

C# Y .NET 9

Parte 5. Estructuras de datos dinámicas

2025-03

Rafael Alberto Moreno Parra
ramsoftware@gmail.com

Contenido

Tabla de ilustraciones.....	4
Acerca del autor.....	6
Licencia de este libro	6
Licencia del software	6
Marcas registradas	7
Dedicatoria	8
El ArrayList	9
Adicionar, tamaño, buscar e imprimir	9
Borrar elemento	11
Cambiar Elemento	13
Insertar Elemento.....	15
Referenciar con una variable, un rango de la lista	17
Tres formas de recorrer un ArrayList.....	19
Borrar completamente un ArrayList	21
Borrar un rango en un ArrayList.....	22
Guardar el ArrayList en un arreglo estático	24
Agregar un arreglo estático a un ArrayList	26
Inserta un arreglo estático en una determinada posición del ArrayList.....	27
Insertar varios ArrayList dentro de un ArrayList.....	28
Invertir un ArrayList.....	30
Invertir un rango de datos en el ArrayList.....	31
Ordena un ArrayList.....	33
Búsqueda Binaria	34
Capacidad del ArrayList	36
Detectar el tipo de dato del elemento del ArrayList	38
List.....	40
Métodos similares a los de ArrayList	40
Métricas: Comparativa de desempeño de ArrayList vs List vs Arreglo estático	45
Ordenando con tipo int.....	45
Con .NET 9	49
Con .NET 8	51
.NET 9 vs .NET 8	53
Ordenando con tipo double	54

Con .NET 9	58
Con .NET 8	60
.NET 9 vs .NET 8	62
Ordenando con tipo char	63
Con .NET 9	67
Con .NET 8	69
.NET 9 vs .NET 8	71
Lista de objetos.....	72
Listas en Listas	75
Dictionary	85
Uso de llaves	85
Llaves tipo string	87
Manejo de objetos en un Dictionary	89
Queue (Cola).....	91
Dato definido en la cola	93
Objetos en la cola.....	95
Stack (Pila)	97
Dato definido en la pila.....	99
Objetos en la pila	101
Hashtable	103
Manejo de objetos en un Hashtable	105
SortedList	107
LinkedList	109
Objetos en LinkedList.....	112

Tabla de ilustraciones

Ilustración 1: ArrayList Adicionar, tamaño, buscar e imprimir.....	10
Ilustración 2: ArrayList, borrar elemento.....	12
Ilustración 3: ArrayList, cambiar elemento	14
Ilustración 4: ArrayList, insertar elemento.....	16
Ilustración 5: ArrayList, referenciar con una variable, un rango de la lista.....	18
Ilustración 6: ArrayList, tres formas de recorrerlo.....	20
Ilustración 7: Borrar completamente un ArrayList.....	21
Ilustración 8: Borrar un rango en un ArrayList	23
Ilustración 9: Guardar el ArrayList en un arreglo estático.....	25
Ilustración 10: Agregar un arreglo estático a un ArrayList.....	26
Ilustración 11: Inserta un arreglo estático en una determinada posición del ArrayList.....	27
Ilustración 12: Insertar varios ArrayList dentro de un ArrayList.....	29
Ilustración 13: Invertir un ArrayList	30
Ilustración 14: Invertir un rango de datos en el ArrayList.....	32
Ilustración 15: Ordena un ArrayList.....	33
Ilustración 16: Búsqueda Binaria	35
Ilustración 17: Capacidad del ArrayList.....	37
Ilustración 18: Detectar el tipo de dato del elemento del ArrayList	39
Ilustración 19: List, métodos.....	44
Ilustración 20: Ordenando enteros con .NET 9	49
Ilustración 21: Ordenando enteros con .NET 9	50
Ilustración 22: Ordenando enteros con .NET 8	51
Ilustración 23: Ordenando enteros con .NET 8	52
Ilustración 24: Ordenando enteros con .NET 9 y .NET 8.....	53
Ilustración 25: Ordenando tipo double con .NET 9	58
Ilustración 26: Ordenando tipo double con .NET 9	59
Ilustración 27: Ordenando tipo double con .NET 8	60
Ilustración 28: Ordenando tipo double con .NET 8	61
Ilustración 29: Ordenando tipo double con .NET 8 y .NET 9.....	62
Ilustración 30: Ordenando tipo char con .NET 9	67
Ilustración 31: Ordenando tipo char con .NET 9	68
Ilustración 32: Ordenando tipo char con .NET 8	69
Ilustración 33: Ordenando tipo char con .NET 8	70
Ilustración 34: Ordenando tipo char con .NET 9 y .NET 8.....	71
Ilustración 35: Lista de objetos	74
Ilustración 36: Uso de listas para simular un sistema de información	83
Ilustración 37: Uso de listas para simular un sistema de información	83
Ilustración 38: Uso de listas para simular un sistema de información	84
Ilustración 39: Dictionary	86
Ilustración 40: Dictionary	88
Ilustración 41: Dictionary, manejo de objetos	90
Ilustración 42: Queue	92
Ilustración 43: Dato definido en la cola	94
Ilustración 44: Objetos en la cola.....	96
Ilustración 45: Stack	98

Ilustración 46: Dato definido en la pila.....	100
Ilustración 47: Objetos en la pila	102
Ilustración 48: Hashtable	104
Ilustración 49: Manejo de objetos en un Hashtable.....	106
Ilustración 50: SortedList	108
Ilustración 51: LinkedList	111
Ilustración 52: Objetos en LinkedList.....	113

Acerca del autor

Rafael Alberto Moreno Parra

ramsoftware@gmail.com o enginelifelife@hotmail.com

Sitio Web: <http://darwin.50webs.com> (dedicado a la investigación de algoritmos evolutivos y vida artificial).

Github: <https://github.com/ramsoftware>

Youtube: <https://www.youtube.com/@RafaelMorenoP>

Licencia de este libro



Licencia del software

Todo el software desarrollado aquí tiene licencia LGPL "Lesser General Public License" [1]



Marcas registradas

En este libro se hace uso de las siguientes tecnologías registradas:

Microsoft ® Windows ® Enlace: <http://windows.microsoft.com/en-US/windows/home>

Microsoft ® Visual Studio 2022 ® Enlace: <https://visualstudio.microsoft.com/es/vs/>

En recuerdo de Michu



El ArrayList

Adicionar, tamaño, buscar e imprimir

En C# está el ArrayList, que es un contenedor de objetos de cualquier tipo.

E/001.cs

```
using System.Collections;

namespace Ejemplo {
    class Program {
        static void Main() {
            //Declara la lista que almacenará cadenas
            ArrayList ListaAnimales = new();

            //Adiciona elementos a la lista
            ListaAnimales.Add("Ballena");
            ListaAnimales.Add("Tortuga marina");
            ListaAnimales.Add("Tiburón");
            ListaAnimales.Add("Estrella de mar");
            ListaAnimales.Add("Hipocampo");
            ListaAnimales.Add("Serpiente marina");
            ListaAnimales.Add("Delfín");
            ListaAnimales.Add("Pulpo");
            ListaAnimales.Add("Pingüinos");
            ListaAnimales.Add("Calamar");

            //Tamaño la lista
            int tamano = ListaAnimales.Count;
            Console.WriteLine("Tamaño de la lista: " + tamano);

            //Traer un determinado elemento de la lista
            int posicion = 7;
            string texto = ListaAnimales[posicion].ToString();
            Console.WriteLine("En posición: " + posicion + " es: " + texto);

            //Nos dice si existe un determinado elemento en la lista
            string buscar = "Pulpo";
            bool Existe = ListaAnimales.Contains(buscar);
            Console.WriteLine("Busca: " + buscar + " Resultado: " + Existe);

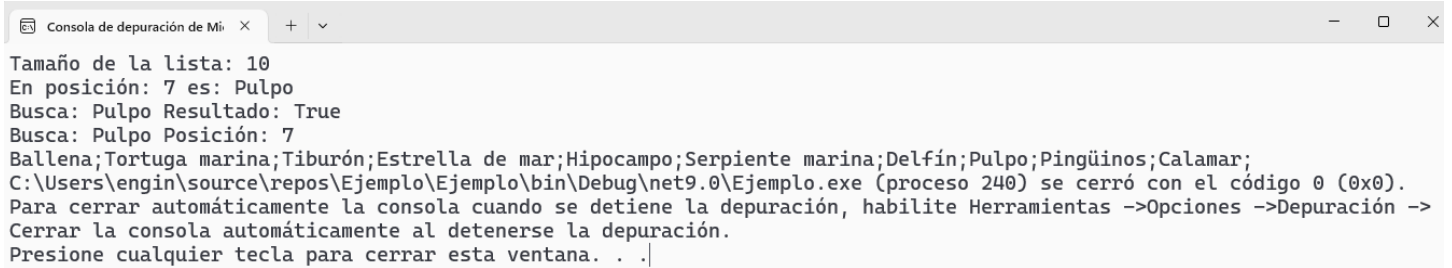
            //Nos dice la posición donde encontró el elemento en la lista
            int posBusca = ListaAnimales.IndexOf(buscar);
            Console.WriteLine("Busca: " + buscar + " Posición: " + posBusca);

            //Imprime la lista
```

```

        for (int Cont = 0; Cont < ListaAnimales.Count; Cont++)
            Console.WriteLine(ListaAnimales[Cont] + ";");
    }
}

```



Consola de depuración de Míi x + v

Tamaño de la lista: 10
 En posición: 7 es: Pulpo
 Busca: Pulpo Resultado: True
 Busca: Pulpo Posición: 7
 Ballena;Tortuga marina;Tiburón;Estrella de mar;Hipocampo;Serpiente marina;Delfín;Pulpo;Pingüinos;Calamar;
 C:\Users\engin\source\repos\Ejemplo\Ejemplo\bin\Debug\net9.0\Ejemplo.exe (proceso 240) se cerró con el código 0 (0x0).
 Para cerrar automáticamente la consola cuando se detiene la depuración, habilite Herramientas ->Opciones ->Depuración ->
 Cerrar la consola automáticamente al detenerse la depuración.
 Presione cualquier tecla para cerrar esta ventana. . .|

Ilustración 1: ArrayList Adicionar, tamaño, buscar e imprimir

Los ArrayLists empiezan en cero. En el código anterior se muestran las funciones para adicionar al ArrayList, determinar el tamaño (número de elementos que tiene), decir si existe un determinado elemento y mostrar la lista.

```
using System.Collections;

namespace Ejemplo {
    class Program {
        static void Main() {
            //Declara la lista
            ArrayList ListaAnimales = new();

            //Adiciona elementos a la lista
            ListaAnimales.Add("Ballena");
            ListaAnimales.Add("Tortuga marina");
            ListaAnimales.Add("Tiburón");
            ListaAnimales.Add("Hipocampo");
            ListaAnimales.Add("Delfín");
            ListaAnimales.Add("Pulpo");
            ListaAnimales.Add("Caballito de mar");
            ListaAnimales.Add("Coral");
            ListaAnimales.Add("Pingüinos");

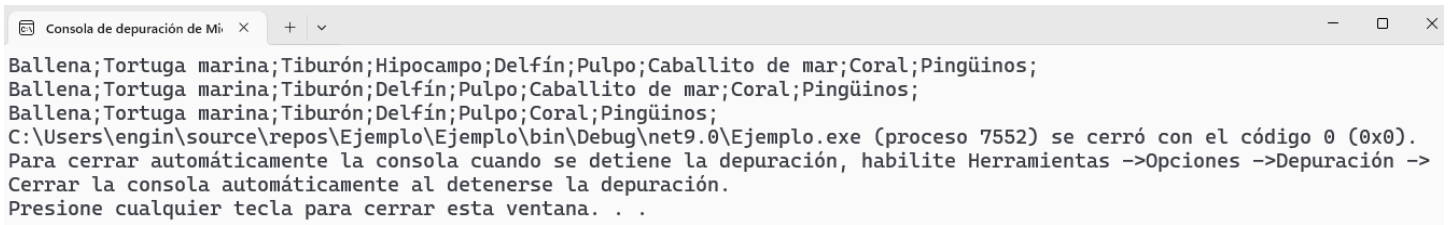
            //Imprime la lista
            for (int Cont = 0; Cont < ListaAnimales.Count; Cont++)
                Console.Write(ListaAnimales[Cont] + ";");
            Console.WriteLine(" ");

            //Retira elemento de la lista
            ListaAnimales.Remove("Hipocampo");

            //Imprime de nuevo la lista
            for (int Cont = 0; Cont < ListaAnimales.Count; Cont++)
                Console.Write(ListaAnimales[Cont] + ";");
            Console.WriteLine(" ");

            //Elimina el objeto de determinada posición.
            ListaAnimales.RemoveAt(5);

            //Imprime de nuevo la lista
            for (int Cont = 0; Cont < ListaAnimales.Count; Cont++)
                Console.Write(ListaAnimales[Cont] + ";");
        }
    }
}
```



```
Consola de depuración de Mi × + v
Ballena;Tortuga marina;Tiburón;Hipocampo;Delfín;Pulpo;Caballito de mar;Coral;Pingüinos;
Ballena;Tortuga marina;Tiburón;Delfín;Pulpo;Caballito de mar;Coral;Pingüinos;
Ballena;Tortuga marina;Tiburón;Delfín;Pulpo;Coral;Pingüinos;
C:\Users\engin\source\repos\Ejemplo\Ejemplo\bin\Debug\net9.0\Ejemplo.exe (proceso 7552) se cerró con el código 0 (0x0).
Para cerrar automáticamente la consola cuando se detiene la depuración, habilite Herramientas ->Opciones ->Depuración ->
Cerrar la consola automáticamente al detenerse la depuración.
Presione cualquier tecla para cerrar esta ventana. . .
```

Ilustración 2: ArrayList, borrar elemento

Dos técnicas para eliminar elementos de un ArrayList, buscando el elemento y eliminándolo, o dada una posición se elimina lo que hay allí.

Cambiar Elemento

Con la posición de la cadena en la lista, se puede cambiar directamente.

E/003.cs

```
using System.Collections;

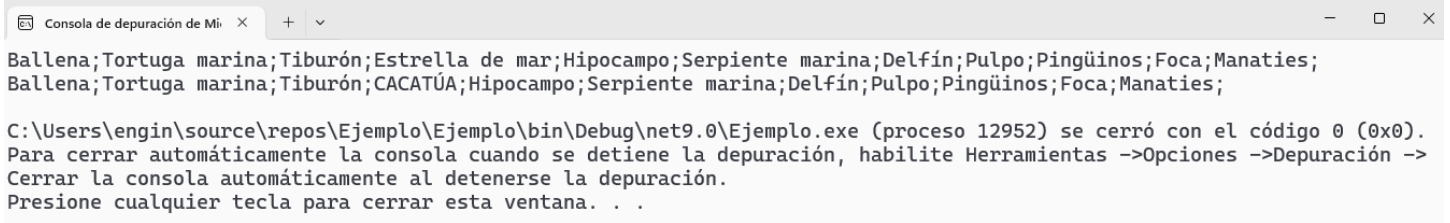
namespace Ejemplo {
    class Program {
        public static void Main() {
            //Declara la lista que almacenará cadenas
            ArrayList ListaAnimales = new();

            //Adiciona elementos a la lista
            ListaAnimales.Add("Ballena");
            ListaAnimales.Add("Tortuga marina");
            ListaAnimales.Add("Tiburón");
            ListaAnimales.Add("Estrella de mar");
            ListaAnimales.Add("Hipocampo");
            ListaAnimales.Add("Serpiente marina");
            ListaAnimales.Add("Delfín");
            ListaAnimales.Add("Pulpo");
            ListaAnimales.Add("Pingüinos");
            ListaAnimales.Add("Foca");
            ListaAnimales.Add("Manatíes");

            //Imprime valores
            for (int Cont = 0; Cont < ListaAnimales.Count; Cont++)
                Console.Write(ListaAnimales[Cont] + ";");
            Console.WriteLine(" ");

            //Cambia una cadena en la lista
            ListaAnimales[3] = "CACATÚA";

            //Imprime valores
            for (int Cont = 0; Cont < ListaAnimales.Count; Cont++)
                Console.Write(ListaAnimales[Cont] + ";");
            Console.WriteLine(" ");
        }
    }
}
```



```
Consola de depuración de Mi... X + v
Ballena;Tortuga marina;Tiburón;Estrella de mar;Hipocampo;Serpiente marina;Delfín;Pulpo;Pingüinos;Foca;Manaties;
Ballena;Tortuga marina;Tiburón;CACATÚA;Hipocampo;Serpiente marina;Delfín;Pulpo;Pingüinos;Foca;Manaties;

C:\Users\engin\source\repos\Ejemplo\Ejemplo\bin\Debug\net9.0\Ejemplo.exe (proceso 12952) se cerró con el código 0 (0x0).
Para cerrar automáticamente la consola cuando se detiene la depuración, habilite Herramientas ->Opciones ->Depuración ->
Cerrar la consola automáticamente al detenerse la depuración.
Presione cualquier tecla para cerrar esta ventana. . .
```

Ilustración 3: ArrayList, cambiar elemento

Insertar Elemento

Se puede insertar un elemento en la lista simplemente señalando su posición.

E/004.cs

```
using System.Collections;

namespace Ejemplo {
    class Program {
        public static void Main() {
            //Declara la lista que almacenará cadenas
            ArrayList ListaAnimales = new();

            //Adiciona elementos a la lista
            ListaAnimales.Add("Ballena");
            ListaAnimales.Add("Tortuga marina");
            ListaAnimales.Add("Tiburón");
            ListaAnimales.Add("Estrella de mar");
            ListaAnimales.Add("Hipocampo");
            ListaAnimales.Add("Serpiente marina");
            ListaAnimales.Add("Delfín");
            ListaAnimales.Add("Pulpo");
            ListaAnimales.Add("Pingüinos");
            ListaAnimales.Add("Calamar");
            ListaAnimales.Add("Foca");
            ListaAnimales.Add("Manatíes");

            //Imprime valores
            for (int Cont = 0; Cont < ListaAnimales.Count; Cont++)
                Console.Write(ListaAnimales[Cont] + ";");
            Console.WriteLine(" ");

            //Inserta una cadena en la posición 4 de la lista
            ListaAnimales.Insert(4, "CACATÚA");

            //Imprime valores
            for (int Cont = 0; Cont < ListaAnimales.Count; Cont++)
                Console.Write(ListaAnimales[Cont] + ";");
            Console.WriteLine(" ");
        }
    }
}
```

;Ballena;Tortuga marina;Tiburón;Estrella de mar;Hi
pocampo;Serpiente marina;Delfín;Pulpo;Pingüinos;Ca
lamar;Foca;Manaties

;Ballena;Tortuga marina;Tiburón;Estrella de mar;CA
CATÚA;Hipocampo;Serpiente marina;Delfín;Pulpo;Ping
üinos;Calamar;Foca;Manaties

Ilustración 4: ArrayList, insertar elemento

Referenciar con una variable, un rango de la lista

Dada una lista A, se crea una variable B de tipo ArrayList que haga referencia a un rango de esa lista A. Como es una referencia, si se modifica un elemento de B, ese cambio ocurrirá en A.

