

ALGORITMOS Y PROGRAMACIÓN

Clase 9 – Primera parte
Ordenamiento

Temario

- Ordenamiento
 - por intercambio
 - por selección
 - por burbuja

Ordenamiento

- Muchos problemas de la vida real requieren que los datos utilizados se encuentren en un determinado orden.
- El ordenamiento de datos es una operación que consiste en disponer u organizar un conjunto de datos bajo algún criterio específico.
- Por lo general el objetivo final de aplicar un método de ordenamiento es facilitar el proceso de recuperación de los datos.

Ordenamiento por intercambio

- Se basa en el recorrido secuencial de la colección a ordenar, comparando el elemento que se encuentra en *una posición fija* de la colección *contra todos los elementos posteriores* y efectuando intercambio de elementos cuando el orden resultante de la comparación no sea el correcto.
- Este algoritmo necesita realizar $n-1$ pasadas para finalizar la tarea, donde n es la cantidad de elementos de la colección.

Ordenamiento por intercambio

- Si la colección de elementos es:

`datos = [4, 7, 5, 1, 3]`

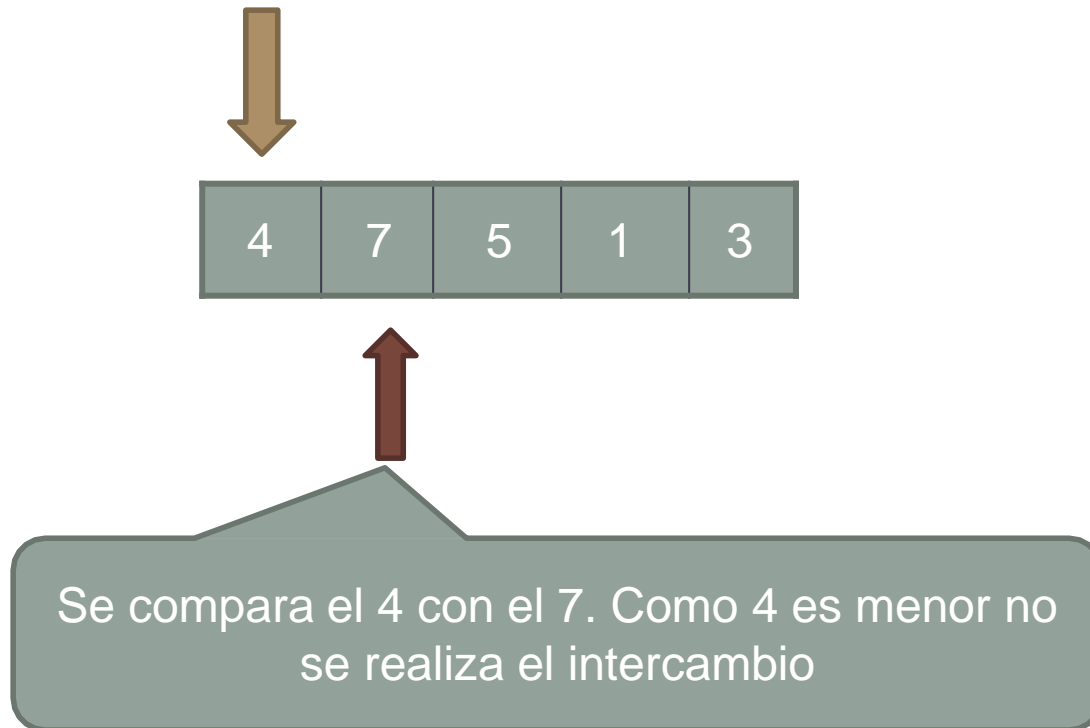
- Se espera que el arreglo quede de esta forma:

1	3	4	5	7
---	---	---	---	---

- Notar que una colección se puede ordenar de manera ascendente o descendente.

Ordenamiento por intercambio

- El elemento que está en la primer posición se compara contra todos los restantes



Ordenamiento por intercambio

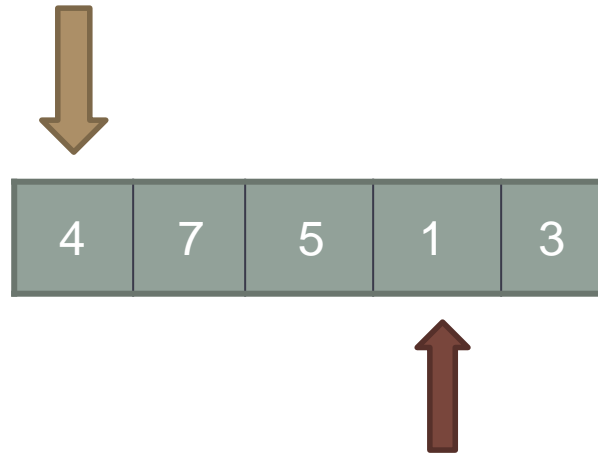
- El elemento que está en la primer posición se compara contra todos los restantes



Se compara el 4 con el 5. Como 4 es menor no se realiza el intercambio

Ordenamiento por intercambio

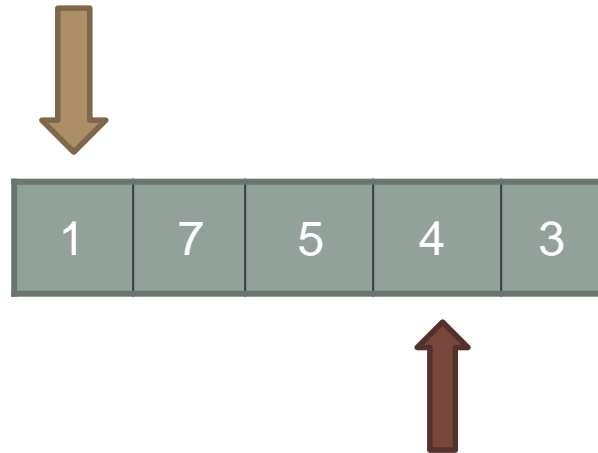
- El elemento que está en la primer posición se compara contra todos los restantes



Se compara el 4 con el 1. En este caso, 1 es menor que 4. Se realiza el intercambio

Ordenamiento por intercambio

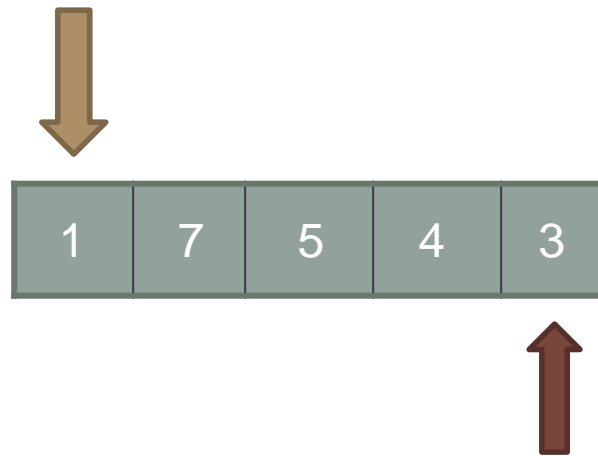
- El elemento que está en la primer posición se compara contra todos los restantes



Se compara el 4 con el 1. En este caso, 1 es menor que 4. Se realiza el intercambio

Ordenamiento por intercambio

- El elemento que está en la primer posición se compara contra todos los restantes



Se compara el 1 con el 3. Como 1 es menor no se realiza el intercambio.

Notar que siempre se compara el elemento que está en la primera posición del arreglo (sin importar cual sea).

Ordenamiento por intercambio

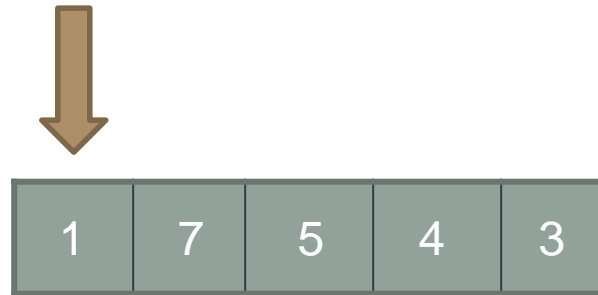
- El elemento que está en la primer posición se compara contra todos los restantes



Finalizó la primer vuelta. ¿Qué se logró?

Ordenamiento por intercambio

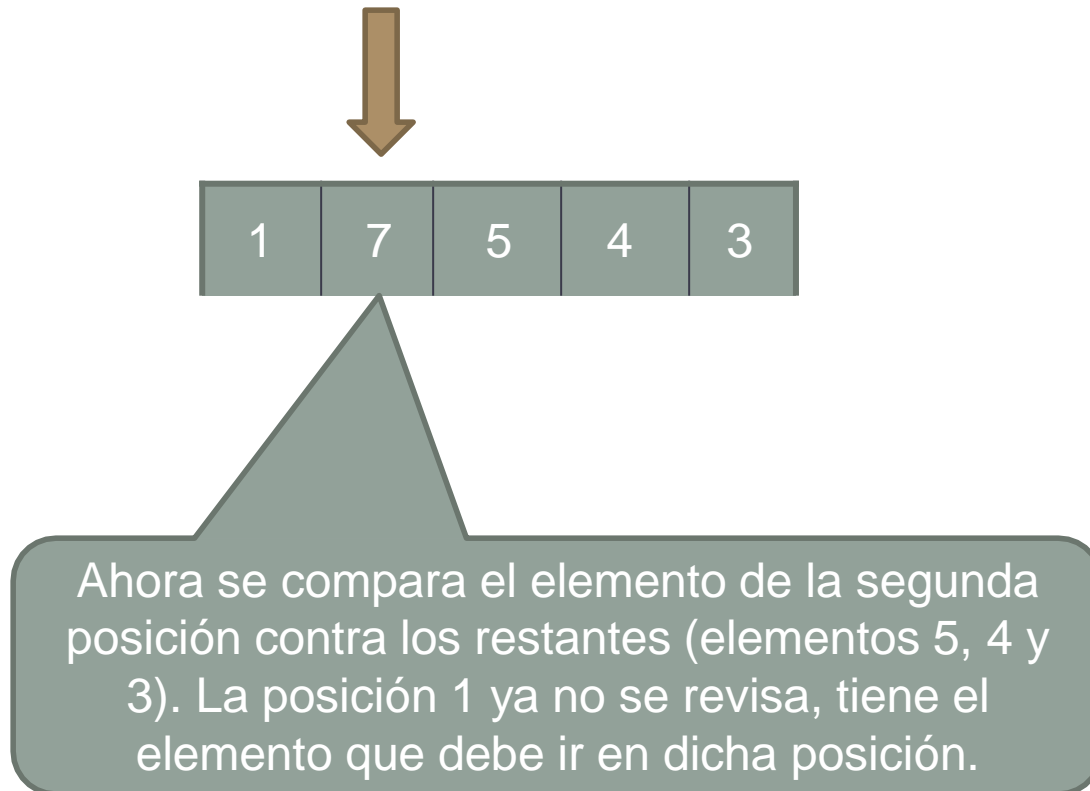
- El elemento que está en la primer posición se compara contra todos los restantes



El elemento más chico está en la primer posición del arreglo.

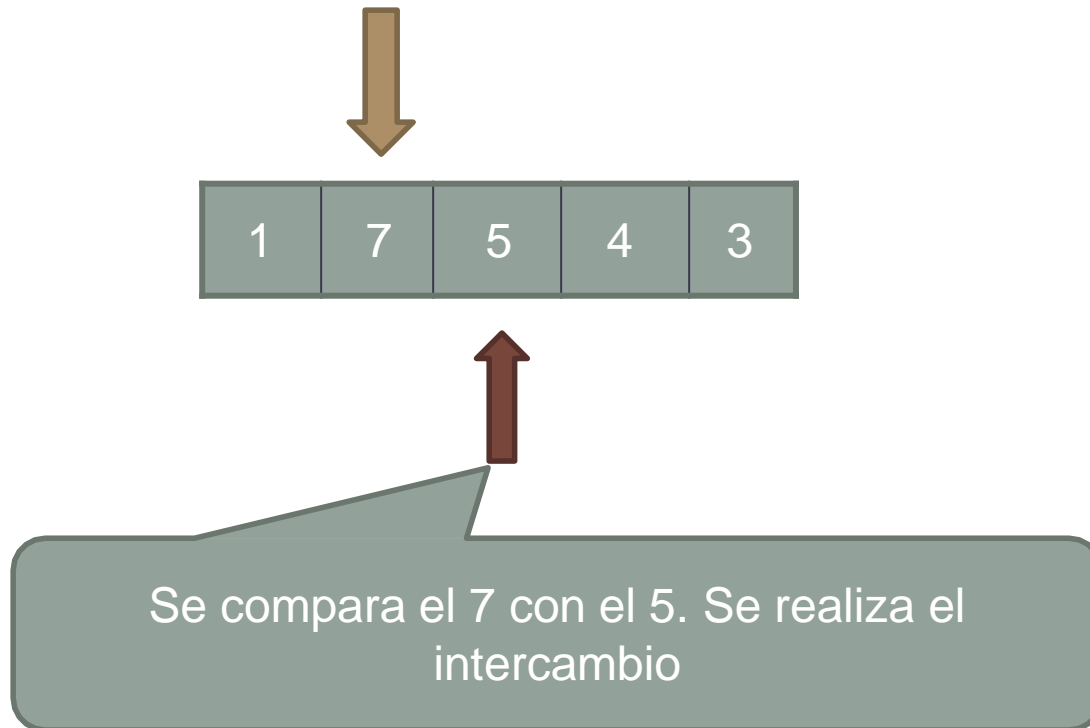
Ordenamiento por intercambio

- El elemento que está en la segunda posición se compara contra todos los restantes



