# **Algoritmos de Ordenamiento de Elementos de un Arreglo**

Rafael Melgarejo Heredia Noviembre 2019

A continuación, se ilustran cinco tipos de algoritmos de ordenamiento: burbuja, selección, inserción, Shell, y el recursivo Quick sort.

## Ordenamiento de intercambio directo (Burbuja o Bubble sort)

El algoritmo examina cada elemento de la lista intercambiando de posición si están en el orden equivocado. El algoritmo revisa varias veces toda la lista hasta que no se necesiten más intercambios, es decir, hasta que la lista está ordenada. El intercambio es directo porque solo usa comparaciones para intercambiar o no elementos.

Su complejidad depende del número de elementos del arreglo: n2

Se llama burbuja por la forma en que “suben” de posición en la lista los elementos durante los intercambios, como pequeñas “burbujas”.

<https://es.wikipedia.org/wiki/Ordenamiento_de_burbuja>

//Código en PSeInt

Algoritmo burbuja

Dimension v1[21];

Definir v1 Como Entero;

// Genera el arreglo v1 con números al azar

Para i<-1 Hasta 20 Con Paso 1 Hacer

v1[i]<- azar(100)

FinPara

// Ordena los elementos de v1 ascendentemente intercambiando posición

Para i<-2 Hasta 20 Hacer

Para j<-1 Hasta 21-i Hacer

Si v1[j]>v1[j+1] Entonces

aux<-v1[j]

v1[j]<-v1[j+1]

v1[j+1]<-aux

FinSi

FinPara

FinPara

// Escribe el arreglo ordenado

Para i<-1 hasta 20 Hacer

Escribir "posición " i " es: " v1[i];

FinPara

FinAlgoritmo

## Ordenamiento por selección

El algoritmo encuentra el mayor elemento del vector e intercambia su posición con el último elemento del arreglo, repitiendo el procedimiento con los elementos 1 hasta N-1, hasta que termine con todos los números. También es un método de intercambio igual que el de burbuja, con complejidad n2, pero mejora en algo el tiempo dependiendo del orden inicial de los elementos, por ello se lo ubica como inestable.

<https://es.wikipedia.org/wiki/Ordenamiento_por_selecci%C3%B3n>

//Código en PSeInt

Algoritmo seleccion

Dimension v1[21];

Definir v1 Como Entero;

// Genera el arreglo v1 con números al azar

Para i<-1 Hasta 20 Con Paso 1 Hacer

v1[i]<- azar(100)

FinPara

// Pone el mayor al último y luego repite con los otros elementos

Para i<-1 hasta 20 hacer

min<-i;

Para j<-i Hasta 20 Hacer

Si v1[j]<v1[min] entonces

min<-j;

FinSi

FinPara

aux<-v1[i];

v1[i]<-v1[min];

v1[min]<-aux;

FinPara

// Escribe el arreglo ordenado

Para i<-1 hasta 20 hacer

Escribir "posicion " i " es: " v1[i];

FinPara

FinAlgoritmo

## Ordenamiento por inserción

Método natural de ordenamiento. No es de intercambio sino de inserción de elementos y su complejidad en el peor de los casos es n2, mejorando en promedio el tiempo de ejecución a los dos anteriores.

Inicia con un elemento, cuando hay k elementos ordenados de menor a mayor toma el elemento k+1 y compara con los anteriores ya ordenados, deteniéndose cuando encuentra un elemento menor “k+1” insertándolo en el arreglo y desplazando los demás elementos hacia la derecha.

<https://es.wikipedia.org/wiki/Ordenamiento_por_inserci%C3%B3n>

Algoritmo OrdenaPorInsercion

Dimension v1[21];

Definir v1 Como Entero;

// Genera el arreglo v1 con números al azar

Para i<-1 Hasta 20 Con Paso 1 Hacer

v1[i]<- azar(100)

FinPara

// Compara e inserta

i<- 1

Mientras i <= 20

aux<-v1[i]

j<- i - 1

Mientras j>0 & v1[j] > aux

v1[j+1]<-v1[j];

j<- j-1

FinMientras

v1[j+1]<-aux

i<-i+1

FinMientras

// Escribe arreglo ordenado

Para i<-1 hasta 20 hacer

Escribir "posición " i " es: " v1[i];

