Práctica 1 – Complejidad Cognitiva.

Ejercicio 1

Escribir una función que determine si un número entero dado es primo o no. La función debe tomar un entero como entrada y devolver un valor booleano que indique si el número es primo o no.

Solución

```
bool EsPrimo(int n)
{
    if (n <= 1)
    {
        return false;
    }
    for (int i = 2; i <= Math.Sqrt(n); i++)
    {
        if (n % i == 0)
        {
            return false;
        }
    }
    return true;
}
```

Siguiendo las reglas de complejidad cognitiva, podemos identificar los siguientes elementos en el código:

Una condición (if) (Regla 1)

Un loop (for) (Regla 1)

Un cálculo matemático (usando la función Math.Sqrt) (Regla 1)

Hay estructuras de control anidadas en el código (Regla 2).

No se cumple la Regla 3, ya que no hay estructuras del lenguaje que permitan incluir varias sentencias en una sola línea.

Por lo tanto, el total de complejidad cognitiva sería de 4 (porque se cumple la Regla 1 tres veces).

Ejercicio 2

Escribir una función que calcule la suma de los dígitos de un número entero. La función debe tomar un entero como entrada y devolver la suma de los dígitos.

Solución

```
int SumaDigitos(int n)
{
    int suma = 0;
    while (n != 0)
    {
        suma += n % 10;
        n /= 10;
    }
    return suma;
}
```

Siguiendo las reglas de complejidad cognitiva, podemos identificar los siguientes elementos en el código:

Un loop while) (Regla 1)

No se cumple la Regla 2, ya que no hay estructuras de control anidadas en el código.

Tampoco se cumple la Regla 3, ya que no hay estructuras del lenguaje que permitan incluir varias sentencias en una sola línea.

Por lo tanto, el total de complejidad cognitiva sería de 1 (porque se cumple la Regla 1 una vez).

Ejercicio 3

Escribir una función que calcule el factorial de un número entero dado. La función debe tomar un entero como entrada y devolver su factorial.

Solución

```
int Factorial(int n)
{
    if (n == 0)
    {
        return 1;
    }
    int resultado = 1;
    for (int i = 1; i <= n; i++)
    {
        resultado *= i;
    }
    return resultado;
}
```

Ejercicio 4

Escribir una función que devuelva el número de palabras en una cadena dada. La función debe tomar una cadena como entrada y devolver un entero que indique el número de palabras.

Solución

```
int ContarPalabras(string cadena)
{
    int contador = 0;
    bool dentroDePalabra = false;
    foreach (char c in cadena)
    {
        if (Char.IsWhiteSpace(c))
        {
            dentroDePalabra = false;
        }
        else if (!dentroDePalabra)
        {
            contador++;
            dentroDePalabra = true;
        }
    }
    return contador;
}
```

Ejercicio 5

Escribir una función que ordene un arreglo de enteros utilizando el algoritmo de selección. La función debe tomar un arreglo de enteros como entrada y ordenarlo en orden ascendente.

Solución

```
void OrdenarSeleccion(int[] numeros)
{
    for (int i = 0; i < numeros.Length - 1; i++)
    {
        int minimoIndice = i;
        for (int j = i + 1; j < numeros.Length; j++)
        {
            if (numeros[j] < numeros[minimoIndice])
            {
                minimoIndice = j;
            }
        }
        int temp = numeros[i];
        numeros[i] = numeros[minimoIndice];
        numeros[minimoIndice] = temp;
    }
}
```

Ejercicio 6

Escribir una función que calcule el producto de dos matrices. La función debe tomar dos matrices como entrada y devolver una matriz que indique el resultado de la multiplicación.

Solución

```
int[,] MultiplicarMatrices(int[,] matriz1, int[,] matriz2)
{
    int filas1 = matriz1.GetLength(0);
    int columnas1 = matriz1.GetLength(1);
    int columnas2 = matriz2.GetLength(1);
    int[,] resultado = new int[filas1, columnas2];
    for (int i = 0; i < filas1; i++)
    {
        for (int j = 0; j < columnas2; j++)
        {
            for (int k = 0; k < columnas1; k++)
            {
                resultado[i, j] += matriz1[i, k] * matriz2[k, j];
            }
        }
    }
    return resultado;
}
```

Ejercicio 7

Escribir una función que determine si una cadena dada es un palíndromo o no. La función debe tomar una cadena como entrada y devolver un valor booleano que indique si la cadena es un palíndromo o no

Solución

```
bool EsPalindromo(string cadena)
{
    for (int i = 0, j = cadena.Length - 1; i < j; i++, j--)
    {
        if (cadena[i] != cadena[j])
        {
            return false;
        }
    }
    return true;
}
```

Ejercicio 8

Escribir una función que determine si una cadena es un palíndromo. La función debe tomar una cadena como entrada y devolver verdadero si la cadena es un palíndromo (es decir, se lee igual de adelante hacia atrás que de atrás hacia adelante) y falso en caso contrario.

