



## 1. Factorial

Implemente un programa que reciba un número entero no negativo  $n$  de la consola y calcule el factorial de ese número.

El factorial de un número  $n$  (denotado como  $n!$ ) se define como el producto de todos los números enteros positivos desde 1 hasta  $n$ , o lo que es lo mismo:

$$n! = \begin{cases} 1 & \text{si } n = 0 \\ n \cdot (n-1)! & \text{si } n > 0 \end{cases}$$

**Respuesta:**

```
public static long CalculateFactorial(int n)
{
    long result = 1;

    for (int i = 1; i <= n; i++)
    {
        result *= i;
    }

    return result;
}
```

## 2. Suma de impares

Implemente un programa que reciba un entero  $n$  e imprima la suma de los primeros  $n$  números impares.

## Respuesta:

```
public static int SumOddNumbers(int n)
{
    int sum = 0;

    for (int i = 1; i < 2 * n; i += 2)
    {
        sum += i;
    }

    return sum;
}
```

Este código usa un ciclo para sumar los primeros  $n$  números impares. Sin embargo, podemos notar una propiedad interesante (que puedes demostrar por inducción):

$$1 + 3 + 5 + \dots + (2n - 1) = n^2$$

Por lo tanto, podemos optimizar nuestro código utilizando esta propiedad matemática.

```
public static int SumOddNumbers(int n)
{
    return n * n;
}
```

## 3. Mayor, menor y promedio de forma perezosa

Implemente un programa que lea una secuencia de números de la consola (uno por línea) hasta que se escriba una línea en blanco y de estos imprimir:

- El mayor
- El menor
- Su promedio

## Respuesta:

```
string input;
int max = int.MinValue;
int min = int.MaxValue;
int sum = 0;
int count = 0;
Console.WriteLine("Introduce números enteros (deja la línea en blanco para finalizar:");

while (!string.IsNullOrWhiteSpace(input = Console.ReadLine()))
{
    if (int.TryParse(input, out int number))
    {
        if (number > max) max = number;
        if (number < min) min = number;
        sum += number;
        count++;
    }
    else
    {
        Console.WriteLine("Entrada inválida, introduce un número entero.");
    }
}
```

```

if (count == 0)
{
    Console.WriteLine("No se introdujeron números.");
}
else
{
    double average = (double)sum / count;

    Console.WriteLine($"El mayor: {max}");
    Console.WriteLine($"El menor: {min}");
    Console.WriteLine($"El promedio: {average}");
}

```

## 4. Recorriendo arrays

Implemente un método para cada inciso, que reciba un array de enteros y devuelva:

1. El mayor elemento de un array.

```

public static int FindMax(int[] array)
{
    int max = int.MinValue;

    for (int i = 0; i < array.Length; i++)
    {
        var num = array[i];
        if (num > max) max = num;
    }

    return max;
}

```

2. El segundo menor elemento de un array.

```

public static int FindSecondMin(int[] array)
{
    int min = int.MaxValue;
    int secondMin = int.MaxValue;

    for (int i = 0; i < array.Length; i++)
    {
        int num = array[i];
        if (num < min)
        {
            secondMin = min;
            min = num;
        }
        else if (num < secondMin && num != min)
        {
            secondMin = num;
        }
    }

    return secondMin;
}

```

3. Si un número  $n$  pertenece al array  $a$ .

```

public static bool Contains(int[] array, int n)
{
    for (int i = 0; i < array.Length; i++)

```

```

    {
        int num = array[i];
        if (num == n)
            return true;
    }

    return false;
}

```

4. El promedio de todos los elementos de un array.

```

public static double CalculateAverage(int[] array)
{
    int sum = 0;
    for (int i = 0; i < array.Length; i++)
    {
        sum += array[i];
    }

    return (double)sum / array.Length;
}

```

5. La cantidad de elementos que son mayor que el promedio en un array.

```

public static int CountAboveAverage(int[] array)
{
    double average = CalculateAverage(array);
    int count = 0;

    for (int i = 0; i < array.Length; i++)
    {
        if (array[i] > average) count++;
    }

    return count;
}

```

## 5. Invirtiendo

Implemente un método que dado el array  $a$  que recibe como entrada, devuelva otro array con los elementos de  $a$  en orden inverso. Ejemplo: recibe:  $[2, 7, -9]$  y devuelve  $[-9, 7, 2]$ .

