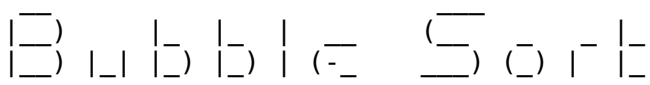
"Análisis de Algoritmos"
José Helo Guzmán



Contenido

▬▬▬ 1. Introducción.

El “bubble sort”, es uno de los algoritmos más sencillos para ordenar números. Realiza el proceso cambiando de forma repetida la posición de los elementos de una lista, si estos elementos no se encuentran en la posición correcta.

▬▬▬ 2. Funcionamiento del Bubble Sort.

Supongamos que tenemos la siguiente lista:

[5, 1, 4, 2, 8]

Y deseamos ordenarla utilizando el algoritmo del “bubble sort”. Lo que se busca son iteraciones sucesivas, cada una lleva al final de la lista al número de mayor valor de la siguiente manera:

>>> Primer paso:

```
[ 5, 1, 4, 2, 8 ]
  ^  ^
```

Se comparan los primeros dos elementos y se cambian, pues 5 > 1.

```
[ 1, 5, 4, 2, 8 ]
     ^  ^
```

Cambia los elementos pues 5 > 4

```
[ 1, 4, 5, 2, 8 ]
        ^  ^
```

Cambia los elementos pues 5 > 2

```
[ 1, 4, 2, 5, 8 ]
           ^  ^
```

No hay cambio pues 5 < 8
Y se ha terminado de procesar la lista.

>>> Segundo Paso.

Se realiza de nuevo el mismo proceso.

```
[ 1, 4, 2, 5, 8 ]
  ^  ^

[ 1, 4, 2, 5, 8 ]
     ^  ^

[ 1, 2, 4, 5, 8 ]
        ^  ^

[ 1, 2, 4, 5, 8 ]
           ^  ^
```

>>> Tercer Paso.

Se realiza de nuevo el proceso.

```
[ 1, 2, 4, 5, 8 ]
  ^  ^

[ 1, 2, 4, 5, 8 ]
     ^  ^

[ 1, 2, 4, 5, 8 ]
        ^  ^

[ 1, 2, 4, 5, 8 ]
           ^  ^
```

El algoritmo ya está ordenado.
Es posible que el algoritmo realice otras iteraciones antes de finalizar.

▬▬▬ 3. Bubble Sort con Subíndices.

Veamos un ejemplo donde se muestran los subíndices:

Supongamos que tenemos la siguiente lista:

[50, 30, 10, 90, 80, 20, 40, 70]

i=1	j	1	2	3	4	5	6	7	8
	1	50*	30*	10	90	80	20	40	70
	2	30	50*	10*	90	80	20	40	70
	3	30	10	50*	90*	80	20	40	70
	4	30	10	50	90*	80*	20	40	70
	5	30	10	50	80	90*	20*	40	70
	6	30	10	50	80	20	90*	40*	70
	7	30	10	50	80	20	40	90*	70*
	-	30	10	50	80	20	40	70	(90)

i=2	j	1	2	3	4	5	6	7	8
	1	30*	10*	50	80	20	40	70	90
	2	10	30*	50*	80	20	40	70	90
	3	10	30	50*	80*	20	40	70	90
	4	10	30	50	80*	20*	40	70	90
	5	10	30	50	20	80*	40*	70	90
	6	10	30	50	20	40	80*	70*	90
	-	10	30	50	20	40	70	(80	90)

i=3	j	1	2	3	4	5	6	7	8
	1	10*	30*	50	20	40	70	80	90
	2	10	30*	50*	20	40	70	80	90
	3	10	30	50*	20*	40	70	80	90
	4	10	30	20	50*	40*	70	80	90
	5	10	30	20	40	50*	70*	80	90
	-	10	30	20	40	50	(70	80	90)

i=4	j	1	2	3	4	5	6	7	8
	1	10*	30*	20	40	50	70	80	90
	2	10	30*	20*	40	50	70	80	90
	3	10	20	30*	40*	50	70	80	90
	4	10	20	30	40*	50*	70	80	90
	-	10	20	30	40	(50	70	80	90)

i=5	j	1	2	3	4	5	6	7	8
	1	10*	20*	30	40	50	70	80	90
	2	10	20*	30*	40	50	70	80	90
	3	10	20	30*	40*	50	70	80	90
	-	10	20	30	(40	50	70	80	90)

i=6	j	1	2	3	4	5	6	7	8
	1	10*	20*	30	40	50	70	80	90
	2	10	20*	30*	40	50	70	80	90
	-	10	20	(30	40	50	70	80	90)

i=7	j	1	2	3	4	5	6	7	8
	1	10*	20*	30	40	50	70	80	90
	-	10	(20	30	40	50	70	80	90)

