

## **BNF de C#:**

<https://learn.microsoft.com/en-us/dotnet/csharp/language-reference/language-specification/grammar#a3-syntactic-grammar>

## **BNF de javascript:**

<https://gist.github.com/avdg/1f10e268e484b1284b46>

## **Breve historia de javascript:**

JavaScript fue creado en 1995 por Brendan Eich mientras trabajaba en Netscape Communications. Originalmente se llamaba "LiveScript", pero se cambió posteriormente a "JavaScript" para capitalizar la popularidad emergente de Java en ese momento. JavaScript fue diseñado como un lenguaje de scripting para navegadores web, permitiendo a los desarrolladores crear interacciones y funcionalidades dinámicas en las páginas web. A pesar de su nombre, JavaScript no tiene relación directa con Java, es un lenguaje independiente. Con el tiempo, JavaScript se convirtió en el lenguaje de programación fundamental para el desarrollo web, habilitando una amplia gama de aplicaciones y experiencias interactivas en línea. Su evolución continuó con la estandarización bajo el nombre de ECMAScript, y las sucesivas versiones han introducido nuevas características y mejoras para mantenerlo relevante y poderoso en el mundo del desarrollo web moderno.

## **Breve historia de C#:**

C# (C Sharp) es un lenguaje de programación moderno desarrollado por Microsoft. Fue creado por Anders Hejlsberg y su equipo en la década de 1990 como parte de la iniciativa de Microsoft para desarrollar una plataforma de desarrollo que fuera fácil de usar y potente al mismo tiempo. La primera versión de C# se lanzó en 2000 como parte de la plataforma .NET.

C# se diseñó con varios objetivos en mente, como la facilidad de uso, la seguridad y la capacidad de aprovechar al máximo la plataforma .NET para desarrollar aplicaciones de software para una amplia gama de dispositivos y sistemas operativos. C# toma prestadas características de otros lenguajes como C++, Java y Visual Basic, lo que lo convierte en un lenguaje versátil.

A lo largo de los años, C# ha evolucionado y ha ganado popularidad en el desarrollo de aplicaciones de escritorio, aplicaciones web, aplicaciones móviles y servicios en la nube. Con la introducción de .NET Core (y su sucesor .NET 5+), C# se volvió aún más versátil, permitiendo desarrollar aplicaciones multiplataforma.

Un hito importante en la historia de C# fue la introducción de LINQ (Language Integrated Query) en C# 3.0, que permitió realizar consultas directamente en colecciones de datos, bases de datos y otros orígenes, de una manera más legible y funcional.

Con el tiempo, C# ha seguido incorporando características modernas de programación, como patrones de diseño asincrónicos y await/async para manejar eficientemente operaciones asíncronas, así como mejoras en la sintaxis y el rendimiento.

En resumen, la historia de C# es la de un lenguaje de programación desarrollado por Microsoft que ha evolucionado a lo largo de los años para convertirse en una herramienta esencial para el desarrollo de una amplia variedad de aplicaciones en diversas plataformas y entornos.

## **Los algoritmos seleccionados fueron los siguientes:**

### -Algoritmo de Ordenamiento de Vectores

En este caso seleccionamos el método de ordenamiento Bubblesort, que se basa en iteraciones que comparan pares de elementos desde el comienzo hasta el final de la lista, si el primer elemento del par es mayor que el segundo entonces intercambian lugar, y así comienza otra iteración con el siguiente par, así sucesivamente hasta ordenar completamente el vector.

A continuación mostraremos el código en cada lenguaje.

## En JavaScript:

```
function bubbleSort(arr) {
  const n = arr.length;
  for (let i = 0; i < n - 1; i++) {
    for (let j = 0; j < n - i - 1; j++) {
      if (arr[j] > arr[j + 1]) {
        const temp = arr[j];
        arr[j] = arr[j + 1];
        arr[j + 1] = temp;
      }
    }
  }
}

function printVector(arr) {
  console.log(arr.join(' '));
}

const vector = [10, 5, 7, 9, 4];

console.log("Vector original:");
printVector(vector);

console.time("Tiempo de ejecución");

const memoryBefore = process.memoryUsage().heapUsed;

bubbleSort(vector);

const memoryAfter = process.memoryUsage().heapUsed;

console.timeEnd("Tiempo de ejecución");

console.log("\nVector ordenado:");
printVector(vector);

const memoryUsed = memoryAfter - memoryBefore;
console.log(`\nMemoria utilizada (bytes): ${memoryUsed}`);
```

## Post Ejecución

```
10 5 7 9 4
Tiempo de ejecución: 0.322ms

Vector ordenado:
4 5 7 9 10

Memoria utilizada (bytes): 1776
```