**Respuesta:**

```

public static int[] Reverse(int[] array)
{
    int[] reversed = new int[array.Length];

    for (int i = 0; i < array.Length; i++)
    {
        reversed[i] = array[array.Length - 1 - i];
    }

    return reversed;
}

```

## 6. Filtrando Positivos

Implemente un método que reciba un array *a* y devuelva un nuevo array con todos los elementos positivos del array *a*.

**Respuesta:**

```
public static int[] FilterPositive(int[] a)
{
    // Contar los elementos positivos
    int count = 0;

    for (int i = 0; i < a.Length; i++)
    {
        if (a[i] > 0)
        {
            count++;
        }
    }

    // Crear un nuevo array para almacenar los elementos positivos
    int[] positiveArray = new int[count];

    // Índice para recorrer el array de positivos
    int index

    for (int i = 0; i < a.Length; i++)
    {
        var num = a[i];
        if (num > 0)
        {
            positiveArray[index++] = num;
        }
    }

    return positiveArray;
}
```

Podemos notar que la parte de contar los elementos mayores que 0 se parece mucho a contar los elementos mayores que el promedio. Para evitar duplicar código y mejorar la mantenibilidad, podríamos crear un método que cuente los elementos mayores que un valor *x* y reutilizarlo en ambas situaciones.

```
public static int CountGreaterThan(int[] a, int x)
{
    int count = 0;

    for (int i = 0; i < a.Length; i++)
    {
        if (a[i] > x)
        {
            count++;
        }
    }

    return count;
}

public static int[] FilterPositive(int[] a)
{
    int count = CountGreaterThan(a, 0);
    int[] positiveArray = new int[count];
}
```

```

int index = 0;

for (int i = 0; i < a.Length; i++)
{
    if (a[i] > 0)
    {
        positiveArray[index++] = a[i];
    }
}

return positiveArray;
}

```

## 7. Rotando

Implemente un método que reciba un array  $a$  y un entero  $veces$  y rote los elementos del array tantas veces como indique el parámetro  $veces$ . Si  $veces$  es positivo, rota los elementos a la derecha; si es negativo, rota los elementos a la izquierda. Si  $veces$  es 0, el array no se modifica. Por ejemplo, si rotamos el array  $[25, 40, 17, 83, 9]$  2 veces, obtenemos el array  $[83, 9, 25, 40, 17]$ , y si lo rotamos -2 veces entonces nos queda  $[17, 83, 9, 25, 40]$ .

### Respuesta:

Imaginemos un array de tamaño  $n = 5$ . Nos podemos dar cuenta de que el resultado de rotar 3 veces es equivalente al de rotar 8, o al de rotar 13. Esto se debe a que al rotar el array tantas veces como su tamaño, el array vuelve a su estado original. Por lo tanto, al rotar más de  $n$  veces, estamos realizando rotaciones adicionales que no cambian el resultado final. Al aplicar la operación módulo (%), obtenemos el menor número de rotaciones necesarias para lograr el mismo resultado.

Veamos qué pasa con las rotaciones negativas. Podemos darnos cuenta de que, en nuestro array de 5 elementos, rotar -2 veces es equivalente a rotar -7, o -22 veces, pero también es equivalente a rotar 3 veces.

Como la operación módulo (%) en C# da como resultado números en el rango  $[-(k-1), k-1]$ . Si el resultado es menor que 0, podemos sumarle  $k$  para obtener un valor positivo. Por ejemplo:

$$-22 \% 5 = -2$$

$$-2 + 5 = 3$$

De esta manera, siempre obtenemos la mínima cantidad de veces que debemos rotar a la derecha para obtener el mismo resultado.

Primero podemos resolver el problema de rotar una vez a la derecha. La rotación una vez a la derecha implica mover cada elemento del array una posición hacia adelante, y mover el último elemento al primer lugar.

```

private static void RotateOnce(int[] array)
{
    int lastElement = array[^1]; // Notación de C# equivalente a array[array.Length - 1]

    for (int i = array.Length - 1; i > 0; i--)
    {
        array[i] = array[i - 1];
    }

    array[0] = lastElement;
}

```

Luego, podemos llamar  $k$  veces a este método para lograr la rotación deseada:

```
public static void Rotate(int[] array, int times)
{
    if (array.Length == 0)
        return;

    // Normalizar times para que esté dentro del rango de la longitud del array
    times %= array.Length;

    if (times < 0)
        times += array.Length;

    for (int i = 0; i < times; i++)
    {
        RotateOnce(array);
    }
}
```