E/005.cs

```
using System.Collections;

namespace Ejemplo {
    class Program {
        public static void Main() {
            //Declara la lista que almacenará cadenas
            ArrayList ListaAnimales = new();

            //Adiciona elementos a la lista
            ListaAnimales.Add("Ballena");
            ListaAnimales.Add("Tortuga marina");
            ListaAnimales.Add("Tiburón");
            ListaAnimales.Add("Estrella de mar");
            ListaAnimales.Add("Hipocampo");
            ListaAnimales.Add("Serpiente marina");
            ListaAnimales.Add("Delfín");
            ListaAnimales.Add("Pulpo");
            ListaAnimales.Add("Caballito de mar");
            ListaAnimales.Add("Coral");
            ListaAnimales.Add("Pingüinos");

            //Imprime valores
            Console.WriteLine("Lista original");
            for (int Cont = 0; Cont < ListaAnimales.Count; Cont++)
                Console.Write(ListaAnimales[Cont] + ";");
            Console.WriteLine(" ");

            //Genera nueva lista
            int posIni = 5;
            int cantidad = 3;
            ArrayList nuevaL = ListaAnimales.GetRange(posIni, cantidad);

            //Imprime valores de esa nueva lista
            Console.WriteLine("\r\nNueva lista");
            for (int Cont = 0; Cont < nuevaL.Count; Cont++)
                Console.Write(nuevaL[Cont] + ";");
            Console.WriteLine(" ");

            //Modifica un valor de la nueva lista
            nuevaL[0] = "CACATÚA";
```

```

        //Imprime la lista nueva con el valor alterado
        Console.WriteLine("\r\nNueva lista, primer valor:");
        for (int Cont = 0; Cont < nuevaL.Count; Cont++)
            Console.Write(nuevaL[Cont] + ";");
        Console.WriteLine(" ");

        //Imprime de nuevo la lista original
        Console.WriteLine("\r\nLista original");
        for (int Cont = 0; Cont < ListaAnimales.Count; Cont++)
            Console.Write(ListaAnimales[Cont] + ";");
        Console.WriteLine(" ");
    }
}
}

```

Lista original

;Ballena;Tortuga marina;Tiburón;Estrella de mar;Hipocampo;
Serpiente marina;Delfín;Pulpo;Caballito de mar;Coral;Pingüinos

Nueva lista

;Serpiente marina;Delfín;Pulpo

Nueva lista, primer valor:

;CACATÚA;Delfín;Pulpo

Lista original

;Ballena;Tortuga marina;Tiburón;Estrella de mar;Hipocampo;
CACATÚA;Delfín;Pulpo;Caballito de mar;Coral;Pingüinos

Ilustración 5: ArrayList, referenciar con una variable, un rango de la lista

```
using System.Collections;

namespace Ejemplo {
    class Program {
        public static void Main() {
            //Declara la lista que almacenará cadenas
            ArrayList ListaAnimales = new();

            //Adiciona elementos a la lista
            ListaAnimales.Add("Ballena");
            ListaAnimales.Add("Tortuga marina");
            ListaAnimales.Add("Tiburón");
            ListaAnimales.Add("Estrella de mar");
            ListaAnimales.Add("Hipocampo");
            ListaAnimales.Add("Serpiente marina");
            ListaAnimales.Add("Delfín");
            ListaAnimales.Add("Pulpo");
            ListaAnimales.Add("Caballito de mar");
            ListaAnimales.Add("Coral");
            ListaAnimales.Add("Pingüinos");

            //Recorrido con foreach
            Console.WriteLine("Recorrido con foreach");
            foreach (Object objeto in ListaAnimales)
                Console.Write(objeto + ";");
            Console.WriteLine("\r\n");

            //Recorrido con for
            Console.WriteLine("Recorrido con for");
            for (int cont = 0; cont < ListaAnimales.Count; cont++)
                Console.Write(ListaAnimales[cont] + ";");
            Console.WriteLine("\r\n");

            //Recorrido con un IEnumerator
            Console.WriteLine("Recorrido con un IEnumerator");
            IEnumerator elemento = ListaAnimales.GetEnumerator();
            while (elemento.MoveNext())
                Console.Write(elemento.Current + ";");
        }
    }
}
```

```
Recorrido con foreach
;Ballena;Tortuga marina;Tiburón;Estrella de mar;Hipocampo;Ser
piente marina;Delfín;Pulpo;Caballito de mar;Coral;Pingüinos
Recorrido con for
;Ballena;Tortuga marina;Tiburón;Estrella de mar;Hipocampo;Ser
piente marina;Delfín;Pulpo;Caballito de mar;Coral;Pingüinos
Recorrido con un IEnumerator
;Ballena;Tortuga marina;Tiburón;Estrella de mar;Hipocampo;Ser
piente marina;Delfín;Pulpo;Caballito de mar;Coral;Pingüinos
```

Ilustración 6: ArrayList, tres formas de recorrerlo

Borrar completamente un ArrayList

Se utiliza el método Clear()

E/007.cs

```
using System.Collections;

namespace Ejemplo {
    class Program {
        public static void Main() {
            //Declara la lista que almacenará cadenas
            ArrayList ListaAnimales = new();

            //Adiciona elementos a la lista
            ListaAnimales.Add("Ballena");
            ListaAnimales.Add("Tortuga marina");
            ListaAnimales.Add("Tiburón");
            ListaAnimales.Add("Estrella de mar");
            ListaAnimales.Add("Hipocampo");
            ListaAnimales.Add("Serpiente marina");
            ListaAnimales.Add("Delfín");
            ListaAnimales.Add("Pulpo");
            ListaAnimales.Add("Caballito de mar");
            ListaAnimales.Add("Coral");
            ListaAnimales.Add("Pingüinos");

            //Limpia el ArrayList
            Console.WriteLine("(Antes) Elementos: " + ListaAnimales.Count);
            ListaAnimales.Clear();
            Console.WriteLine("(Después) Elementos: " + ListaAnimales.Count);
        }
    }
}
```

(Antes) Elementos: 11
(Después) Elementos: 0

Ilustración 7: Borrar completamente un ArrayList

Borrar un rango en un ArrayList

Se utiliza el método RemoveRange()

E/008.cs

```
using System.Collections;

namespace Ejemplo {
    class Program {
        public static void Main() {
            //Declara la lista que almacenará cadenas
            ArrayList ListaAnimales = new();

            //Adiciona elementos a la lista
            ListaAnimales.Add("Ballena");
            ListaAnimales.Add("Tortuga marina");
            ListaAnimales.Add("Tiburón");
            ListaAnimales.Add("Estrella de mar");
            ListaAnimales.Add("Hipocampo");
            ListaAnimales.Add("Serpiente marina");
            ListaAnimales.Add("Delfín");
            ListaAnimales.Add("Pulpo");
            ListaAnimales.Add("Caballito de mar");
            ListaAnimales.Add("Coral");
            ListaAnimales.Add("Pingüinos");

            //Elimina un rango de elementos del ArrayList
            Console.WriteLine("Antes");
            for (int Cont = 0; Cont < ListaAnimales.Count; Cont++)
                Console.Write(ListaAnimales[Cont] + ";");
            Console.WriteLine(" ");

            int posicion = 1;
            int cantidad = 4;
            ListaAnimales.RemoveRange(posicion, cantidad);

            Console.WriteLine("Después");
            //Imprime valores
            for (int Cont = 0; Cont < ListaAnimales.Count; Cont++)
                Console.Write(ListaAnimales[Cont] + ";");
            Console.WriteLine(" ");
        }
    }
}
```

Antes

;Ballena;Tortuga marina;Tiburón;Estrella de mar;Hipocampo;Serpiente marina;Delfín;Pulpo;Caballito de mar;Coral;Pingüinos

Después

;Ballena;Serpiente marina;Delfín;Pulpo;Caballito de mar;Coral;Pingüinos

Ilustración 8: Borrar un rango en un ArrayList

Guardar el ArrayList en un arreglo estático

Todos los elementos del ArrayList pasan a un arreglo estático, sea de tipo object() o de algún tipo de dato como string.

E/009.cs

```
using System.Collections;

namespace Ejemplo {
    class Program {
        public static void Main() {
            //Declara la lista que almacenará cadenas
            ArrayList ListaAnimales = new();

            //Adiciona elementos a la lista
            ListaAnimales.Add("Ballena");
            ListaAnimales.Add("Tortuga marina");
            ListaAnimales.Add("Tiburón");
            ListaAnimales.Add("Estrella de mar");
            ListaAnimales.Add("Hipocampo");
            ListaAnimales.Add("Serpiente marina");
            ListaAnimales.Add("Delfín");
            ListaAnimales.Add("Pulpo");
            ListaAnimales.Add("Caballito de mar");
            ListaAnimales.Add("Coral");
            ListaAnimales.Add("Pingüinos");

            //Guarda el ArrayList en un arreglo estático
            //de tipo objeto
            Console.WriteLine("Arreglo estático de tipo objeto");
            object[] arregloEstatico = ListaAnimales.ToArray();
            for (int Cont = 0; Cont < ListaAnimales.Count; Cont++)
                Console.Write(ListaAnimales[Cont] + ";");
            Console.WriteLine(" ");

            //Guarda el ArrayList en un arreglo estático de tipo string
            Console.WriteLine("Arreglo estático de tipo cadena");
            string[] Cadenas = ListaAnimales.ToArray(typeof(string)) as
string[];
            for (int cont = 0; cont < Cadenas.Length; cont++)
                Console.Write(Cadenas[cont] + ";");
        }
    }
}
```



```
Arreglo estático de tipo objeto
;Ballena;Tortuga marina;Tiburón;Estrella de mar;Hipocampo;Serpiente marina;Delfín;Pulpo;Caballito de mar;Coral;Pingüinos
Arreglo estático de tipo cadena
Ballena;Tortuga marina;Tiburón;Estrella de mar;Hipocampo;Serpiente marina;Delfín;Pulpo;Caballito de mar;Coral;Pingüinos;
```

Ilustración 9: Guardar el ArrayList en un arreglo estático

Agregar un arreglo estático a un ArrayList

Toma el contenido del arreglo estático y lo copia en el ArrayList usando el método AddRange()

E/010.cs

```
using System.Collections;

namespace Ejemplo {
    class Program {
        public static void Main() {
            //Declara la lista que almacenará cadenas
            ArrayList ListaAnimales = new();

            //Adiciona elementos a la lista
            ListaAnimales.Add("Ballena");
            ListaAnimales.Add("Tortuga marina");
            ListaAnimales.Add("Tiburón");
            ListaAnimales.Add("Estrella de mar");
            ListaAnimales.Add("Hipocampo");
            ListaAnimales.Add("Serpiente marina");
            ListaAnimales.Add("Delfín");
            ListaAnimales.Add("Pulpo");
            ListaAnimales.Add("Caballito de mar");
            ListaAnimales.Add("Coral");
            ListaAnimales.Add("Pingüinos");

            //Un arreglo estático
            string[] Cadenas = { "Gato", "Conejo", "Liebre" };

            //Adiciona el arreglo estático al ArrayList
            ListaAnimales.AddRange(Cadenas);

            //Imprime el ArrayList
            for (int cont = 0; cont < ListaAnimales.Count; cont++)
                Console.WriteLine(ListaAnimales[cont] + "; ");
        }
    }
}
```

Ballena; Tortuga marina; Tiburón; Estrella de mar; Hipocampo; Serpiente marina; Delfín; Pulpo; Caballito de mar; Coral; Pingüinos; Gato; Conejo; Liebre;

Ilustración 10: Agregar un arreglo estático a un ArrayList

Inserta un arreglo estático en una determinada posición del ArrayList

Toma el contenido del arreglo estático, lo copia y lo inserta en alguna posición del ArrayList usando el método `InsertRange()`

E/011.cs

```
using System.Collections;

namespace Ejemplo {
    class Program {
        public static void Main() {
            //Declara la lista que almacenará cadenas
            ArrayList ListaAnimales = new();

            //Adiciona elementos a la lista
            ListaAnimales.Add("Ballena");
            ListaAnimales.Add("Tortuga marina");
            ListaAnimales.Add("Tiburón");
            ListaAnimales.Add("Estrella de mar");
            ListaAnimales.Add("Hipocampo");
            ListaAnimales.Add("Serpiente marina");
            ListaAnimales.Add("Delfín");
            ListaAnimales.Add("Pulpo");
            ListaAnimales.Add("Caballito de mar");
            ListaAnimales.Add("Coral");
            ListaAnimales.Add("Pingüinos");

            //Un arreglo estático
            string[] Cadenas = { "Gato", "Conejo", "Liebre" };

            //Inserta el arreglo estático en una determinada
            //posición del ArrayList
            int posicionInserta = 4;
            ListaAnimales.InsertRange(posicionInserta, Cadenas);

            //Imprime el ArrayList
            for (int cont = 0; cont < ListaAnimales.Count; cont++)
                Console.Write(ListaAnimales[cont] + "; ");
        }
    }
}
```

Ballena; Tortuga marina; Tiburón; Estrella de mar; Gato; Conejo; Liebre; Hipocampo; Serpiente marina; Delfín; Pulpo; Caballito de mar; Coral; Pingüinos;

Ilustración 11: Inserta un arreglo estático en una determinada posición del ArrayList

Insertar varios ArrayList dentro de un ArrayList

El contenido de varios ArrayList es copiado dentro de otro ArrayList. Como es una copia, no importa si se modifica el ArrayList fuente.

E/012.cs

```
using System.Collections;

namespace Ejemplo {
    class Program {
        public static void Main() {
            //Declara tres ArrayList
            ArrayList ListaA = new();
            ArrayList ListaB = new();
            ArrayList ListaC = new();

            //Adiciona elementos a esos ArrayList
            ListaA.Add("A");
            ListaA.Add("B");
            ListaA.Add("C");
            ListaB.Add("7");
            ListaB.Add("8");
            ListaB.Add("9");
            ListaC.Add("qw");
            ListaC.Add("er");
            ListaC.Add("ty");

            //Inserta los dos primeros ArrayList en el tercero
            int posicionInserta = 1;
            ListaC.InsertRange(posicionInserta, ListaA);

            posicionInserta = 5;
            ListaC.InsertRange(posicionInserta, ListaB);

            //Imprime el ArrayList
            for (int cont = 0; cont < ListaC.Count; cont++)
                Console.Write(ListaC[cont] + "; ");
            Console.WriteLine("\r\n");

            //Modifica ListaA y se chequea si eso afectó a ListaC
            ListaA[0] = "CACATÚA";
            for (int cont = 0; cont < ListaC.Count; cont++)
                Console.Write(ListaC[cont] + "; ");

        }
    }
}
```

```
qw; A; B; C; er; 7; 8; 9; ty;  
qw; A; B; C; er; 7; 8; 9; ty;
```

Ilustración 12: Insertar varios ArrayList dentro de un ArrayList

Invertir un ArrayList

Le da la vuelta al ArrayList

E/013.cs

```
using System.Collections;

namespace Ejemplo {
    class Program {
        public static void Main() {
            //Declara la lista que almacenará cadenas
            ArrayList Listado = new();

            //Adiciona elementos a la lista
            Listado.Add("AB");
            Listado.Add("CD");
            Listado.Add("EF");
            Listado.Add("GH");
            Listado.Add("IJ");
            Listado.Add("KL");
            Listado.Add("MN");
            Listado.Add("OP");

            //Imprime el ArrayList
            for (int cont = 0; cont < Listado.Count; cont++)
                Console.Write(Listado[cont] + "; ");
            Console.WriteLine("\r\n");

            //Aplica Reverse()
            Listado.Reverse();

            //Imprime de nuevo el ArrayList
            for (int cont = 0; cont < Listado.Count; cont++)
                Console.Write(Listado[cont] + "; ");
        }
    }
}
```

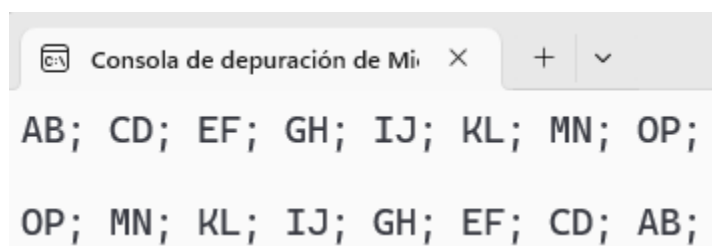


Ilustración 13: Invertir un ArrayList

```
using System.Collections;

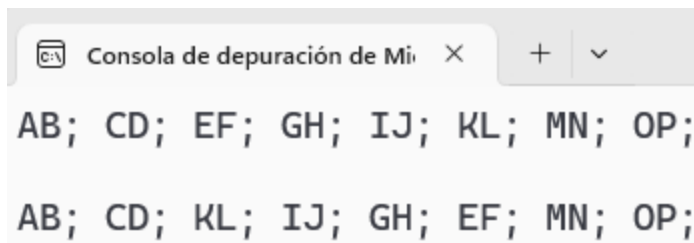
namespace Ejemplo {
    class Program {
        public static void Main() {
            //Declara la lista que almacenará cadenas
            ArrayList Listado = new();

            //Adiciona elementos a la lista
            Listado.Add("AB");
            Listado.Add("CD");
            Listado.Add("EF");
            Listado.Add("GH");
            Listado.Add("IJ");
            Listado.Add("KL");
            Listado.Add("MN");
            Listado.Add("OP");

            //Imprime el ArrayList
            for (int cont = 0; cont < Listado.Count; cont++)
                Console.Write(Listado[cont] + "; ");
            Console.WriteLine("\r\n");

            //Aplica Reverse(posicion, cantidad)
            int posicion = 2;
            int cantidad = 4;
            Listado.Reverse(posicion, cantidad);

            //Imprime de nuevo el ArrayList
            for (int cont = 0; cont < Listado.Count; cont++)
                Console.Write(Listado[cont] + "; ");
        }
    }
}
```



```
AB; CD; EF; GH; IJ; KL; MN; OP;  
AB; CD; KL; IJ; GH; EF; MN; OP;
```

Ilustración 14: Invertir un rango de datos en el ArrayList


```
using System.Collections;

namespace Ejemplo {
    class Program {
        public static void Main() {
            //Declara la lista que almacenará cadenas
            ArrayList Listado = new();

            //Adiciona elementos a la lista
            Listado.Add("GH");
            Listado.Add("MN");
            Listado.Add("AB");
            Listado.Add("OP");
            Listado.Add("IJ");
            Listado.Add("KL");
            Listado.Add("CD");
            Listado.Add("EF");

            //Imprime el ArrayList
            for (int cont = 0; cont < Listado.Count; cont++)
                Console.Write(Listado[cont] + "; ");
            Console.WriteLine("\r\n");

            //Ordena el ArrayList
            Listado.Sort();

            //Imprime de nuevo el ArrayList
            for (int cont = 0; cont < Listado.Count; cont++)
                Console.Write(Listado[cont] + "; ");
        }
    }
}
```

```
GH; MN; AB; OP; IJ; KL; CD; EF;

AB; CD; EF; GH; IJ; KL; MN; OP;
```

Ilustración 15: Ordena un ArrayList

Búsqueda Binaria

Una vez el ArrayList es ordenado, se puede hacer una búsqueda binaria.

