Técnicas de Programación

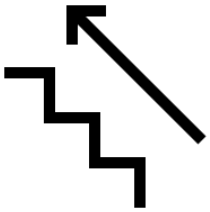
Carrera Programador full-stack

Algoritmos Básicos (Conceptos)

Algoritmos Básicos

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

Recorrido



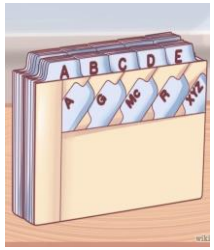
Búsqueda

Ordenamiento

Algoritmos de Ordenamiento

Objetivo y Alternativas

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



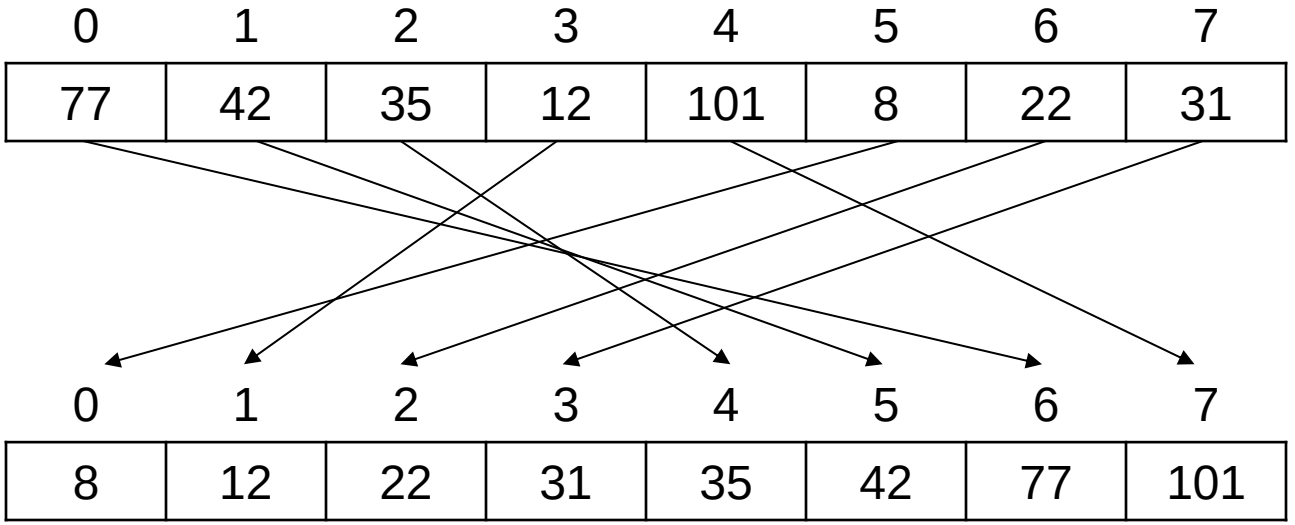
  - Orden alfabético descendente (de la Z a la A)
  - Orden numérico ascendente (0 a infinito)

- 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

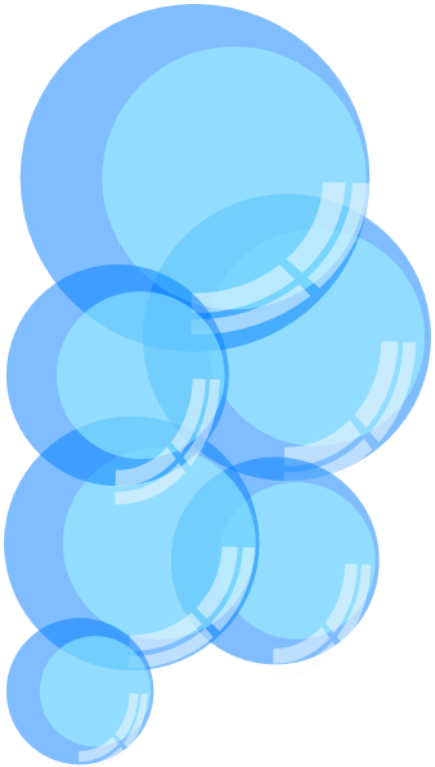
Tipos de Algoritmos

- 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**



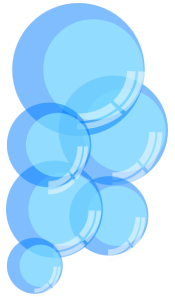
Algoritmos de Ordenamiento

Burbuja (video)

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

Algoritmos de Ordenamiento

Burbuja (razonamiento)

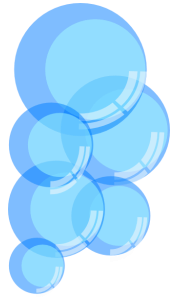


Como se codifica:

- 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)

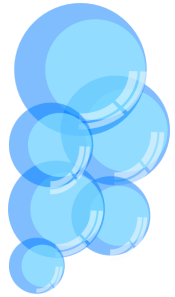


Este método permite cargar un arreglo “arreglo” de dimensión “cantidad” y llenarlo de valores generados al azar entre 0 y “numAzar” (parámetro)

```
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)

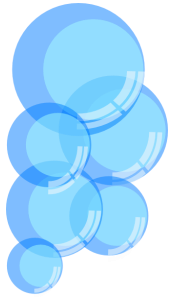


Este método permite mostrar un arreglo “arreglo” de dimensión “cantidad” en una única línea, separando los valores con un espacio

