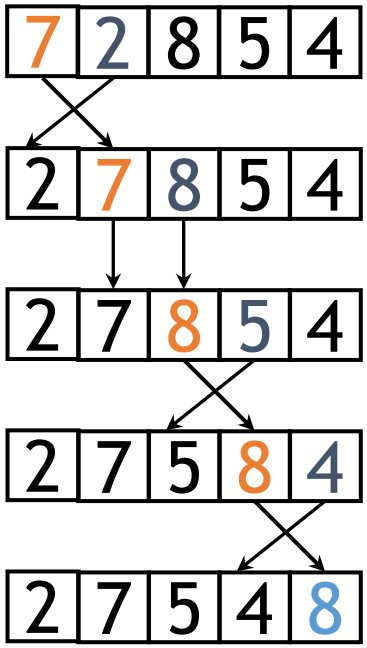
```
function comparar(arreglo : number[], i : number, j :
number) : number {
        let comparacion : number;
        if (arreglo[i] === arreglo[j]) {
                comparacion = 0;
        } else if (arreglo[i] < arreglo[j]) {
                comparacion = -1;
        } else {
                comparacion = 1;
        }
        return comparacion;
}
```

Algoritmos de Ordenamiento

Burbuja (ejemplo)

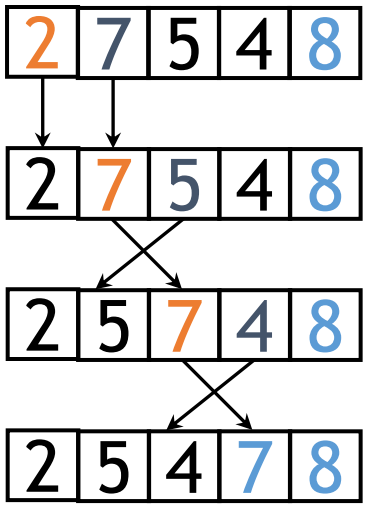


i=2



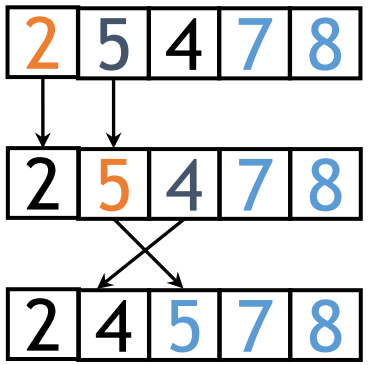
j=3 j+1=4

i=3



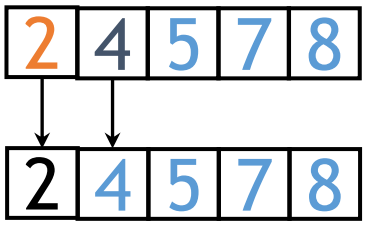
j=2 j+1=3

i=4



j=1 j+1=2

i=5



j=0 j+1=1

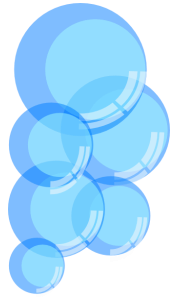
N=5

1er Ciclo: i = 2 Hasta 5

2do Ciclo: j = 0 Hasta 5-i

Algoritmos de Ordenamiento

Burbuja (código)



```
function burbuja(arreglo : number[], cantidad : number)
        let i : number, j : number;
        for (i = 2 ; i < cantidad; i++) {
                for (j = 0 ; j < (cantidad - 1); j++) {
                        if (comparar(arreglo, j, j+1) == 1 ) {
                                intercambiar(arreglo, j, j+1) ;
                        }
                }
        }
}
```

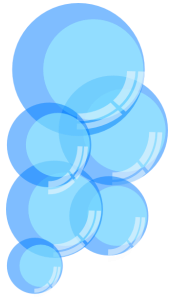
Desde 2 hasta n (el primer elemento esta ordenado en la ultima vuelta)

Desde 0 hasta n – 1 (vamos achicando el rango a medida que se ubican los valores al final del arreglo)

Si los adyacentes j y j + 1 no están ordenados, intercambiarlos

Algoritmos de Ordenamiento

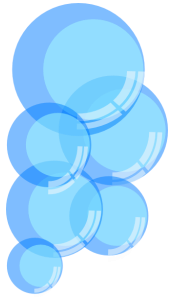
Burbuja (código)



```
//Algoritmo Orden

let lim : number = 10;
let a : number[]= new Array(lim);
cargar(a, lim, 100);
escribirEnUnaLinea(a, lim);
burbuja(a, lim);
escribirEnUnaLinea(a, lim);
```