## En C#

```
class BubbleSortExample
{
    static void Main(string[] args)
    {
        int[] vector = { 10, 5, 7, 9, 4 };

        Console.WriteLine("Vector original:");
        PrintVector(vector);

        long memoryBefore = GC.GetTotalMemory(true);

        BubbleSort(vector);

        long memoryAfter = GC.GetTotalMemory(true);

        Console.WriteLine("\nVector ordenado:");
        PrintVector(vector);

        long memoryUsed = memoryAfter - memoryBefore;
        Console.WriteLine($"Memoria utilizada (bytes): {memoryUsed}");
    }

    static void BubbleSort(int[] arr)
    {
        int n = arr.Length;
        for (int i = 0; i < n - 1; i++)
        {
            for (int j = 0; j < n - i - 1; j++)
            {
                if (arr[j] > arr[j + 1])
                {
                    int temp = arr[j];
                    arr[j] = arr[j + 1];
                    arr[j + 1] = temp;
                }
            }
        }
    }
}
```

```
static void PrintVector(int[] arr)
{
    foreach (int num in arr)
    {
        Console.Write(num + " ");
    }
    Console.WriteLine();
}
```

## Post Ejecución

Vector original:

10 5 7 9 4

Vector ordenado:

4 5 7 9 10

Memoria utilizada (bytes): -104

Y con un tiempo de ejecución de 0.006 segundos.

Lenguaje	Tiempo de ejecución	Memoria Utilizada
Javascript	0.332ms	1776
C#	6ms	104

Otro aspecto a comparar es la complejidad de los códigos, pero en este caso son fáciles de entender y no toman funciones que puedan generar una complicación al momento de ejecutar y/o leer el código desde fuera, lo único que poseen distinto es que dentro del código de JS añadimos una función para lograr obtener el tiempo de ejecución en la terminal (y en ambas la memoria utilizada).

Estos resultados están sujetos a la máquina y los programas con los que se realizaron las pruebas, siendo LINQPad 7 para C# y Visual Studio Code para JS.

### Justificación de rendimiento:

#### \*Memoria:

JavaScript es un lenguaje interpretado y con tipado dinámico, lo que puede llevar a ciertas características de ejecución que pueden afectar el rendimiento y la memoria. C# es un lenguaje tipado estáticamente y a menudo se compila en código nativo antes de la ejecución, lo que puede resultar en una mayor eficiencia en términos de rendimiento. En los resultados se puede ver esta situación.

#### \*Tiempo de ejecución:

El compilador de JavaScript utilizado tiene optimizaciones un poco mas agresivas, que resulta en una ejecución más rápida del código Bubble Sort en

comparación con un compilador de C# que no realiza optimizaciones similares. (cosa que tambien gasta mas memoria)

### -Algoritmo para representación de árboles binarios

#### En C#:

```
void Main(){}  
class TreeNode  
{  
    public int Value;  
    public TreeNode Left;  
    public TreeNode Right;  
  
    public TreeNode(int value)  
    {  
        Value = value;  
        Left = null;  
        Right = null;  
    }  
}  
  
class BinarySearchTree  
{  
    public TreeNode Root;  
  
    public BinarySearchTree()  
    {  
        Root = null;  
    }  
  
    public void Insert(int value)  
    {  
        TreeNode newNode = new TreeNode(value);  
  
        if (Root == null)  
        {  
            Root = newNode;  
            return;  
        }  
    }  
}
```

```

        TreeNode current = Root;
        while (true)
        {
            if (value < current.Value)
            {
                if (current.Left == null)
                {
                    current.Left = newNode;
                    return;
                }
                current = current.Left;
            }
            else
            {
                if (current.Right == null)
                {
                    current.Right = newNode;
                    return;
                }
                current = current.Right;
            }
        }
    }
}