▬▬▬ 4. Tiempo de Ejecución.

El peor caso para este algoritmo ocurre cuando se tiene una lista ordenada de forma inversa. Es decir, si ordenamos de menor a mayor, el peor caso ocurre cuando debemos de ordenar esta lista de menor a mayor.

El ejemplo anterior brinda un ejemplo de este proceso. Veamos el número de comparaciones que se necesitan.

De esta forma:

En el paso 1 se requieren: (n - 1) comparaciones.
En el paso 2 se requieren: (n - 2) comparaciones.
En el paso 3 se requieren: (n - 3) comparaciones.
.
.
.
En el paso (n-1) se requiere: 1 comparación.

Entonces, para estimar el número total de comparaciones se tiene:

$$(n-1) + (n-2) + \ldots + 1 = \left(\frac{n*(n-1)}{2} \right)$$

$$\left(\frac{n*(n-1)}{2} \right) = \left(\frac{n^2 - n}{2} \right) = \left(\frac{n^2}{2} - \frac{n}{2} \right)$$

Que es un polinomio de grado 2 y su orden es de $O(n^2)$

▬▬▬ 5. Una pequeña modificación.

Se puede mejorar levemente el algoritmo si se utiliza un indicador que se activa cuando se han producido cambios en una pasada. Si no se han producido cambios, significa que la lista se encuentra ordenada y se finaliza el proceso.

▬▬▬ 6. Resumen.

• Algunos algoritmos de ordenamiento es más fácil ejecutarlos con listas enlazadas y otros es más fácil con arreglos.

• El “bubble sort” va ordenando en cada iteración un valor. Y de esta manera disminuye poco a poco el número de iteraciones que tiene que realizar.

• Para implementar el algoritmo se necesitan dos procedimientos “for” anidados.

• El tiempo de ejecución del algoritmo, en el peor caso, es de $O(n^2)$

▬▬▬ 7. Ejercicios.

●● Ejercicio 1.

Ejecute una corrida en papel, paso a paso, para ordenar cada uno de los siguientes vectores, utilizando el algoritmo de “bubble sort”.

• vec: [1, 2, 4, 6, 8]
• vec: [9, 5, 3, 1]
• vec: [9, 3, 5, 2, 4]

●● Ejercicio 2.

El “bubble sort” se puede mejorar de manera que se detenga cuando la lista está ordenada. Llamaremos a este algoritmo “bubble sort mejorado”.

Una forma algo ineficiente de realizar este trabajo es revisando que la lista esté ordenada en cada iteración.

Otra forma de programar este proceso, es poner una variable booleana en “false” antes de iniciar una iteración, y se ponen en “true”, si se realiza un “swap”.

Programe cada uno de estos procedimientos y pruébelos.

●● Ejercicio 3.

Brinde un ejemplo, de la clase de datos de entrada, con los cuales no sirve para nada revisar si la lista ya ha sido ordenada.
(Sugerencia: Intente un ejemplo con una lista en orden inverso al que se necesita)

●● Ejercicio 4.

Hay alguna manera, en caso que los datos de entrada estén ordenados al revés, de evitar que el algoritmo realice un gran número de operaciones innecesarias?

Se le ocurre alguna solución?

●● Ejercicio 5.

Dado el siguiente arreglo

vec : [1, 2, 4, 3]

Cuántas iteraciones se necesitan para que el algoritmo ponga en orden todos los elementos?

●● Ejercicio 6.

Dado el siguiente arreglo

vec : [1, 2, 4, 3]

Cuántas iteraciones se necesitan para que el algoritmo del “bubble sort mejorado” ponga en orden todos los elementos?