E/016.cs

```
using System.Collections;

namespace Ejemplo {
    class Program {
        public static void Main() {
            //Declara la lista que almacenará cadenas
            ArrayList Listado = new();

            //Adiciona elementos a la lista
            Listado.Add("GH");
            Listado.Add("MN");
            Listado.Add("AB");
            Listado.Add("KL");
            Listado.Add("OP");
            Listado.Add("IJ");
            Listado.Add("CD");
            Listado.Add("EF");

            //Imprime el ArrayList
            Console.WriteLine("ArrayList Original");
            for (int cont = 0; cont < Listado.Count; cont++)
                Console.Write(Listado[cont] + "; ");
            Console.WriteLine("\r\n");

            //Ordena el ArrayList
            Listado.Sort();

            //Imprime de nuevo el ArrayList
            Console.WriteLine("ArrayList Ordenado");
            for (int cont = 0; cont < Listado.Count; cont++)
                Console.Write(Listado[cont] + "; ");
            Console.WriteLine("\r\n");

            //Busca en forma binaria en el ArrayList
            string Buscar = "KL";
            int pos = Listado.BinarySearch(Buscar);
            Console.WriteLine("Buscando: " + Buscar);
            Console.WriteLine("Encontrado en: " + pos);
        }
    }
}
```

```
ArrayList Original  
GH; MN; AB; KL; OP; IJ; CD; EF;  
  
ArrayList Ordenado  
AB; CD; EF; GH; IJ; KL; MN; OP;  
  
Buscando: KL  
Encontrado en: 5
```

Ilustración 16: Búsqueda Binaria

Capacidad del ArrayList

A medida que el ArrayList se le van adicionando elementos, su capacidad va aumentando siempre por encima del número de elementos almacenados.

E/017.cs

```
using System.Collections;

namespace Ejemplo {
    class Program {
        public static void Main() {
            //Declara la lista que almacenará cadenas
            ArrayList Listado = new();

            //Para agregar elementos al azar
            Random azar = new();

            //Va agregando elementos al azar,
            //imprime el tamaño y la capacidad
            for (int veces = 1; veces <= 50; veces++) {
                Console.WriteLine("\r\nIteración: " + veces);
                Console.WriteLine("Tamaño: " + Listado.Count);
                Console.WriteLine("Capacidad: " + Listado.Capacity);
                for (int cont = 1; cont <= 30; cont++) {
                    Listado.Add(azar.NextDouble());
                }
            }
        }
    }
}
```

```
Iteración: 1  
Tamaño: 0  
Capacidad: 0  
  
Iteración: 2  
Tamaño: 30  
Capacidad: 32  
  
Iteración: 3  
Tamaño: 60  
Capacidad: 64  
  
Iteración: 4  
Tamaño: 90  
Capacidad: 128  
  
Iteración: 5  
Tamaño: 120  
Capacidad: 128  
  
Iteración: 6  
Tamaño: 150  
Capacidad: 256
```

Ilustración 17: Capacidad del ArrayList

Detectar el tipo de dato del elemento del ArrayList

Un ArrayList puede almacenar cualquier tipo de dato, luego si no se está seguro del tipo de dato, se hace uso de GetType()

E/018.cs

```
using System.Collections;

namespace Ejemplo {
    class Program {
        public static void Main() {
            //Declara la lista que almacenará cadenas
            ArrayList Listado = new();

            //Agrega diferentes tipos de datos
            Listado.Add("Rafael Alberto Moreno Parra");
            Listado.Add(720626);
            Listado.Add(1.6832929);
            Listado.Add('J');
            Listado.Add(true);

            //Muestra el contenido y el tipo de cada elemento
            for (int cont = 0; cont < Listado.Count; cont++) {
                Console.Write(Listado[cont]);
                Console.WriteLine(" tipo: " + Listado[cont].GetType());
            }
            Console.WriteLine("\r\n");

            //Y compara
            for (int cont = 0; cont < Listado.Count; cont++) {
                Console.Write(Listado[cont]);
                if (Listado[cont].GetType() == typeof(int))
                    Console.WriteLine(" es un entero");

                if (Listado[cont].GetType() == typeof(char))
                    Console.WriteLine(" es un caracter");

                if (Listado[cont].GetType() == typeof(double))
                    Console.WriteLine(" es un real");

                if (Listado[cont].GetType() == typeof(string))
                    Console.WriteLine(" es una cadena");

                if (Listado[cont].GetType() == typeof(bool))
                    Console.WriteLine(" es un booleano");
            }
        }
    }
}
```

```
}  
}
```

```
Rafael Alberto Moreno Parra tipo: System.String  
720626 tipo: System.Int32  
1.6832929 tipo: System.Double  
J tipo: System.Char  
True tipo: System.Boolean
```

```
Rafael Alberto Moreno Parra es una cadena  
720626 es un entero  
1.6832929 es un real  
J es un caracter  
True es un booleano
```

Ilustración 18: Detectar el tipo de dato del elemento del ArrayList

List

List es similar a ArrayList. La diferencia es que List exige un mismo tipo de datos para todos los elementos. Los métodos de List son similares a ArrayList. La ventaja es que List es mucho más rápido que ArrayList.

La sintaxis para definir un List es:

```
List<tipo de dato> Nombre = new List<tipo de dato>();
```

Métodos similares a los de ArrayList

Los métodos que se documentaron anteriormente para ArrayList (excepto el de detectar el tipo de dato) se utilizan en List. Se muestra su uso a continuación:

E/019.cs

```
using System.Collections;

namespace Ejemplo {
    class Program {
        public static void Main() {
            //Declara la lista que almacenará cadenas
            List<string> Lista = [];

            //Adiciona elementos a la lista
            Lista.Add("Ballena");
            Lista.Add("Tortuga");
            Lista.Add("Tiburón");
            Lista.Add("Hipocampo");
            Lista.Add("Delfín");
            Lista.Add("Pulpo");
            Lista.Add("Coral");
            Lista.Add("Pingüinos");
            Lista.Add("Calamar");
            Lista.Add("Gaviota");
            Lista.Add("Foca");
            Lista.Add("Manatíes");
            Lista.Add("Orca");
            Lista.Add("Medusas");
            Lista.Add("Mejillones");
            Lista.Add("Caracoles");

            //Tamaño de la lista
            int tamano = Lista.Count;
            Console.WriteLine("Tamaño: " + tamano);
        }
    }
}
```



```

//Traer un determinado elemento de la lista
int posicion = 7;
string texto = Lista[posicion].ToString();
Console.Write("Posición: " + posicion);
Console.WriteLine(" : " + texto);

//Nos dice si existe un determinado
//elemento en la lista
string buscar = "Foca";
bool Existe = Lista.Contains(buscar);
Console.Write("Busca: " + buscar);
Console.WriteLine(" Resultado: " + Existe);

//Nos dice la posición donde encontró
//el elemento en la lista
int posBusca = Lista.IndexOf(buscar);
Console.Write("Busca: " + buscar);
Console.WriteLine(" Posición: " + posBusca + "\r\n");

//Imprime la lista
Console.WriteLine("Lista Original");
ImprimeLista(Lista);

//Retira elemento de la lista
Console.WriteLine("Retira HipoCampo");
Lista.Remove("Hipocampo");
ImprimeLista(Lista);

//Elimina el objeto de determinada posición.
Console.WriteLine("Retira Elemento posición 5");
Lista.RemoveAt(5);
ImprimeLista(Lista);

//Cambia una cadena en la lista
Console.WriteLine("Modifica elemento posición 3");
Lista[3] = "ORNITORRINCO";
ImprimeLista(Lista);

//Inserta una cadena en la posición 4 de la lista
Console.WriteLine("Inserta elemento posición 4");
Lista.Insert(4, "CACATÚA");
ImprimeLista(Lista);

//Genera nueva lista
int pos = 5;
int cantidad = 3;
List<string> nueva = Lista.GetRange(pos, cantidad);

```

```

Console.WriteLine("Nueva lista");
ImprimeLista(nueva);

//Recorrido con foreach
Console.WriteLine("Recorrido con foreach");
foreach (Object objeto in Lista)
    Console.Write(objeto + ";");
Console.WriteLine("\r\n");

//Recorrido con for
Console.WriteLine("Recorrido con for");
for (int cont = 0; cont < Lista.Count; cont++)
    Console.Write(Lista[cont] + ";");
Console.WriteLine("\r\n");

//Recorrido con un IEnumerator
Console.WriteLine("Recorrido con un IEnumerator");
IEnumerator Iobjeto = Lista.GetEnumerator();
while (Iobjeto.MoveNext())
    Console.Write(Iobjeto.Current + ";");
Console.WriteLine("\r\n");

//Guarda el List en un arreglo
//estático de tipo string
Console.WriteLine("List a arreglo estático");
string[] cadenas = Lista.ToArray();
for (int cont = 0; cont < cadenas.Length; cont++)
    Console.Write(cadenas[cont] + ";");
Console.WriteLine("\r\n");

//Adiciona un arreglo estático al List
Console.WriteLine("Adiciona arreglo estático al List");
string[] Cadenas = { "Gato", "Perro", "Conejo" };
Lista.AddRange(Cadenas);
ImprimeLista(Lista);

//Inserta un arreglo estático al List
Console.WriteLine("Inserta arreglo estático al List");
string[] Aves = { "Azulejo", "Bichofue", "Gavilán" };
int posicionInserta = 4;
Lista.InsertRange(posicionInserta, Aves);
ImprimeLista(Lista);

//Invierte la lista
Console.WriteLine("Invierte la lista");
Lista.Reverse();
ImprimeLista(Lista);

```

```

//Ordena el List
Console.WriteLine("Ordena la lista");
Lista.Sort();
ImprimeLista(Lista);

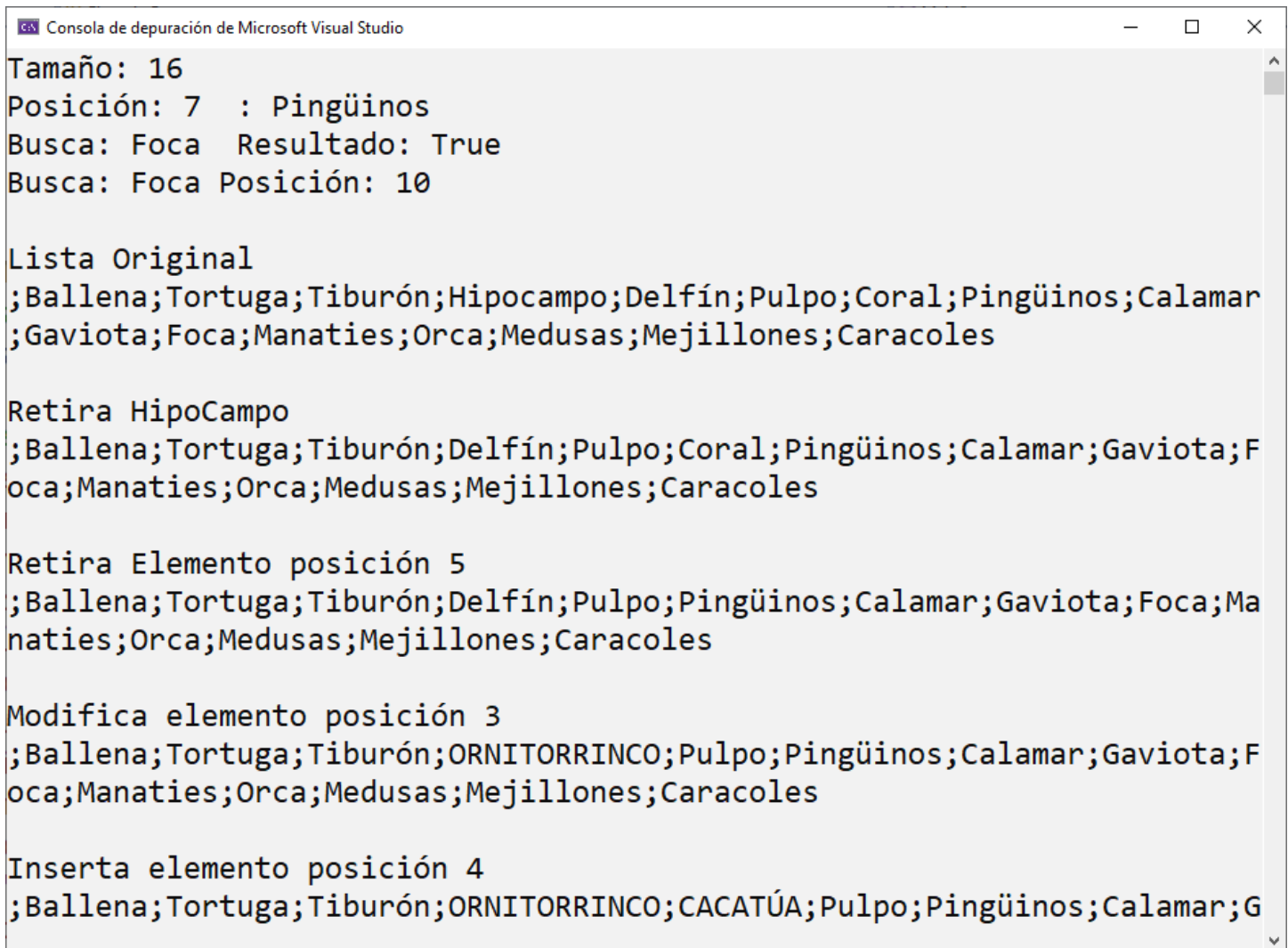
//Busca en forma binaria en el List
Console.WriteLine("Búsqueda binaria en la lista");
string Buscar = "Gato";
int buscado = Lista.BinarySearch(Buscar);
Console.Write("Buscando a: " + Buscar);
Console.WriteLine(" encontrado en: " + buscado + "\r\n");

//Elimina un rango de elementos del List
Console.WriteLine("Elimina un rango de elementos");
int PosBorra = 1;
int CantidadBorra = 4;
Lista.RemoveRange(PosBorra, CantidadBorra);
ImprimeLista(Lista);

//Limpia el List
Console.WriteLine("Borra el List");
Console.WriteLine("(Antes) Elementos: " + Lista.Count);
Lista.Clear();
Console.WriteLine("(Después) Elementos: " + Lista.Count);
}

static void ImprimeLista(List<string> Listado) {
    for (int cont = 0; cont < Listado.Count; cont++)
        Console.Write(Listado[cont] + ";");
    Console.WriteLine("\r\n");
}
}
}

```



```
Consola de depuración de Microsoft Visual Studio

Tamaño: 16
Posición: 7 : Pingüinos
Busca: Foca Resultado: True
Busca: Foca Posición: 10

Lista Original
;Ballena;Tortuga;Tiburón;Hipocampo;Delfín;Pulpo;Coral;Pingüinos;Calamar
;Gaviota;Foca;Manaties;Orca;Medusas;Mejillones;Caracoles

Retira HipoCampo
;Ballena;Tortuga;Tiburón;Delfín;Pulpo;Coral;Pingüinos;Calamar;Gaviota;F
oca;Manaties;Orca;Medusas;Mejillones;Caracoles

Retira Elemento posición 5
;Ballena;Tortuga;Tiburón;Delfín;Pulpo;Pingüinos;Calamar;Gaviota;Foca;Ma
naties;Orca;Medusas;Mejillones;Caracoles

Modifica elemento posición 3
;Ballena;Tortuga;Tiburón;ORNITORRINCO;Pulpo;Pingüinos;Calamar;Gaviota;F
oca;Manaties;Orca;Medusas;Mejillones;Caracoles

Inserta elemento posición 4
;Ballena;Tortuga;Tiburón;ORNITORRINCO;CACATÚA;Pulpo;Pingüinos;Calamar;G
```

Ilustración 19: List, métodos

Métricas: Comparativa de desempeño de ArrayList vs List vs Arreglo estático

¿Cuál estructura tiene mejor desempeño? Para lograr esta comparativa, se utilizó el algoritmo de ordenación de burbuja, el cual hace uso intensivo de operaciones de lectura y cambio de valores en la estructura de memoria (por eso es tan lento ese algoritmo), pero será útil para hacer la comparativa. El programa a continuación:

Ordenando con tipo int

E/020a.cs

```
using System.Collections;
using System.Diagnostics;

namespace Ejemplo {
    class Program {
        static void Main() {
#if DEBUG
            Console.WriteLine("Modo DEBUG detectado. Las pruebas se deben
hacer en RELEASE");
            Environment.Exit(0);
#endif

            /* Prueba de velocidad de los diferentes tipos de estructuras:
            * arreglo estático, ArrayList, List
            * Se usará el método de ordenación de burbuja en el que
            * hace una gran cantidad de lectura y escritura sobre
            * la estructura (por eso es el más lento pero muy bueno para
            * hacer esta comparativa)
            * */

            int minOrden = 500; //Mínimo número de elementos a ordenar
            int maxOrden = 9000; //Máximo número de elementos a ordenar
            int avanceOrden = 500; //Avance de elementos a ordenar

            /* Número de pruebas por ordenamiento
            * Luego el tiempo de ordenar N elementos es el promedio
            * de esas pruebas así se evita que por algún motivo los
            * tiempos tengan picos o valles */
            int numPruebas = 40;

            //Limite es el tamaño de datos que se van a ordenar
            Console.WriteLine(".NET Versión: " + Environment.Version);
            Console.WriteLine("Ordenación. Tiempo promedio en milisegundos");
```

```

        Console.WriteLine("Elementos;Arreglo;ArrayList;List");
        for (int Lim = minOrden; Lim <= maxOrden; Lim += avanceOrden)
            Ordenamiento(Lim, numPruebas);

        Console.WriteLine("\r\nFinal de la prueba");
    }

    static void Ordenamiento(int Limite, int numPruebas) {
        Random azar = new();

        //Las estructuras usadas: arreglo estático, ArrayList, List
        int[] numerosA = new int[Limite];
        int[] numerosB = new int[Limite];
        ArrayList arraylist = [];
        List<int> list = [];

        //Medidor de tiempos
        Stopwatch temporizador = new();

        //Almacena los tiempos de cada método de ordenación
        long TParreglo = 0, TPararraylist = 0, TPlist = 0;

        //Para disminuir picos o valles en el tiempo,
        //se hacen varias pruebas
        for (int prueba = 1; prueba <= numPruebas; prueba++) {

            //Llena con valores al azar el arreglo
            LlenaAzar(numerosA, azar);

            //Ordenación por Burbuja ArrayList
            arraylist.Clear();
            arraylist.AddRange(numerosA);
            temporizador.Reset();
            temporizador.Start();
            BurbujaArrayList(arraylist);
            TPararraylist += temporizador.ElapsedMilliseconds;

            //Ordenación por Burbuja List
            list.Clear();
            list.AddRange(numerosA);
            temporizador.Reset();
            temporizador.Start();
            BurbujaList(list);
            TPlist += temporizador.ElapsedMilliseconds;

            //Ordenación por Burbuja Arreglo estático
            Array.Copy(numerosA, 0, numerosB, 0, numerosA.Length);
            temporizador.Reset();

```

```

    temporizador.Start();
    BurbujaArreglo(numerosB);
    TParreglo += temporizador.ElapsedMilliseconds;

    //Compara las listas ordenadas
    for (int cont = 0; cont < numerosB.Length; cont++) {
        if (numerosB[cont] != list[cont] ||
            list[cont] != Convert.ToInt32(arraylist[cont]))
            Console.WriteLine("Error en la ordenación");
    }
}

double Tarreglo = (double)TParreglo / numPruebas;
double Tarraylist = (double)TParraylist / numPruebas;
double Tlist = (double)Tplist / numPruebas;

Console.Write(Limite + ";" + Tarreglo);
Console.Write ";" + Tarraylist);
Console.WriteLine ";" + Tlist);
}


//Llena el arreglo unidimensional con valores aleatorios
static void LlenaAzar(int[] numerosA, Random azar) {
    for (int cont = 0; cont < numerosA.Length; cont++)
        numerosA[cont] = azar.Next(0, 10000);
}

//Ordenamiento por burbuja usando ArrayList
static void BurbujaArrayList(ArrayList arraylist) {
    int tamano = arraylist.Count;
    object tmp;
    for (int i = 0; i < tamano - 1; i++) {
        for (int j = 0; j < tamano - 1; j++) {
            int iA = Convert.ToInt32(arraylist[j]);
            int iB = Convert.ToInt32(arraylist[j + 1]);
            if (iA > iB) {
                tmp = arraylist[j];
                arraylist[j] = arraylist[j + 1];
                arraylist[j + 1] = tmp;
            }
        }
    }
}

//Ordenamiento por burbuja usando List
static void BurbujaList(List<int> list) {
    int tamano = list.Count;
    int tmp;