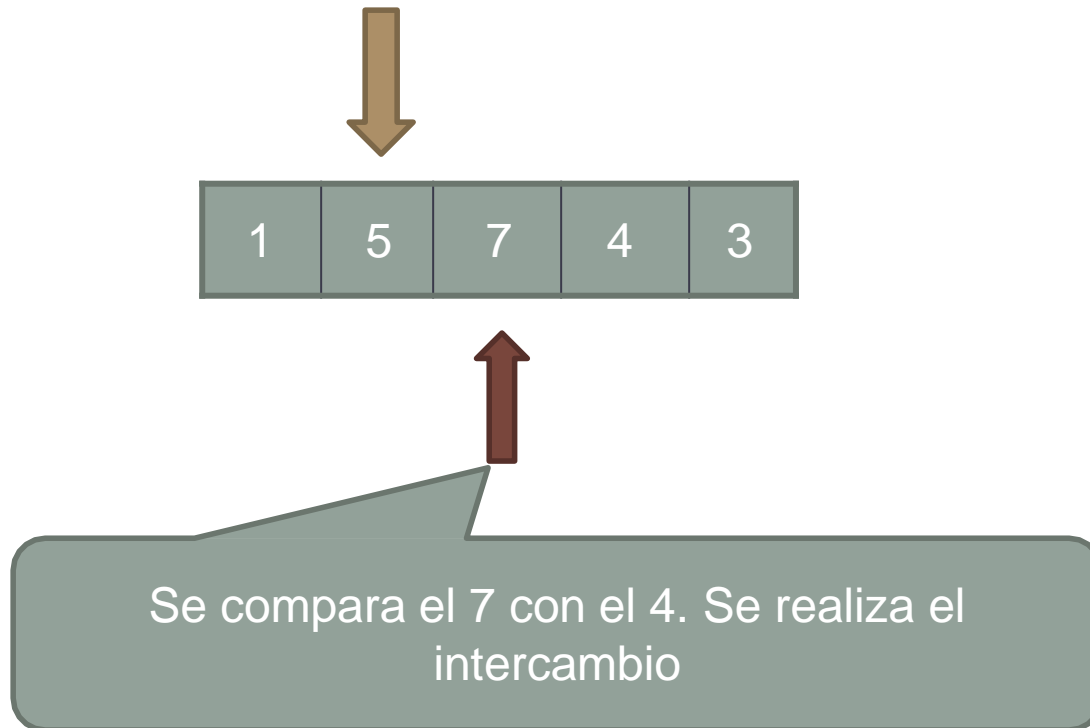
Ordenamiento por intercambio

- El elemento que está en la segunda posición se compara contra todos los restantes



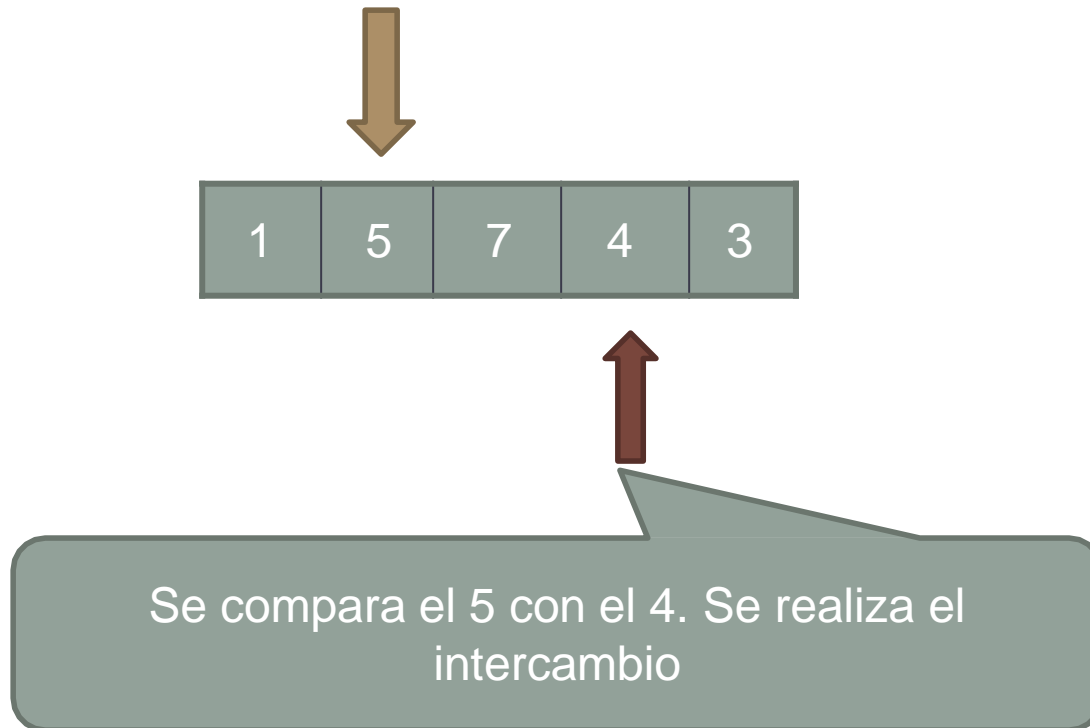
Ordenamiento por intercambio

- El elemento que está en la segunda posición se compara contra todos los restantes



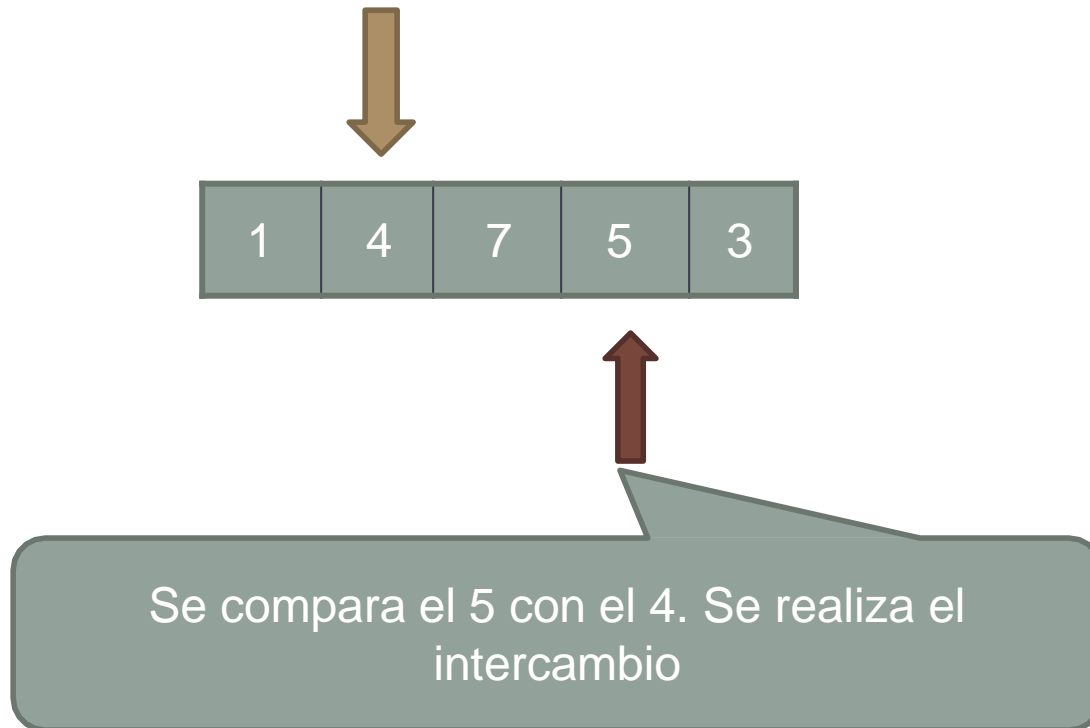
Ordenamiento por intercambio

- El elemento que está en la segunda posición se compara contra todos los restantes



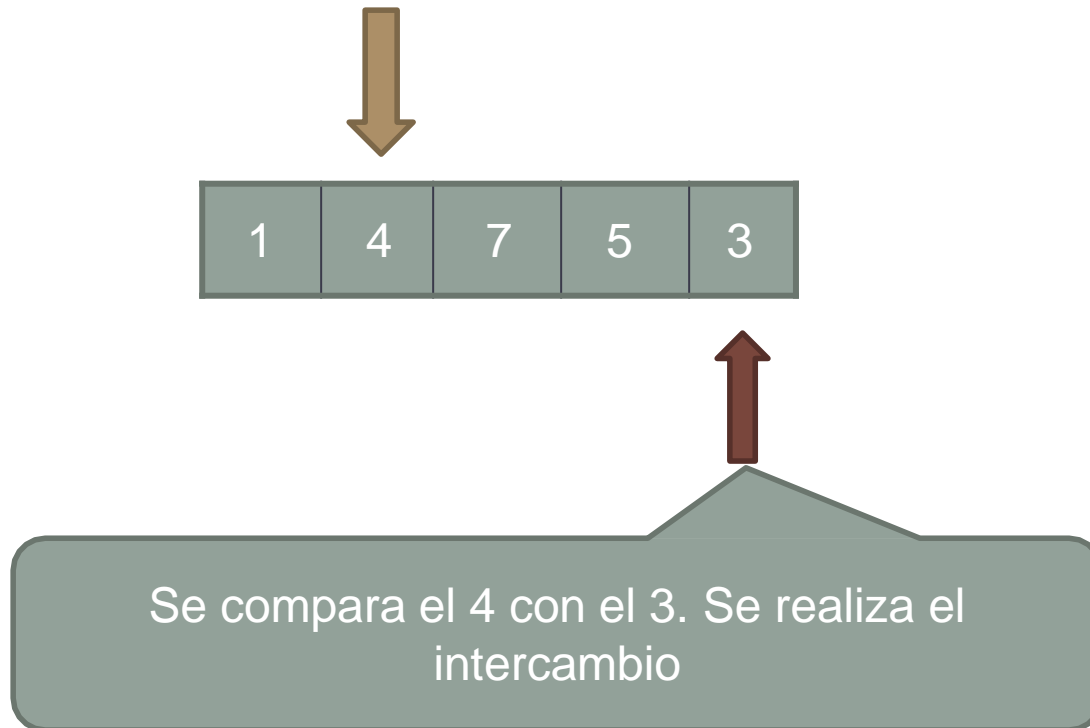
Ordenamiento por intercambio

- El elemento que está en la segunda posición se compara contra todos los restantes



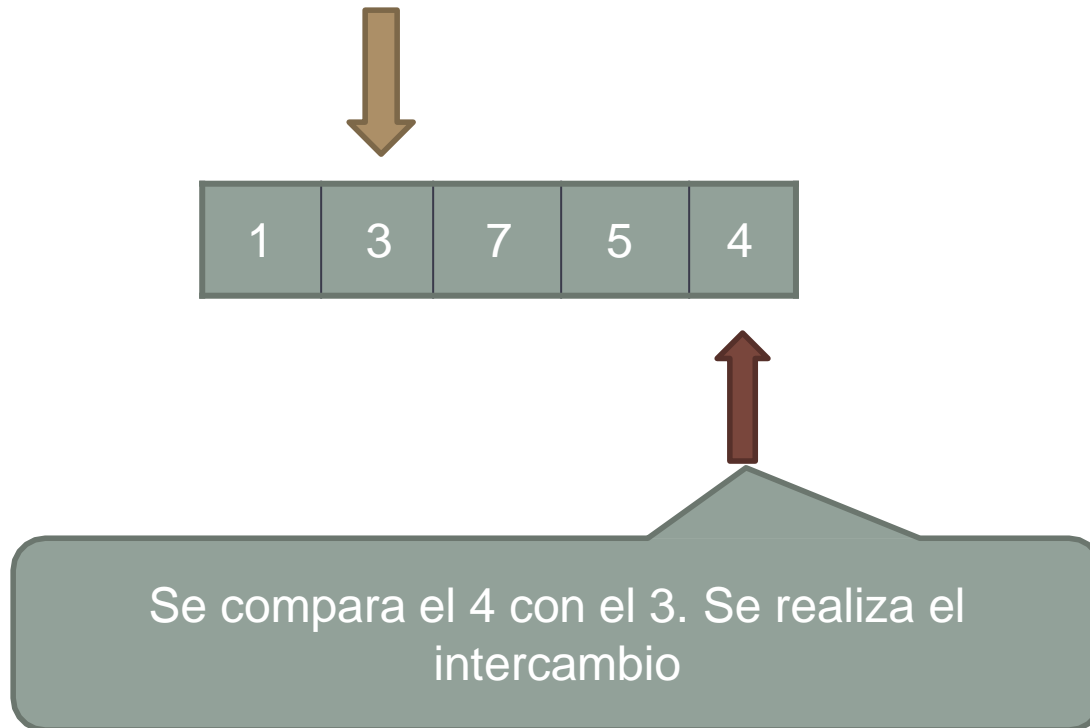
Ordenamiento por intercambio

- El elemento que está en la segunda posición se compara contra todos los restantes



