11074 - Programación I

División Computación - Departamento de Ciencias Básicas

VECTORES: ALGORITMOS DE ORDENAMIENTO

Introducción

Ya hemos visto que en muchos casos es muy útil tener el arreglo ordenado de alguna manera. Esto permite que la forma de encontrar algo y de realizar varias operaciones adicionales sea mucho más eficiente y se consuman menos recursos computacionales. A medida que vayamos avanzando en este tema iremos develando estas situaciones.

Métodos de Ordenamiento

A continuación vamos a analizar y diseñar algoritmos para ordenar arreglos (o vectores) que son fácilmente transportables a otros tipos de estructuras.

La idea es pensar juntos cómo se puede realizar esta operación.

Para desarrollar esta actividad, pensemos en lo siguiente:

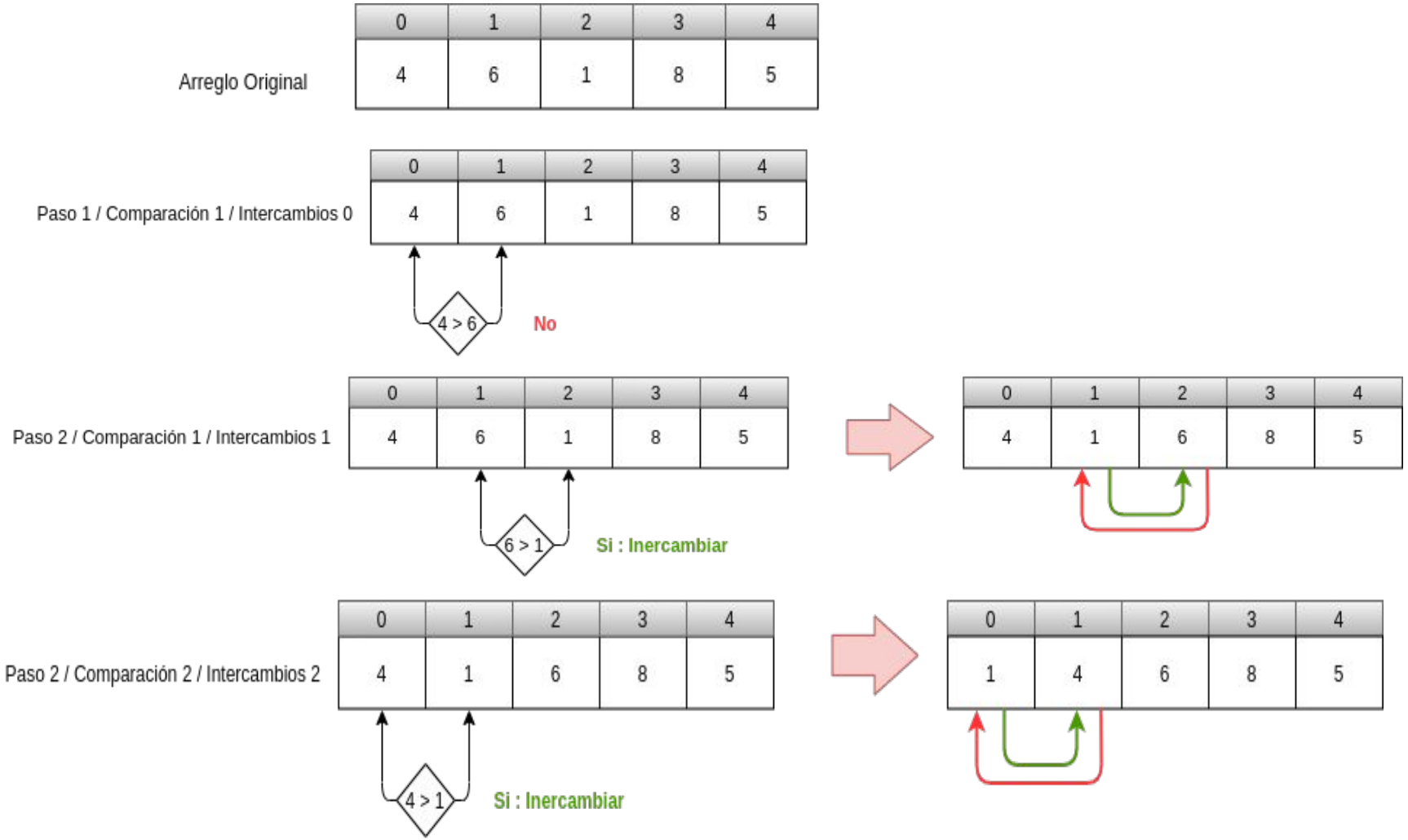
Tenemos un conjunto de rifas que nos dieron para vender. Supongamos que son unos 100 números. Los números van del 1 al 100 y la distribución no es uniforme (es decir que no tengo los números del 1 al 100 ni los números pares entre tal y cual, sino que me dieron números al azar. De hecho, quien lo hizo, primero cortó todos los números, los puso en una bolsa, los batió y me dio las 100 rifas.