```

```

class Program
{
    static void Main(string[] args)
    {
        int[] values = { 1, 8, 6, 34, 12, 21, 72, 15, 50, 45 };
        BinarySearchTree tree = new BinarySearchTree();

        foreach (int value in values)
        {
            tree.Insert(value);
        }

        PrintTree(tree.Root, 0);
    }

    static void PrintTree(TreeNode node, int indent)
    {
        if (node == null)
        {
            return;
        }

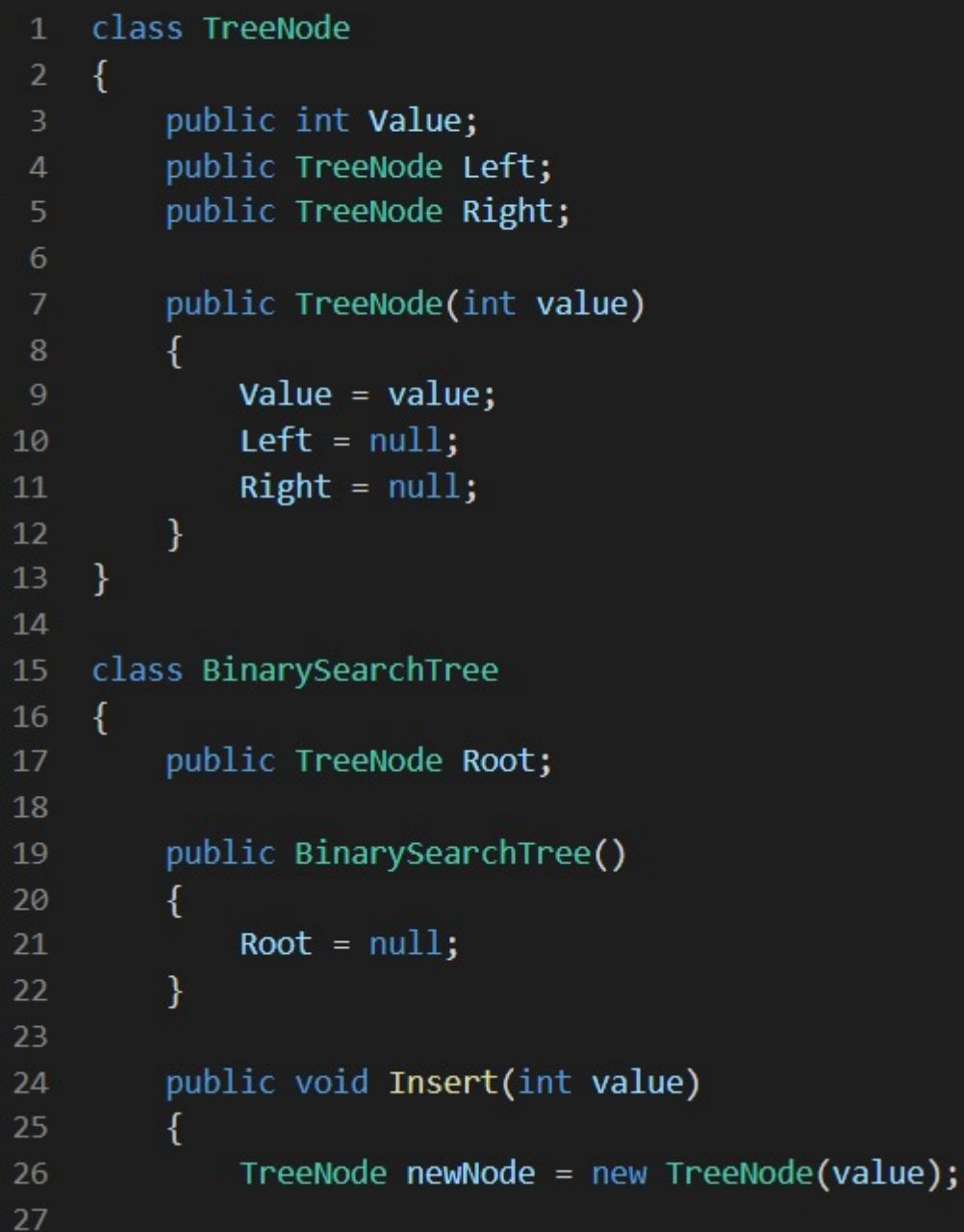
        PrintTree(node.Right, indent + 4);
        Console.WriteLine(new string(' ', indent) + node.Value);
        PrintTree(node.Left, indent + 4);
    }
}

```

### Benchmark de C#:

```
1
5 Tiempo de ejecución: 34 ms
6 Uso máximo de memoria: 26452 KB
7
```

### En JavaScript:



```
1  class TreeNode
2  {
3      public int Value;
4      public TreeNode Left;
5      public TreeNode Right;
6
7      public TreeNode(int value)
8      {
9          Value = value;
10         Left = null;
11         Right = null;
12     }
13 }
14
15 class BinarySearchTree
16 {
17     public TreeNode Root;
18
19     public BinarySearchTree()
20     {
21         Root = null;
22     }
23
24     public void Insert(int value)
25     {
26         TreeNode newNode = new TreeNode(value);
27
```



```
27
28     if (Root == null)
29     {
30         Root = newNode;
31         return;
32     }
33
34     TreeNode current = Root;
35     while (true)
36     {
37         if (value < current.Value)
38         {
39             if (current.Left == null)
40             {
41                 current.Left = newNode;
42                 return;
43             }
44             current = current.Left;
45         }
46         else
47         {
48             if (current.Right == null)
49             {
50                 current.Right = newNode;
51                 return;
52             }
53             current = current.Right;
54         }
55     }
56 }
57 }
```

### **Benchmark de Javascript:**

```
}  
Tiempo de ejecución: 13 ms  
Uso aproximado de memoria: 44 KB  
|
```

Lenguaje	Tiempo de ejecución	Memoria Utilizada
Javascript	13ms	44 KB
C#	34ms	26452 KB

Como se ve javascript otra vez ganó frente al tiempo de ejecución relacionado con lo explicado anteriormente pero la diferencia mayor y notoria en este caso, se da en la memoria utilizada donde en esta ocasión javascript ganó.

La justificación se da en que C# se ejecuta en una máquina virtual que proporciona administración de memoria más sofisticada que JavaScript. Esto puede llevar a una mayor asignación de memoria inicial para alojar objetos y estructuras de datos en C# en comparación con JavaScript.

También puede ser por los tipos de datos en C#, que suelen ser más ricos y pueden contener más metadatos que en JavaScript. Esto puede llevar a un aumento en el uso de memoria.