Algoritmos de Ordenamiento

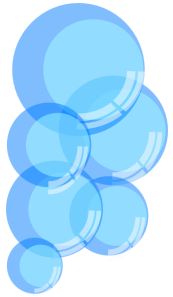
Burbuja (Ejemplo en clase)



```
let arregloBur = [1,7,4,8,5,9,3];
console.log(burbuja(arregloBur,7));


function burbuja(arreglo : number[], cantidad : number):number[]{
    let i : number, j : number;
    for (i = 2 ; i < cantidad; i++) {
        for (j = 0 ; j < (cantidad - 1); j++) {
            if (comparar(arreglo, j, j+1) == 1 ) {
                intercambiar(arreglo, j, j+1) ;
            }
        }
    }
    return arreglo;
}
function comparar(arreglo : number[], i : number, j : number) : number {
    let comparacion : number;
    if (arreglo[i] === arreglo[j]) {
        comparacion = 0;
    } else if (arreglo[i] < arreglo[j]) {
        comparacion = -1;
    } else {
        comparacion = 1;
    }
    return comparacion;
}
function intercambiarBur(arreglo:number[], i:number, j:number) {
    let aux:number;
    aux = arreglo[i] ;
    arreglo[i] = arreglo[j] ;
    arreglo[j] = aux ;
}
```

Algoritmos de Ordenamiento

Burbuja (eficiencia)



- Complejidad: n^2 (dos loops)

- Mejor caso: todo ordenado de antemano

- Peor caso: ordenado en sentido inverso

Algoritmos de Ordenamiento

Selección (selection-sort)

- Permite **ordenar** un estructura de forma **natural**
- Funciona **buscando** el elemento que corresponde en una ubicación y moviéndolo al **lugar correcto** (es decir, ordenado)
- Ejemplo para orden ascendente:
  - se localiza el mínimo de un arreglo y se lo coloca en el primer lugar
  - se localiza el segundo mínimo y se lo coloca en el segundo,
  - y así hasta que no queden elementos que colocar
- Es **ligeramente mejor** que “burbuja” porque **intercambia menos** valores

Algoritmos de Ordenamiento

Selección (video)

https://www.youtube.com/watch?v=Ns4TPTC8whw

Algoritmos de Ordenamiento

Selección (razonamiento)

Cómo se codifica:

- Dos bucles (con índices i y j)
- El primero itera por la cantidad de elementos en el arreglo, y el índice i denota la posición que se está buscando ordenar
- El segundo delimita las posiciones que todavía no han sido ordenadas
- Se busca el mínimo/máximo valor en el arreglo en el rango del segundo bucle (índice j)
- Al terminar el segundo bucle, intercambiamos lo que haya en la índice i con lo que haya en la posición con el valor mínimo/máximo

Algoritmos de Ordenamiento

Selección (código)

```
function seleccion(arreglo:number[], cantidad:number) {
    let i:number, j:number, posicion:number;
    for (i = 0; i < (cantidad-1); i++) {
        posicion = i;
        for (j = i + 1; j < cantidad; j++) {
            if (comparar(arreglo, posicion, j) == 1) {
                posicion = j;
            }
        }
        intercambiar(arreglo, i, posicion);
    }
}
```

Desde 0 hasta n-2 (el último elemento queda ordenado al final del ciclo)

Desde i+1 hasta n-1 (vamos moviendo el rango izquierdo a medida que se ubican los valores al comienzo del arreglo)

Si el valor en el índice "j" es menor/mayor que el que hay en "posición", actualizar "posición" con "j"

Una vez que encontré el valor en el índice "posición" que corresponde en el índice "i", intercambiarlos

Algoritmos de Ordenamiento

Selección (código)

```
//Algoritmo Orden
let lim: number = 10;
let a: number[] = new Array(lim);
cargar(a, lim, 100);
escribirEnUnaLinea(a, lim);
//seleccion
seleccion(a, lim);
escribirEnUnaLinea(a, lim);
```