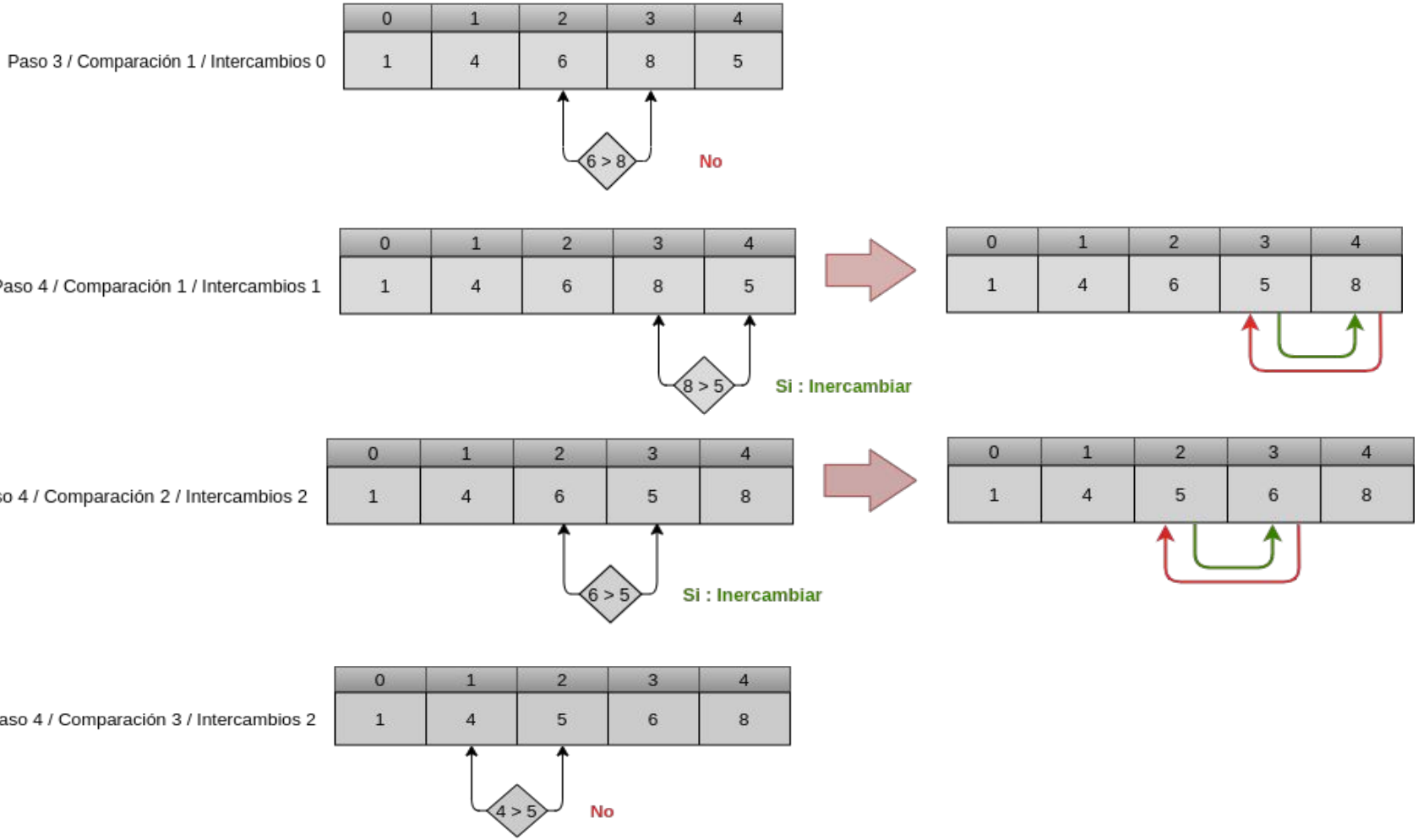
Creo que sería bueno tenerlos ordenados para que la gente que busca los números localice el que quiere más rápido si es que está disponible.