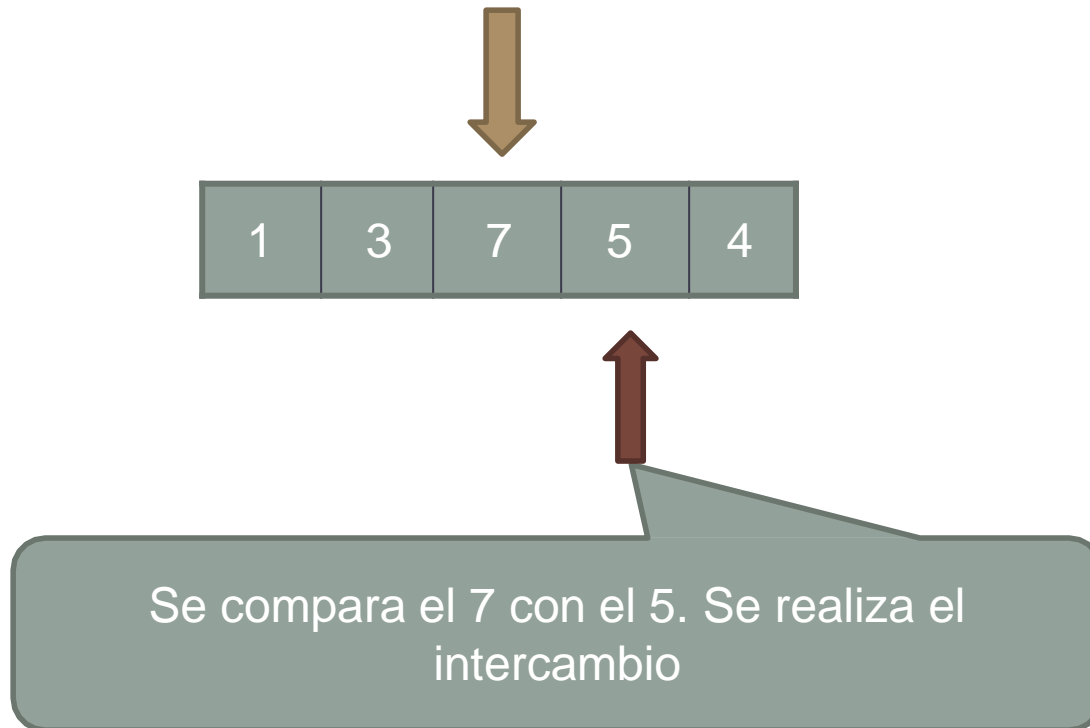
Ordenamiento por intercambio

- El elemento que está en la segunda posición se compara contra todos los restantes



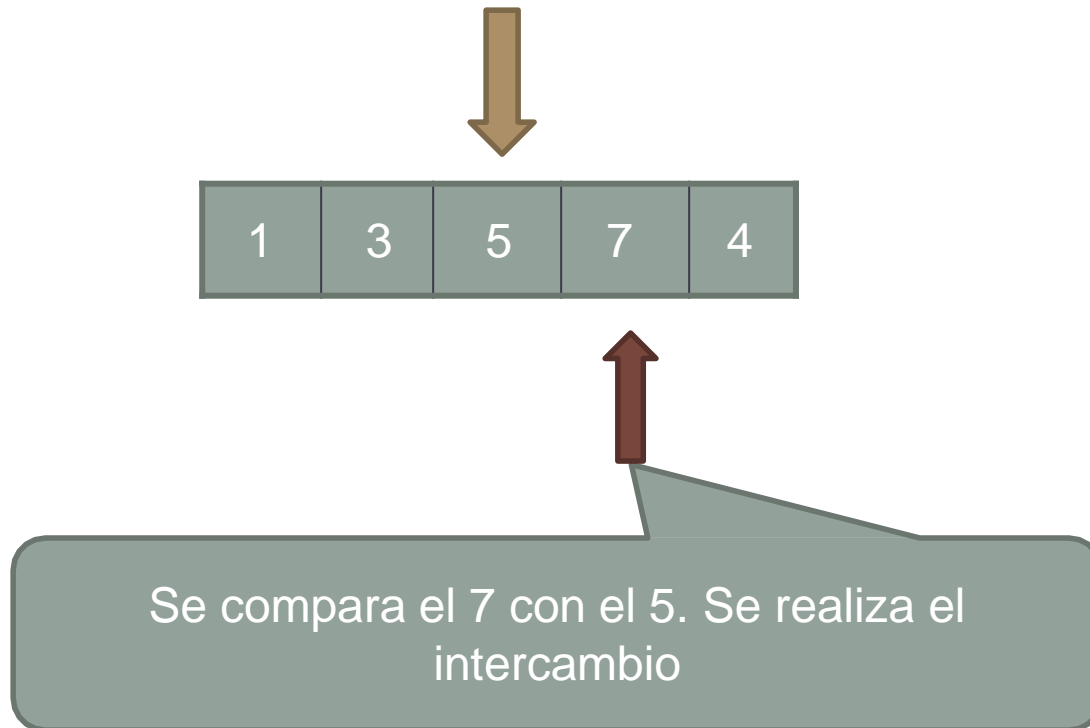
Ordenamiento por intercambio

- El elemento que está en la tercera posición se compara contra todos los restantes



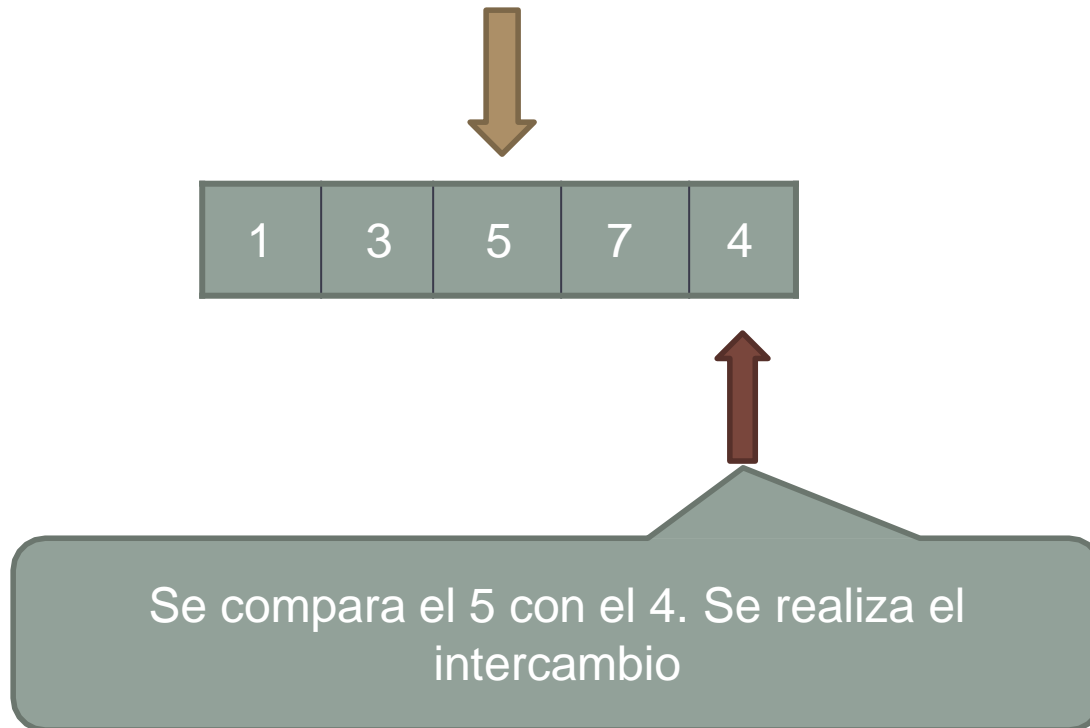
Ordenamiento por intercambio

- El elemento que está en la tercera posición se compara contra todos los restantes



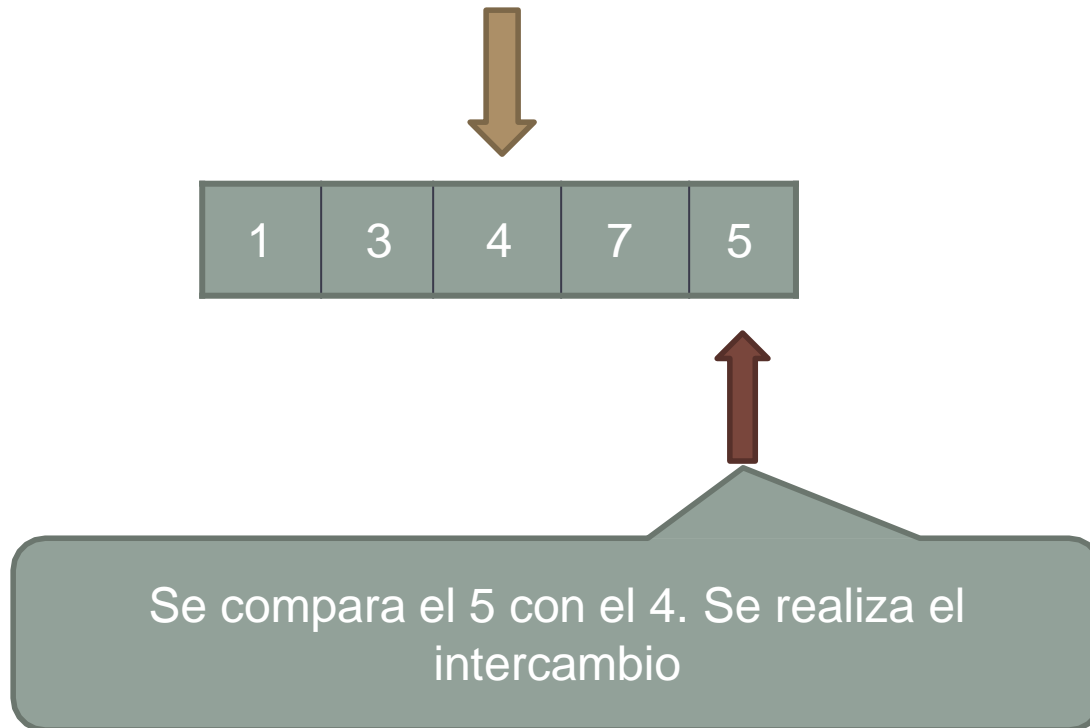
Ordenamiento por intercambio

- El elemento que está en la tercera posición se compara contra todos los restantes



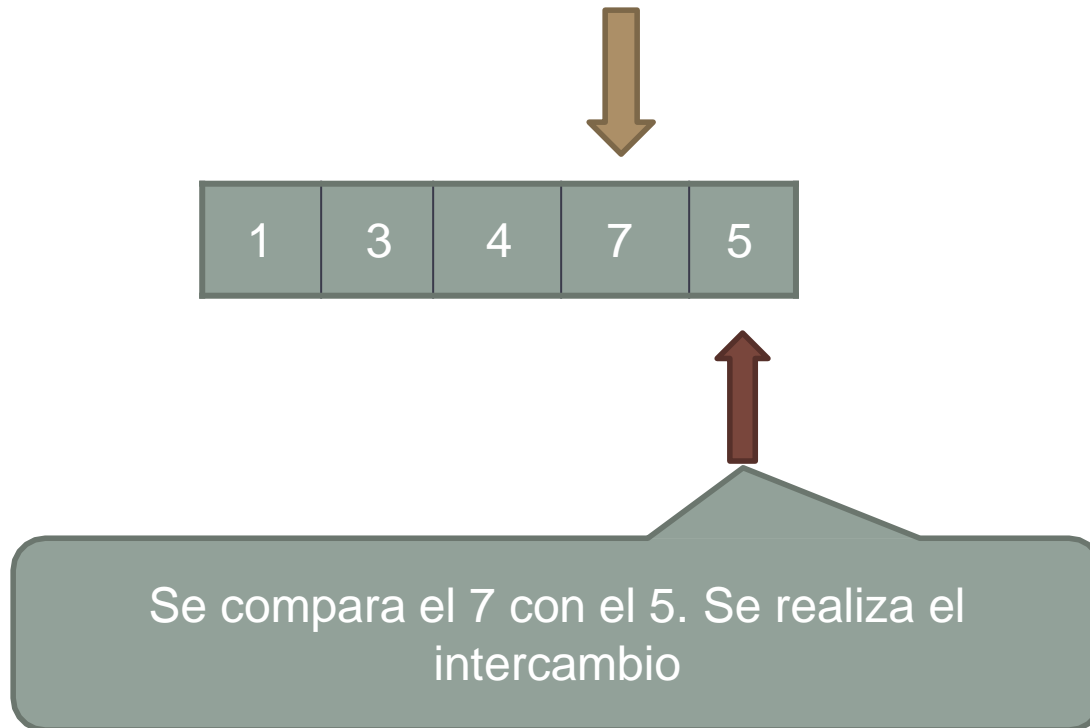
Ordenamiento por intercambio

- El elemento que está en la tercera posición se compara contra todos los restantes



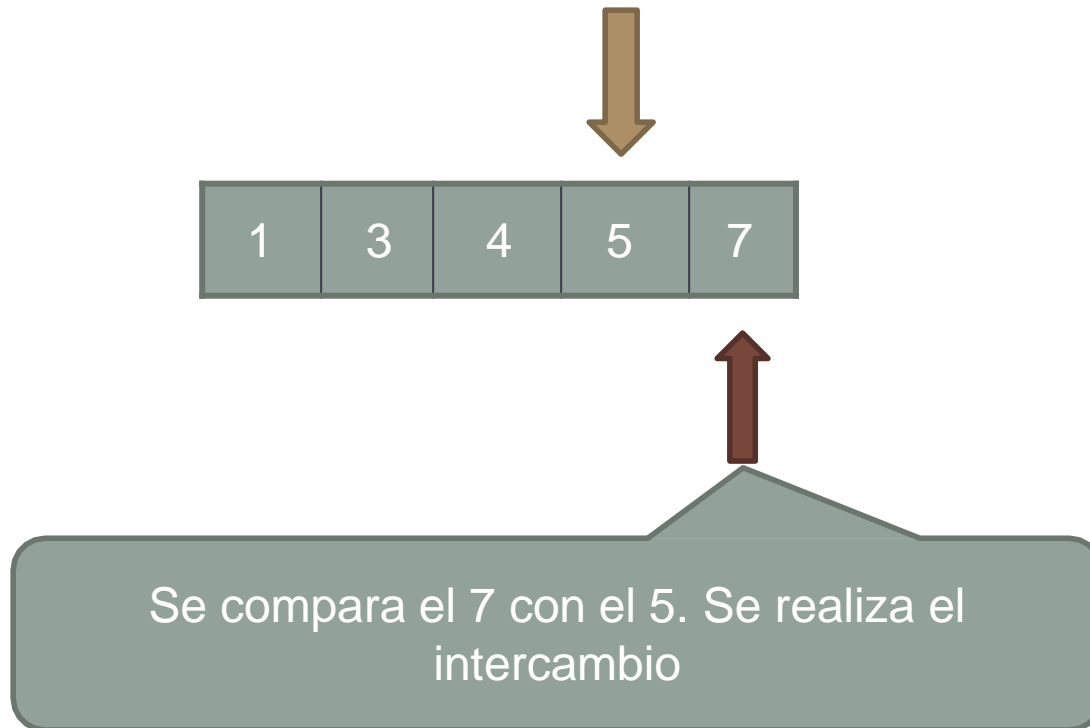
Ordenamiento por intercambio

- El elemento que está en la cuarta posición se compara contra todos los restantes