```
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)

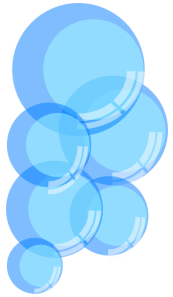


Este método permite intercambiar los valores en las posiciones “i” y “j” de un arreglo “arreglo” utilizando una variable auxiliar

```
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)



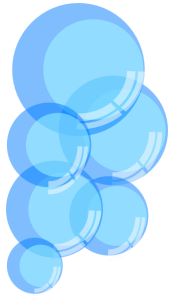
Este método permite comparar los valores en las posiciones “i” y “j” del arreglo “arreglo”

- 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”

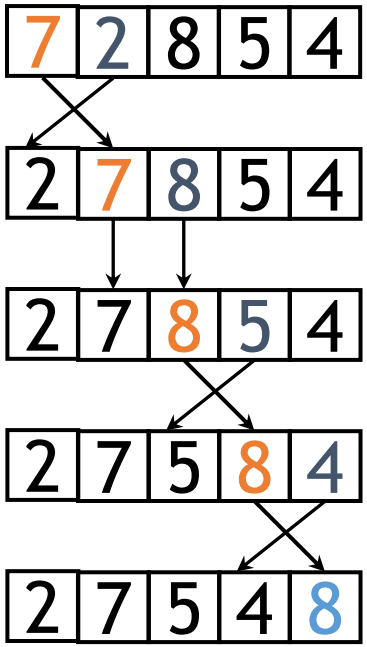
```
function comparar(arreglo : number[], i : number, j :
number) : number {
        let comparacion : number;
        if (arreglo[i] === arreglo[j]) {
                comparacion = 0;
        } else if (arreglo[i] < arreglo[j]) {
                comparacion = -1;
        } else {
                comparacion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

Algoritmos de Ordenamiento

Burbuja (ejemplo)

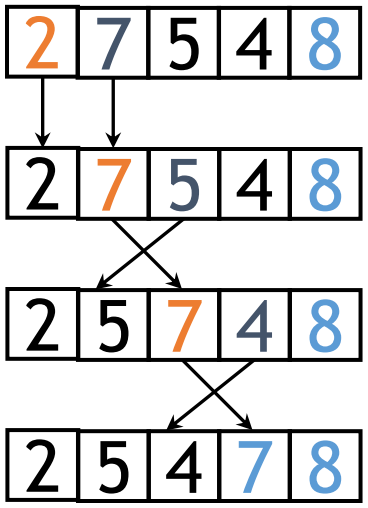


i=2



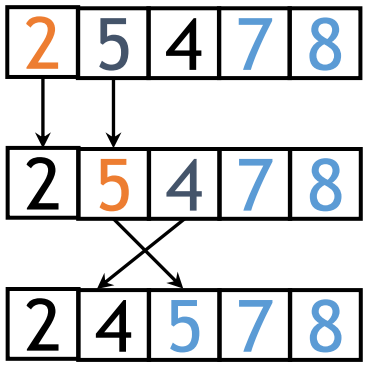
j=3 j+1=4

i=3



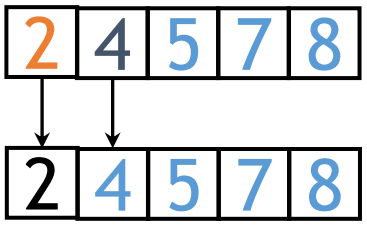
j=2 j+1=3

i=4



j=1 j+1=2

i=5



j=0 j+1=1

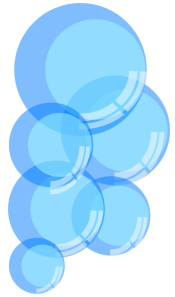
N=5

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) ;
                        }
                }
        }
}
```

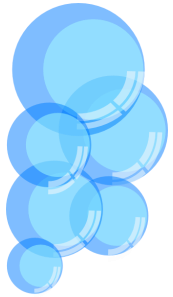
Desde 2 hasta n (el primer elemento esta ordenado en la ultima vuelta)

Desde 0 hasta n – i (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)

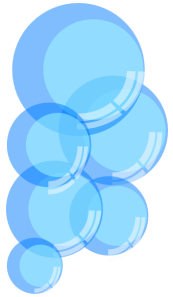


```
//Algoritmo Orden

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)

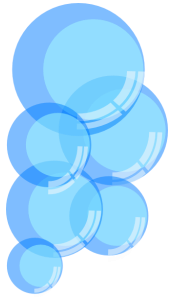


```
let arregloBur = [1,7,4,8,5,9,3];
console.log(burbuja(arregloBur,7));


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 (eficiencia)

- Complejidad: n^2 (dos loops)

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

Técnicas de Programación

Carrera Programador full-stack

Ejercicio

EJERCICIO ENTREGABLE – 28/10

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