¿De qué manera podríamos proceder para ordenarlos?

Un método natural que usamos algunas personas es tomar el segundo número y compararlo con el primero. Si el segundo es menor al primero, lo intercambiamos, sino, tomamos el tercero y vemos si es menor al segundo. Si lo es, los intercambiamos y preguntamos si es menor al primero y si lo es lo intercambiamos. Luego seguimos por el cuarto y así vamos ordenando todos.

El siguiente esquema muestra este procedimiento con 5 números paso a paso


Arreglo Original
0 1 2 3 4
4 6 1 8 5
Paso 1 / Comparación 1 / Intercambios 0
0 1 2 3 4
4 6 1 8 5
4 > 6
No
Paso 2 / Comparación 1 / Intercambios 1
0 1 2 3 4
4 6 1 8 5
6 > 1
Si : Inercambiar
0 1 2 3 4
4 1 6 8 5
Paso 2 / Comparación 2 / Intercambios 2
0 1 2 3 4
4 1 6 8 5
4 > 1
Si : Inercambiar
0 1 2 3 4
1 4 6 8 5


Paso 3 / Comparación 1 / Intercambios 0
0 1 2 3 4
1 4 6 8 5
6 > 8
No
Paso 4 / Comparación 1 / Intercambios 1
0 1 2 3 4
1 4 6 8 5
8 > 5
Si : Inercambiar
0 1 2 3 4
1 4 6 5 8
Paso 4 / Comparación 2 / Intercambios 2
0 1 2 3 4
1 4 6 5 8
6 > 5
Si : Inercambiar
0 1 2 3 4
1 4 5 6 8
Paso 4 / Comparación 3 / Intercambios 2
0 1 2 3 4
1 4 5 6 8
4 > 5
No

La idea de este algoritmo consiste en considerar que la parte izquierda está ordenada. Pensalo así, si el arreglo tuviera un solo elemento, ¿Estaría ordenado? Claro!! no hay forma de que no. Entonces, si el primer elemento está en orden, ¿qué pasa con el segundo? Si no lo está se lo inserta en el lugar donde va (para esto se va desplazando cada elemento mayor a la derecha). Ahora ya tenemos dos elementos ordenados, ¿Qué pasa con el tercero? …

La cantidad de pasos que va a tener es la cantidad de elementos menos 1. En cada paso, la cantidad de comparaciones y de intercambios serán, como máximo, el paso en el que estemos. Si estamos en el paso 2, solo podemos tener dos comparaciones y dos intercambios (puede haber menos que eso). Mirá en el ejemplo que pasó con el paso 2. Después observa el paso 4. Las comparaciones se detienen cuando detectamos que el elemento analizado ya está ordenado, que puede suceder cuando llegamos al primer elemento del arreglo o cuando el elemento que está en la posición anterior es menor o igual.

A este algoritmo se lo llama: **Ordenamiento por Inserción**

```
void ordenar_insercion(int datos[N], int n) {
    int i, j;
    int temp;
    for (i = 1; i < n; i++) {
        j=i;
        while ((j >0) && (datos[j-1] >datos[j])) {
            temp = datos[j-1];
            datos[j-1] = datos[j];
            datos[j] = temp;
            j=j-1;
        }
    }
}
```

Seguimos pensando ...