Ordenamiento por intercambio

- El elemento que está en la cuarta posición se compara contra todos los restantes



Ordenamiento por intercambio

- Fin del algoritmo

1	3	4	5	7
---	---	---	---	---

El arreglo está ordenado

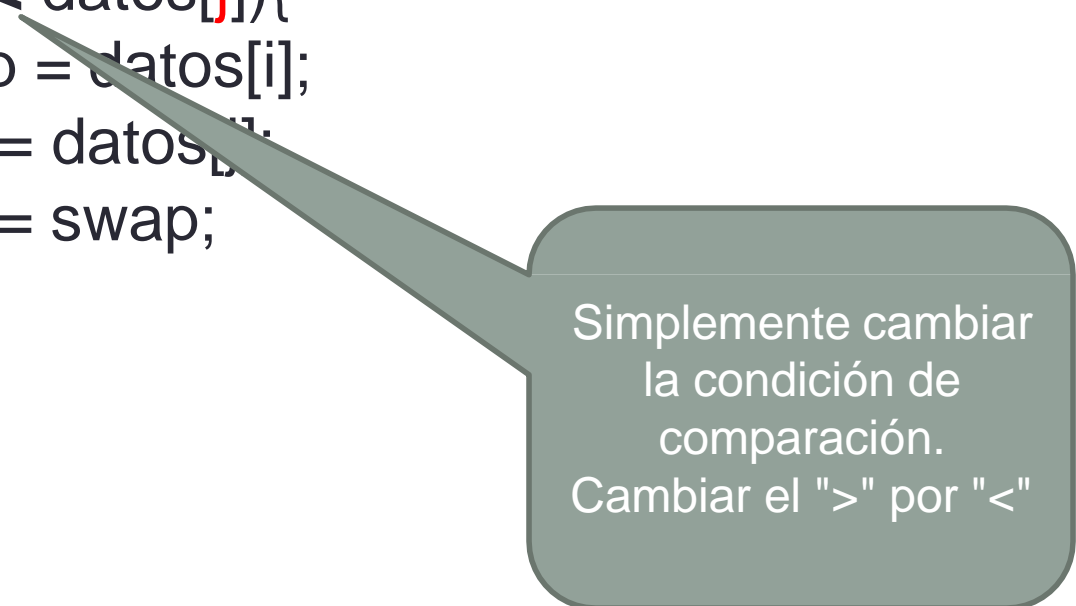
Ordenamiento por intercambio

```
public static void ordenarPorIntercambio(ref int [] datos){  
  
    int n = datos.Length;  
    for(int i=0; i<(n-1); i++)  
        for(int j=i+1; j<n; j++)  
            if(datos[i] > datos[j])  
            {  
                int swap = datos[i];  
                datos[i] = datos[j];  
                datos[j] = swap;  
            }  
}
```

¿Qué modificación
hay que hacerle al
algoritmo para
ordenar de mayor a
menor?

Ordenamiento por intercambio

```
public static void ordenarPorIntercambio(ref int [] datos){  
    int n = datos.Length;  
    for(int i=0; i<(n-1); i++)  
        for(int j=i+1; j<n; j++)  
            if(datos[i] < datos[j]){  
                int swap = datos[i];  
                datos[i] = datos[j];  
                datos[j] = swap;  
            }  
}
```



Simplemente cambiar
la condición de
comparación.
Cambiar el ">" por "<"

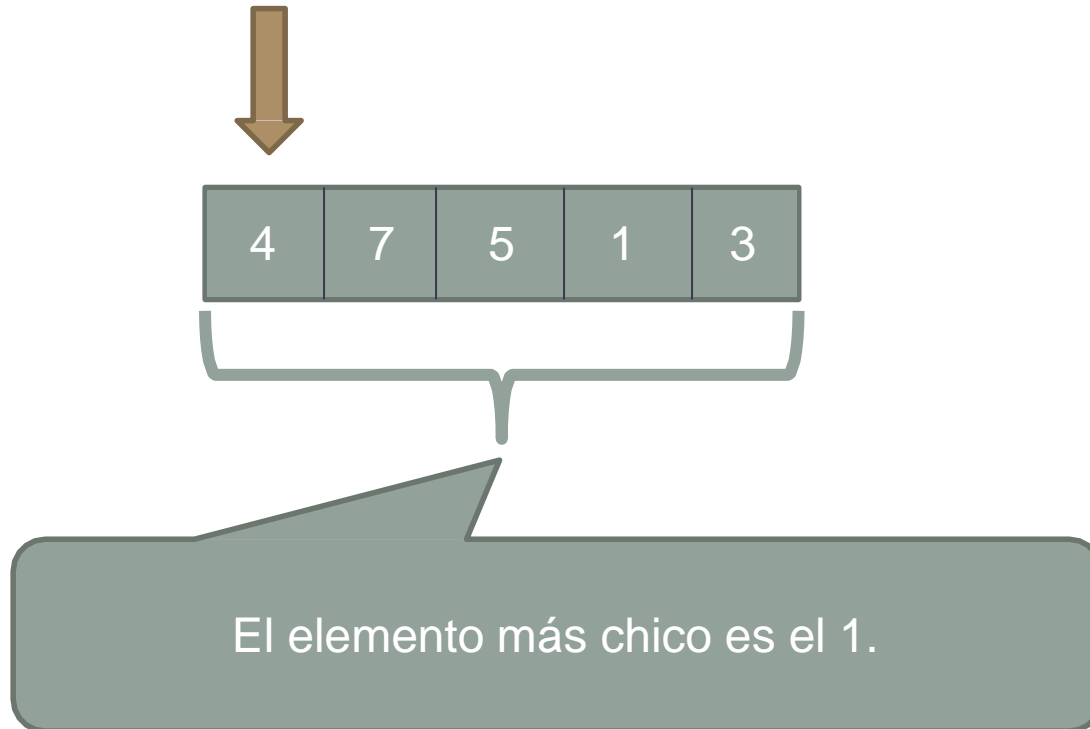
Ordenamiento por selección

Este algoritmo realiza los siguientes pasos:

1. *Seleccionar el elemento más pequeño de la lista*; intercambiarlo con *el primer elemento*. Ahora la entrada más pequeña está en la primera posición.
2. Considerar las posiciones restantes de la lista y *seleccionar el elemento más pequeño* e intercambiarlo con *el segundo elemento*.
Ahora las dos primeras entradas están en orden.
3. *Continuar este proceso* seleccionando los elementos más pequeño de los restantes elementos de la lista e intercambiándolos adecuadamente.

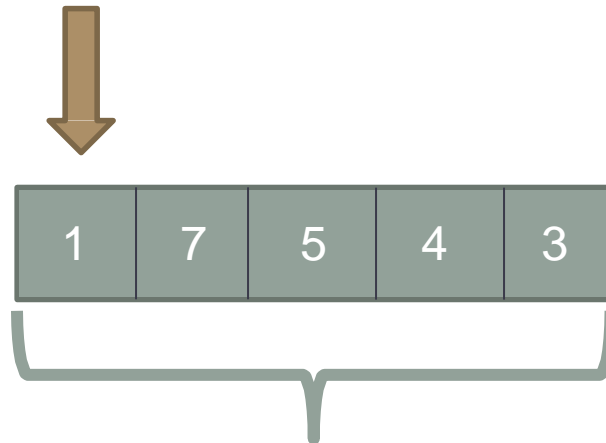
Ordenamiento por selección

- Buscar el elemento más chico y ponerlo en la primer posición



Ordenamiento por selección

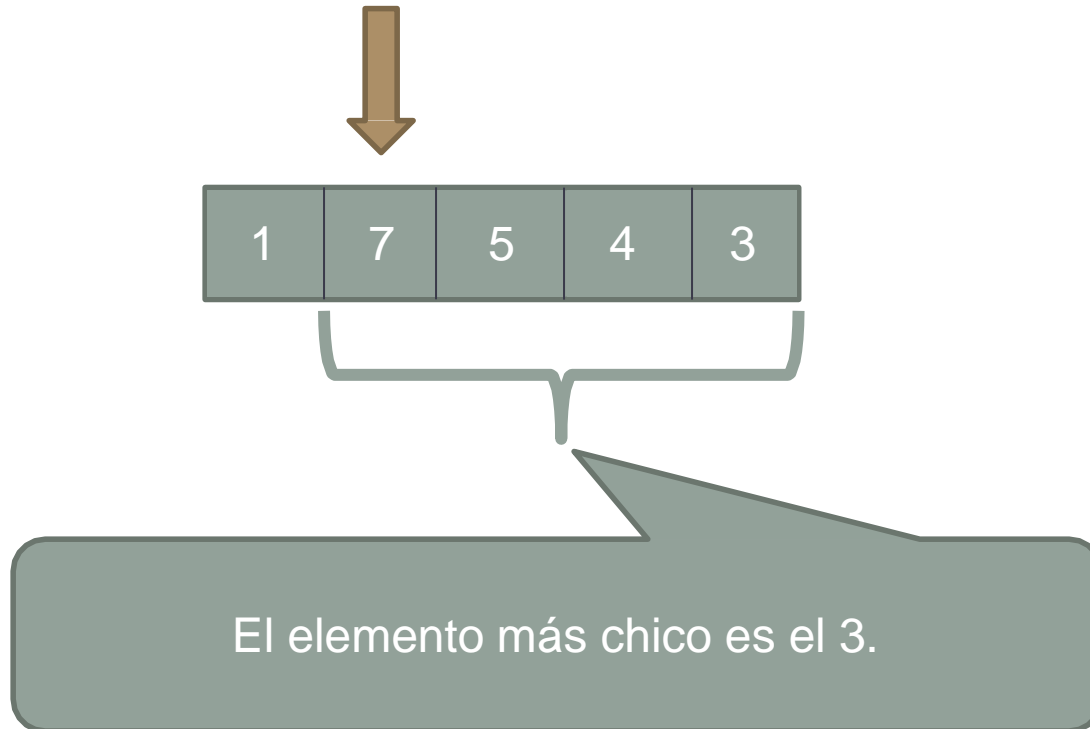
- Buscar el elemento más chico y ponerlo en la primer posición



Se coloca en la primer posición haciendo el intercambio de elementos.

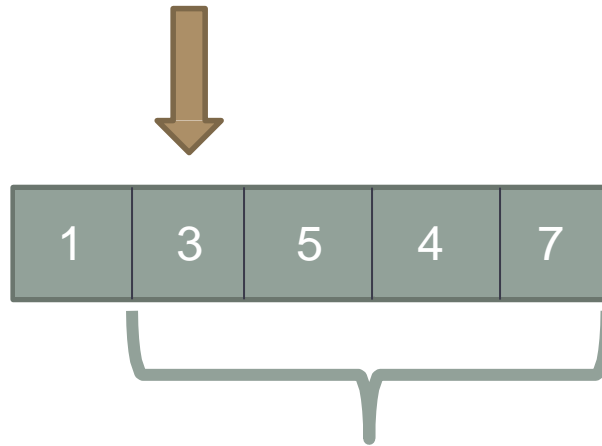
Ordenamiento por selección

- Buscar el elemento más chico y ponerlo en la segunda posición



Ordenamiento por selección

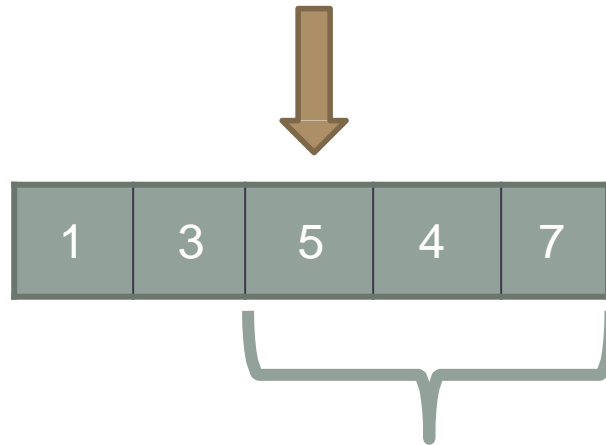
- Buscar el elemento más chico y ponerlo en la segunda posición



Se coloca en la segunda posición haciendo el intercambio de elementos.

Ordenamiento por selección

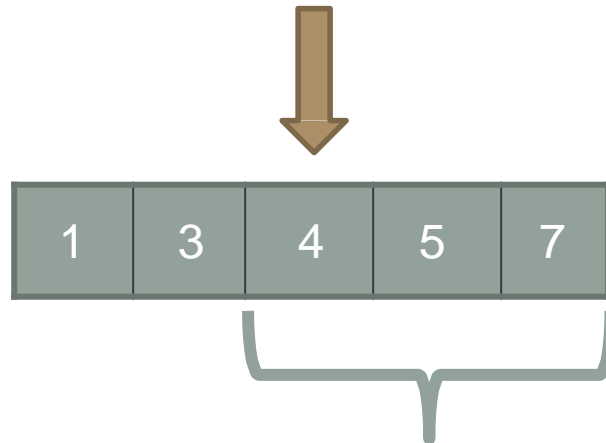
- Buscar el elemento más chico y ponerlo en la tercera posición



El elemento más chico es el 4.