```

}



```
.NET Versión: 9.0.2
Ordenación. Tiempo promedio en milisegundos
Elementos;Arreglo;ArrayList;List
500;0;3,1;0,025
1000;1;9,325;1
1500;2;12,4;2,725
2000;3,675;23,2;5,2
2500;6;39,85;8,375
3000;8,375;52,35;12,2
3500;12,05;78,05;17,025
4000;15,875;94,55;22,4
4500;20,675;136,025;29
5000;25,6;149,875;36,05
5500;31,475;182,525;44,3
6000;38,075;217,825;53,25
6500;45,15;256,35;63,3
7000;53;299,5;74,125
7500;61,475;343,5;86
8000;69,775;392,2;99
8500;79,175;443,15;112,3
9000;89,325;497,775;126,5

Final de la prueba

C:\Users\engin\source\repos\Ejemplo\Ejemplo\bin\Release\net9
.0-windows10.0.26100.0\Ejemplo.exe (proceso 4692) se cerró c
on el código 0 (0x0).
Para cerrar automáticamente la consola cuando se detiene la
depuración, habilite Herramientas ->Opciones ->Depuración ->
Cerrar la consola automáticamente al detenerse la depuración
```

Ilustración 20: Ordenando enteros con .NET 9

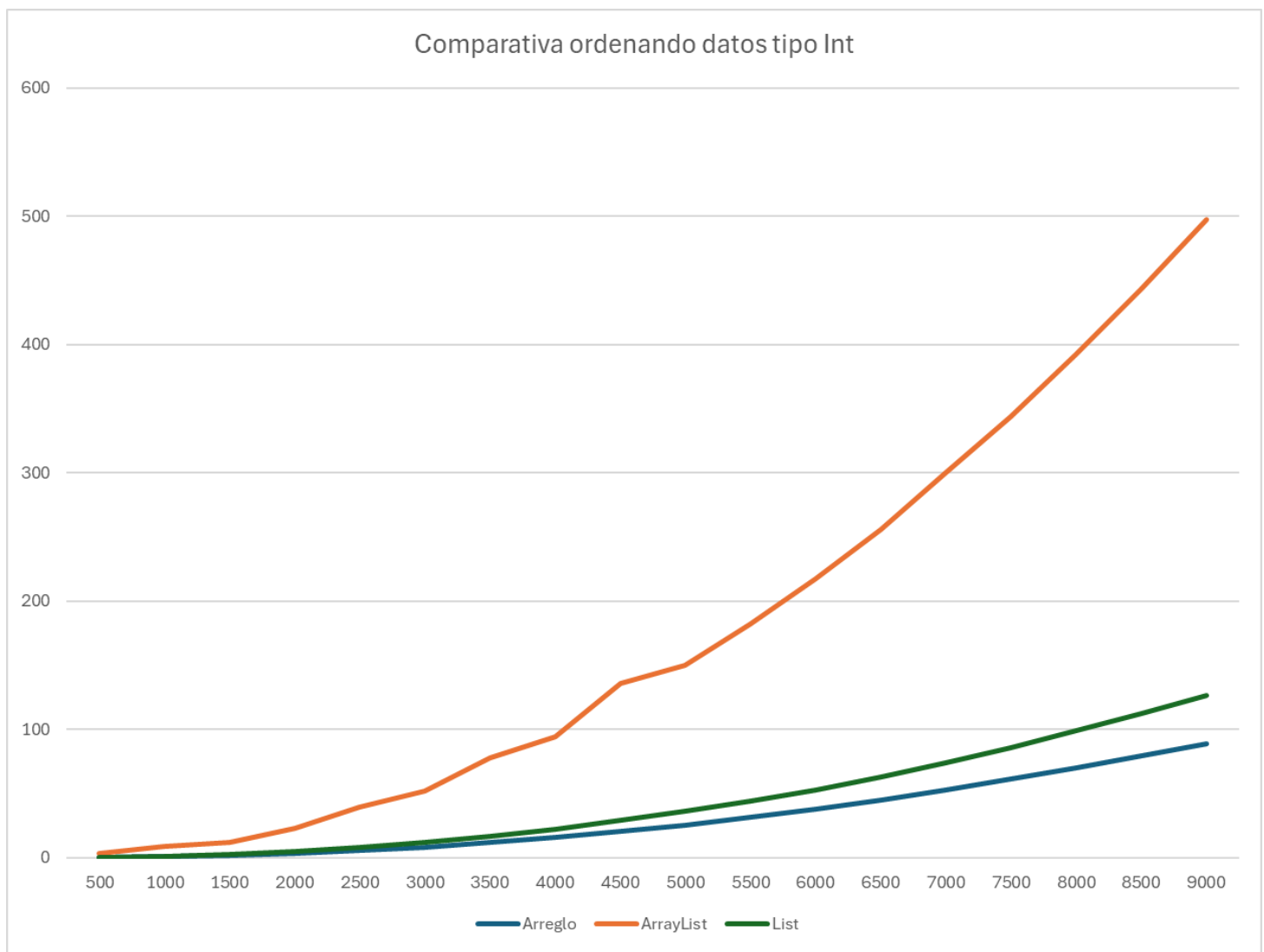
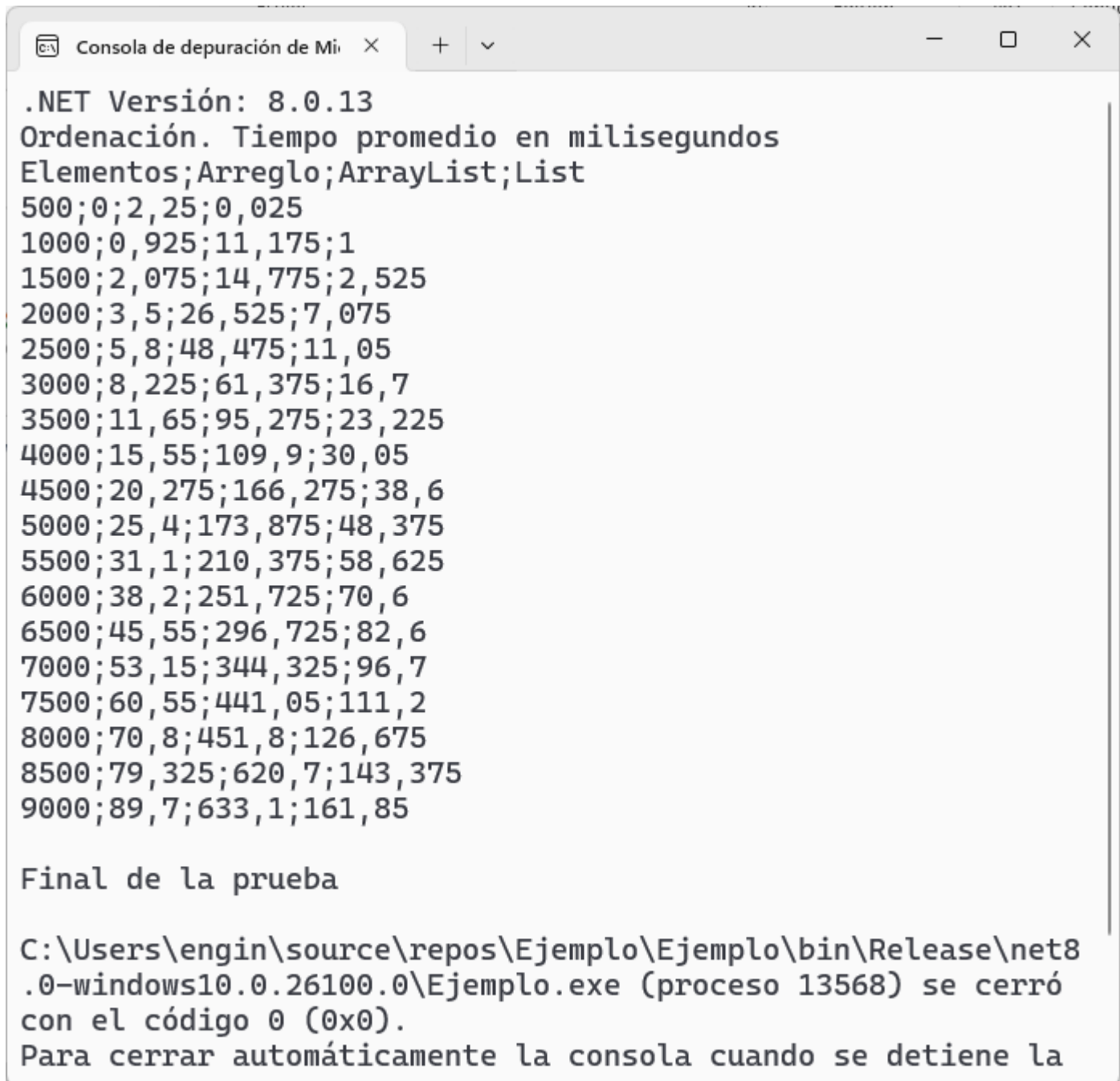


Ilustración 21: Ordenando enteros con .NET 9



```
.NET Versión: 8.0.13
Ordenación. Tiempo promedio en milisegundos
Elementos;Arreglo;ArrayList;List
500;0;2,25;0,025
1000;0,925;11,175;1
1500;2,075;14,775;2,525
2000;3,5;26,525;7,075
2500;5,8;48,475;11,05
3000;8,225;61,375;16,7
3500;11,65;95,275;23,225
4000;15,55;109,9;30,05
4500;20,275;166,275;38,6
5000;25,4;173,875;48,375
5500;31,1;210,375;58,625
6000;38,2;251,725;70,6
6500;45,55;296,725;82,6
7000;53,15;344,325;96,7
7500;60,55;441,05;111,2
8000;70,8;451,8;126,675
8500;79,325;620,7;143,375
9000;89,7;633,1;161,85

Final de la prueba

C:\Users\engin\source\repos\Ejemplo\Ejemplo\bin\Release\net8
.0-windows10.0.26100.0\Ejemplo.exe (proceso 13568) se cerró
con el código 0 (0x0).
Para cerrar automáticamente la consola cuando se detiene la
```

Ilustración 22: Ordenando enteros con .NET 8

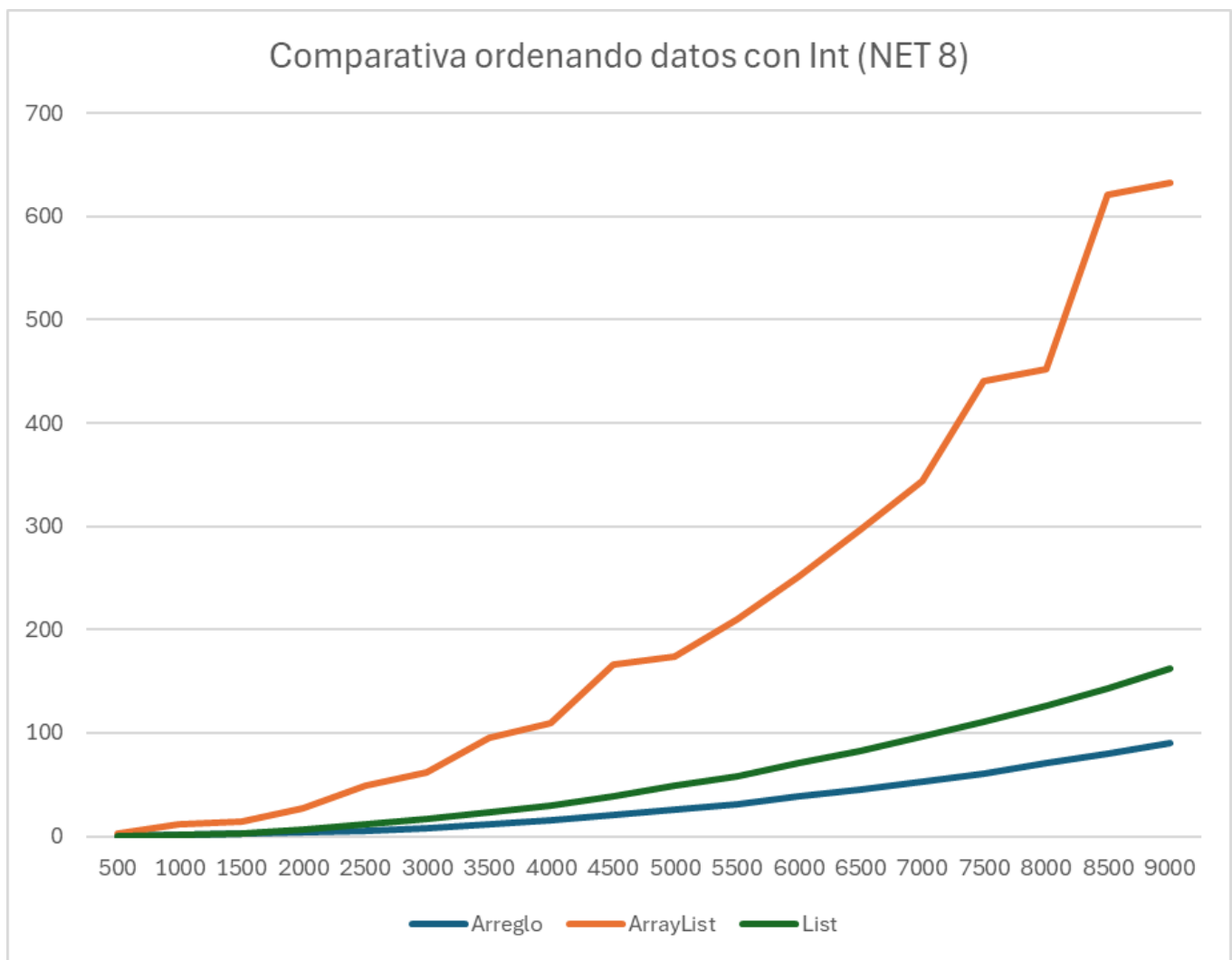


Ilustración 23: Ordenando enteros con .NET 8

.NET 9 vs .NET 8

En las pruebas, .NET 9 mostró la mayor parte de las pruebas ser más rápido que .NET 8. En el gráfico se muestra el tiempo que tomó cada prueba (entre menor, mejor)

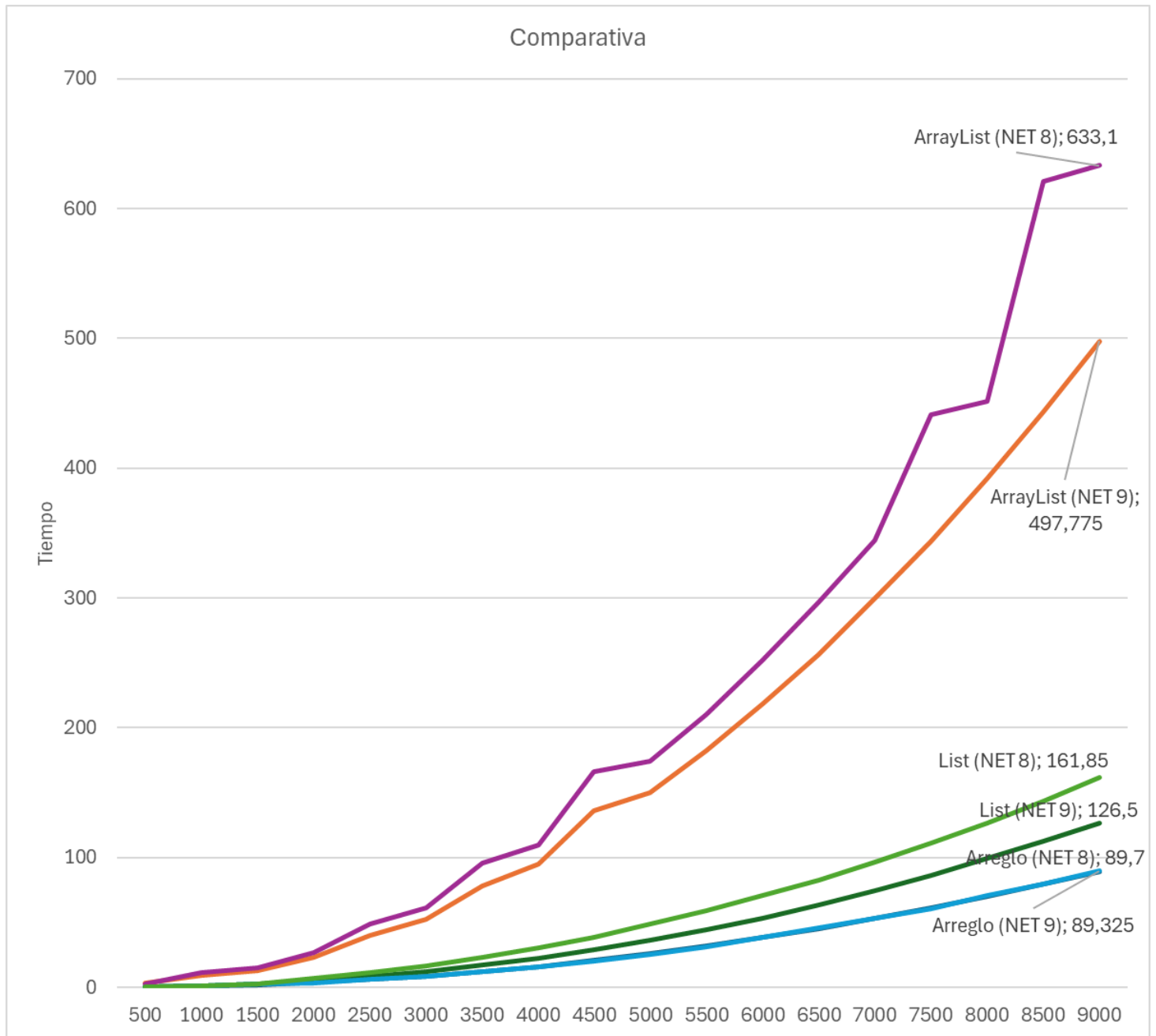


Ilustración 24: Ordenando enteros con .NET 9 y .NET 8

Los arreglos unidimensionales estáticos mostraron mejores tiempos. La estructura dinámica List es ligeramente más lenta y ArrayList es notablemente más lenta.

```
using System.Collections;
using System.Diagnostics;

namespace Ejemplo {
    class Program {
        static void Main() {
#if DEBUG
            Console.WriteLine("Modo DEBUG detectado. Las pruebas se deben
hacer en RELEASE");
            Environment.Exit(0);
#endif

            /* Prueba de velocidad de los diferentes tipos de estructuras:
            * arreglo estático, ArrayList, List
            * Se usará el método de ordenación de burbuja en el que
            * hace una gran cantidad de lectura y escritura sobre
            * la estructura (por eso es el más lento pero muy bueno para
            * hacer esta comparativa)
            * */

            int minOrden = 500; //Mínimo número de elementos a ordenar
            int maxOrden = 9000; //Máximo número de elementos a ordenar
            int avanceOrden = 500; //Avance de elementos a ordenar

            /* Número de pruebas por ordenamiento
            * Luego el tiempo de ordenar N elementos es el promedio
            * de esas pruebas así se evita que por algún motivo los
            * tiempos tengan picos o valles */
            int numPruebas = 40;

            //Limite es el tamaño de datos que se van a ordenar
            Console.WriteLine(".NET Versión: " + Environment.Version);
            Console.WriteLine("Ordenación. Tiempo promedio en milisegundos");
            Console.WriteLine("Elementos;Arreglo;ArrayList;List");
            for (int Lim = minOrden; Lim <= maxOrden; Lim += avanceOrden)
                Ordenamiento(Lim, numPruebas);

            Console.WriteLine("\r\nFinal de la prueba");
        }

        static void Ordenamiento(int Limite, int numPruebas) {
            Random azar = new();

            //Las estructuras usadas: arreglo estático, ArrayList, List
```

```

double[] numerosA = new double[Limite];
double[] numerosB = new double[Limite];
ArrayList arraylist = [];
List<double> list = [];

//Medidor de tiempos
Stopwatch temporizador = new();

//Almacena los tiempos de cada método de ordenación
long TParreglo = 0, TPararraylist = 0, TPlist = 0;

//Para disminuir picos o valles en el tiempo,
//se hacen varias pruebas
for (int prueba = 1; prueba <= numPruebas; prueba++) {

    //Llena con valores al azar el arreglo
    LlenaAzar(numerosA, azar);

    //Ordenación por Burbuja ArrayList
    arraylist.Clear();
    arraylist.AddRange(numerosA);
    temporizador.Reset();
    temporizador.Start();
    BurbujaArrayList(arraylist);
    TPararraylist += temporizador.ElapsedMilliseconds;

    //Ordenación por Burbuja List
    list.Clear();
    list.AddRange(numerosA);
    temporizador.Reset();
    temporizador.Start();
    BurbujaList(list);
    TPlist += temporizador.ElapsedMilliseconds;

    //Ordenación por Burbuja Arreglo estático
    Array.Copy(numerosA, 0, numerosB, 0, numerosA.Length);
    temporizador.Reset();
    temporizador.Start();
    BurbujaArreglo(numerosB);
    TParreglo += temporizador.ElapsedMilliseconds;

    //Compara las listas ordenadas
    for (int cont = 0; cont < numerosB.Length; cont++) {
        if (numerosB[cont] != list[cont] ||
            list[cont] != Convert.ToDouble(arraylist[cont]))
            Console.WriteLine("Error en la ordenación");
    }
}

