

ALGORITMIA – V2

RUBEN FELIPE TOVAR

Instructor: JHON CORREDOR

SENA-CIES

ADSO 2900177

NEIVA-HUILA

EASY

Nombre	Ruben Felipe Tovar Aviles	Fecha	dia	mes	año
Profesor	Jhon Corredor	Materia	ADSO		
Institución	SENA - CIES	Curso	2900177	Nota	

ALGORITMIA - V2

EASY

1. Calculadora de Distancia Manhattan

```
public static int DistanciaManhattan (int x1, int y1, int x2, int y2) {  
    int valorAbsX = Math.Abs (x1 - x2);  
    int valorAbsY = Math.Abs (y1 - y2);  
    int distManhattan = valorAbsX + valorAbsY;  
    return distManhattan;  
}
```

2. Calculadora de Resistencias en Serie/Paralelo

```
public static double CalcResistencia (double [] resistencias, bool enSerie) {  
    if (enSerie) {  
        return resistencias.Sum ();  
    } else {  
        double suma = 0;  
        foreach (double r in resistencias) {  
            if (r != 0) suma += 1.0 / r;  
        }  
        return suma != 0 ? 1.0 / suma : 0;  
    }  
}
```

3. Calculadora de Índice de Masa Corporal (IMC)

```
public static (double imc, string clasific) CalcularIMC (double peso, double alt)  
{  
    double imc = peso / (alt * alt);  
    string clasificacion;  
    if (imc < 18.5) clasificacion = "Bajo peso";  
    else if (imc < 25) clasificacion = "Normal";  
    else if (imc < 30) clasificacion = "Sobre peso";  
    else clasificacion = "Obesidad";  
    return (Math.Round (imc, 2), clasificacion);  
}
```

4. Simulador de Lanzamiento de Proyectil

```
public static (double alcance, double altMax) CalcProyectil (  
    double VInit, double angulo) {  
    double anguloRad = angulo * Math.PI / 180;  
    double g = 9.81;  
    double alcance = (VInit * VInit * Math.Sin(2 * anguloRad)) / g;  
    double altMax = (VInit * VInit * Math.Sin(anguloRad)) *  
        * Math.Sin(anguloRad)) / (2 * g);  
    return (Math.Round(alcance, 2), Math.Round(altMax, 2));  
}
```

5. Verificador de Formato de Email Básico

```
public static bool ValidEmail (string email) {  
    if (string.IsNullOrEmpty(email)) return false;  
    intarrobaIndex = email.IndexOf('@');  
    if (arrobaIndex <= 0 || arrobaIndex == email.Length - 1) return false;  
    string dominio = email.Substring(arrobaIndex + 1);  
    if (!dominio.Contains(".")) return false;  
    if (dominio.StartsWith(".") || dominio.EndsWith(".")) return false;  
    return email.All(c => char.IsLetterOrDigit(c)  
        || c == '@' || c == '.' || c == '-' || c == '+');  
}
```

6. Calculadora de Interés Compuesto

```
public static double CalcInteresCompuesto (double principal, double Tasa,  
    int Tiempo) {  
    return Math.Round(principal * Math.Pow(1 + Tasa / 100, Tiempo), 2);  
}
```

7. Calculadora de Área de Polígonos Regulares

```
public static double CalcAreaPolReg(int numLados, double longLado){  
    double angulo = Math.PI / numLados;  
    double area = (numLados * longLado * longLado) / (4 * Math.Tan(angulo));  
    return Math.Round(area, 2);  
}
```

8. Pedir una frase y contar las veces que aparece 'a'

```
public static int ContarA(string palabra){  
    int contador = 0;  
    foreach (char c in palabra)  
        if (c == 'a')  
            contador++;  
    return contador;  
}
```

9. Recibir un Cadena y contar Letras

```
public static int ContarLetras(string palabra){  
    int contador = 0;  
    foreach (char c in palabra)  
        if ((c >= 'A' && c <= 'Z') || (c >= 'a' && c <= 'z'))  
            contador++;  
    return contador;  
}
```

10. Pedir palabra y Mostrarla al revés

```
public static string PalabraReverso(string palabra){  
    string reverso = "";  
    for (int i = palabra.Length - 1; i >= 0; i--)  
        reverso += palabra[i];  
    return reverso;  
}
```

HARD

HARD

11. Verificar Número Auto compuesto

```
public static bool EsNumCompuesto (int numero) {  
    List<int> divisores = new List<int>();  
    for (int i = 1; i < numero; i++) {  
        if (numero % i == 0)  
            divisores.Add(i);  
    }  
    for (int i = 0; i < divisores.Count; i++) {  
        for (int z = i + 1; z < divisores.Count; z++) {  
            String concu = divisores[i].ToString() + divisores[z].ToString();  
            if (int.Parse(concu) == numero)  
                return true;  
        }  
    }  
    return false;  
}
```