Ordenamiento por selección

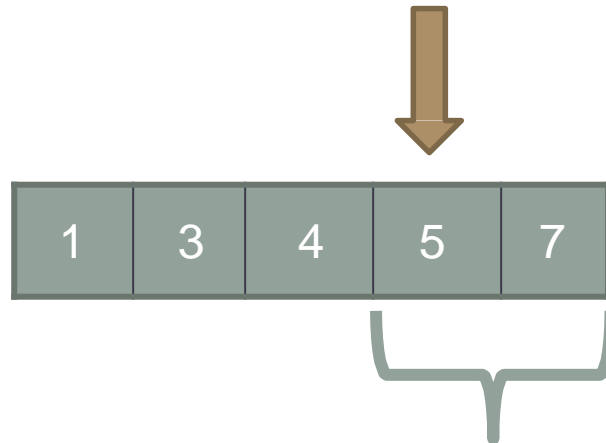
- Buscar el elemento más chico y ponerlo en la tercera posición



Se coloca en la tercera posición haciendo el intercambio de elementos.

Ordenamiento por selección

- Buscar el elemento más chico y ponerlo en la cuarta posición



Notar que, más allá de que el arreglo ya quedó ordenado con el último intercambio, el algoritmo no lo sabe.

La búsqueda del elemento más chico se hace igual (el 5) y se coloca en la cuarta posición.

Ordenamiento por selección

- Fin del algoritmo

1	3	4	5	7
---	---	---	---	---

El arreglo está ordenado

Ordenamiento por selección

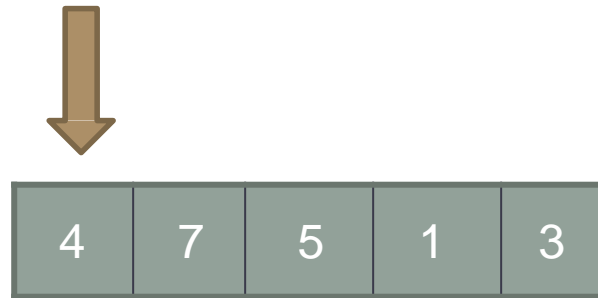
```
public static void ordenarPorSeleccion(ref int [] datos){  
    int n = datos.Length;  
    for(int i=0; i<(n-1); i++)  
    {  
        int menor = i;  
        for(int j=i+1; j<n; j++)  
            if(datos[j] < datos[menor])  
                menor = j;  
  
        if(menor != i){  
            int swap = datos[i];  
            datos[i] = datos[menor];  
            datos[menor] = swap;  
        }  
    }  
}
```

Ordenamiento por burbuja

- En el caso de una colección con n elementos, el Ordenamiento por burbuja requiere hasta $n - 1$ pasadas.
- Por cada pasada **se comparan elementos adyacentes** y **se intercambian sus valores** cuando el primer elemento es mayor que el segundo elemento.
- Al final de cada pasada, el elemento mayor ha «burbujeado» hasta la cima de la subcolección actual.

Ordenamiento por burbuja

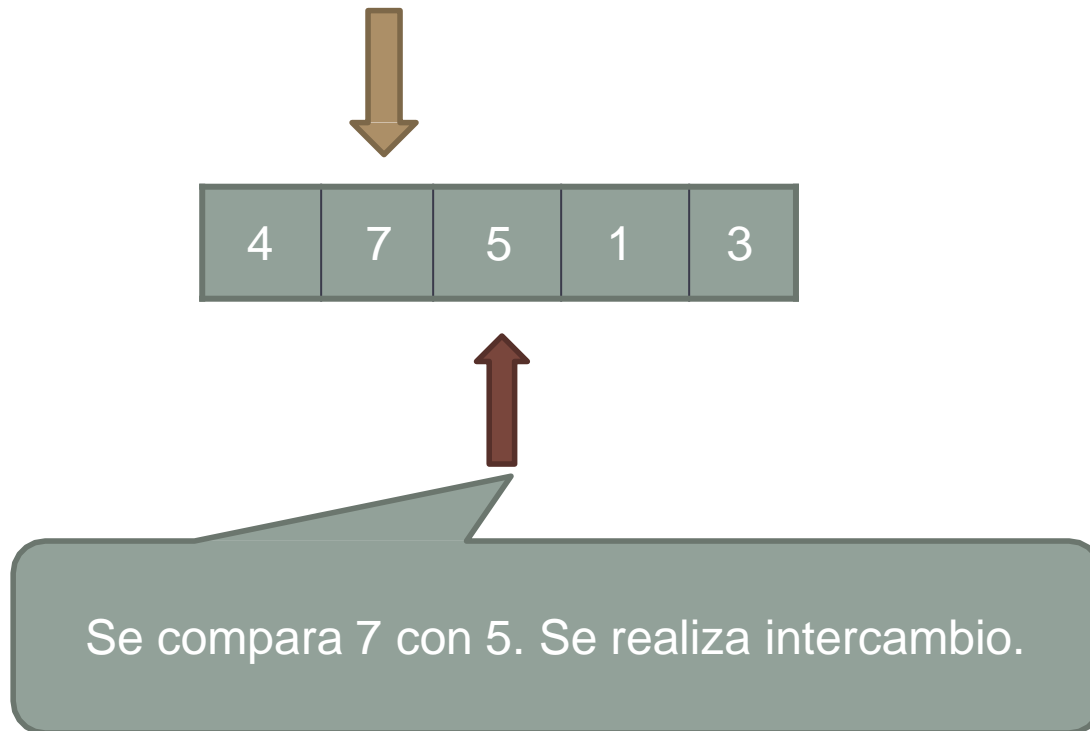
- Llevar a la quinta posición el elemento más grande.



Se compara 4 con 7. 7 es más grande, no se realiza intercambio.

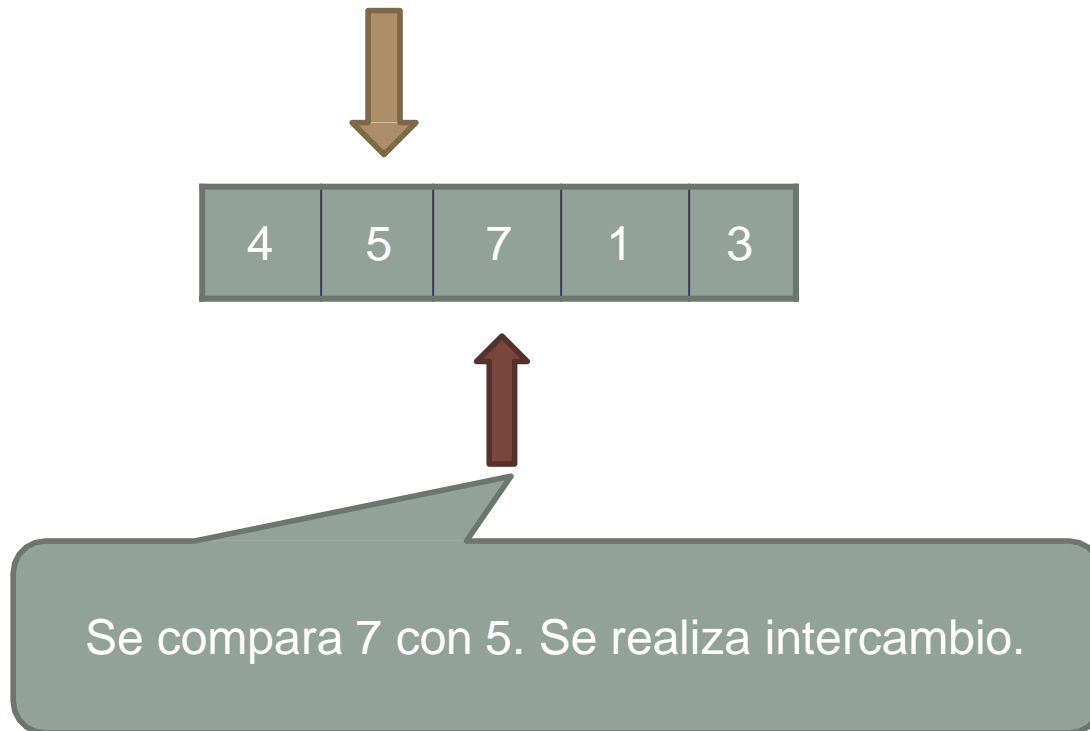
Ordenamiento por burbuja

- Llevar a la quinta posición el elemento más grande.



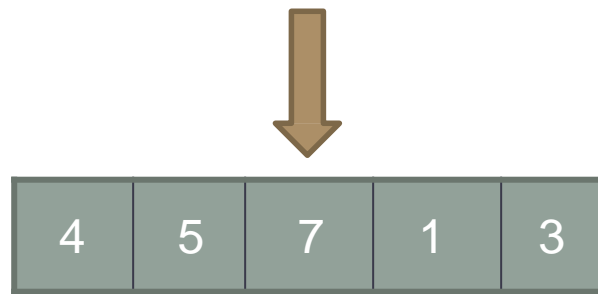
Ordenamiento por burbuja

- Llevar a la quinta posición el elemento más grande.



Ordenamiento por burbuja

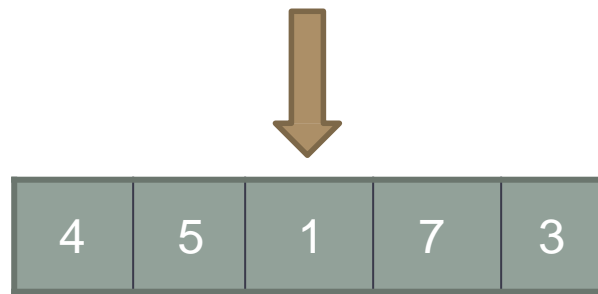
- Llevar a la quinta posición el elemento más grande.



Se compara 1 con 7. Se realiza intercambio.

Ordenamiento por burbuja

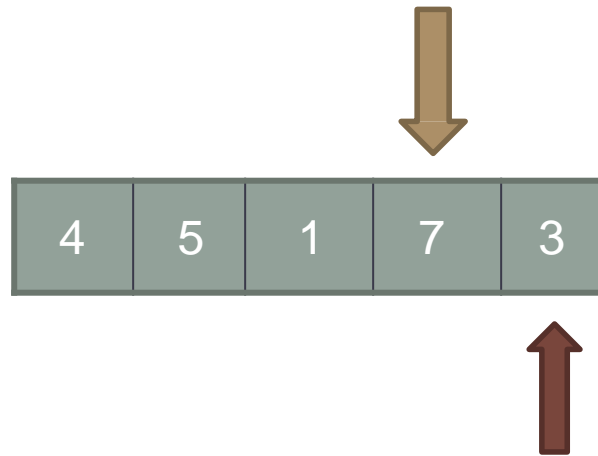
- Llevar a la quinta posición el elemento más grande.



Se compara 1 con 7. Se realiza intercambio.

Ordenamiento por burbuja

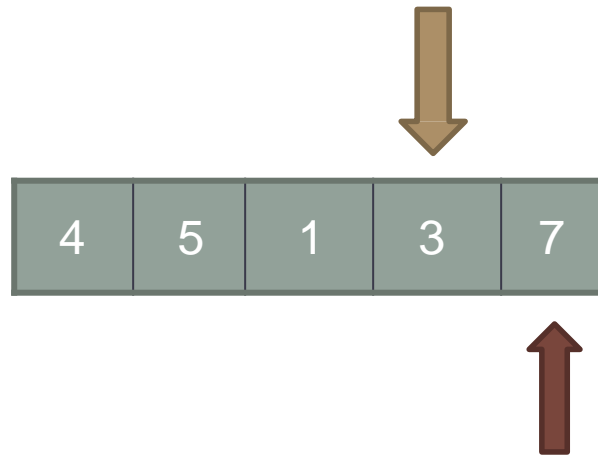
- Llevar a la quinta posición el elemento más grande.



Se compara 7 con 3. Se realiza intercambio.

Ordenamiento por burbuja

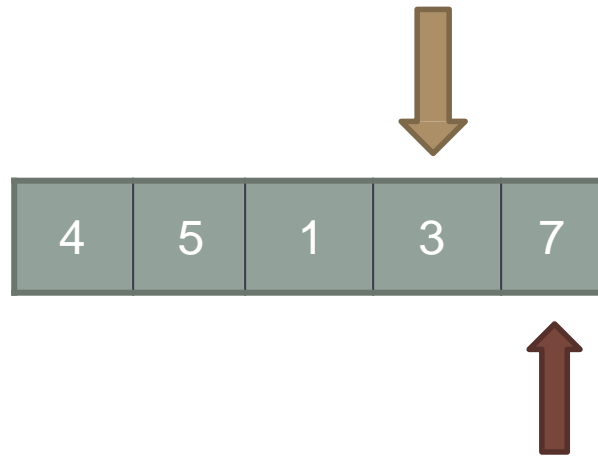
- Llevar a la quinta posición el elemento más grande.



Se compara 7 con 3. Se realiza intercambio.