Otra forma de ordenar un arreglo sería la siguiente: recorremos el vector buscando el número más grande. Lo intercambiamos por el que ocupa la última posición, luego buscamos el más grande entre el primero y el anteúltimo, y lo intercambiamos por éste. Luego buscamos el más grande entre el primero y el antepenúltimo y lo intercambiamos. Y así hasta que queden todos ordenados.

Como se observará, en esta representación los máximos van acomodándose uno a uno al final del arreglo, dejando ordenada la última parte. Es decir, se selecciona el mayor y luego se intercambia. Por eso, a este algoritmo se le llama **Ordenamiento por Selección.**

Te lo representamos en el siguiente esquema:

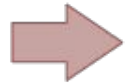
Arreglo Original

0	1	2	3	4
4	6	1	8	5

Paso 1 / localizamos el mayor entre 0 y 4

0	1	2	3	4
4	6	1	8	5

El mayor es el 8 en la posición 3

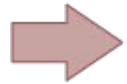


0	1	2	3	4
4	6	1	5	8

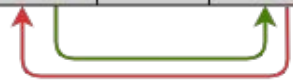
Paso 2 / localizamos el mayor entre 0 y 3

0	1	2	3	4
4	6	1	5	8

El mayor es el 6 en la posición 1



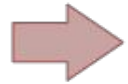
0	1	2	3	4
4	5	1	6	8



Paso 3 / localizamos el mayor entre 0 y 2

0	1	2	3	4
4	5	1	6	8

El mayor es el 5 en la posición 1

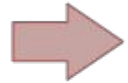


0	1	2	3	4
4	1	5	6	8

Paso 4 / localizamos el mayor entre 0 y 1

0	1	2	3	4
4	1	5	6	8

El mayor es el 4 en la posición 0



0	1	2	3	4
1	4	5	6	8

```
void ordenar_seleccion(int datos[], int n) {
    int i, j, aux;
    for (i = 0; i < n-1; i++)
        for (j = i+1; j < n; j++)
            if (datos[i] > datos[j]) {
                /* Intercambio */
                aux = datos[i];
                datos[i] = datos[j];
                datos[j] = aux;
            }
}
```

¿Se puede hacer de otra forma?