Algoritmos de Ordenamiento

Selección (Ejemplo en clase)

```
function seleccion(arreglo: number[]): number[] {
    for (let i = 0; i < 9 - 1; i++) {
        let minimoIndex = i;
        for (let j = i + 1; j < 9; j++) {
            if (arreglo[j] < arreglo[minimoIndex]) {
                minimoIndex = j;
            }
        }
        if (minimoIndex !== i) {
           intercambiar(arreglo,i,minimoIndex);
        }
    }
    return arreglo;
}

const arregloSel = [3, 6, 2, 9, 7, 1, 4, 8, 5];
console.log(seleccion(arregloSel));

function intercambiar(arreglo: number[], indice:number, minimo:number){
    let aux= arreglo[indice];
    arreglo[indice]  = arreglo[minimo];
    arreglo[minimo] = aux;
}
```

Algoritmos de Ordenamiento

Selección (eficiencia)

- Complejidad: n^2 (dos loops)

- Mejor y peor caso: siempre hace la misma cantidad de comparaciones

Técnicas de Programación

Carrera Programador full-stack

Algoritmos Básicos (Ejercicios)

Ordenamiento

Implemente un algoritmo de ordenamiento con el método Bubble Sort, para que ordene un arreglo de longitud N en orden descendente.

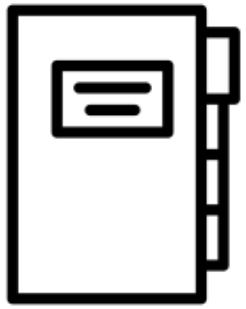
Algoritmos de Ordenamiento

Ordenar por Dos Criterios

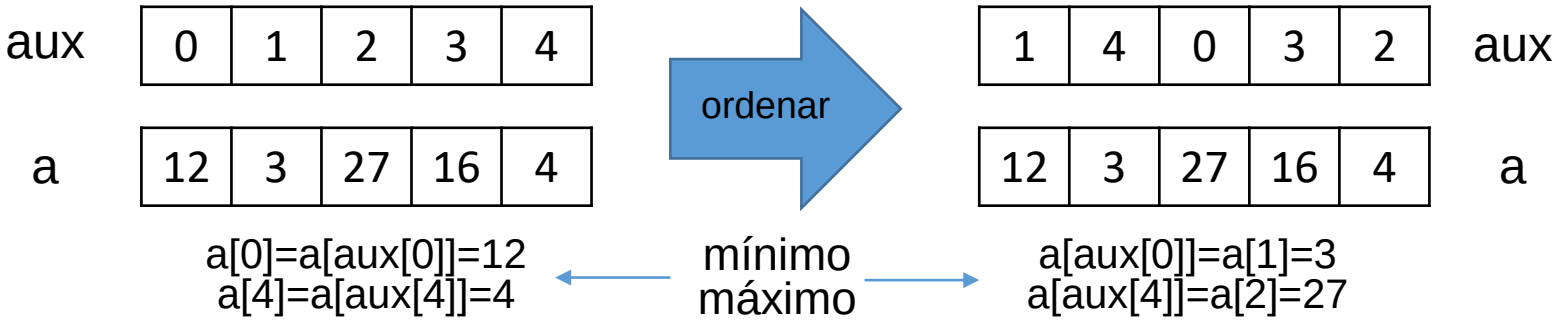
- Dados un arreglo de texto y dos arreglos de enteros de tamaño n:
  - nombres Como Texto
  - años Como Entero y altura Como Entero
- Ordénelos los tres vectores a la vez según los años, y en caso que haya un empate, utilice la altura para desempatar
- Tener en cuenta que los intercambios tienen que cambiar los elementos de los tres vectores a la vez