Ordenamiento por burbuja

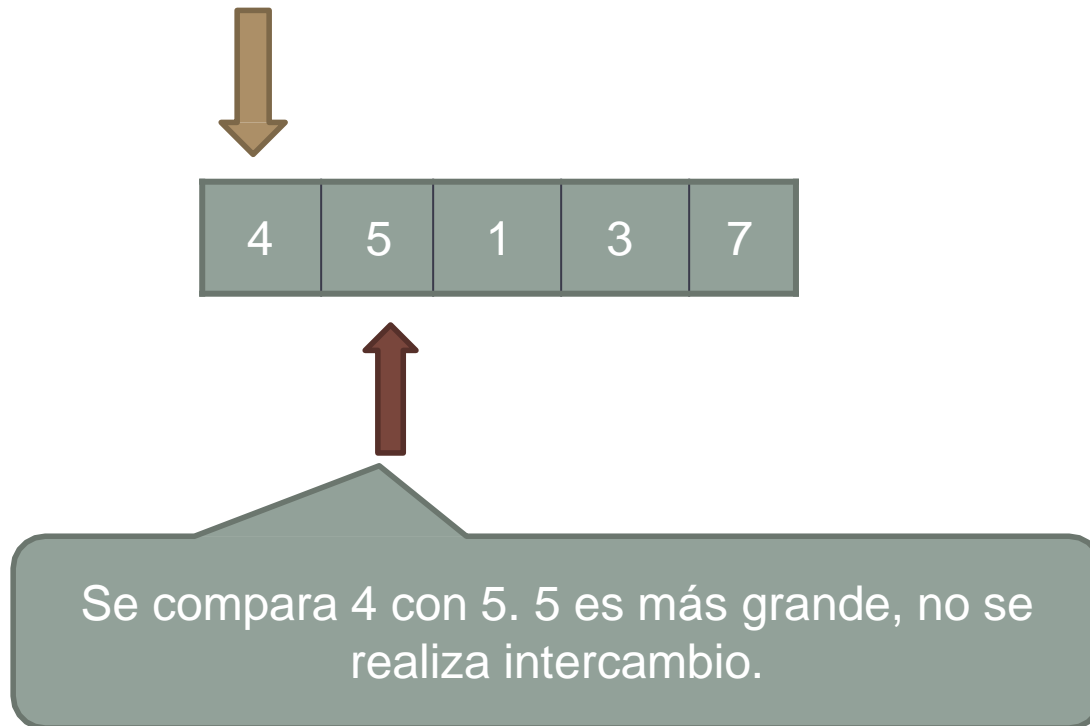
- Llevar a la quinta posición el elemento más grande.



El elemento más grande quedó en el final del arreglo.

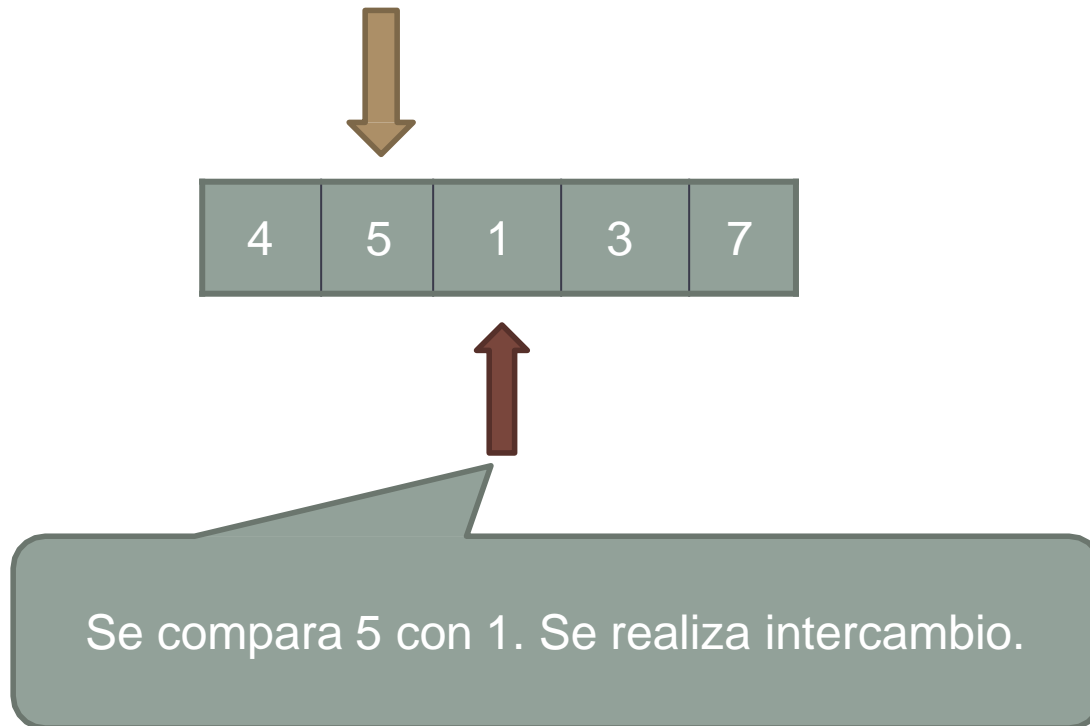
Ordenamiento por burbuja

- Llevar a la cuarta posición el elemento más grande.



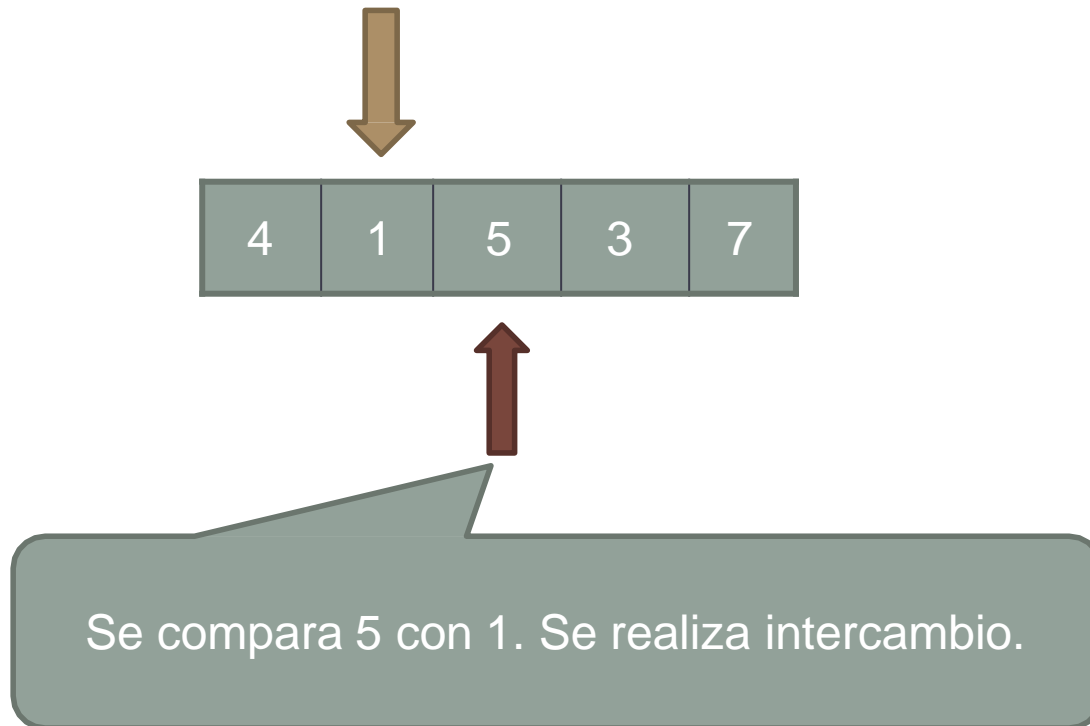
Ordenamiento por burbuja

- Llevar a la cuarta posición el elemento más grande.



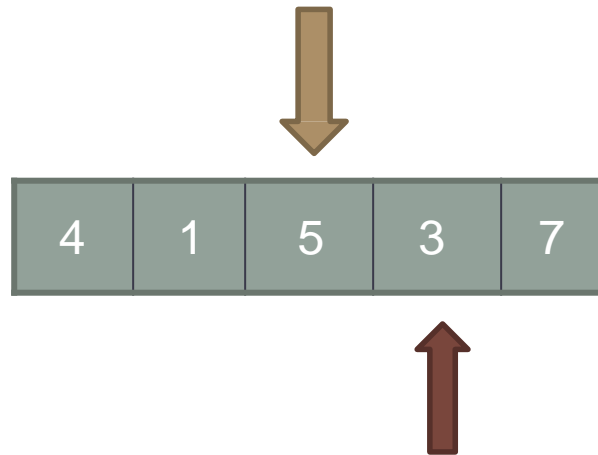
Ordenamiento por burbuja

- Llevar a la cuarta posición el elemento más grande.



Ordenamiento por burbuja

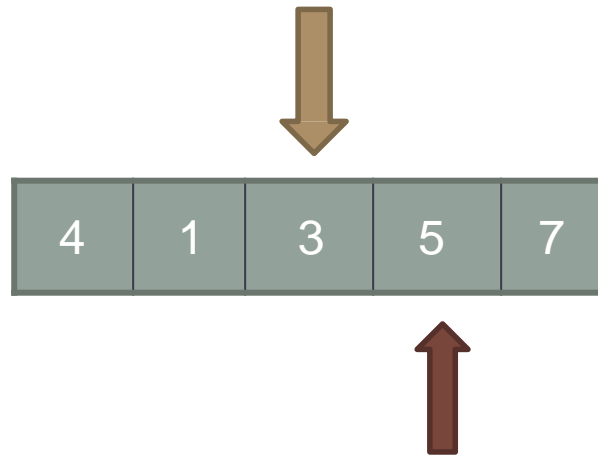
- Llevar a la cuarta posición el elemento más grande.



Se compara 5 con 3. Se realiza intercambio.

Ordenamiento por burbuja

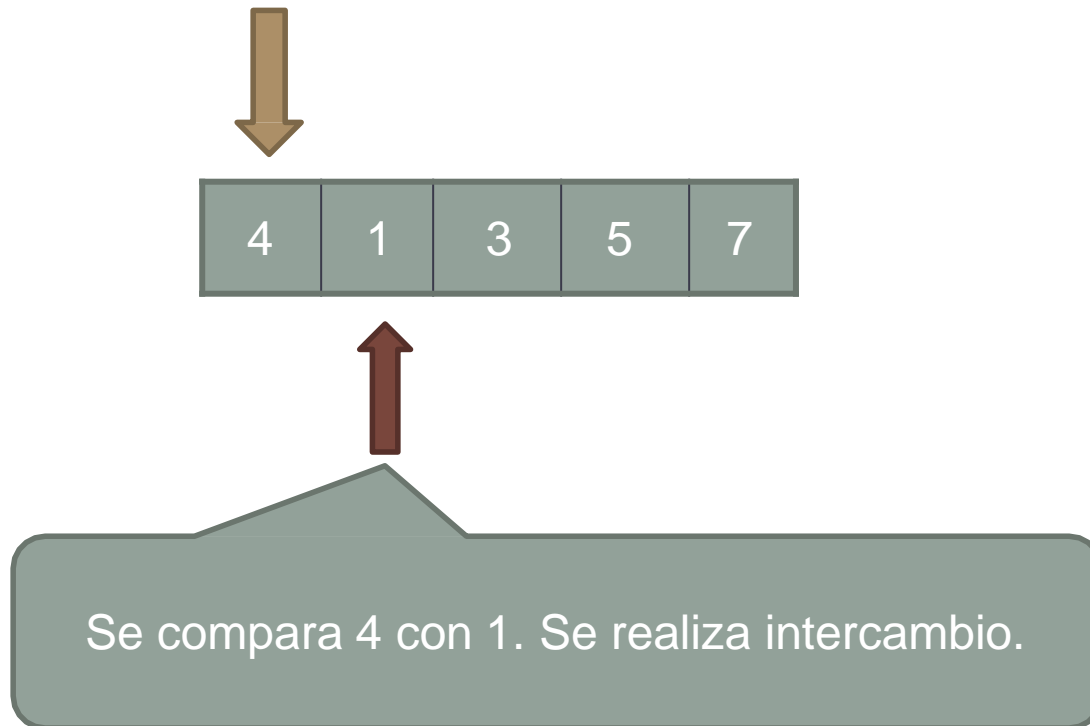
- Llevar a la cuarta posición el elemento más grande.



Se compara 5 con 3. Se realiza intercambio.

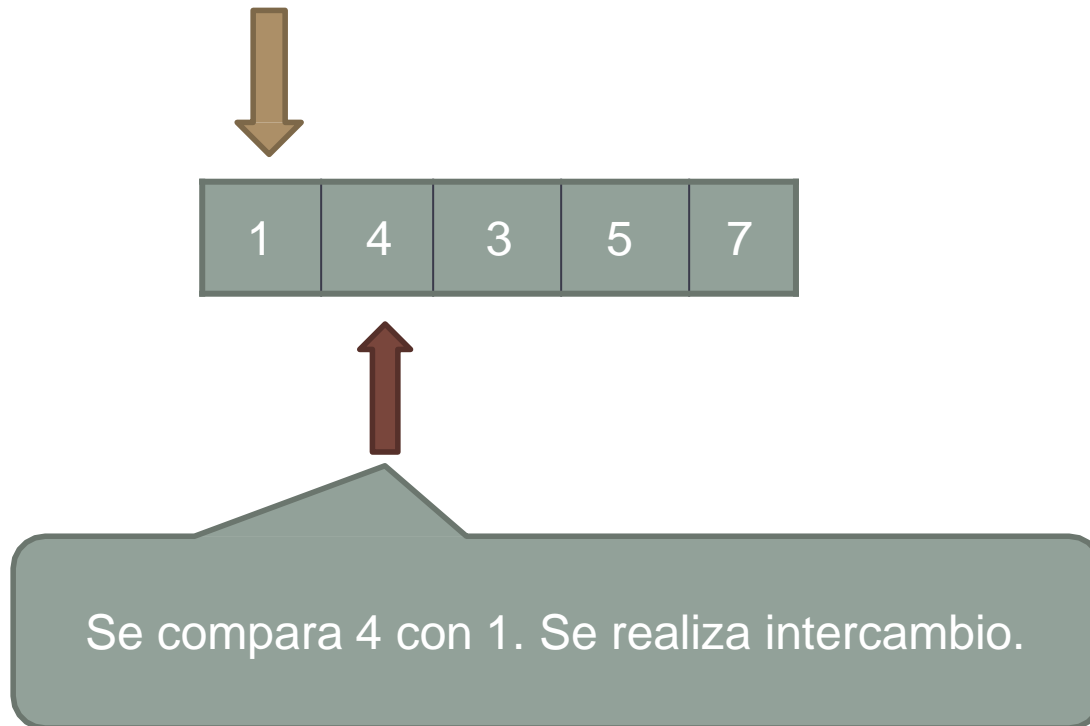
Ordenamiento por burbuja

- Llevar a la tercera posición el elemento más grande.