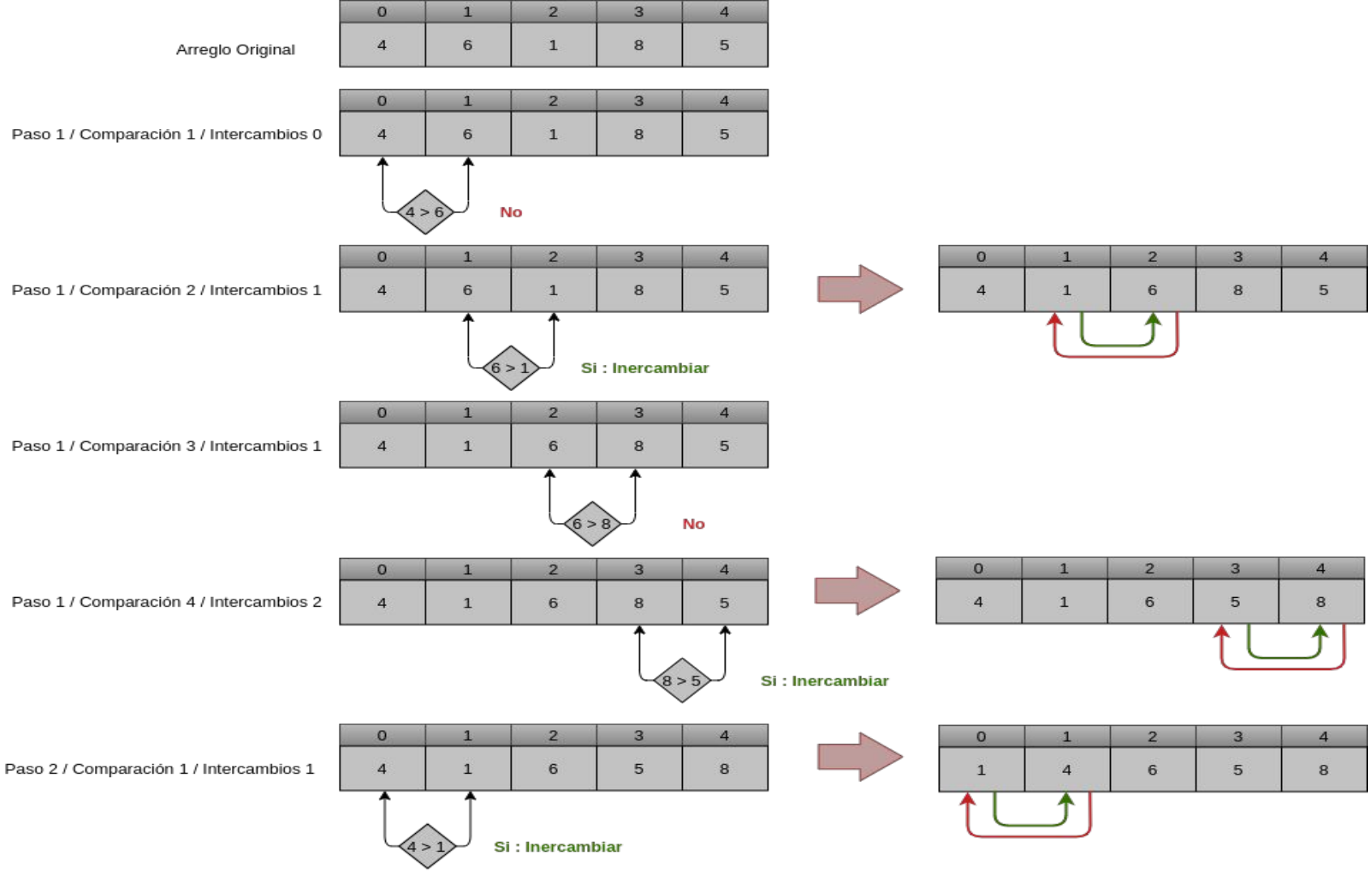
¡ Claro !

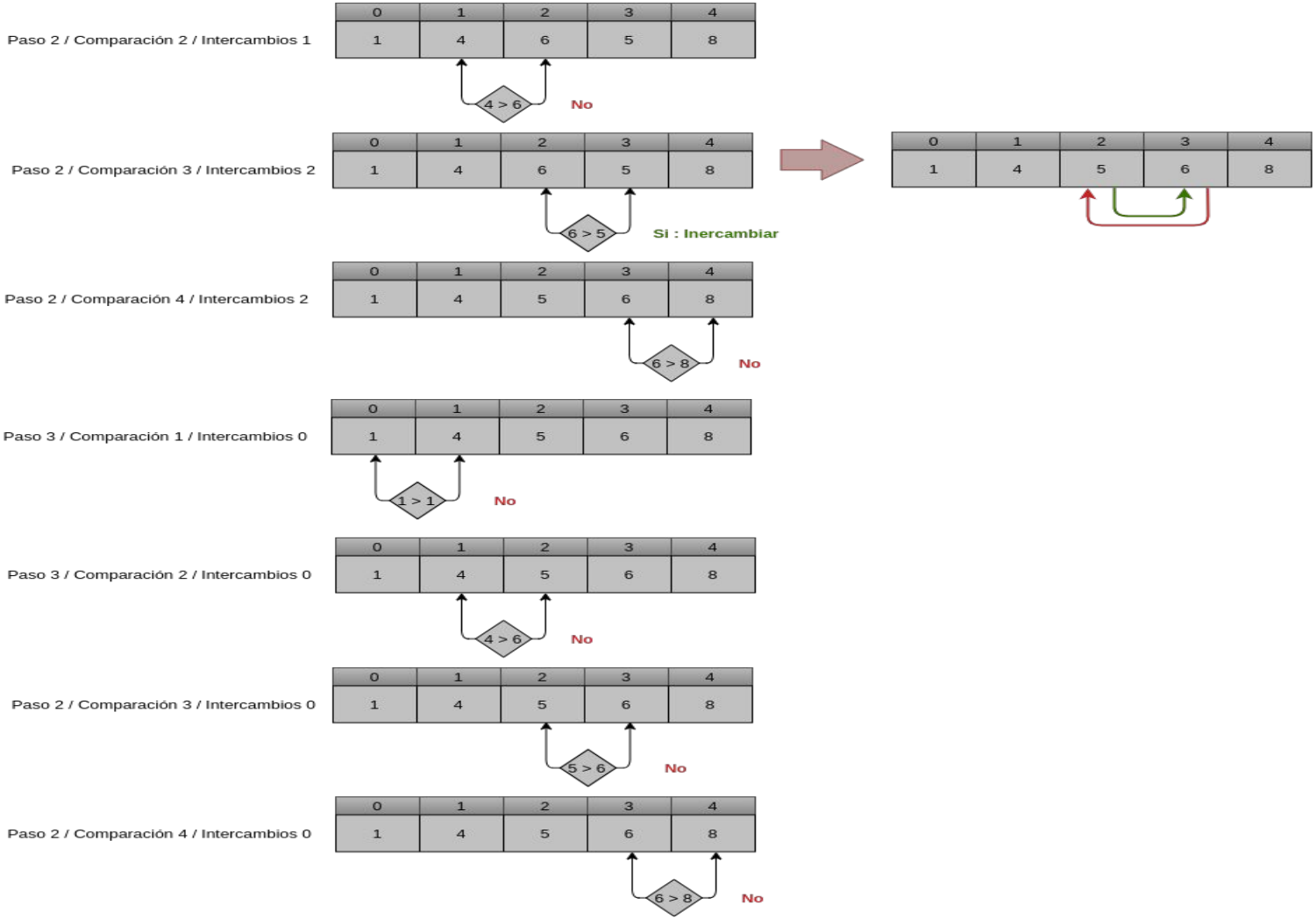
Otra forma de ordenar es aprovechando la idea anterior de buscar un máximo, pero hacer intercambios cada vez que encontramos un nuevo valor más grande.