FinPara

FinAlgoritmo

## Ordenamiento Shell

Parecido al de la burbuja, pero no ordena elementos adyacentes, sino que utiliza una segmentación entre los datos que puede ser de cualquier tamaño de acuerdo a una secuencia de valores que empiezan con un valor grande (pero menor al tamaño total de la estructura) y van disminuyendo hasta llegar a 1. Su complejidad es n log2 n que es menor a n2. La segmentación de datos es aleatoria y puede mejorar o empeorar el rendimiento del algoritmo. El algoritmo a continuación utiliza la secuencia de Knuth.

<https://en.wikipedia.org/wiki/Shellsort>

//Código en PSeInt

Algoritmo OrdenaShell

Dimension v1[21];

Definir v1 Como Entero

Definir h Como Entero

// Genera el arreglo v1 con números al azar

Para i<-1 Hasta 20 Con Paso 1 Hacer

v1[i]<- azar(100)

FinPara

// Genera un valor grande respecto al número de elementos iniciales

h<-1

Mientras h<=20 Hacer

h<-h\*3

h<-h+1

FinMientras

Escribir h

Repetir

h<-trunc(h/3); // Reduce el tamaño de h

Para j<-h+1 Hasta 20 Con Paso 1 Hacer

aux<-v1[j]

i<-j-h

Mientras i>0 & v1[i]>aux Hacer

v1[i+h]<-v1[i]

i<-i-h

Fin Mientras

v1[i+h]<-aux

Fin Para

Hasta Que h<=1

// Escribe el arreglo ordenado

Para i<-1 hasta 20 hacer

Escribir "posicion " i " es: " v1[i];

FinPara

FinAlgoritmo

## Ordenamiento rápido

Quicksort se considera el algoritmo más eficiente. El algoritmo selecciona un valor del arreglo llamado Pivote (comparador) y divide el arreglo en dos partes, a un lado los elementos mayores o iguales al pivote y al otro lado todos los elementos menores. Este proceso se repite por cada segmento restante hasta que el arreglo esté ordenado. Este es un algoritmo recursivo por naturaleza. Su complejidad es n log n.

El pivote se obtiene de dos formas, sea aleatoriamente, o haciendo la media de un subconjunto de valores del arreglo. Es preferible utilizar como pivote al valor que este precisamente en medio del rango de valores, pero no siempre será la manera más óptima.

<https://en.wikipedia.org/wiki/Quicksort>

Código en JavaScript

<!DOCTYPE html>

<HTML>

<HEAD>

<TITLE>Quick Sort</TITLE>

</HEAD>

<BODY onload="empezar();">

<SCRIPT type="text/javascript">

function quickSort(arr, left, right)

{

var len = arr.length, pivot, partitionIndex;

if(left < right)

{

pivot = right;

partitionIndex = partition(arr, pivot, left, right);

//sort left and right

quickSort(arr, left, partitionIndex - 1);

quickSort(arr, partitionIndex + 1, right);

}

return arr;

}

function partition(arr, pivot, left, right)

{

var pivotValue = arr[pivot],

partitionIndex = left;

for(var i = left; i < right; i++)

{

if(arr[i] < pivotValue){

swap(arr, i, partitionIndex);

partitionIndex++;

}

}

swap(arr, right, partitionIndex);

return partitionIndex;

}

function swap(arr, i, j)

{

var temp = arr[i];

arr[i] = arr[j];

arr[j] = temp;

}

function empezar(){

var v1 = new Array(20);

var v1 = new Number();

// Genera cada uno de los 20 elementos del arreglo v1

for (i=1;i<=20;i++) {

v1[i-1] = Math.floor(Math.random()\*10000);

}

quickSort(v1,0,19);

for (i=1;i<=20;i++) {

document.write("posicion ",i," es: ",v1[i-1],'<BR/>');

}

}

</SCRIPT>

</BODY>

</HTML>

REFERENCIAS

<http://artemisa.unicauca.edu.co/~nediaz/EDDI/cap03.htm>