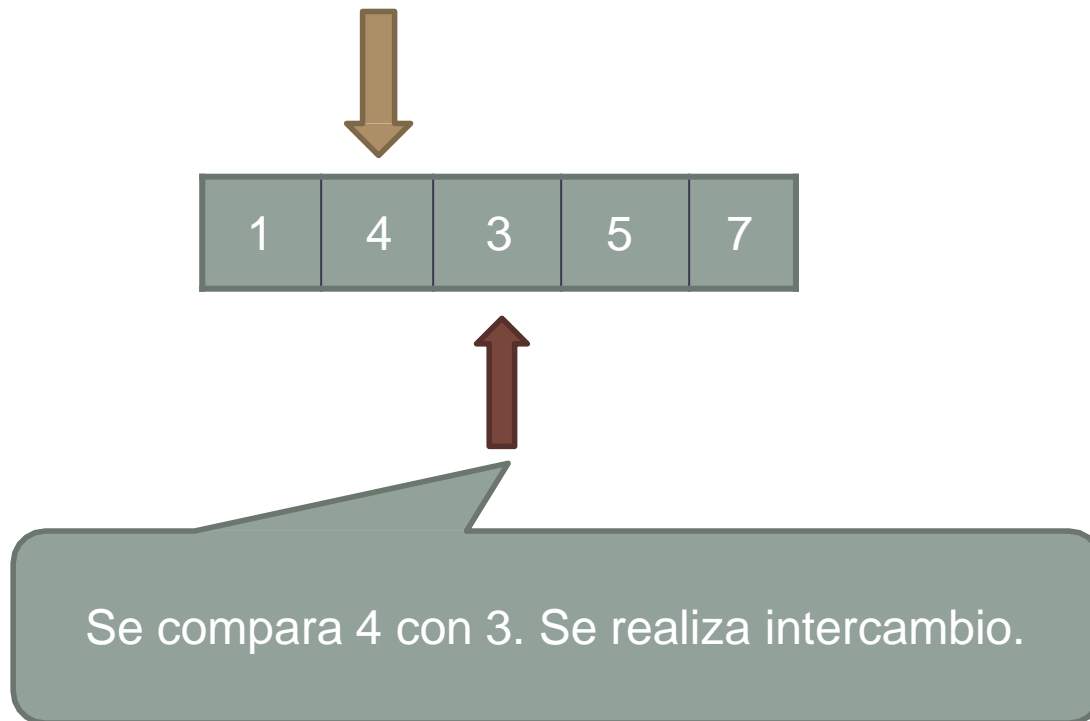
Ordenamiento por burbuja

- Llevar a la tercera posición el elemento más grande.



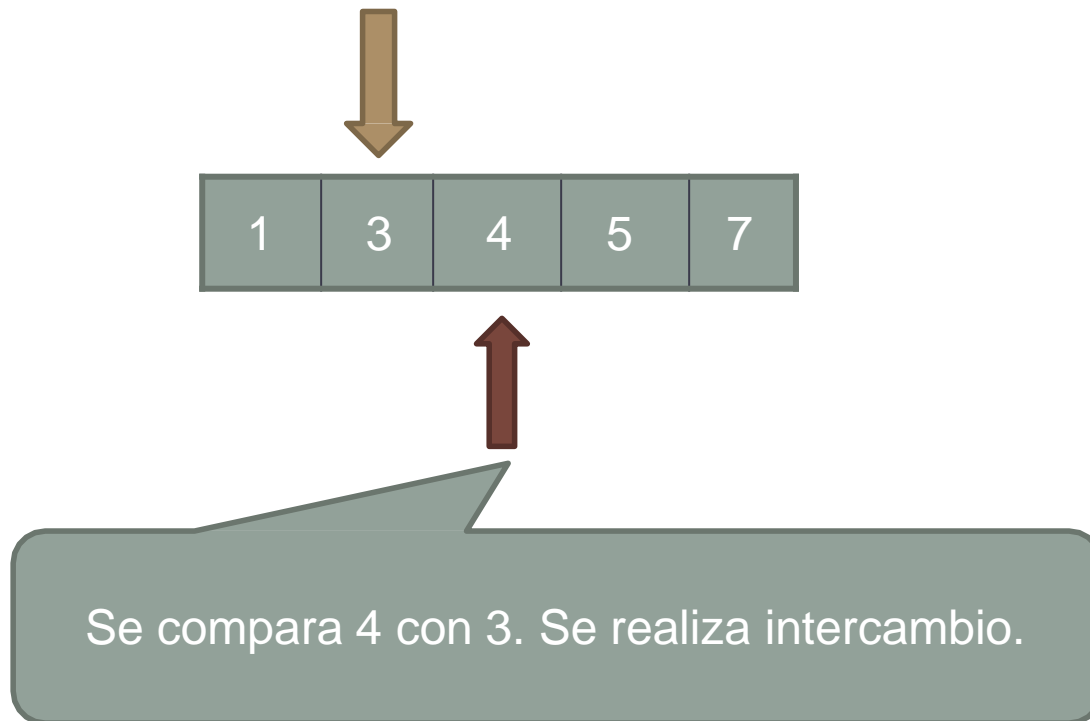
Ordenamiento por burbuja

- Llevar a la tercera posición el elemento más grande.



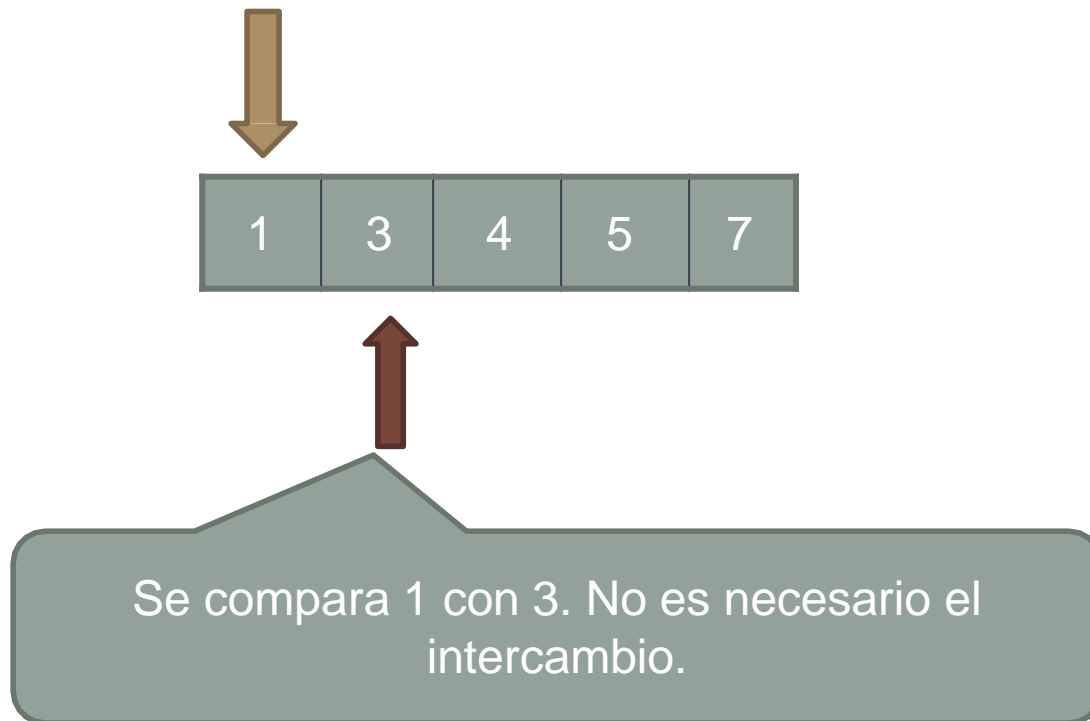
Ordenamiento por burbuja

- Llevar a la tercera posición el elemento más grande.



Ordenamiento por burbuja

- Llevar a la segunda posición el elemento más grande.



Ordenamiento por burbuja

- Fin del algoritmo

1	3	4	5	7
---	---	---	---	---

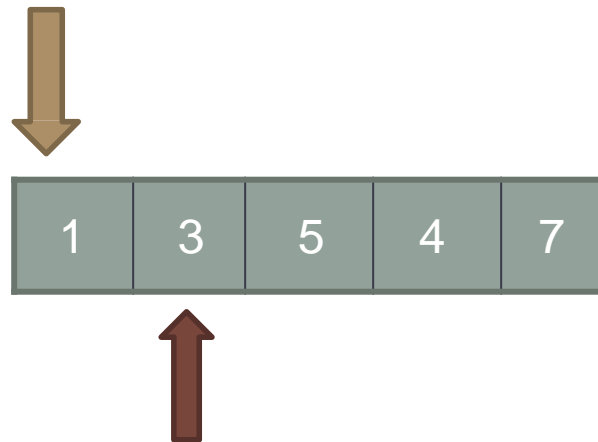
El arreglo está ordenado

Ordenamiento por burbuja

- El algoritmo tiene una mejora, el proceso de ordenamiento puede terminar antes de las $n-1$ pasadas.
- Si en una pasada no se produce intercambio alguno entre elementos a ordenar es porque ya está ordenado, entonces no es necesario realizar más pasadas. Esto se programa mediante el uso de una marca.

Ordenamiento por burbuja

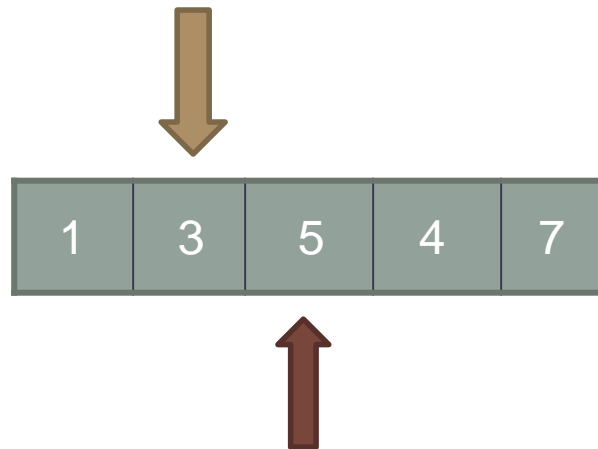
- Ejemplo de fin de algoritmo al darse cuenta que el arreglo ya está ordenado.



Se compara 1 con 3. No hay intercambio

Ordenamiento por burbuja

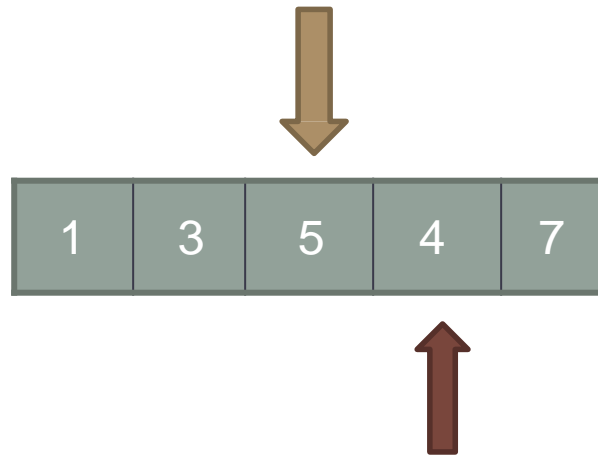
- Ejemplo de fin de algoritmo al darse cuenta que el arreglo ya está ordenado.



Se compara 3 con 5. No hay intercambio

Ordenamiento por burbuja

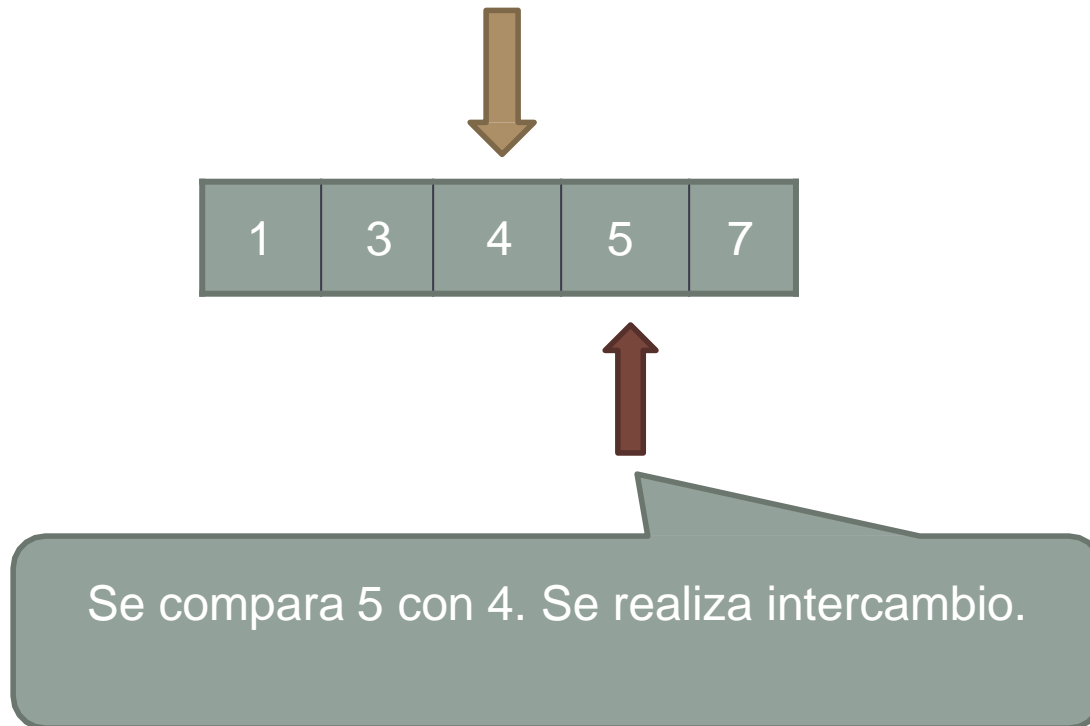
- Ejemplo de fin de algoritmo al darse cuenta que el arreglo ya está ordenado.