Algoritmos de Ordenamiento

Ordenar con Arreglo Auxiliar



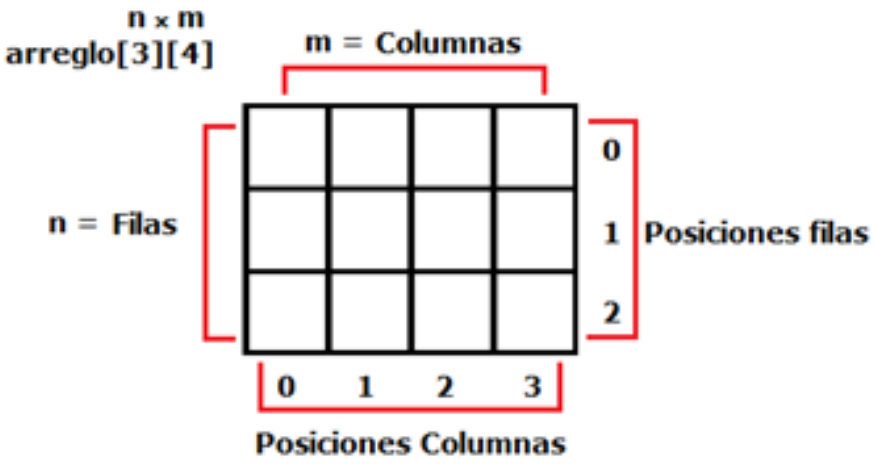
- Desarrollar un programa que permita ordenar un arreglo “a” de tamaño “n” sin modificarlo, es decir, sin hacer los intercambios sobre la estructura “a”
- Utilizar un arreglo auxiliar “aux” cargado con los índices del arreglo “a” (de 0 a n)
- El ordenamiento tiene que hacerse mirando los valores de “a” pero haciendo los intercambios en “aux”
- Crear un método que permita imprimir ordenado que reciba como parámetros “a”, “aux” y “n”



Algoritmos de Ordenamiento

Ordenar Matriz por Fila

- Crear un algoritmo que permita ordenar las filas de una matriz de nxm en orden descendente según la suma de todas sus elementos (es decir, todas las columnas)
- Tener en cuenta que la comparación se hace entre filas (y no entre elementos puntuales de la matriz)
- Considerar que el intercambio tiene que mover filas enteras (en vez de un solo número)



Técnicas de Programación

Carrera Programador full-stack

Algoritmos Básicos (Resolución)

Algoritmos de Ordenamiento

Ordenar por Dos Criterios

- Dados un arreglo de texto y dos arreglos de enteros de tamaño n:
  - nombres Como Texto
  - años Como Entero y altura Como Entero
- Ordénelos los tres vectores a la vez según los años, y en caso que haya un empate, utilice la altura para desempatar
- Tener en cuenta que los intercambios tienen que cambiar los elementos de los tres vectores a la vez

Algoritmos de Ordenamiento

Ordenar por Dos Criterios

Permite cargar los nombres, la edad y la altura de un número dado de personas y almacenarlos en arreglos

```
function cargarPersonas(nombres:string[], anios:number[], altura:number[], n:number) {
    let i:number;
    for (i = 0; i < n; i++) {
        nombres[i] = rls.question("Nombre: ");
        anios[i] = rls.questionFloat("Edad: ");
        altura[i] = rls.questionFloat("Altura (en cm): ");
    }
}
```

Algoritmos de Ordenamiento

Ordenar por Dos Criterios

Sirve para escribir por pantalla los datos de las personas de a una línea por vez (por ejemplo, Alejandro - 32 años - 170 cm)

```
function escribirPorPantalla(nombres:string[], anios:number[], altura:number[], n:number) {
    let i:number;
    for (i = 0; i < n; i++) {
        console.log(`${nombres[i]} - ${anios[i]} años - ${altura[i]} cm`);
    }
}
```

Algoritmos de Ordenamiento

Ordenar por Dos Criterios

```
function burbuja(nombres:string[], anios:number[], altura:number[], n:number) {
    let i:number, j:number;
    for (i = 2; i < n; i++) {
        for (j = 0; j < (n - i); j++) {
            if (comparar(anios, altura, j, j+1) == 1) {
                intercambiar(nombres, j, j+1);
                intercambiar(anios, j, j+1);
                intercambiar(altura, j, j+1);
            }
        }
    }
}
```

Se reciben como parámetros los tres arreglos y su tamaño

Se comparan años y altura para determinar si burbujea

Si corresponde, se intercambian las valores en los tres arreglos

Algoritmos de Ordenamiento

Ordenar por Dos Criterios

```
function comparar(anios:number[], altura:number[], i:number, j:number):number {
    let comparacion:number;
    if (anios[i] == anios[j]) {
        if (altura[i] == altura[j] ) {
            comparacion = 0;
        } else if (altura[i] < altura[j] ) {
            comparacion = -1;
        } else {
            comparacion = 1;
        }
    } else if (anios[i] < anios[j] ) {
        comparacion = -1;
    } else {
        comparacion = 1;
    }
    return comparacion;
}
```

Si los valores almacenados en “años” son iguales, entonces se comparan los valores almacenados en “altura”

Si los valores almacenados en “años” son diferentes, entonces se determina si son mayores o menores para establecer el orden de los arreglos

