Operaciones avanzadas con arrays

Recorrido de arrays con foreach

Útil cuando queremos recorrer TODAS las posiciones del array SIN modificar ninguna (sólo examinando sus valores).

También la podemos aplicar a tablas si no nos interesa saber la posición de los elementos.

```
int[] numeros = new int[10];
int[,] tabla = new int[4, 10];
...
foreach(int numero in numeros)
    Console.WriteLine(numero);

foreach(int numero in tabla)
    Console.WriteLine(numero);
```

Ordenación de arrays

Existen varias estrategias para ordenar (ascendente o descendentemente) los datos de un array.

Casi todas se basan en un doble bucle anidado que recorre cada posición y la compara con otras para ver si los datos están desordenados, intercambiándolos en ese caso.

La complejidad de estas operaciones es cuadrática (n2), lo que significa que para ordenar un array de N elementos se hacen en torno a N * N repeticiones en total.

Método de la burbuja

Compara cada elemento de una posición i con la adyacente i+1 y los intercambia si están desordenados.

Al final de este proceso, tendremos ordenado el último elemento del array.

Repetir el proceso deteniéndose cada vez una posición antes.

Se le llama "burbuja" porque los números fluyen por el array de forma similar a las burbujas en una botella.

Cada paso se muestra verticalmente

```
for (int i = 0; i < numeros.Length; i++)
{
    for (int j = 0; j < numeros.Length - i - 1; j++)
    {
        if (numeros[j] > numeros[j+1])
        {
            int auxiliar = numeros[j];
            numeros[j] = numeros[j+1];
            numeros[j+1] = auxiliar;
        }
    }
}
```

Método de intercambio directo

A veces se le confunde con el algoritmo de burbuja.

Para cada posición i del array, se recorre el mismo desde esa posición hasta el final buscando el menor elemento, y se inserta en esa posición i.

Buscamos el menor elemento y lo colocamos en 1ª posición.Buscamos el segundo menor elemento y lo colocamos en 2ª posición.

...

Cada elemento que se encuentre que sea menor que el que actualmente esté en la posición i, se intercambia.

Cada paso se muestra horizontalmente

```
for (int i = 0; i < numeros.Length - 1; i++)
{
    for (int j = i+1; j < numeros.Length; j++)
    {
        if (numeros[i] > numeros[j])
        {
            int auxiliar = numeros[i];
            numeros[i] = numeros[j];
            numeros[j] = auxiliar;
        }
    }
}
```

Ordenación automática

La instrucción **Array.Sort** ordena automáticamente el array que le indiquemos de menor a mayor

Sólo funciona para arrays de tipos simples (enteros, reales, strings...), no para structs, por ejemplo

Búsqueda binaria

La búsqueda secuencial tiene una complejidad lineal O(n)

Significa que, en el peor de los casos, tendremos que hacer n repeticiones para encontrar el dato (si está al final del array)

La búsqueda binaria tiene una complejidad O(log(n)), menor que la lineal

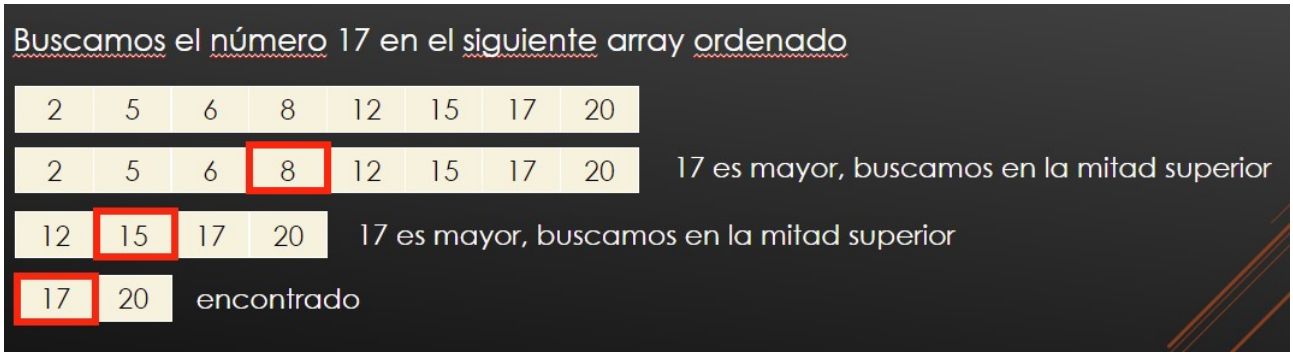
Es adecuada si el array está previamente ordenado (ascendente o descendente)

1-Tomamos el elemento del centro del array

2-Si es el que estamos buscando, hemos terminado

3-Si es menor, sabemos que tenemos que buscar en la mitad superior del array

4-Si es mayor, sabemos que tenemos que buscar en la mitad inferior del array



Suponemos que buscamos el 15 en un array

```
int[] numeros = new int[50];
// Suponemos un array ya relleno y ordenado
ascendentemente
int mitad, limiteInferior = 0, limiteSuperior =
numeros.Length - 1;
bool encontrado = false;
while (!encontrado && limiteInferior <= limiteSuperior)
{
    mitad = (limiteInferior + limiteSuperior) / 2;
    if (numeros[mitad] == 15)
        encontrado = true;
    else if (numeros[mitad] < 15)
    {
        // Buscamos en mitad superior
        limiteInferior = mitad + 1;
    }
    else
    {
        // Buscamos en mitad inferior
        limiteSuperior = mitad - 1;
    }
}
```