Esto sería así, se comparan los elementos adyacentes en un arreglo y se intercambian en caso de que no se encuentren en el orden deseado. Simplemente eso.

¿Cómo? Claro, vamos comparando uno a uno los elementos con el que sigue, si está desordenado, se ordena, sino se toma el que sigue. Luego se vuelve a empezar. ¿Cuántas veces? El primer planteo podría ser, hacerlo una vez por cada elemento que tenga el arreglo. Pero en realidad, no es necesario. Podríamos poner una bandera que indique si en la última pasada se realizó o no un intercambio. Si no se realizaron intercambios, ya no se sigue más porque esto implica que no importa cuantas veces me falta recorrerlo, todos sus elementos ya están ordenados.

Mirá este esquema:


Arreglo Original
0 1 2 3 4
4 6 1 8 5
Paso 1 / Comparación 1 / Intercambios 0
4 > 6
No
Paso 1 / Comparación 2 / Intercambios 1
6 > 1
Si : Inercambiar
0 1 2 3 4
4 1 6 8 5
Paso 1 / Comparación 3 / Intercambios 1
6 > 8
No
Paso 1 / Comparación 4 / Intercambios 2
8 > 5
Si : Inercambiar
4 1 6 5 8
Paso 2 / Comparación 1 / Intercambios 1
4 > 1
Si : Inercambiar
1 4 6 5 8


Paso 2 / Comparación 2 / Intercambios 1
0 1 2 3 4
1 4 6 5 8
4 > 6
No
Paso 2 / Comparación 3 / Intercambios 2
0 1 2 3 4
1 4 6 5 8
6 > 5
Si : Inercambiar
0 1 2 3 4
1 4 5 6 8
Paso 2 / Comparación 4 / Intercambios 2
0 1 2 3 4
1 4 5 6 8
6 > 8
No
Paso 3 / Comparación 1 / Intercambios 0
0 1 2 3 4
1 4 5 6 8
1 > 1
No
Paso 3 / Comparación 2 / Intercambios 0
0 1 2 3 4
1 4 5 6 8
4 > 6
No
Paso 2 / Comparación 3 / Intercambios 0
0 1 2 3 4
1 4 5 6 8
5 > 6
No
Paso 2 / Comparación 4 / Intercambios 0
0 1 2 3 4
1 4 5 6 8
6 > 8
No