Algoritmos de Ordenamiento

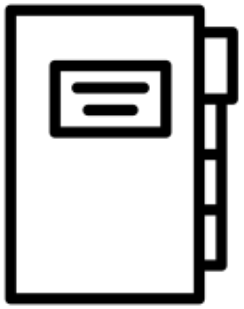
Ordenar por Dos Criterios

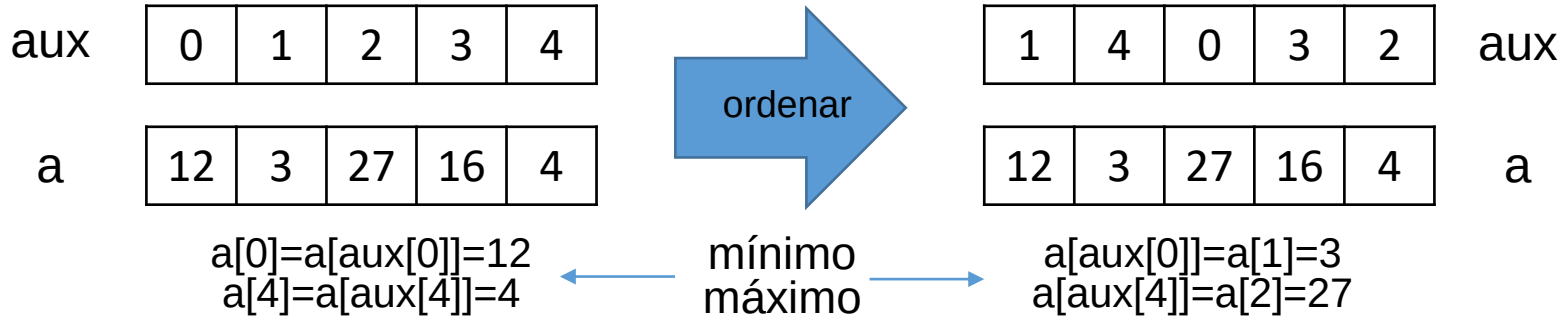
```
//Algoritmo DosCriterios

let n : number = 6;
let nombres : string[] = new Array(n);
let anios : number[] = new Array(n);
let altura : number[] = new Array(n);
cargarPersonas(nombres, anios, altura, n);
console.log ("Sin ordenar");
escribirPorPantalla(nombres, anios, altura, n);
console.log ("Ordenado");
burbuja(nombres, anios, altura, n);
escribirPorPantalla(nombres, anios, altura, n);
```

Algoritmos de Ordenamiento

Ordenar con Arreglo Auxiliar

- Desarrollar un programa que permita ordenar un arreglo “a” de tamaño “n” sin modificarlo, es decir, sin hacer los intercambios sobre la estructura “a”
- Utilizar un arreglo auxiliar “aux” cargado con los índices del arreglo “a” (de 0 a n)
- El ordenamiento tiene que hacerse mirando los valores de “a” pero haciendo los intercambios en “aux”
- Crear un método que permita imprimir ordenado que reciba como parámetros “a”, “aux” y “n”





Algoritmos de Ordenamiento

Ordenar con Arreglo Auxiliar

Completa un arreglo “a” de valores enteros al azar cuyo rango es entre 0 y el valor “numAzar” ingresado como parámetro (menos 1)

```
function cargar(a : number[], n : number, numAzar : number) {
    let i : number;
    for (i = 0; i < n; i++) {
        a[i] = Azar(numAzar);
    }
}
```

Algoritmos de Ordenamiento

Ordenar con Arreglo Auxiliar

Completa un arreglo “a” de valores enteros consecutivos entre 0 y “n” - 1

```
function inicializar(a : number[], n : number) {
    let i : number;
    for (i = 0; i < n; i++) {
        a[i] = i;
    }
}
```

Algoritmos de Ordenamiento

Ordenar con Arreglo Auxiliar

Permite escribir por pantalla un arreglo “a” de tamaño “n” en una única línea separados por espacios (por ejemplo, “1 10 25 50 22”)

```
function escribirEnUnaLinea(a : number[], n : number) {
    let i : number;
    let cadena : string = " ";
    for (i = 0; i < n; i++) {
                cadena += `${a[i]} `;
    }
    console.log (cadena);
}
```