12. Verificar Sudoku Parcial

```
public static bool VerifSudoku (int[] numeros) {  
    HashSet<int> visitos = new HashSet<int>();  
    foreach (int num in numeros) {  
        if (num != 0) {  
            if (num < 1 || num > 9 || visitos.Contains(num))  
                return false;  
            visitos.Add(num);  
        }  
    }  
    return true;  
}
```

Nombre	Fecha	dia	mes	año
Profesor	Materia			
Institución	Curso			Nota

13. Verificador de Números Vampiro

```

public static bool EsNumVampiro (int num) {
    if (num < 1000 || num > 9999) return False;
    string numSTR = num.ToString();
    for (int i = 10; i <= 99; i++) {
        if (num % i == 0) {
            int Factor = num / i;
            if (Factor >= 10 && Factor <= 99) {
                string Factores = i.ToString() + Factor.ToString();
                if (TienenMisMosDig (numSTR, Factores))
                    return true;
            }
        }
    }
    return False;
}

private static bool TienenMisMosDig (string sTR1, string sTR2) {
    return sTR1.Concat(sTR1.OrderBy(c => c)) == sTR2.Concat(sTR2.OrderBy(c => c));
}

```

14. Simulador de Crecimiento Poblacional

```

public static double CalcCrecimientoPoblacional (double poblacionInit,
    TasaNatalidad, TasaMortalidad, int años) {
    double poblacion = poblacionInit;
    double tasaCrecimiento = (TasaNatalidad - TasaMortalidad) / 100.0;
    for (int i = 0; i < años; i++) {
        poblacion += poblacion + tasaCrecimiento;
    }
    return Math.Round (poblacion, 2);
}

```

15. Detector de Subcadenas Repetidas

```
public static String FindSubString (String Texto) {
    for (int longitud = 1; longitud <= Texto.Length; i++) {
        String subcadena = Texto.Substring (0, longitud);
        String repetida = "";
        for (int i = 0; i < Texto.Length / longitud; i++) {
            repetida += subcadena;
        }
        if (repetida == Texto) {
            return subcadena;
        }
    }
    return Texto;
}
```

16. Simulador de Probabilidad de dados

```
public static Dictionary<int, int> SimularDados (int numDados, int numLanzas) {
    Random random = new Random();
    Dictionary<int, int> resultados = new Dictionary<int, int> ();
    for (int i = 0; i < numLanzas; i++) {
        int suma = 0;
        for (int j = 0; j < numDados; j++) {
            suma += random.Next (1, 7);
        }
        if (resultados.ContainsKey (suma)) {
            resultados [suma]++;
        } else {
            resultados [suma] = 1;
        }
    }
}
```

17. Simulador de Calificaciones Ponderadas

```
public static double CalcPromPonderado ( double[] clasifi, double[] pesos ) {  
    if ( clasifi.Length != pesos.Length ) return 0;  
    double sumProductos = 0;  
    double sumPesos = 0;  
    for ( int i = 0; i < clasificaciones.Length; i++ ) {  
        sumProductos += clasificaciones [i] * pesos [i];  
        sumPesos += pesos [i];  
    }  
    return sumPesos != 0 ? Math.Round ( sumProductos / sumPesos, 2 ) : 0;  
}
```

18. Contador de Ensamblajes Válidos de Fichas

```
public static int contadorTorreEstable ( int[] Fichas ) {  
    int contador = 0;  
    for ( int i = 0; i < Fichas.Length - 2; i++ ) {  
        int suma = Fichas [i] + Fichas [i + 1] + Fichas [i + 2];  
        if ( suma % 3 == 0 ) contador++;  
    }  
    return contador;  
}
```

19. Simulador de Parpadeo Alternante

```
public static bool tieneParpadeoALT ( int[] leds ) {  
    if ( leds.Length < 6 ) return false;  
    for ( int i = 0; i < leds.Length - 6; i++ ) {  
        bool valido = true;  
        for ( int z = 0; z < 5; z++ )  
            if ( leds [i + z] == leds [i + z + 1] ) { valido = false; break; }  
        if ( valido ) return true;  
    }  
    return false;  
}
```

20. Contador de Giros Equilibrados

```
public static bool GirosEquilibrados (char[] mov) {
    int horiz = 0, Verti = 0;
    foreach (char mov in mov) {
        if (mov == 'L' || mov == 'R') {
            horiz++;
        } else if (mov == 'U' || mov == 'D') {
            Verti++;
        }
    }
    return horiz == Verti;
}
```