Como se observa, en esta representación los máximos van acomodándose uno a uno al final del arreglo, dejando ordenada la última parte. Se dice que los máximos van burbujeando hasta llegar a su posición y por eso se le llama **Ordenamiento por Burbuja .**

```
void ordenar_burbuja(int datos[N], int n) {
    int i, j, aux;
        for(i=0;i<n;i++)
            for (j = 1; j <n - i; j++){
                if (datos[j-1] > datos[j]) {
                    /* Intercambio */
                    aux =  datos[j];
                    datos[j] = datos[j-1];
                    datos[j-1] = aux;
                }
            }
}
```

Dijimos que podemos optimizarlo, usando una bandera que indica que ya no es necesario seguir el procedimiento pues el vector ya está ordenado

```
void ordenar_burbuja(int datos[N], int n) {
    int i, j, nointer, aux;
    i=1;     nointer=0;
    for(i=0;i<n && nointer==0;i++)  {
        nointer=1;
        for (j = 1; j <n - i; j++){
            if (datos[j-1] > datos[j]) {                    /* Intercambio */
                aux =  datos[j];
                datos[j] = datos[j-1];
                datos[j-1] = aux;
                nointer=0;
            }
        }
    }
}
```

Mezcla de arreglos.

La mezcla de arreglos consiste en generar un arreglo ordenado a partir de otros ordenados con anterioridad. En lugar de concatenar los dos subarreglos en otro mayor y ordenar el conjunto, se aprovecha que ya existe un orden parcial en las entradas. El mecanismo de mezcla consiste en comparar sendos elementos de cada uno de los arreglos de entrada y se coloca el más pequeño en el arreglo destino, tomando el elemento siguiente del arreglo correspondiente, hasta agotar todos los elementos de uno de ellos. Por último, se copiarán los elementos no procesados del otro arreglo. A continuación se presenta la implementación en lenguaje **C**:

```c
#include <stdio.h>
#define M 4
#define N 5
int main() {
    int arreglo1[] = {2, 3, 5, 8};
    int arreglo2[] = {1, 2, 6, 9, 10};
    int destino[M+N];
    int i, j, k;
    i = 0;
    j = 0;
    k = 0;
```

```
 /* Mezcla */
    while ( i < M && j < N ) {
        if ( arreglo1[i] <= arreglo2[j] ) {
            destino[k] = arreglo1[i];
            i++;
        }
        else {
            destino[k] = arreglo2[j];
            j++;
        }
        k++;
    }


}
```

```
    if (i == M)
        for (; j < N; j++) {  /* 'arreglo1' no tiene más elementos, copiar
'arreglo2' */
            destino[k] = arreglo2[j];
            k++;
        }
    else
        if (j == N)
            for (; i < M; i++)    {
                /* 'arreglo2' no tiene más elementos, copiar 'arreglo1' */
                destino[k] = arreglo1[i];
            k++;
            }
}
```

Hay muchas más formas de ordenar arreglos. Hay un método llamado **Quick Sort** que es muy bueno pero es recursivo y como aún no vimos recursividad en la asignatura, no lo vamos a describir. Durante mucho tiempo fue un tema de investigación permanente. Hoy ya es un tema superado. La mayoría de los lenguajes ofrece un método ya implementado para ordenar colecciones. Nuestra intención en esta guía fue justamente tomar el desafío de pensar distintas formas de ordenar y de esa manera aprender a pensar algoritmos.

Un muy buen sitio sobre métodos de ordenamiento puedes encontrarlo aquí : http://lwh.free.fr/pages/algo/tri/tri_es.htm Lo que tiene de interesante es que podés probar vos mismo con una simulación como se realizan los pasos. Para cada método ofrece un gráfico que se va ordenando y código de ejemplo en varios lenguajes.

"Enseñar no es transferir conocimientos, sino crear las posibilidades de su construcción; quien enseña aprende al enseñar y quien aprende enseña al aprender"

Paulo Freire