```

```

double Tarreglo = (double)TParreglo / numPruebas;
double Tarraylist = (double)TParraylist / numPruebas;
double Tlist = (double)TPlist / numPruebas;

Console.Write(Limite + ";" + Tarreglo);
Console.Write(";" + Tarraylist);
Console.WriteLine(";" + Tlist);
}

//Llena el arreglo unidimensional con valores aleatorios
static void LlenaAzar(double[] numerosA, Random azar) {
    for (int cont = 0; cont < numerosA.Length; cont++)
        numerosA[cont] = azar.NextDouble();
}

//Ordenamiento por burbuja usando ArrayList
static void BurbujaArrayList(ArrayList arraylist) {
    int tamano = arraylist.Count;
    object tmp;
    for (int i = 0; i < tamano - 1; i++) {
        for (int j = 0; j < tamano - 1; j++) {
            double dA = Convert.ToDouble(arraylist[j]);
            double dB = Convert.ToDouble(arraylist[j + 1]);
            if (dA > dB) {
                tmp = arraylist[j];
                arraylist[j] = arraylist[j + 1];
                arraylist[j + 1] = tmp;
            }
        }
    }
}

//Ordenamiento por burbuja usando List
static void BurbujaList(List<double> list) {
    int tamano = list.Count;
    double tmp;
    for (int i = 0; i < tamano - 1; i++) {
        for (int j = 0; j < tamano - 1; j++) {
            if (list[j] > list[j + 1]) {
                tmp = list[j];
                list[j] = list[j + 1];
                list[j + 1] = tmp;
            }
        }
    }
}

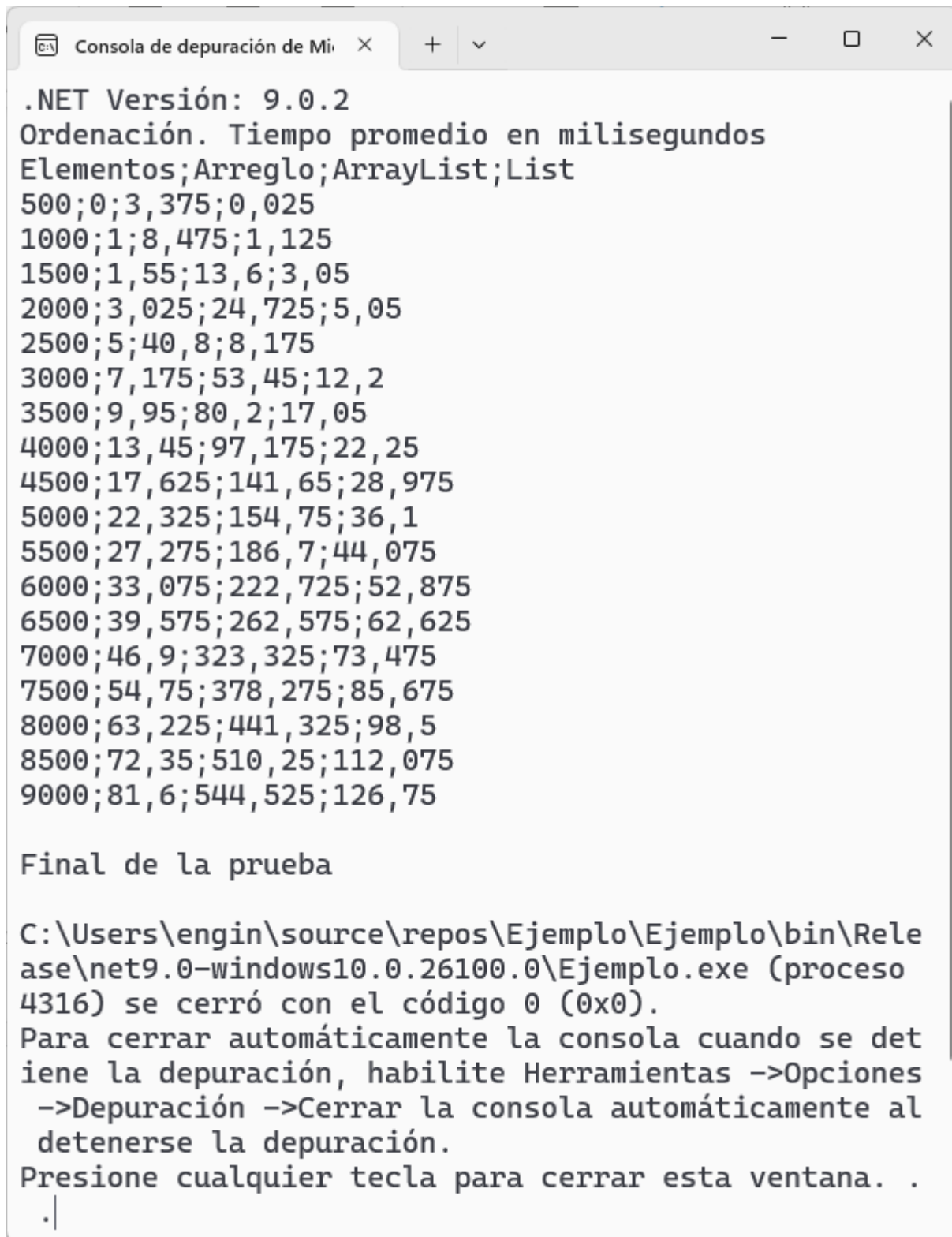
```



```

//Ordenamiento por burbuja usando arreglo unidimensional estático
static void BurbujaArreglo(double[] arregloestatico) {
    int tamano = arregloestatico.Length;
    double tmp;
    for (int i = 0; i < tamano - 1; i++) {
        for (int j = 0; j < tamano - 1; j++) {
            if (arregloestatico[j] > arregloestatico[j + 1]) {
                tmp = arregloestatico[j];
                arregloestatico[j] = arregloestatico[j + 1];
                arregloestatico[j + 1] = tmp;
            }
        }
    }
}

```



```
.NET Versión: 9.0.2
Ordenación. Tiempo promedio en milisegundos
Elementos;Arreglo;ArrayList;List
500;0;3,375;0,025
1000;1;8,475;1,125
1500;1,55;13,6;3,05
2000;3,025;24,725;5,05
2500;5;40,8;8,175
3000;7,175;53,45;12,2
3500;9,95;80,2;17,05
4000;13,45;97,175;22,25
4500;17,625;141,65;28,975
5000;22,325;154,75;36,1
5500;27,275;186,7;44,075
6000;33,075;222,725;52,875
6500;39,575;262,575;62,625
7000;46,9;323,325;73,475
7500;54,75;378,275;85,675
8000;63,225;441,325;98,5
8500;72,35;510,25;112,075
9000;81,6;544,525;126,75

Final de la prueba

C:\Users\engin\source\repos\Ejemplo\Ejemplo\bin\Release\net9.0-windows10.0.26100.0\Ejemplo.exe (proceso 4316) se cerró con el código 0 (0x0).
Para cerrar automáticamente la consola cuando se detiene la depuración, habilite Herramientas ->Opciones ->Depuración ->Cerrar la consola automáticamente al detenerse la depuración.
Presione cualquier tecla para cerrar esta ventana. .
.
```

Ilustración 25: Ordenando tipo double con .NET 9

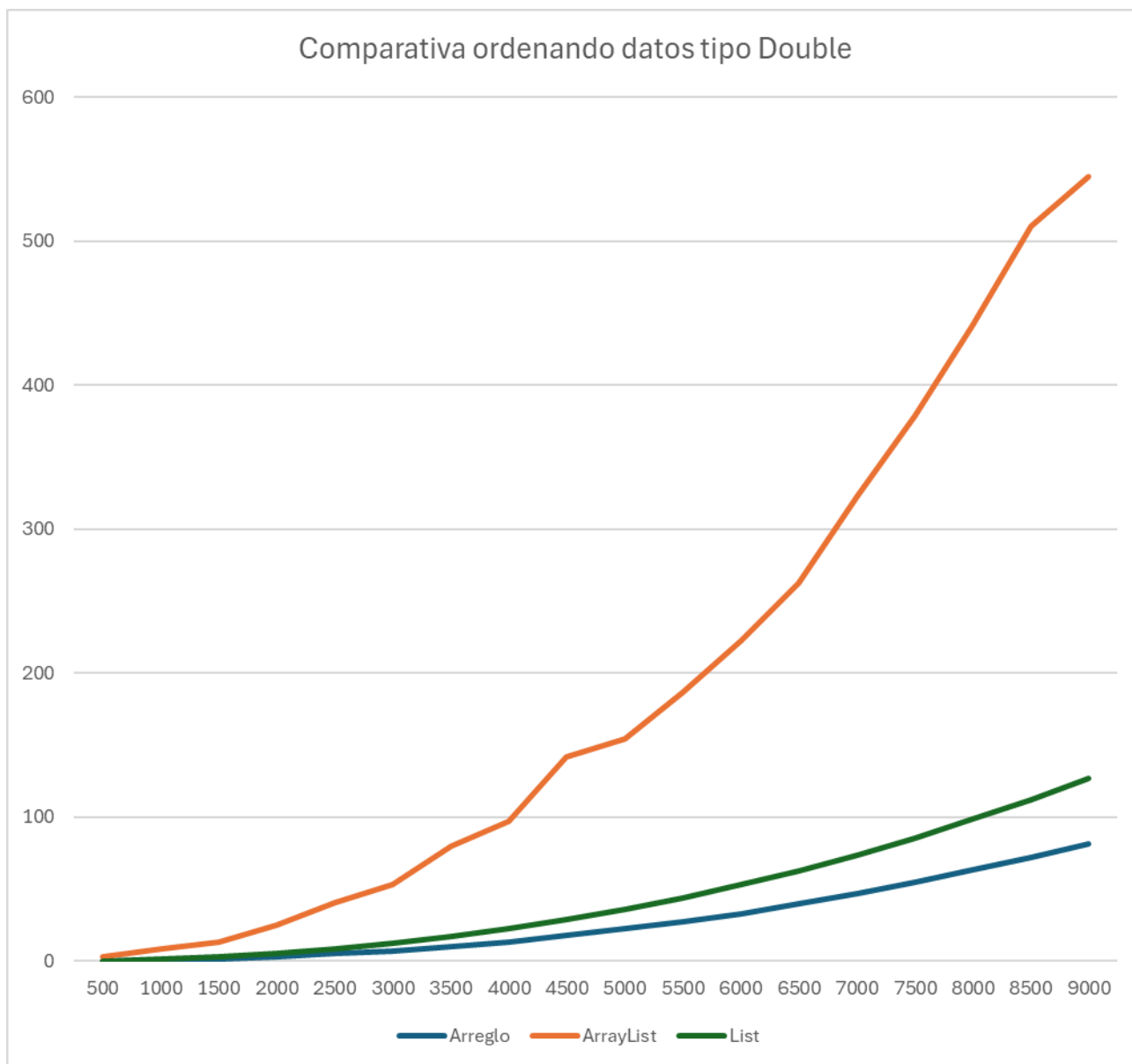
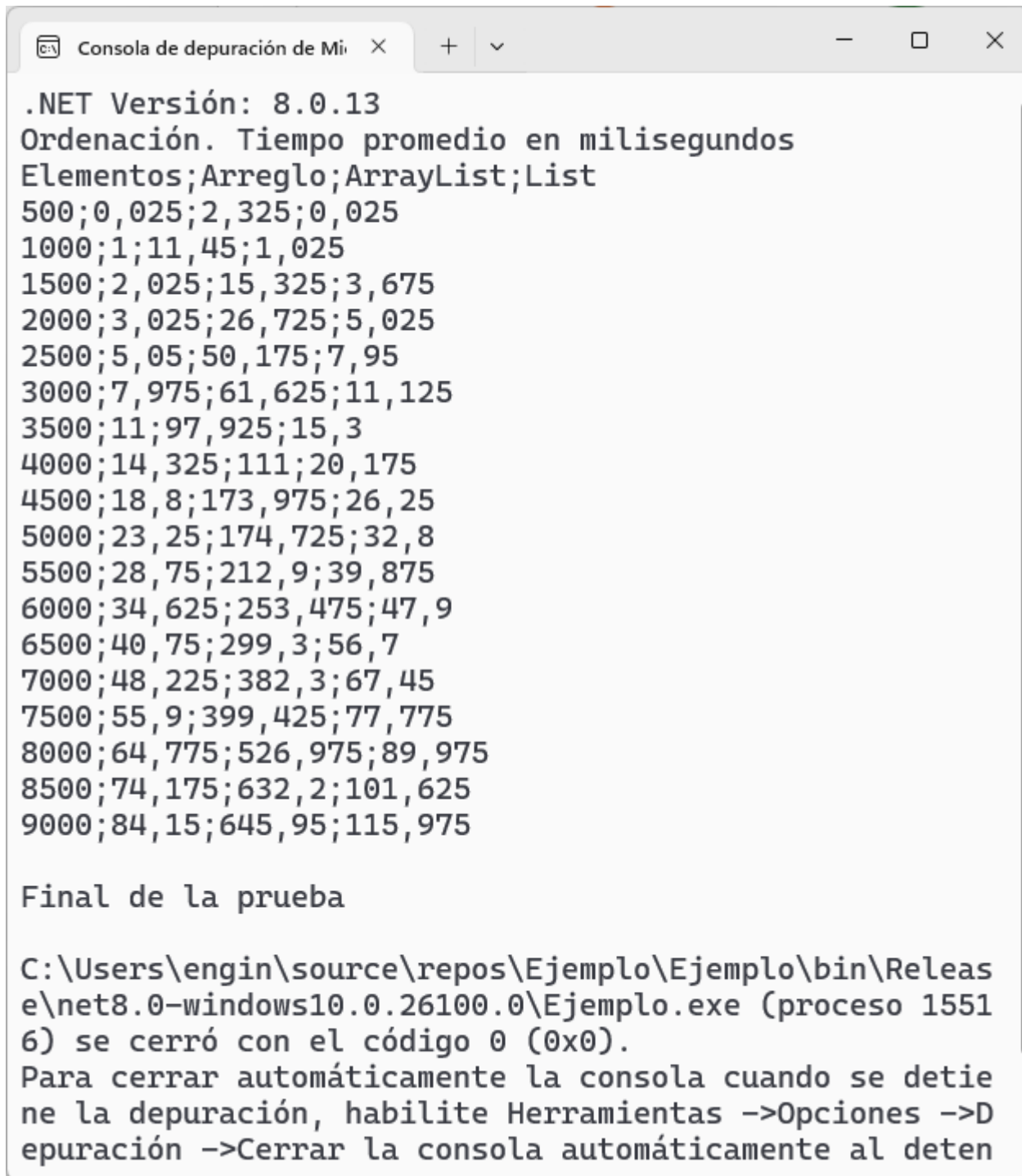


Ilustración 26: Ordenando tipo double con .NET 9



```
.NET Versión: 8.0.13
Ordenación. Tiempo promedio en milisegundos
Elementos;Arreglo;ArrayList;List
500;0,025;2,325;0,025
1000;1;11,45;1,025
1500;2,025;15,325;3,675
2000;3,025;26,725;5,025
2500;5,05;50,175;7,95
3000;7,975;61,625;11,125
3500;11;97,925;15,3
4000;14,325;111;20,175
4500;18,8;173,975;26,25
5000;23,25;174,725;32,8
5500;28,75;212,9;39,875
6000;34,625;253,475;47,9
6500;40,75;299,3;56,7
7000;48,225;382,3;67,45
7500;55,9;399,425;77,775
8000;64,775;526,975;89,975
8500;74,175;632,2;101,625
9000;84,15;645,95;115,975

Final de la prueba

C:\Users\engin\source\repos\Ejemplo\Ejemplo\bin\Release\net8.0-windows10.0.26100.0\Ejemplo.exe (proceso 15516) se cerró con el código 0 (0x0).
Para cerrar automáticamente la consola cuando se detiene la depuración, habilite Herramientas ->Opciones ->Depuración ->Cerrar la consola automáticamente al deten
```