21. Agrupador por Dígito Final

```
public static int[] AgruparPorDigitoFinal (int[] g) {
    int[] grupos = new int[10];
    foreach (int num in grupos) {
        int ultimoDigito = Math.Abs (sum) % 10;
        grupos [ultimoDigito]++;
    }
    return grupos;
}
```

22. Detector de Cambios Bruscos

```
public static int ContarCambiosBruscos (int[] valores) {
    int contador = 0;
    for (int i = 1; i < valores.Length; i++) {
        if (Math.Abs (valores [i] - valores [i - 1]) > 10) {
            contador++;
        }
    }
    return contador;
}
```

Nombre	Fecha	dia	mes	año
Profesor	Materia			
Institución	Curso			Nota

23. Verificador de Encuadre de Fichas

```
public static bool PuedenEncuadrar (int[] Fichas1, int[] Fichas2) {
    if (Fichas1.Length != Fichas2.Length)
        return false;
    for (int i = 0; i < Fichas1.Length; i++) {
        // i Fic Fichas
        if (Fichas1[i] + Fichas2[i] != 0)
            return false;
    }
    return true;
}
```

24. Contador de Zonas Oscuras

```
public static int ContarZonasOscuras (int[] leds) {
    int contador = 0, seguidos = 0;
    for (int i = 0; i < leds.Length; i++) {
        if (leds[i] == 0) {
            seguidos++;
            if (seguidos == 3) count++;
        } else {
            seguidos = 0;
        }
    }
    return count;
}
```

25. Medidor Progreso del Cubo

```
public static double CalcPorcResuelto (int [] piezas) {
    if (piezas.length == 0)
        return 0.0;
    int correctas = 0;
    foreach (int pieza in piezas) {
        if (pieza == 1)
            correctas++;
    }
    return (double) correctas / piezas.length * 100.0;
}
```

26. Analizador de Rachu Gondora

```
public static int RachuMasLargo (int [] resultados) {
    int rachuMax = 0;
    int rachuAct = 0;
    foreach (int resul in resultados) {
        if (resultado == 1)
            rachuAct++;
        if (rachuAct > rachuMax)
            rachuMax = rachuAct;
        else
            rachuAct = 0;
    }
    return rachuMax;
}
```

27. Filtro de Suma Numérica

```
public static int FiltroDeSumaNumerica(int[] arreglo) {
    int[] resultado = new int[arreglo.Length];
    for (int i = 0; i < arreglo.Length; i++) {
        int count = 0;
        for (int z = 0; z < arreglo.Length; z++) {
            if (arreglo[z] == arreglo[i]) count++;
        }
        resultado[i] = arreglo[i] + count;
    }
    return resultado;
}
```

28. Orden Pseudorandom Numérica

```
public static int OrdenPseudorandom(int[] arreglo) {
    int[] resultado = new int[arreglo.Length];
    bool[] usados = new bool[arreglo.Length];
    int pos = 0;
    for (int i = 0; i < arreglo.Length; i++) {
        if (usados[i]) continue;
        int count = 1;
        for (int z = i + 1; z < arreglo.Length; z++) {
            if (arreglo[z] == arreglo[i] && count++ && usados[z] == false) {
                for (int y = 0; y < arreglo.Length; y++) {
                    if (arreglo[y] == arreglo[z]) {
                        resultado[pos++] = arreglo[y];
                    }
                }
            }
        }
        for (int i = 0; i < resultado.Length - 1; i++) {
            for (int z = i + 1; z < resultado.Length; z++) {
                int ci = 0, cz = 0;
                for (int y = 0; y < arreglo.Length; y++) {
                    if (arreglo[y] == resultado[i]) ci++;
                    if (arreglo[y] == resultado[z]) cz++;
                }
                if (ci > cz) {
                    int fomp = resultado[i];
                    resultado[i] = resultado[z];
                    resultado[z] = fomp;
                }
            }
        }
    }
    return resultado;
}
```

29. Valido de Valor Segun Posicion

```
public static bool EsImpulsoAlterno(int[] arreglo) {
    for (int i = 1; i < arreglo.Length; i++) {
        if (i % 2 == 0) {
            if (arreglo[i] <= arreglo[i - 1]) return false;
        } else {
            if (arreglo[i] >= arreglo[i - 1]) return false;
        }
    }
    return true;
}
```

30. Ajustador de Arreglos por Mitades

```
public static int AjustarMitades(int[] arreglo) {
    if (arreglo.Length % 2 != 0) return -1;
    int mitad = arreglo.Length / 2;
    int suma1 = 0, suma2 = 0;
    for (int i = 0; i < mitad; i++) suma1 += arreglo[i];
    for (int i = mitad; i < arreglo.Length; i++) suma2 += arreglo[i];
    int diferencia = Math.Abs(suma1 - suma2);
    if (diferencia % 2 != 0) return -1;
    return diferencia / 2;
}
```