Algoritmos de Ordenamiento

Ordenar con Arreglo Auxiliar

Permite escribir por pantalla un arreglo “a” de tamaño “n” en una única línea, pero utilizando el arreglo “aux” como índice para acceder a las posiciones de “a”

```
function escribirConIndice(a : number[], aux : number[], n : number) {
    let i : number;
    let cadena : string = " ";
    for (i = 0; i < n; i++) {
                cadena += `${a[aux[i]]} `;
    }
    console.log (cadena);
}
```

Algoritmos de Ordenamiento

Ordenar con Arreglo Auxiliar

Permite escribir los tres arreglos: el arreglo original “a”, el arreglo índice “aux” y los valores de “a” accedidos a través de “aux” en una única línea

```
function mostrarArreglos(a : number[], aux : number[], n : number) {
    console.log ("a[i]=");
    escribirEnUnaLinea(a, n));
    console.log ("aux[i]=");
    escribirEnUnaLinea(aux, n));
    console.log ("a[aux[i]]=");
    escribirConIndice(a, aux, n));
}
```

Algoritmos de Ordenamiento

Ordenar con Arreglo Auxiliar

```
function comparar(a:number[], aux:number[], i:number, j:number)
:number {
        let comparacion:number;
        if (a[aux[i]] == a[aux[j]]) {}
            comparacion = 0;
        } else if (a[aux[i]] < a[aux[j]]) {
            comparacion = -1;
        } else {
            comparacion = 1;
              }
        return comparacion;
}
```

También se recibe como parámetro al arreglo “aux”

A la hora de comparar los valores, los mismos se acceden a través del índice “aux” (en vez de accederlos directamente)

Se puede evitar modificar el método? Que pasa si pasamos los índices de “aux” en vez de los de “a” como parámetros?

Algoritmos de Ordenamiento

Ordenar con Arreglo Auxiliar

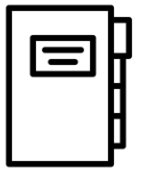
```
function seleccion(a:number[], aux:number[], n:number) {
    let i:number, j:number, pos:number;
    for (i = 0; i < n - 1; i++) {
        pos = i;
        for (j = i + 1; j <= (n - 1); j++) {
            if (comparar(a, aux, pos, j) == 1) {
                pos = j;
            }
        }
        intercambiar(aux, i, pos);
    }
}
```

Para comparar, se utiliza tanto el arreglo “a” como el arreglo “aux”

Al intercambiar los valores, solamente se hacen las modificaciones en el arreglo “aux” donde están los índices

Algoritmos de Ordenamiento

Ordenar con Arreglo Auxiliar

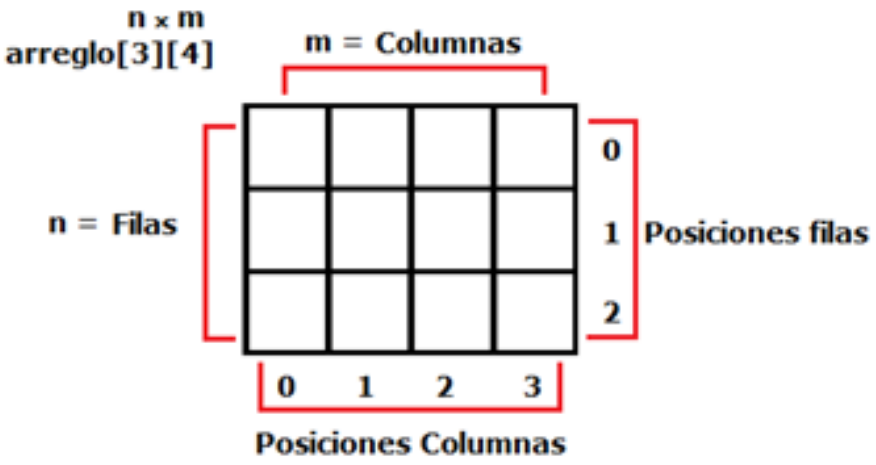


```
//Algoritmo OrdenarConIndice
let n : number = 10;
let a : number[] = new Array(n);
let aux : number[] = new Array(n);
cargar(a, n, 100);
inicializar(aux, n);
console.log ("Antes de ordenar");
mostrarArreglos(a, aux, n);
console.log ("Ordenando...");
seleccion(a, aux, n);
console.log ("Despues de ordenar");
mostrarArreglos(a, aux, n);
```