Ilustración 27: Ordenando tipo double con .NET 8

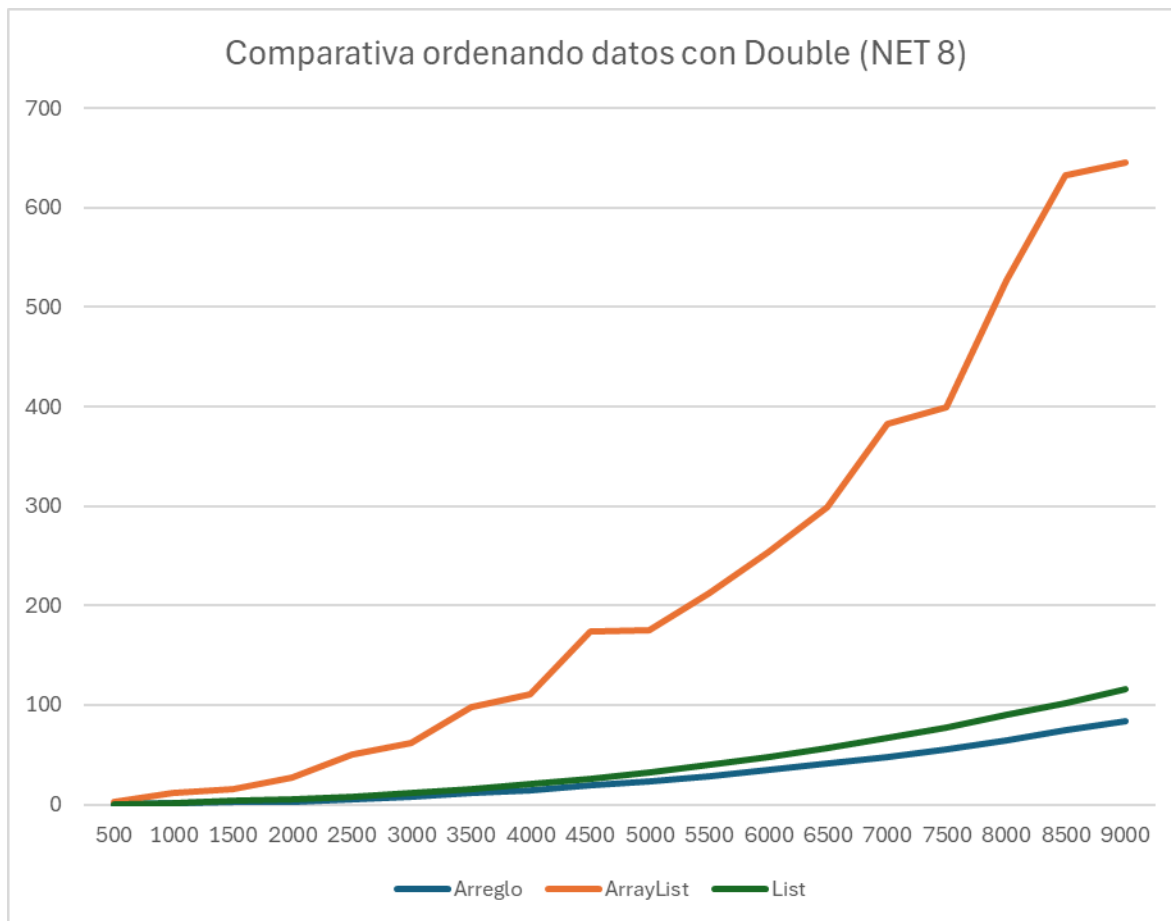


Ilustración 28: Ordenando tipo double con .NET 8

.NET 9 vs .NET 8

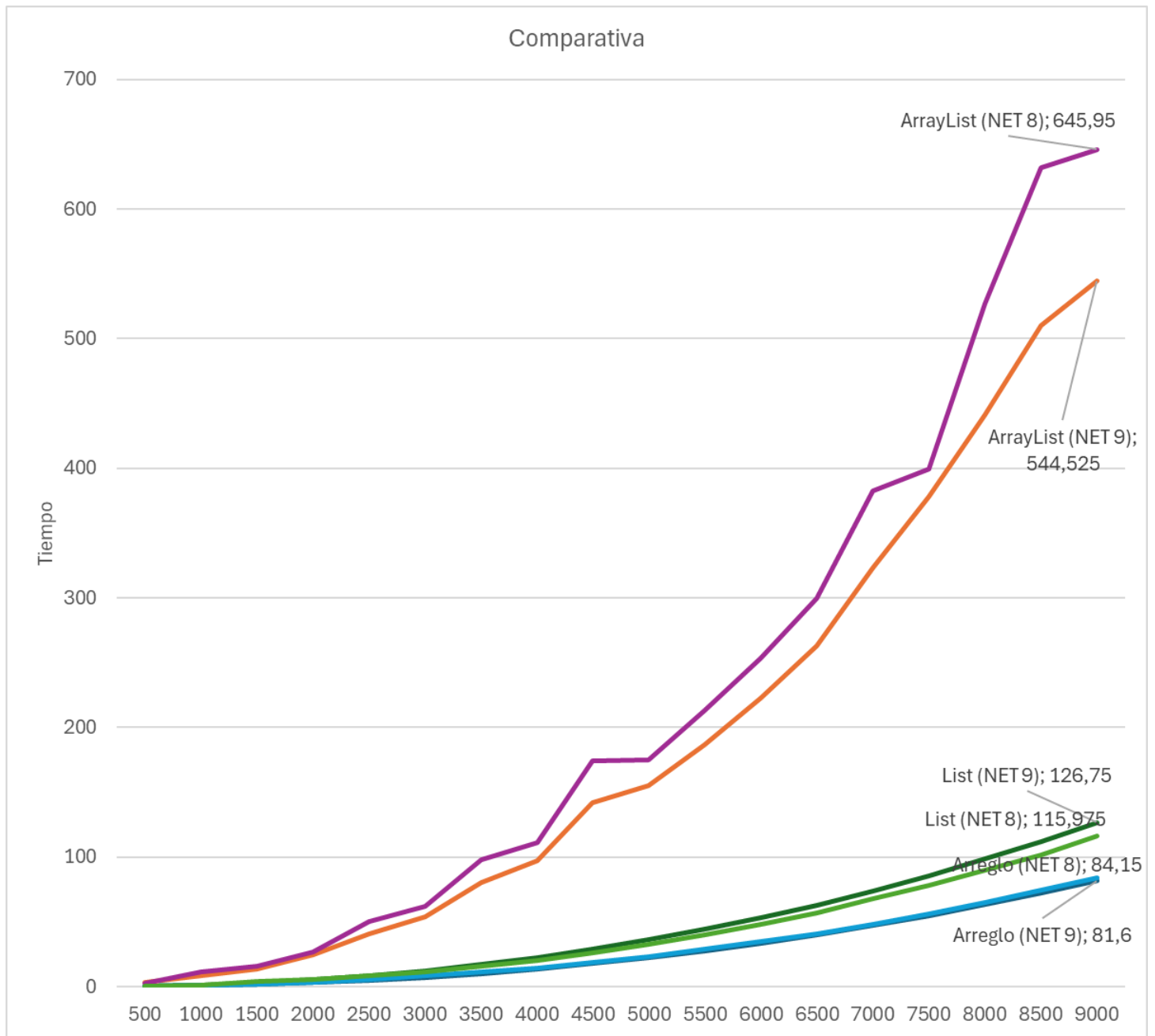


Ilustración 29: Ordenando tipo double con .NET 8 y .NET 9

Para el tipo de datos double, .NET 8 fue ligeramente más rápido que .NET 9 usando la estructura List, en las otras estructuras .NET 9 lo hizo mejor.

```
using System.Collections;
using System.Diagnostics;

namespace Ejemplo {
    class Program {
        static void Main() {
#if DEBUG
            Console.WriteLine("Modo DEBUG detectado. Las pruebas se deben
hacer en RELEASE");
            Environment.Exit(0);
#endif

            /* Prueba de velocidad de los diferentes tipos de estructuras:
            * arreglo estático, ArrayList, List
            * Se usará el método de ordenación de burbuja en el que
            * hace una gran cantidad de lectura y escritura sobre
            * la estructura (por eso es el más lento pero muy bueno para
            * hacer esta comparativa)
            * */

            int minOrden = 500; //Mínimo número de elementos a ordenar
            int maxOrden = 9000; //Máximo número de elementos a ordenar
            int avanceOrden = 500; //Avance de elementos a ordenar

            /* Número de pruebas por ordenamiento
            * Luego el tiempo de ordenar N elementos es el promedio
            * de esas pruebas así se evita que por algún motivo los
            * tiempos tengan picos o valles */
            int numPruebas = 40;

            //Limite es el tamaño de datos que se van a ordenar
            Console.WriteLine(".NET Versión: " + Environment.Version);
            Console.WriteLine("Ordenación. Tiempo promedio en milisegundos");
            Console.WriteLine("Elementos;Arreglo;ArrayList;List");
            for (int Lim = minOrden; Lim <= maxOrden; Lim += avanceOrden)
                Ordenamiento(Lim, numPruebas);

            Console.WriteLine("\r\nFinal de la prueba");
        }

        static void Ordenamiento(int Limite, int numPruebas) {
            Random azar = new();

            //Las estructuras usadas: arreglo estático, ArrayList, List
```

```

char[] numerosA = new char[Limite];
char[] numerosB = new char[Limite];
ArrayList arraylist = [];
List<char> list = [];

//Medidor de tiempos
Stopwatch temporizador = new();

//Almacena los tiempos de cada método de ordenación
long TParreglo = 0, TPararraylist = 0, TPlist = 0;

//Para disminuir picos o valles en el tiempo,
//se hacen varias pruebas
for (int prueba = 1; prueba <= numPruebas; prueba++) {

    //Llena con valores al azar el arreglo
    LlenaAzar(numerosA, azar);

    //Ordenación por Burbuja ArrayList
    arraylist.Clear();
    arraylist.AddRange(numerosA);
    temporizador.Reset();
    temporizador.Start();
    BurbujaArrayList(arraylist);
    TPararraylist += temporizador.ElapsedMilliseconds;

    //Ordenación por Burbuja List
    list.Clear();
    list.AddRange(numerosA);
    temporizador.Reset();
    temporizador.Start();
    BurbujaList(list);
    TPlist += temporizador.ElapsedMilliseconds;

    //Ordenación por Burbuja Arreglo estático
    Array.Copy(numerosA, 0, numerosB, 0, numerosA.Length);
    temporizador.Reset();
    temporizador.Start();
    BurbujaArreglo(numerosB);
    TParreglo += temporizador.ElapsedMilliseconds;

    //Compara las listas ordenadas
    for (int cont = 0; cont < numerosB.Length; cont++) {
        if (numerosB[cont] != list[cont] ||
            list[cont] != Convert.ToChar(arraylist[cont]))
            Console.WriteLine("Error en la ordenación");
    }
}

```



```

double Tarreglo = (double)TParreglo / numPruebas;
double Tarraylist = (double)TParraylist / numPruebas;
double Tlist = (double)TPlist / numPruebas;

Console.Write(Limite + ";" + Tarreglo);
Console.Write(";" + Tarraylist);
Console.WriteLine(";" + Tlist);
}

//Llena el arreglo unidimensional con valores aleatorios
static void LlenaAzar(char[] numerosA, Random azar) {
    string Permitido = "abcdefghijklmnñopqrstuvwxyz";
    for (int cont = 0; cont < numerosA.Length; cont++)
        numerosA[cont] = Permitido[azar.Next(Permitido.Length)];
}

//Ordenamiento por burbuja usando ArrayList
static void BurbujaArrayList(ArrayList arraylist) {
    int tamano = arraylist.Count;
    object tmp;
    for (int i = 0; i < tamano - 1; i++) {
        for (int j = 0; j < tamano - 1; j++) {
            char cA = Convert.ToChar(arraylist[j]);
            char cB = Convert.ToChar(arraylist[j + 1]);
            if (cA > cB) {
                tmp = arraylist[j];
                arraylist[j] = arraylist[j + 1];
                arraylist[j + 1] = tmp;
            }
        }
    }
}

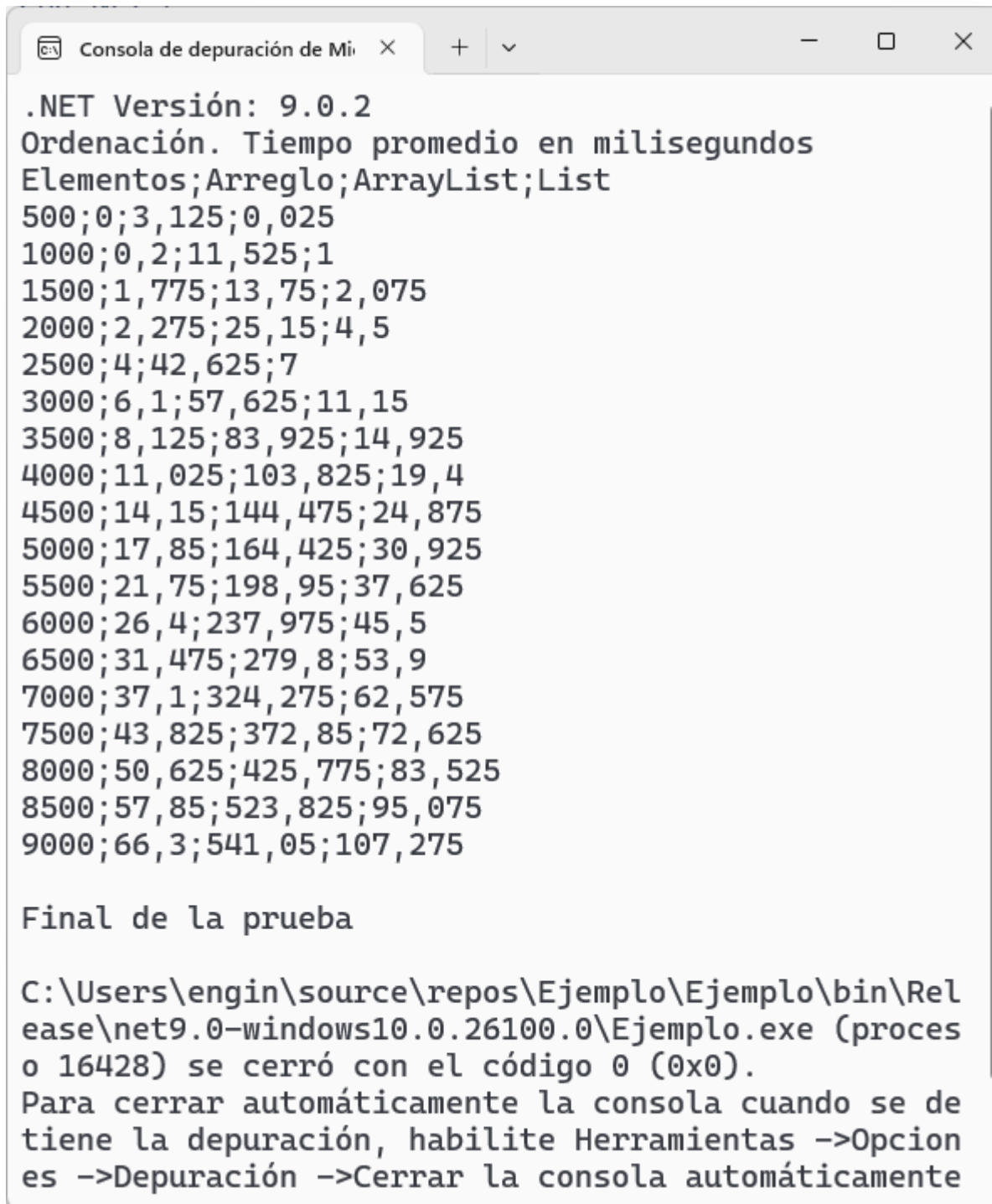
//Ordenamiento por burbuja usando List
static void BurbujaList(List<char> list) {
    int tamano = list.Count;
    char tmp;
    for (int i = 0; i < tamano - 1; i++) {
        for (int j = 0; j < tamano - 1; j++) {
            if (list[j] > list[j + 1]) {
                tmp = list[j];
                list[j] = list[j + 1];
                list[j + 1] = tmp;
            }
        }
    }
}

```

```

//Ordenamiento por burbuja usando arreglo unidimensional estático
static void BurbujaArreglo(char[] arregloestatico) {
    int tamano = arregloestatico.Length;
    char tmp;
    for (int i = 0; i < tamano - 1; i++) {
        for (int j = 0; j < tamano - 1; j++) {
            if (arregloestatico[j] > arregloestatico[j + 1]) {
                tmp = arregloestatico[j];
                arregloestatico[j] = arregloestatico[j + 1];
                arregloestatico[j + 1] = tmp;
            }
        }
    }
}

```



```
.NET Versión: 9.0.2
Ordenación. Tiempo promedio en milisegundos
Elementos;Arreglo;ArrayList;List
500;0;3,125;0,025
1000;0,2;11,525;1
1500;1,775;13,75;2,075
2000;2,275;25,15;4,5
2500;4;42,625;7
3000;6,1;57,625;11,15
3500;8,125;83,925;14,925
4000;11,025;103,825;19,4
4500;14,15;144,475;24,875
5000;17,85;164,425;30,925
5500;21,75;198,95;37,625
6000;26,4;237,975;45,5
6500;31,475;279,8;53,9
7000;37,1;324,275;62,575
7500;43,825;372,85;72,625
8000;50,625;425,775;83,525
8500;57,85;523,825;95,075
9000;66,3;541,05;107,275

Final de la prueba

C:\Users\engin\source\repos\Ejemplo\Ejemplo\bin\Release\net9.0-windows10.0.26100.0\Ejemplo.exe (proceso 16428) se cerró con el código 0 (0x0).
Para cerrar automáticamente la consola cuando se detiene la depuración, habilite Herramientas ->Opciones ->Depuración ->Cerrar la consola automáticamente
```