Solución

```
bool EsPalindromo2(string cadena)
{
    int izquierda = 0;
    int derecha = cadena.Length - 1;
    while (izquierda < derecha)
    {
        if (cadena[izquierda] != cadena[derecha])
        {
            return false;
        }
        izquierda++;
        derecha--;
    }
    return true;
}
```

Ejercicio 9

Escribir una función que calcule el número de formas en que se pueden colocar k objetos de n posibles en orden, sin repetición. La función debe tomar dos enteros como entrada y devolver un entero que indique el número de formas posibles.

Solución

```
int Permutaciones(int n, int k)
{
    if (n < k)
    {
        return 0;
    }
    int resultado = 1;
    for (int i = n; i > n - k; i--)
    {
        resultado *= i;
    }
    return resultado;
}
```

Ejercicio 10

Escribir una función que determine si una cadena dada es un anagrama o no. La función debe tomar dos cadenas como entrada y devolver un valor booleano que indique si son anagramas o no.

Solución

```
bool EsAnagrama(string cadena1, string cadena2)
{
    if (cadena1.Length != cadena2.Length)
    {
        return false;
    }
    char[] chars1 = cadena1.ToCharArray();
    Array.Sort(chars1);
    string ordenada1 = new string(chars1);
    char[] chars2 = cadena2.ToCharArray();
    Array.Sort(chars2);
    string ordenada2 = new string(chars2);
    return ordenada1.Equals(ordenada2);
}
```

Ejercicio 11

Escribir una función que calcule el máximo común divisor de dos enteros utilizando el algoritmo de Euclides. La función debe tomar dos enteros como entrada y devolver un entero que indique el máximo común divisor.

Solución

```
int MaximoComunDivisor(int a, int b)
{
    while (b != 0)
    {
        int temp = b;
        b = a % b;
        a = temp;
    }
    return a;
}
```

Ejercicio 12

Escribir una función que calcule el enésimo número de Fibonacci. La función debe tomar un entero como entrada y devolver el enésimo número de Fibonacci.

Solución

```
int Fibonacci(int n)
{
    if (n <= 1)
    {
        return n;
    }
    int a = 0;
    int b = 1;
    for (int i = 2; i <= n; i++)
    {
        int temp = a + b;
        a = b;
        b = temp;
    }
    return b;
}
```

Ejercicio 13

Escribir una función que calcule la n-ésima serie de Fibonacci. La función debe tomar un entero n como entrada y devolver un entero que indique el valor de la n-ésima serie de Fibonacci.

Solución

```
int Fibonacci2(int n)
{
    if (n == 0)
    {
        return 0;
    }
    else if (n == 1)
    {
        return 1;
    }
    else
    {
        return Fibonacci(n - 1) + Fibonacci(n - 2);
    }
}
```

Ejercicio 14

Escribir una función que calcule la suma de los elementos en un arreglo de enteros. La función debe tomar un arreglo de enteros como entrada y devolver un entero que indique la suma de todos los elementos.

Solución

```
int SumaArreglo(int[] numeros)
{
    int suma = 0;
    foreach (int numero in numeros)
    {
        suma += numero;
    }
    return suma;
}
```

Ejercicio 15

Escribir una función que calcule el máximo común divisor de dos números enteros dados. La función debe tomar dos enteros como entrada y devolver su máximo común divisor.

Solución

```
int MCD(int a, int b)
{
    while (b != 0)
    {
        int temp = b;
        b = a % b;
        a = temp;
    }
    return a;
}
```

Ejercicio 16

Escribir una función que calcule la suma de los elementos en una matriz de enteros. La función debe tomar una matriz de enteros como entrada y devolver un entero que indique la suma total de los elementos en la matriz.

Solución

```
int SumaMatriz(int[,] matriz)
{
    int suma = 0;
    int filas = matriz.GetLength(0);
    int columnas = matriz.GetLength(1);
    for (int i = 0; i < filas; i++)
    {
        for (int j = 0; j < columnas; j++)
        {
            suma += matriz[i, j];
        }
    }
    return suma;
}
```

Ejercicio 17

Escribir una función que encuentre el número más grande en un arreglo de enteros. La función debe tomar un arreglo de enteros como entrada y devolver un entero que indique el valor del número más grande.

Solución

```
int NumeroMasGrande(int[] numeros)
{
    int maximo = numeros[0];
    for (int i = 1; i < numeros.Length; i++)
    {
        if (numeros[i] > maximo)
        {
            maximo = numeros[i];
        }
    }
    return maximo;
}
```

Ejercicio 18

Escribir una función que calcule la distancia euclidiana entre dos puntos en un plano cartesiano. La función debe tomar las coordenadas (x,y) de los dos puntos como entrada y devolver un número que indique la distancia entre ellos.

Solución

```
double DistanciaEuclidiana(double x1, double y1, double x2, double y2)
{
    double dx = x2 - x1;
    double dy = y2 - y1;
    return Math.Sqrt(dx * dx + dy * dy);
}
```