Algoritmos de Ordenamiento

Ordenar Matriz por Fila

- let un algoritmo que permita ordenar las filas de una matriz de nxm en orden descendente según la suma de todas sus elementos (es decir, todas las columnas)
- Tener en cuenta que la comparación se hace entre filas (y no entre elementos puntuales de la matriz)
- Considerar que el intercambio tiene que mover filas enteras (en vez de un solo número)

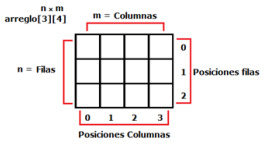


Algoritmos de Ordenamiento

Ordenar Matriz por Fila

Completa una matriz de $f_x c$ con valores aleatorios entre 0 y “numAzar” - 1

```
function cargar(matriz:number[][], f:number, c:number, numAzar:number) {
    let fil:number, col:number;
    for (fil = 0; fil < f; fil++) {
        for (col = 0; col < c; col++) {
            matriz[fil][col] = Azar(numAzar);
        }
    }
}
```

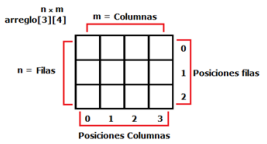


Algoritmos de Ordenamiento

Ordenar Matriz por Fila

Muestra por pantalla una matriz de fxc de a una fila por línea

```
function mostrarMatriz(matriz:number[][], f:number, c:number) {
        let fil:number, col:number;
        let cadena:string;
        for (fil = 0; fil < f; fil++) {
                cadena = " ";
                for (col = 0; col < c; col++) {
                                cadena += ` ${matriz[fil][col]} `";
                                }
                console.log(cadena);
        }
}
```

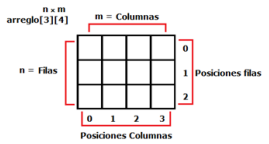


Algoritmos de Ordenamiento

Ordenar Matriz por Fila

Calcula y retorna la suma de todos los valores de una fila “f” en una matriz de “c” columnas

```
function calcularSumaFila(matriz:number[][], f:number, c:number):number {
    let suma:number = 0;
    let col:number;
    for (col = 0; col < c; col++) {}
        suma += matriz[f][col];
    }
    return suma;
}
```

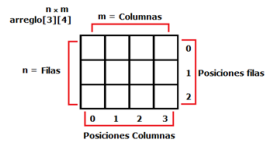


Algoritmos de Ordenamiento

Ordenar Matriz por Fila

Intercambia las filas “f” y “fs” de una matriz

```
function intercambiarFilaCompleta(matriz:number[][], f:number, fs:number,
c:number) {
        let aux:number[] = new Array(c);
        aux[0] = matriz[f];
        matriz[f] = matriz[fs];
        matriz[fs] = aux[0];
}
```



Algoritmos de Ordenamiento

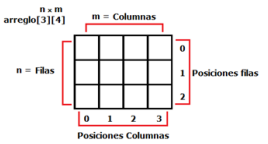
Ordenar Matriz por Fila

```
function compararFila(matriz:number[][], f:number, fs:number, c:number):number {
        let comparacion:number;
        let sumaFilaF:number;
        let sumaFilaFS:number;
        sumaFilaF = calcularSumaFila(matriz, f, c);
        sumaFilaFS = calcularSumaFila(matriz, fs, c);
        if (sumaFilaF == sumaFilaFS) {
                comparacion = 0;
        } else if (sumaFilaF < sumaFilaFS) {
                comparacion = -1;
        } else {
                comparacion = 1;
        }
        return comparacion;
}
```

Se para como parámetro la cantidad de columnas de la matriz

Se calcula la suma de la fila f y de la fila fs, almacenandolas en dos variables

La comparación se hace con los resultados de las sumas de las filas



Algoritmos de Ordenamiento

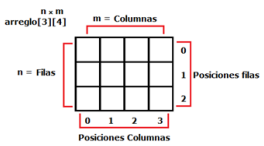
Ordenar Matriz por Fila

```
function burbuja(matriz:number[][], f:number, c:number) {
    let i:number, j:number;
    for (i = 2; i < f; i++) {
        for (j = 0; j < (f - i); j++) {
            if (compararFila(matriz, j, j+1, c) == 1) {

                intercambiarFilaCompleta(matriz, j, j+1, c);
            }
        }
    }
}
```

Se hace el burbujeo para las filas solamente (matriz[**m**, n])