Ilustración 30: Ordenando tipo char con .NET 9

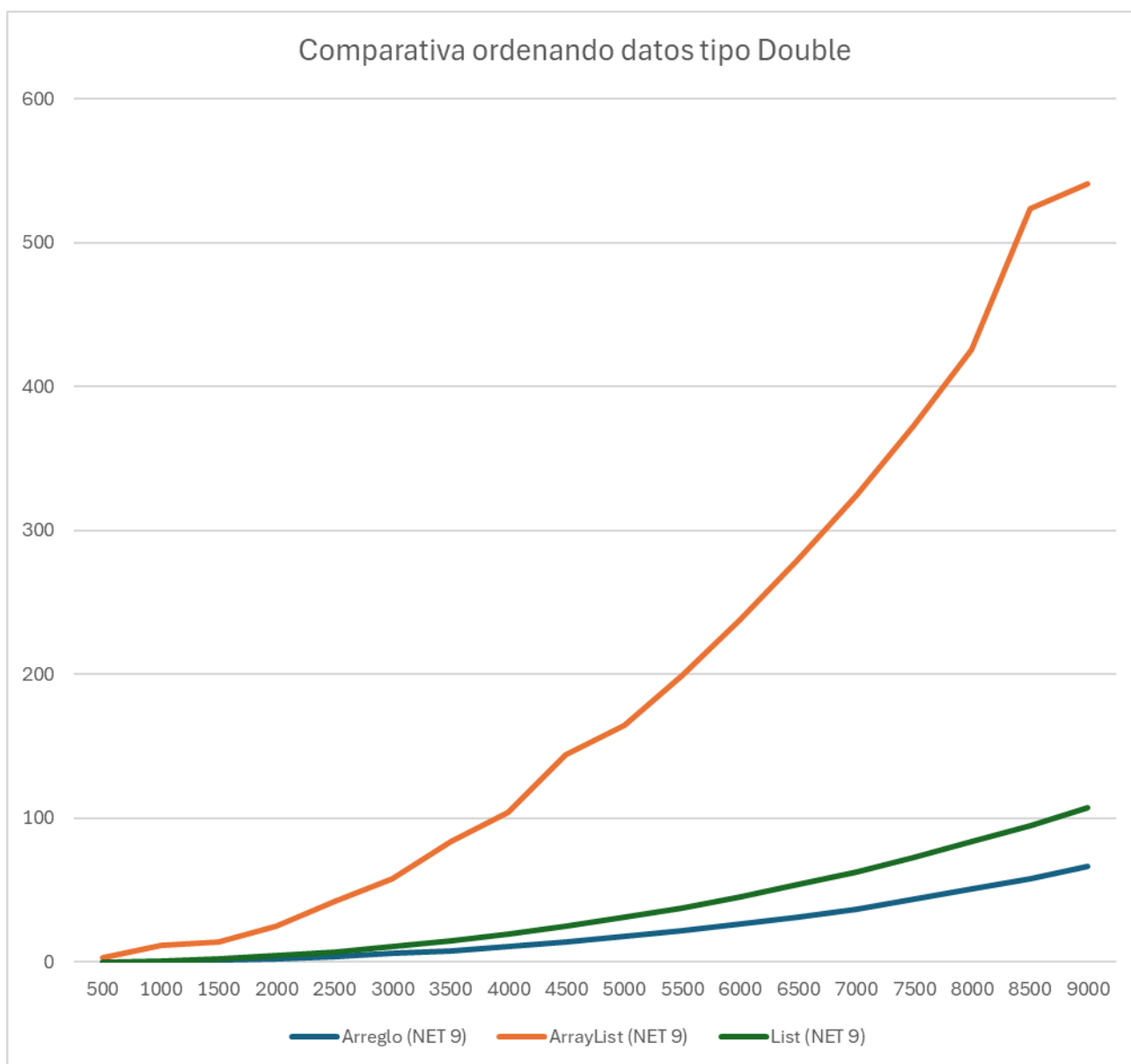
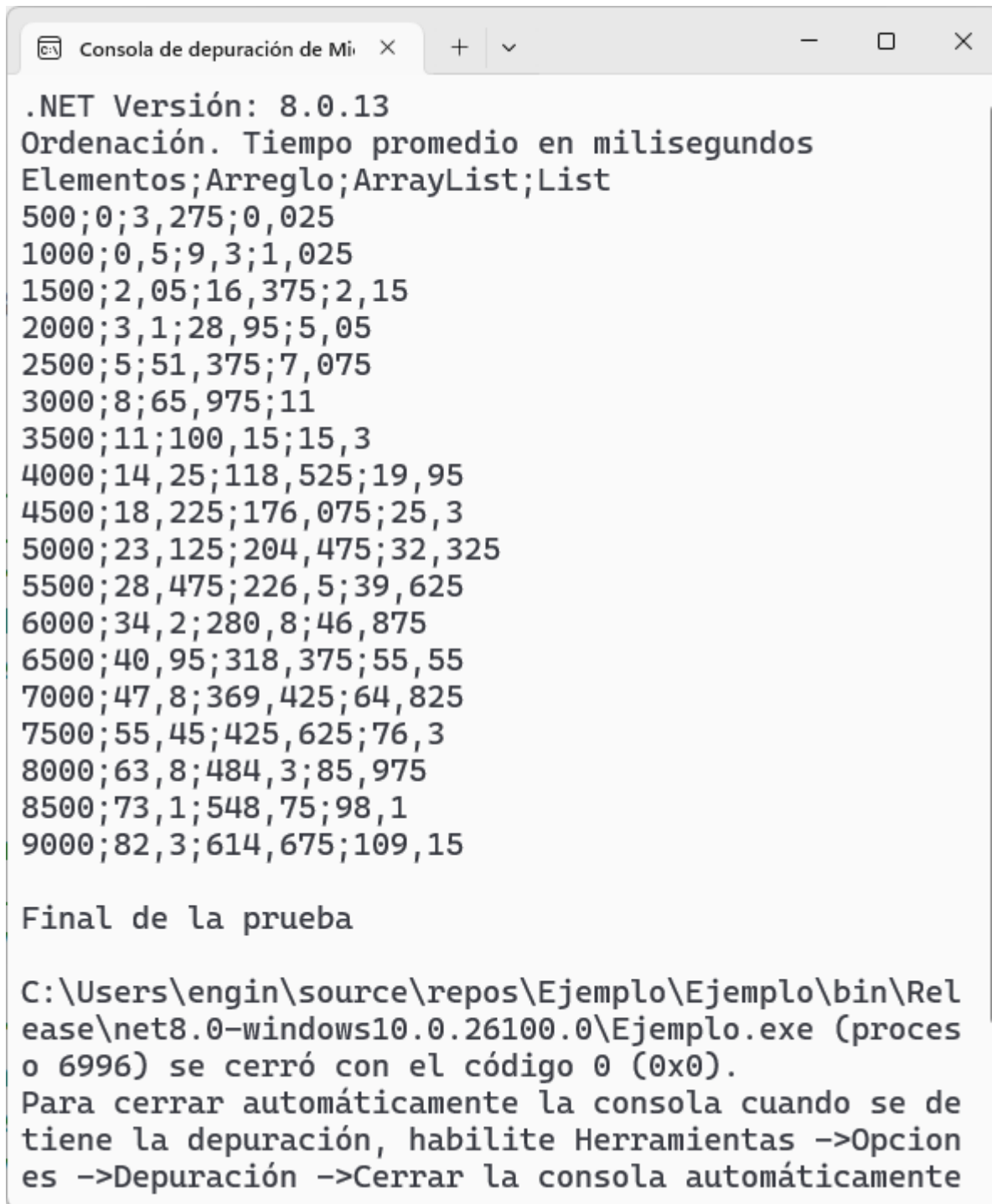


Ilustración 31: Ordenando tipo char con .NET 9



```
.NET Versión: 8.0.13
Ordenación. Tiempo promedio en milisegundos
Elementos;Arreglo;ArrayList;List
500;0;3,275;0,025
1000;0,5;9,3;1,025
1500;2,05;16,375;2,15
2000;3,1;28,95;5,05
2500;5;51,375;7,075
3000;8;65,975;11
3500;11;100,15;15,3
4000;14,25;118,525;19,95
4500;18,225;176,075;25,3
5000;23,125;204,475;32,325
5500;28,475;226,5;39,625
6000;34,2;280,8;46,875
6500;40,95;318,375;55,55
7000;47,8;369,425;64,825
7500;55,45;425,625;76,3
8000;63,8;484,3;85,975
8500;73,1;548,75;98,1
9000;82,3;614,675;109,15

Final de la prueba

C:\Users\engin\source\repos\Ejemplo\Ejemplo\bin\Release\net8.0-windows10.0.26100.0\Ejemplo.exe (proceso 6996) se cerró con el código 0 (0x0).
Para cerrar automáticamente la consola cuando se detiene la depuración, habilite Herramientas -> Opciones -> Depuración -> Cerrar la consola automáticamente
```

Ilustración 32: Ordenando tipo char con .NET 8

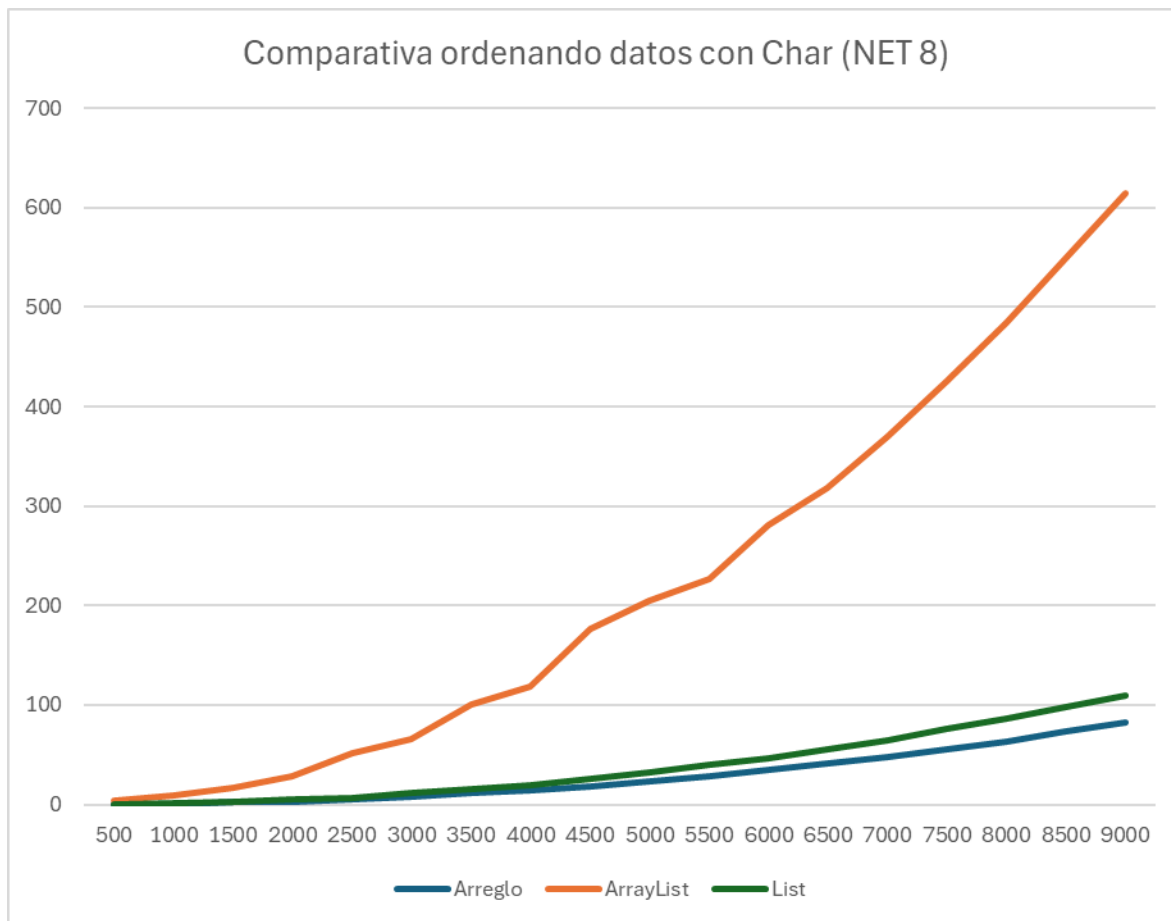


Ilustración 33: Ordenando tipo char con .NET 8

.NET 9 vs .NET 8

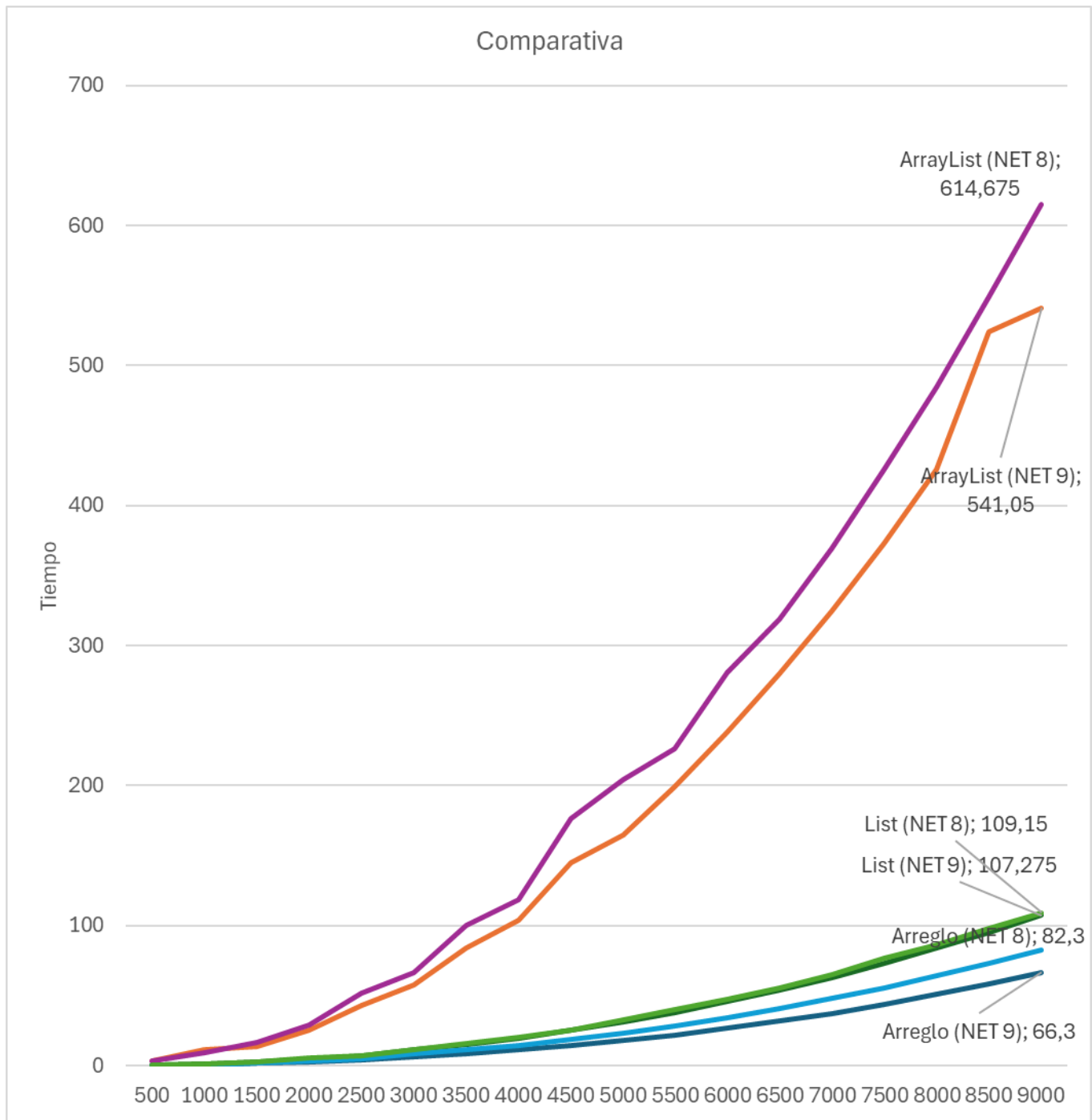


Ilustración 34: Ordenando tipo char con .NET 9 y .NET 8

.NET 9 en las tres estructuras fue más rápido que .NET 8 manejando el tipo de datos char

Lista de objetos

Un frecuente uso de List es para tener un listado de objetos del mismo tipo, no solo tipos de datos nativo. Se requiere entonces dos clases, en una está definido el tipo de objeto a guardar y en la otra se crean, adicionan, actualizan y borran del List

E/021.cs

```
namespace Ejemplo {
    class MiClase {
        //Atributos variados
        public int Entero { get; set; }
        public double Num { get; set; }
        public char Car { get; set; }
        public string Cad { get; set; }

        //Constructor
        public MiClase(int Entero, double Num, char Car, string Cad) {
            this.Entero = Entero;
            this.Num = Num;
            this.Car = Car;
            this.Cad = Cad;
        }

        //Imprime los valores
        public void Imprime() {
            Console.WriteLine("\r\nEntero: " + Entero);
            Console.WriteLine("Real: " + Num);
            Console.WriteLine("Caracter: " + Car);
            Console.WriteLine("Cadena: [" + Cad + "]");
        }
    }

    class Program {
        static void Main() {
            List<MiClase> Listado = [];

            //Adiciona objetos a la lista
            Listado.Add(new MiClase(16, 83.29, 'R', "Rui señor"));
            Listado.Add(new MiClase(29, 89.7, 'A', "Águila"));
            Listado.Add(new MiClase(2, 80.19, 'M', "Manatí"));
            Listado.Add(new MiClase(95, 7.21, 'P', "Puma"));

            //Llama al método de imprimir del objeto
            for (int cont = 0; cont < Listado.Count; cont++)
                Listado[cont].Imprime();
        }
    }
}
```



```
//Inserta un objeto
Listado.Insert(1, new MiClase(88, 3.33, 'Z', "QQQQQ"));

//Elimina un objeto
Listado.RemoveAt(3);

//Llama al método de imprimir del objeto
Console.WriteLine("\r\nDespués de modificar");
for (int cont = 0; cont < Listado.Count; cont++)
    Listado[cont].Imprime();

Console.WriteLine("\r\nFinal");
}
}
}
```

Entero: 16
Real: 83,29
Caracter: R
Cadena: [Ruiñeñor]

Entero: 29
Real: 89,7
Caracter: A
Cadena: [Águila]

Entero: 2
Real: 80,19
Caracter: M
Cadena: [Manatí]

Entero: 95
Real: 7,21
Caracter: P
Cadena: [Puma]

Después de modificar

Entero: 16
Real: 83,29
Caracter: R
Cadena: [Ruiñeñor]

Entero: 88
Real: 3,33
Caracter: Z
Cadena: [QQQQQQQQ]

Entero: 29
Real: 89,7
Caracter: A
Cadena: [Águila]

Ilustración 35: Lista de objetos

Listas en Listas

Un objeto de una lista, a su vez tiene listas. Un ejemplo con series de TV, personajes y actores:

1. La serie tiene un nombre y se puede ver información sobre esta en IMDB.
2. Los personajes son interpretados por actores y actrices.
3. No es nada extraño que los actores participen en diversas series interpretando algún personaje en alguna serie, sólo es ver su ficha en IMDB.

E/022.cs

```
namespace Ejemplo {

    //Datos del actor o actriz
    class ActorActriz {
        public string Nombre { get; set; }
        public string URLIMDB { get; set; }

        //Constructor
        public ActorActriz(string Nombre, string URLIMDB) {
            this.Nombre = Nombre;
            this.URLIMDB = "https://www.imdb.com/name/" + URLIMDB;
        }
    }

    //Datos de la serie de televisión
    class Serie {
        public string Nombre { get; set; }
        public string URLIMDB { get; set; }

        //Listado de actores y actrices que actúan en la serie
        public List<ActorActriz> Actor = [];

        //Constructor
        public Serie(string Nombre, string URLIMDB) {
            this.Nombre = Nombre;
            this.URLIMDB = "https://www.imdb.com/title/" + URLIMDB;
        }
    }

    //La parte que simula la capa de persistencia
    class Persistencia {
        List<ActorActriz> Actores;
        List<Serie> Series;

        //Carga datos de prueba
        public Persistencia() {
            Actores = [];
        }
    }
}
```

```

Series = [];

//Un listado de actores y actrices
Actores.Add(new ActorActriz("Ana María Orozco", "nm0650450"));
Actores.Add(new ActorActriz("Laura Londoño", "nm2256810"));
Actores.Add(new ActorActriz("Carolina Ramírez", "nm1329835"));
Actores.Add(new ActorActriz("Catherine Siachoque", "nm0796171"));
Actores.Add(new ActorActriz("Carmenza González", "nm1863990"));
Actores.Add(new ActorActriz("Andrés Londoño", "nm2150265"));

//Un listado de series
Series.Add(new Serie("Yo soy Betty, la fea", "tt0233127"));
Series.Add(new Serie("La reina del flow", "tt8560918"));
Series.Add(new Serie("Café con Aroma de Mujer", "tt14471346"));
Series.Add(new Serie("Los Briceño", "tt10348478"));
Series.Add(new Serie("Distrito Salvaje", "tt8105958"));
Series.Add(new Serie("Mil Colmillos", "tt9701670"));
Series.Add(new Serie("Perdida", "tt10064124"));

//Añado actores y actrices a la tercera serie
Series[2].Actor.Add(Actores[1]);

//Observe que un mismo actor o actriz puede
//estar en dos series distintas
Series[0].Actor.Add(Actores[0]);
Series[6].Actor.Add(Actores[0]);
}

//Trae datos de la serie
public string SerieNombre(int pos) { return Series[pos].Nombre; }
public string SerieIMDB(int pos) { return Series[pos].URLIMDB; }

//Trae datos del actor
public string ActorNombre(int pos) { return Actores[pos].Nombre; }
public string ActorURL(int pos) { return Actores[pos].URLIMDB; }

//Total de registros
public int ActorTotal() { return Actores.Count; }
public int SerieTotal() { return Series.Count; }

//Adicionar actor
public void ActorAdiciona(string Nombre, string URL) {
    Actores.Add(new ActorActriz(Nombre, URL));
}

//Editar actor
public void ActorEdita(int codigo, string Nombre, string URL) {
    Actores[codigo].Nombre = Nombre;