En general, podemos determinar la posición final de cada elemento en la rotación, ya que al mover cada elemento  $k$  posiciones a la derecha, este se ubicará en una posición específica calculable dentro del array.

```
public static void Rotate(int[] array, int times)
{
    if (array.Length == 0)
        return;

    // Normalizar times para que esté dentro del rango de la longitud del array
    times %= array.Length;

    if (times < 0)
        times += array.Length;

    // Copiar los elementos del array original en el array temporal
    // en sus nuevas posiciones rotadas
    int[] temp = new int[array.Length];

    for (int i = 0; i < array.Length; i++)
    {
        temp[(i + times) % array.Length] = array[i];
    }

    // Copiar los elementos ya rotados al array original
    for (int i = 0; i < length; i++)
    {
        array[i] = temp[i];
    }
}
```

Podemos solucionar el problema sin usar un array auxiliar. Notemos que siempre sabemos en qué posición terminará el elemento, pero no podemos moverlo y ya, pues sobrescribiríamos el elemento que estaba en la posición a donde lo estamos moviendo.

Supongamos que tenemos el array  $[1, 2, 3, 4, 5, 6]$ , de tamaño  $n = 6$  y queremos rotar  $k = 4$  veces. Comenzamos en la posición 0 y sigamos estos pasos:

1. El valor en la posición 0 (valor 1) se mueve a la posición 4.

$[1, 2, 3, 4, 1, 6]$

2. El valor que estaba en la posición 4 (valor 5) se mueve a la posición 2.

$[1, 2, 5, 4, 1, 6]$

3. El valor que estaba en la posición 2 (valor 3) se mueve a la posición 0 (la posición de inicio).

[3,2,5,4,1,6]

Notemos que ya hemos cubierto los índices [0, 2, 4], o sea, los valores que están en estos índices están correctamente rotados.

Ahora hagamos lo mismo comenzando en la posición 1 en el array resultante [3,2,5,4,1,6]:

1. El valor en la posición 1 (valor 2) se mueve a la posición 5.

[3,2,5,4,1,2]

2. El valor que estaba en la posición 5 (valor 2) se mueve a la posición 3.

[3,2,5,6,1,2]

3. El valor que estaba en la posición 3 (valor 4) se mueve a la posición 1 (la posición de inicio).

[3,4,5,6,1,2]

Notemos que ya hemos cubierto correctamente todos los índices del array, por tanto ya quedó completamente rotado.

El siguiente código rota correctamente un array  $k$  veces:

```
public static void Rotate(int[] array, int times)
{
    if (array.Length == 0)
        return;

    // Normalizar times para que esté dentro del rango de la longitud del array
    times %= array.Length;
    if (times < 0)
        times += array.Length;

    int startIndex = 0;
    int totalCount = 0;

    while (totalCount < array.Length)
    {
        totalCount += RotateCycle(array, times, startIndex);
    }
}

// Ejecuta un ciclo de rotaciones comenzando desde 'startIndex'
// y devuelve la cantidad de elementos que fueron rotados en ese ciclo
public static int RotateCycle(int[] array, int times, int startIndex)
{
    int currentIndex = startIndex;
    int currentElement = array[startIndex];
    int count = 0;
    do
    {
        int nextIndex = (currentIndex + times) % array.Length;

        int temp = array[nextIndex];
        array[nextIndex] = currentElement;
        currentElement = temp;

        currentIndex = nextIndex;
        count++;
    } while (currentIndex != startIndex);
    return count;
}
```



## 8. Mezcla ordenada

Implemente un método que a partir de los arrays ordenados *a* y *b* deberá devolver un nuevo array que sea la mezcla ordenada de estos. Por ejemplo, si el array *a* es [23,40,83] y el array *b* es [5,17,23,24,51], entonces el resultado será el array [5,17,23,23,24,40,51,83].