Se compara 5 con 4. Se realiza intercambio.

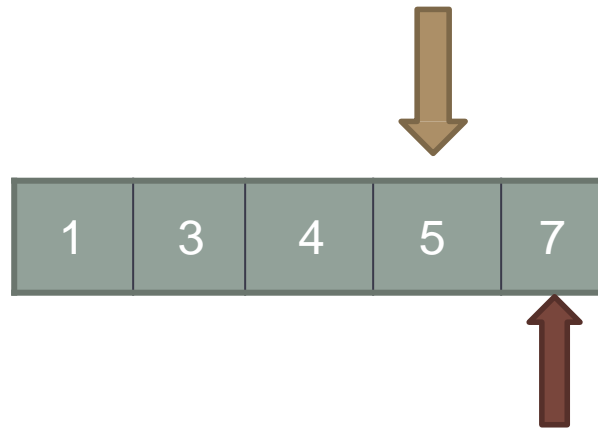
Ordenamiento por burbuja

- Ejemplo de fin de algoritmo al darse cuenta que el arreglo ya está ordenado.



Ordenamiento por burbuja

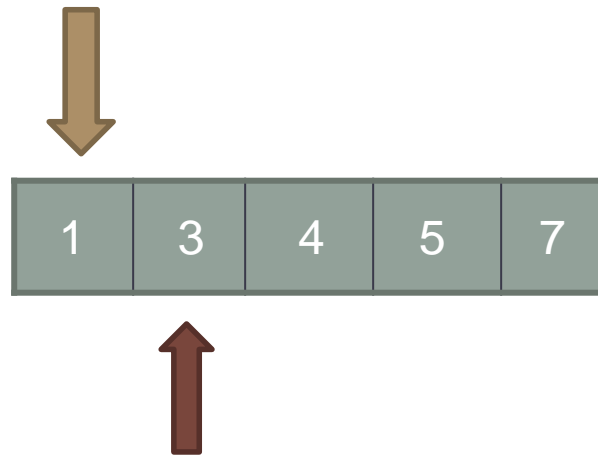
- Ejemplo de fin de algoritmo al darse cuenta que el arreglo ya está ordenado.



Se compara 5 con 7. No realiza intercambio.
Finaliza una pasada

Ordenamiento por burbuja

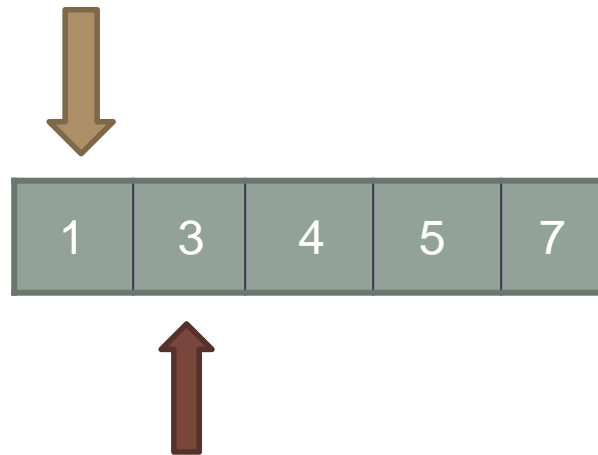
- Ejemplo de fin de algoritmo al darse cuenta que el arreglo ya está ordenado.



En esta segunda pasada se detiene en la tercer posición

Ordenamiento por burbuja

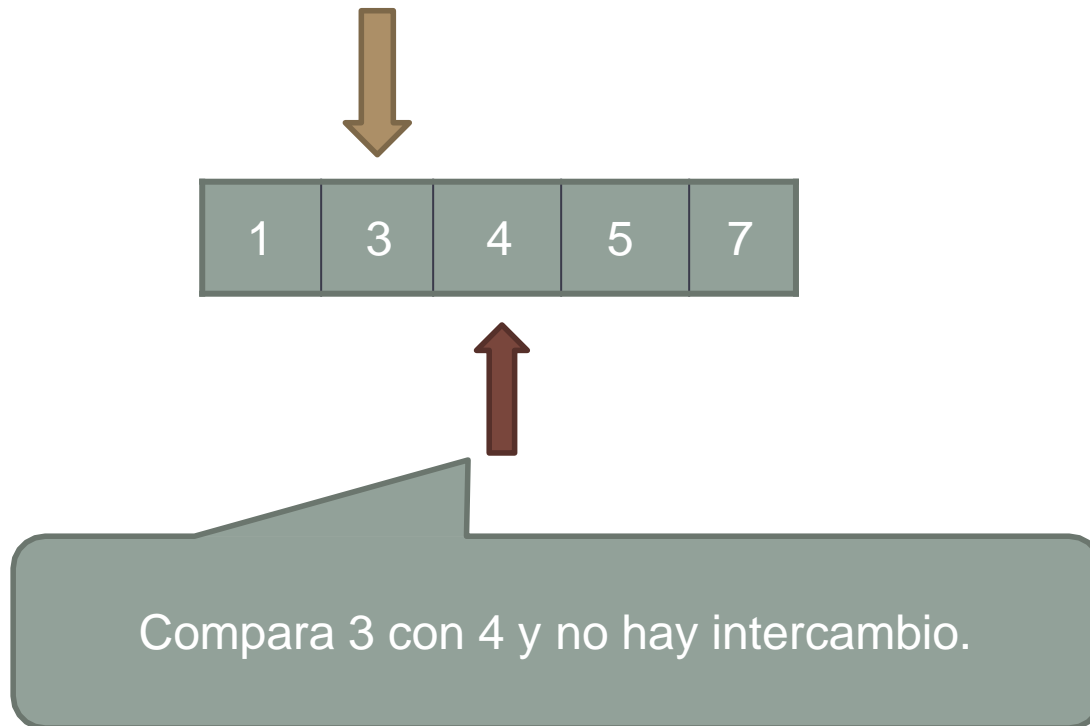
- Ejemplo de fin de algoritmo al darse cuenta que el arreglo ya está ordenado.



Compara 1 con 3 y no hay intercambio

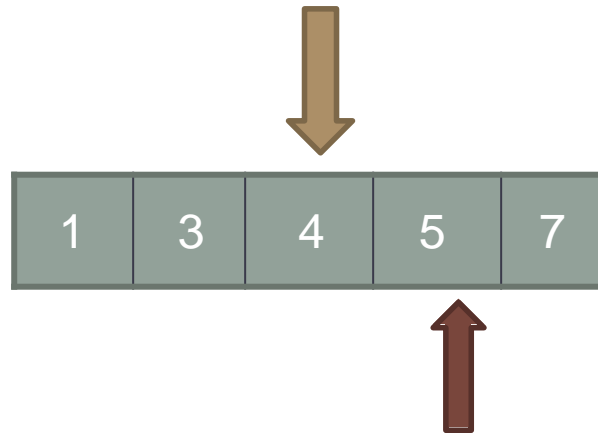
Ordenamiento por burbuja

- Ejemplo de fin de algoritmo al darse cuenta que el arreglo ya está ordenado.



Ordenamiento por burbuja

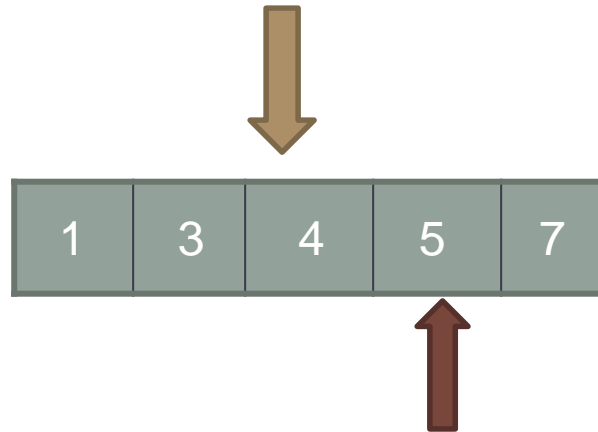
- Ejemplo de fin de algoritmo al darse cuenta que el arreglo ya está ordenado.



Compara 4 con 5 y no hay intercambio.

Ordenamiento por burbuja

- Ejemplo de fin de algoritmo al darse cuenta que el arreglo ya está ordenado.



Se realizó una pasada sin intercambios. Eso ocurre cuando el arreglo está ordenado. No sigue aplicándolo. Fin del algoritmo

Ordenamiento por burbuja

```
public static void ordenarPorBurbuja(ref int [] datos){
```

```
    int n = datos.Length;
    int i = 0;
    boolean ordenado=false;
    while((i<(n-1)) && (ordenado==false))
    {
        ordenado=true;
        for(int j=0; j<(n-1- i); j++){
            if(datos[j] > datos[j+1]){
                ordenado=false;
                int swap = datos[j];
                datos[j] = datos[j+1];
                datos[j+1] = swap;
            }
        }
    }
}
```

Ordenamiento por burbuja

```
public static void ordenarPorBurbuja(ref int [] datos){  
  
    int n = datos.Length;  
    int i = 0;  
    int cont=1;           /*cuenta la cantidad de intercambios*/  
    while((i<(n-1)) && (cont!=0))  
    {  
        cont=0;  
        for(int j=0; j<(n-1- i); j++){  
            if(datos[j] > datos[j+1]){  
                cont++;  
                int swap = datos[j];  
                datos[j] = datos[j+1];  
                datos[j+1] = swap;  
            }  
        }  
    }  
}
```

Se puede usar un contador de cambios en lugar de una marca booleana

Ordenamiento

- En problemas reales se puede necesitar ordenar colecciones de objetos.
- Por ejemplo, supongamos que tenemos la clase alumno con el promedio de cada alumno de una universidad

```
public class Alumno {  
    private double promedio;  
    public Alumno(double p)  
    {  
        promedio = p;  
    }  
    public double Promedio { get { return promedio; } }  
}
```


Ordenamiento

- Y tenemos un ArrayList con alumnos

```
public static void Main(string[] args) {  
    ArrayList alumnos = new ArrayList();  
    alumnos.Add(new Alumno(4.89));  
    alumnos.Add(new Alumno(7.23));  
    alumnos.Add(new Alumno(5.04));  
    alumnos.Add(new Alumno(1.92));  
    alumnos.Add(new Alumno(9.78));  
  
    ordenarPorIntercambio(alumnos)  
}
```

- ¿Cómo podemos ordenar esa colección para imprimir un listado de los alumnos ordenados por promedio?

Ordenamiento

- Podríamos adaptar una versión de alguno de los algoritmos de ordenamiento sobre los elementos de un ArrayList

```
public static void ordenarPorIntercambio(ref ArrayList datos){  
  
    int n = datos.Count;  
    for(int i=0; i<(n-1); i++)  
        for(int j=i+1; j<n; j++)  
            if(datos[i] > datos[j]){  
                object swap = datos[i];  
                datos[i] = datos[j];  
                datos[j] = swap;  
            }  
}
```

Ordenamiento

- Este algoritmo serviría para cualquier tipo de objetos, ya que en un ArrayList se guardan objetos.

```
public static void ordenarPorIntercambio(ref ArrayList datos){  
    int n = datos.Count;  
    for(int i=0; i<(n-1); i++)  
        for(int j=i+1; j<n; j++)  
            if(datos[i] > datos[j]){  
                object swap = datos[i];  
                datos[i] = datos[j];  
                datos[j] = swap;  
            }  
    }
```

Ordenamiento

- El problema es que C# no permite la comparación por mayor o menor entre datos de tipo object.

```
public static void ordenarPorIntercambio(ref ArrayList datos){  
    int n = datos.Count;  
    for(int i=0; i<(n-1); i++)  
        for(int j=i+1; j<n; j++)  
            if(datos[i] > datos[j]){  
                object swap = datos[i];  
                datos[i] = datos[j];  
                datos[j] = swap;  
            }  
}
```

El compilador no permite hacer esto entre variables object.

Ordenamiento

- Una alternativa simple es utilizar las propiedades de los objetos al aplicar ordenamiento:

```
public static void ordenarPorIntercambio( ref ArrayList datos)
{
    int n = datos.Count;
    for(int i=0; i<(n-1); i++)
        for(int j=i+1; j<n; j++)
        {
            if( ((Alumno)datos[i]).Promedio > ((Alumno) datos[j]).Promedio)
            {
                Alumno swap = (Alumno) datos[i];
                (Alumno) datos[i] = (Alumno) datos[j];
                (Alumno) datos[j] = swap;
            }
        }
}
```