Se compara la fila j con la j+1

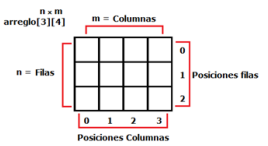
Se intercambian las filas completas para ordenar



Algoritmos de Ordenamiento

Ordenar Matriz por Fila

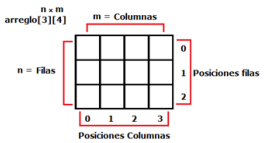
```
//Algoritmo OrdenMatrices
let indice : number ;
let f : number = 10;
let c : number = 8;
let matriz : number[][] = new Array(f);
for (indice=0; indice< f; indice++) {
   matriz[indice] = new Array(c);
}
cargar(matriz, f, c, 10);
//Sin ordenar
console.log("Sin ordenar");
mostrarMatriz(matriz, f, c);
//Ordeno
burbuja(matriz, f, c);
console.log("Ordenada");
mostrarMatriz(matriz, f, c);
```



Algoritmos de Ordenamiento

Ordenar Matriz por Fila (Versión 2.0)

- Calculamos **muchas veces** la suma de los elementos de cada fila
- Que pasa si la matriz tiene un **número muy grande** de filas y columnas?
- Podemos utilizar alguna **estructura** para asegurarnos solamente calcular una **única** vez la suma de los elementos en una fila?



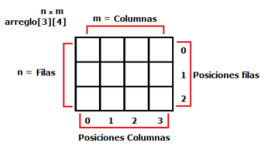
Algoritmos de Ordenamiento

Ordenar Matriz por Fila (Versión 2.0)

Carga en un arreglo “arregloSuma” de tamaño m la suma de los valores de cada fila en una matriz

(es decir, cada posición i del arreglo representa la suma de la fila i de la matriz)

```
function calcularArregloSuma(matriz:number[][], f:number, c:number,
arregloSuma:number[]) {
        let i:number;
        for (i = 0; i < f; i++) {
                arregloSuma[i] = calcularSumaFila(matriz, i, c);
        }
}
```

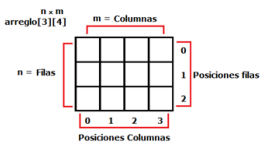


Algoritmos de Ordenamiento

Ordenar Matriz por Fila (Versión 2.0)

```
function comparar(a:number[][], i:number, j:number):number {
    let comparacion:number;
    if (a[i] = a[j]) {
        comparacion = 0;
    } else if (a[i] < a[j]) {
        comparacion = -1;
    } else {
        comparacion = 1;
    }
    return comparacion;
}
```

Es la misma comparación de arreglos de siempre, pero ahora lo vamos a utilizar con “arregloSuma”



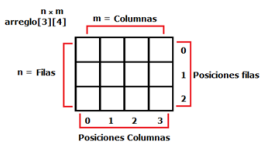
Algoritmos de Ordenamiento

Ordenar Matriz por Fila (Versión 2.0)

```
function burbuja(matriz:number[][], f:number, c:number, arregloSuma:number[]) {
    let i:number, j:number;
    for (i = 2; i < f; i++) {
        for (j = 0; j < (f - i); j++) {
            if (comparar(arregloSuma, j, j+1) == 1 ) {
                intercambiarFilaCompleta(matriz, j, j+1, c);
                intercambiar(arregloSuma, j, j+1);
            }
        }
    }
}
```

Calculamos previamente la suma de las filas y las pasamos como parámetro

Comparamos los valores de "arregloSuma"



Intercambiamos no solo las filas de la matriz sino también los resultados de "arregloSuma"

Algoritmos de Ordenamiento

Ordenar Matriz por Fila (Versión 2.0)

```
//Algoritmo OrdenMatrices
let indice :number;
let f :number = 10;
let c :number = 8;
let matriz :number[][] = new Array(f);
for (indice=0; indice< f; indice++) {
   matriz[indice] = new Array(c);
};
cargar(matriz, f, c, 10);
let  arregloSuma :number[] = new Array(f);
calcularArregloSuma(matriz, f, c, arregloSuma);
//Sin ordenar
console.log("Sin ordenar");
mostrarMatriz(matriz, f, c);
console.log("ArregloSuma = ");
escribirEnUnaLinea(arregloSuma, f);
//Ordenado
burbuja(matriz, f, c, arregloSuma);
console.log("Ordenada");
mostrarMatriz(matriz, f, c);
console.log("ArregloSuma = ");
escribirEnUnaLinea(arregloSuma, f);
```