**Respuesta:**

```
public static int[] MergeSortedArrays(int[] a, int[] b)
{
    int[] result = new int[a.Length + b.Length];

    // Inicializa los índices para recorrer los arrays 'a', 'b' y 'result'
    int i = 0, j = 0, k = 0;

    // Itera mientras haya elementos en ambos arrays 'a' y 'b'
    while (i < a.Length && j < b.Length)
    {
        // Si el elemento actual de 'a' es menor que el de 'b'
        if (a[i] < b[j])
        {
            // Copia el elemento de 'a' en 'result' y avanza los índices 'i' y 'k'
            result[k++] = a[i++];
        }
        else
        {
            // Copia el elemento de 'b' en 'result' y avanza los índices 'j' y 'k'
            result[k++] = b[j++];
        }
    }

    // Copia los elementos restantes de 'a' en 'result', si los hay
    while (i < a.Length)
    {
        result[k++] = a[i++];
    }

    // Copia los elementos restantes de 'b' en 'result', si los hay
    while (j < b.Length)
    {
        result[k++] = b[j++];
    }

    return result;
}
```

## 9. Añadiendo al Final

Implemente un método que reciba un valor *val* y añada dicho valor al final del array *a*, devolviendo un nuevo array con el elemento añadido.

**Respuesta:**

```
public static int[] Add(int[] array, int val)
{
    int[] newArray = new int[array.Length + 1];

    for (int i = 0; i < array.Length; i++)
```

```

    {
        newArray[i] = array[i];
    }

    newArray[array.Length] = val;

    return newArray;
}

```

## 10. Insertando

Implemente un método que, dado un entero *pos* y un valor *val*, inserte el valor *val* en la posición *pos* de *a*, desplazando los elementos existentes hacia la derecha, devuelva un nuevo array con el elemento insertado.

**Respuesta:**

```

public static int[] Insert(int[] array, int pos, int val)
{
    int[] newArray = new int[array.Length + 1];

    for (int i = 0; i < pos; i++)
    {
        newArray[i] = array[i];
    }

    newArray[pos] = val;

    for (int i = pos; i < array.Length; i++)
    {
        newArray[i + 1] = array[i];
    }

    return newArray;
}

```

## 11. Eliminando

Implemente un método que, dado un entero *pos* referente a determinada posición del array *a*, elimine el elemento que se encuentra en dicha posición, devuelva un nuevo array sin ese elemento.

**Respuesta:**

```

public static int[] RemoveAt(int[] array, int pos)
{
    int[] newArray = new int[array.Length - 1];

    for (int i = 0; i < pos; i++)
    {
        newArray[i] = array[i];
    }

    for (int i = pos + 1; i < array.Length; i++)
    {
        newArray[i - 1] = array[i];
    }
}

```

```

    }

    return newArray;
}

```

## 12. Representación binaria

1. Implemente un método que reciba un número entero no negativo y devuelva un string con su representación binaria.
2. Implemente un método que convierta un número de binario a decimal. (El número binario está representado por un string compuesto de 0s y 1s).

## 13. Es primo

Determinar si un número entero positivo es primo. Un número es primo si solo tiene dos divisores: 1 y el propio número.

## 14. Número perfecto

Determinar si un número entero positivo es perfecto. Un número es perfecto si la suma de sus divisores propios es igual a él. Ejemplo:  $28 = 1 + 2 + 4 + 7 + 14$ .

## 15. Substring

Un substring (subcadena) es una secuencia de caracteres que aparece consecutivamente en una cadena mayor. Dados los string *s* y *x*, implemente un método que diga si *x* es substring de *s*. Ejemplo: "" es substring de toda cadena, "a" es substring de "casa", "asap" no es substring de "casa".

## 16. Es palíndromo

Implemente un método que determine si *s* es palíndromo (se lee igual al derecho que al revés). Ejemplos: ana, anitalavalatina, zz.

## 17. Menor sufijo para ser palíndromo

Implemente un método que compute el menor string *t* tal que *s* + *t* es palíndromo.

## 18. Ordenando

Implemente un método que reciba un array de enteros *a* y devuelva un nuevo array con los mismos elementos ordenados de menor a mayor.

## 19. Mediana

Implemente un método que reciba un array de enteros y devuelva el elemento mediana. La mediana de un array es el elemento que tiene la misma cantidad de elementos mayores y menores en el array.

Ejemplos:

- Para el array [3, 5, 2, 8, 1]: La cantidad de números menores que 3 es la misma que la cantidad de números mayores que 3, por lo que 3 es la mediana.
- Para el array [3, 5, 2, 8]: El tamaño del array es par, por tanto definiremos la mediana como el elemento que tiene  $\frac{n}{2}$  elementos menores y  $\frac{n}{2} - 1$  elementos mayores, por lo tanto, la mediana es 5.

### Hint:

Ordena el array primero.