```

```

    Actores[codigo].URLIMDB = URL;
}

//Borrar actor
public void ActorBorra(int codigo) {
    Actores.RemoveAt(codigo);
}

//Retorna una lista de series donde el actor trabaja
public List<string> ActorTrabaja(int codigo) {
    List<string> ListaSeries = [];
    for (int cont = 0; cont < Series.Count; cont++) {
        for (int num = 0; num < Series[cont].Actor.Count; num++) {
            if (Actores[codigo] == Series[cont].Actor[num])
                ListaSeries.Add(Series[cont].Nombre);
        }
    }
    return ListaSeries;
}

//Adicionar serie
public void SerieAdiciona(string Nombre, string URL) {
    Series.Add(new Serie(Nombre, URL));
}

//Editar serie
public void SerieEdita(int codigo, string Nombre, string URL) {
    Series[codigo].Nombre = Nombre;
    Series[codigo].URLIMDB = URL;
}

//Borrar serie
public void SerieBorra(int codigo) {
    Series.RemoveAt(codigo);
}

//Retornar los actores que trabajan en la serie
public List<string> SerieActores(int codigo) {
    List<ActorActriz> actores = Series[codigo].Actor;
    List<string> Nombres = [];
    for (int cont = 0; cont < actores.Count; cont++)
        Nombres.Add(actores[cont].Nombre);
    return Nombres;
}

//Añade un actor a una serie
public void SerieAsocia(int serie, int actor) {
    Series[serie].Actor.Add(Actores[actor]);
}

```

```

    }

    //Quita un actor de una serie
    public void SerieDisocia(int serie, int actor) {
        Series[serie].Actor.RemoveAt(actor);
    }
}

//La parte visual del programa
class Visual {
    public Persistencia Datos;

    //Conecta con la capa de persistencia
    public Visual(Persistencia objDatos) {
        Datos = objDatos;
    }

    //Menú principal
    public void Menu() {
        int opcion;
        do {
            Console.Clear();
            Console.WriteLine("\nSoftware TV Show 1.6 (Marzo de 2025)");
            Console.WriteLine("1. CRUD de actores y actrices");
            Console.WriteLine("2. CRUD de series");
            Console.WriteLine("3. Salir");
            Console.Write("¿Opción? ");
            opcion = Convert.ToInt32(Console.ReadLine());
            switch (opcion) {
                case 1: CRUDactores(); break;
                case 2: CRUDseries(); break;
            }
        } while (opcion != 3);
    }

    //Menú de actores y actrices
    public void CRUDactores() {
        int opcion;
        do {
            Console.Clear();
            Console.WriteLine("\nSoftware TV Show. Actores/Actrices");
            for (int cont = 0; cont < Datos.ActorTotal(); cont++) {
                Console.Write "[" + cont + "] ";
                Console.Write(Datos.ActorNombre(cont));
                Console.WriteLine(" URL: " + Datos.ActorURL(cont));
            }
            Console.WriteLine(" \n1. Adicionar");
            Console.WriteLine("2. Editar");

```

```

        Console.WriteLine("3. Borrar");
        Console.WriteLine("4. ¿En cuáles series trabaja?");
        Console.WriteLine("5. Volver a menú principal");
        Console.Write("¿Opción? ");
        opcion = Convert.ToInt32(Console.ReadLine());
        switch (opcion) {
            case 1: ActorAdiciona(); break;
            case 2: ActorEdita(); break;
            case 3: ActorBorra(); break;
            case 4: ActorTrabaja(); break;
        }
    } while (opcion != 5);
}

//Menú de series de TV
public void CRUDseries() {
    int opcion;
    do {
        Console.Clear();
        Console.WriteLine("\nSoftware TV Show. Series");
        for (int cont = 0; cont < Datos.SerieTotal(); cont++) {
            Console.Write "[" + cont + "] ";
            Console.Write(Datos.SerieNombre(cont));
            Console.WriteLine(" URL: " + Datos.SerieIMDB(cont));
        }
        Console.WriteLine("\n1. Adicionar");
        Console.WriteLine("2. Editar");
        Console.WriteLine("3. Borrar");
        Console.WriteLine("4. Detalles de la serie");
        Console.WriteLine("5. Asociar actor a serie");
        Console.WriteLine("6. Disociar actor a serie");
        Console.WriteLine("7. Volver a menú principal");
        Console.Write("¿Opción? ");
        opcion = Convert.ToInt32(Console.ReadLine());
        switch (opcion) {
            case 1: SerieAdiciona(); break;
            case 2: SerieEdita(); break;
            case 3: SerieBorra(); break;
            case 4: SerieDetalle(); break;
            case 5: SerieAsocia(); break;
            case 6: SerieDisocia(); break;
        }
    } while (opcion != 7);
}

//Pantalla para adicionar actores
public void ActorAdiciona() {
    Console.WriteLine("\tAdicionar actor al listado");
}

```

```

        Console.WriteLine("¿Nombre? ");
        string nombre = Console.ReadLine();
        Console.WriteLine("¿URL de IMDB? ");
        string URL = Console.ReadLine();
        Datos.ActorAdiciona(nombre, URL);
        Console.WriteLine("\nActor adicionado.");
        Console.ReadKey();
    }

    //Pantalla para editar actores
    public void ActorEdita() {
        Console.WriteLine("\tEditar actor");
        Console.WriteLine("¿Cuál? Escriba el número que está entre [ ]: ");
        int codigo = Convert.ToInt32(Console.ReadLine());
        Console.WriteLine("¿Nombre? ");
        string nombre = Console.ReadLine();
        Console.WriteLine("¿URL de IMDB? ");
        string URL = Console.ReadLine();
        Datos.ActorEdita(codigo, nombre, URL);
        Console.WriteLine("\nActor editado.");
        Console.ReadKey();
    }

    //Pantalla para borrar actores
    public void ActorBorra() {
        Console.WriteLine("\tBorrar actor o actriz");
        Console.WriteLine("¿Cuál? Escriba el número que está entre [ ]: ");
        int codigo = Convert.ToInt32(Console.ReadLine());
        Datos.ActorBorra(codigo);
        Console.WriteLine("\nActor borrado.");
        Console.ReadKey();
    }

    //Pantalla para mostrar en que series trabaja el actor
    public void ActorTrabaja() {
        List<string> ListaSeries;
        Console.WriteLine("\tListar series donde actúa");
        Console.WriteLine("¿Cuál? Escriba el número que está entre [ ]: ");
        int codigo = Convert.ToInt32(Console.ReadLine());
        ListaSeries = Datos.ActorTrabaja(codigo);
        for (int cont = 0; cont < ListaSeries.Count; cont++)
            Console.WriteLine(ListaSeries[cont]);
        Console.WriteLine("\nPresione");
        Console.ReadKey();
    }

    //Pantalla para adicionar series
    public void SerieAdiciona() {

```



```

        Console.WriteLine("\tAdicionar serie al listado");
        Console.Write("¿Nombre? ");
        string nombre = Console.ReadLine();
        Console.Write("¿URL en IMDB? ");
        string url = Console.ReadLine();
        Datos.SerieAdiciona(nombre, url);
        Console.WriteLine("\nSerie adicionada.");
        Console.ReadKey();
    }

    //Pantalla para editar series
    public void SerieEdita() {
        Console.WriteLine("\tEditar serie");
        Console.Write("¿Cuál? Escriba el número que está entre [ ]: ");
        int codigo = Convert.ToInt32(Console.ReadLine());
        Console.Write("¿Nombre? ");
        string nombre = Console.ReadLine();
        Console.Write("¿URL en IMDB? ");
        string url = Console.ReadLine();
        Datos.SerieEdita(codigo, nombre, url);
        Console.WriteLine("\nSerie editada.");
        Console.ReadKey();
    }

    //Pantalla para borrar series
    public void SerieBorra() {
        Console.WriteLine("\tBorrar serie");
        Console.Write("¿Cuál? Escriba el número que está entre [ ]: ");
        int codigo = Convert.ToInt32(Console.ReadLine());
        Datos.SerieBorra(codigo);
        Console.WriteLine("\nSerie borrada.");
        Console.ReadKey();
    }

    //Pantalla para ver el detalle de la serie
    public void SerieDetalle() {
        List<string> ListaActores;

        Console.WriteLine("\t === Detalle de una serie ===");
        Console.Write("¿Cuál? Número[ ]: ");
        int codigo = Convert.ToInt32(Console.ReadLine());

        Console.WriteLine("Nombre: " + Datos.SerieNombre(codigo));
        Console.WriteLine("URL: " + Datos.SerieIMDB(codigo));
        Console.WriteLine("Actores");
        ListaActores = Datos.SerieActores(codigo);
        for (int cont = 0; cont < ListaActores.Count; cont++)
            Console.WriteLine("\t" + ListaActores[cont]);
    }

```

```

        Console.WriteLine("\nENTER para continuar");
        Console.ReadKey();
    }

    //Asociar actor o actriz a una serie
    public void SerieAsocia() {
        Console.WriteLine("\tAsocia un actor o actriz a una serie");
        Console.Write("¿Cuál serie? Número[ ]: ");
        int serie = Convert.ToInt32(Console.ReadLine());
        for (int cont = 0; cont < Datos.ActorTotal(); cont++) {
            Console.Write "[" + cont + " ] ";
            Console.Write(Datos.ActorNombre(cont));
            Console.WriteLine(" URL: " + Datos.ActorURL(cont));
        }
        Console.Write("¿Cuál Actor? Número[ ]: ");
        int actor = Convert.ToInt32(Console.ReadLine());
        Datos.SerieAsocia(serie, actor);
        Console.WriteLine("\nActor asociado a la serie.");
        Console.ReadKey();
    }

    //Pantalla para disociar actor de alguna serie
    public void SerieDisocia() {
        List<string> ListaActores;

        Console.WriteLine("\t === Disociar actor de la serie ===");
        Console.Write("¿Cuál serie? Número[ ]: ");
        int serie = Convert.ToInt32(Console.ReadLine());

        ListaActores = Datos.SerieActores(serie);
        for (int cont = 0; cont < ListaActores.Count; cont++)
            Console.WriteLine "[" + cont + " ] " + ListaActores[cont]);

        Console.Write("¿Cuál actor quiere quitar? Número[ ]: ");
        int actor = Convert.ToInt32(Console.ReadLine());

        Datos.SerieDisocia(serie, actor);
        Console.WriteLine("\nActor retirado de la serie.");
        Console.ReadKey();
    }

}

class Program {
    static void Main() {
        //Se debe llamar primero la capa de persistencia
        //(carga datos de ejemplo)
    }
}

```

```

        Persistencia objDatos = new();

        //Luego se llama la capa visual
        Visual objVisual = new(objDatos);
        objVisual.Menu();
    }
}

```

Ilustración 36: Uso de listas para simular un sistema de información

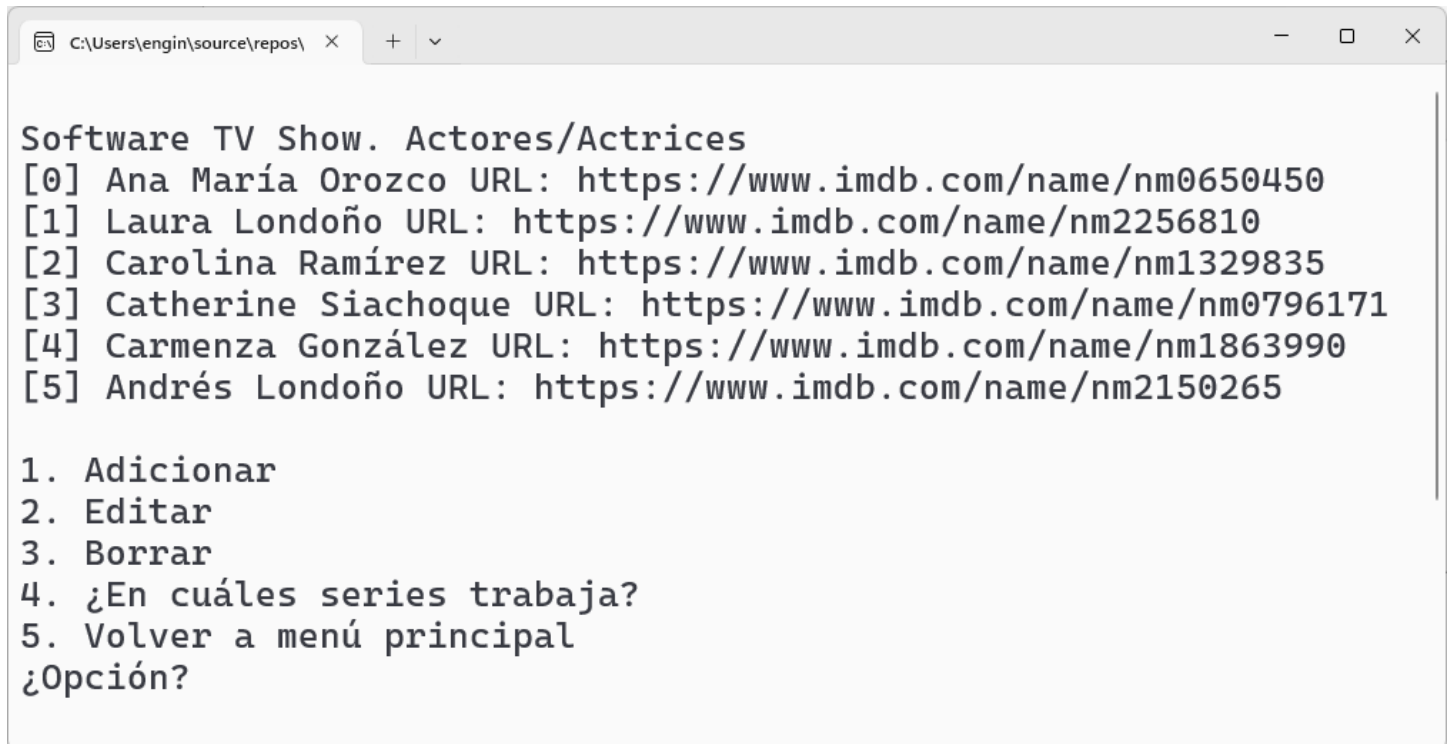
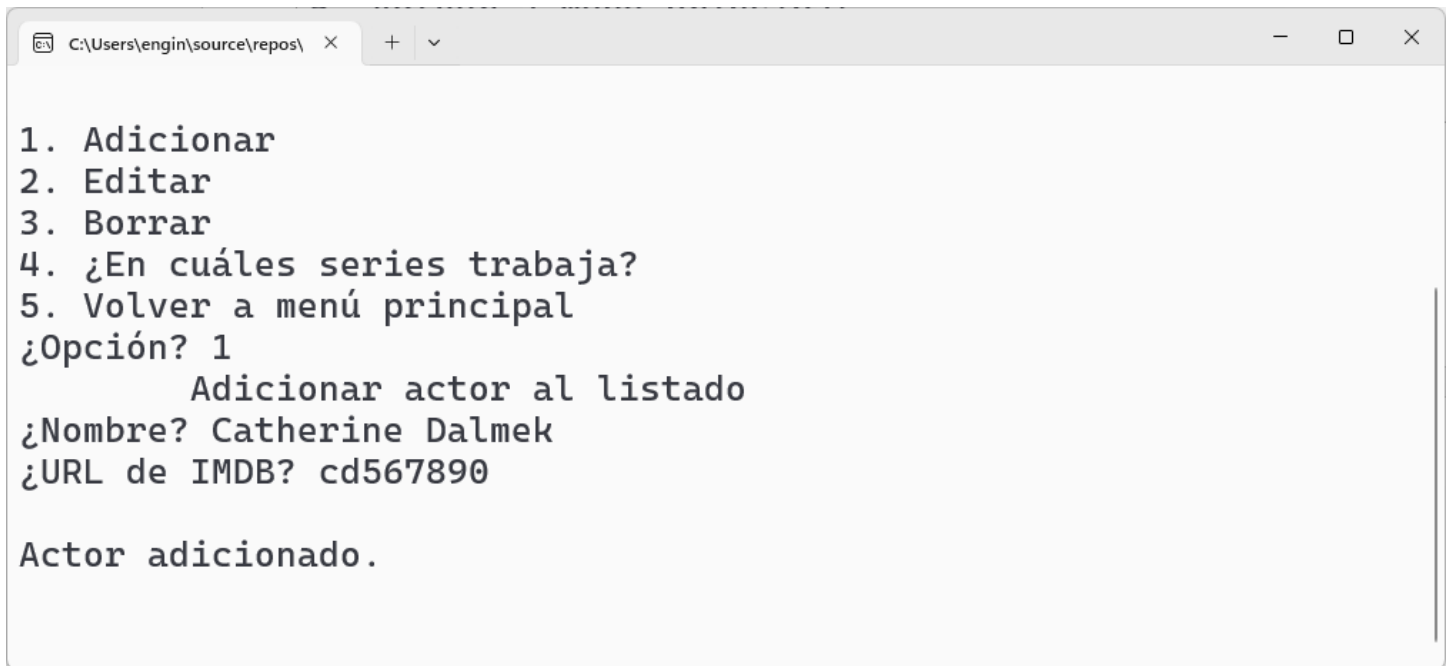


Ilustración 37: Uso de listas para simular un sistema de información



A screenshot of a terminal window with a light gray background. The window's title bar shows a file path 'C:\Users\engin\source\repos\' and standard window controls. The terminal text is as follows:

```
1. Adicionar
2. Editar
3. Borrar
4. ¿En cuáles series trabaja?
5. Volver a menú principal
¿Opción? 1
        Adicionar actor al listado
¿Nombre? Catherine Dalmek
¿URL de IMDB? cd567890

Actor adicionado.
```

Ilustración 38: Uso de listas para simular un sistema de información

Dictionary

Uso de llaves

En una estructura diccionario, hay una llave y un valor (entero, cadena, objeto). Se puede llegar a ese valor usando la llave. Con la instrucción: NombreDiccionario[Llave]. Ejemplo:

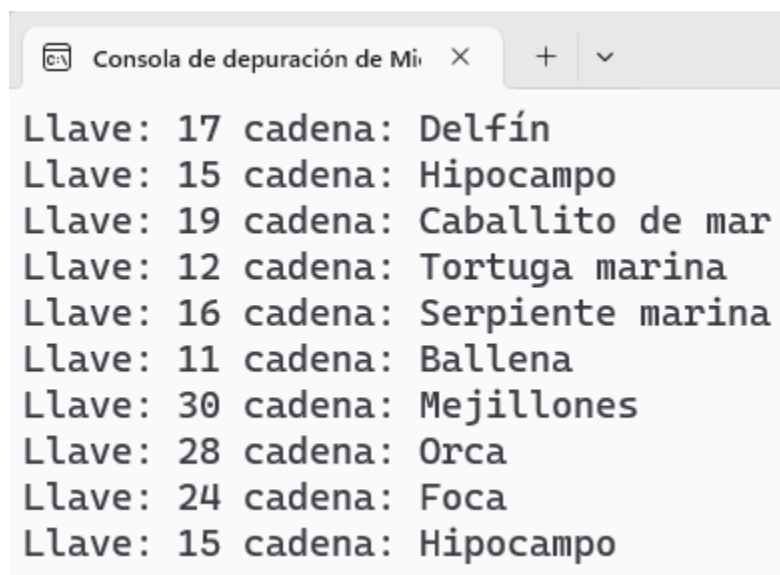
E/023.cs

```
namespace Ejemplo {
    class Program {
        static void Main() {
            Random Azar = new();

            //Se define un diccionario: llave, cadena
            //En este caso la llave es un número entero
            Dictionary<int, string> Animales = new() {
                {11, "Ballena"},
                {12, "Tortuga marina"},
                {13, "Tiburón"},
                {14, "Estrella de mar"},
                {15, "Hipocampo"},
                {16, "Serpiente marina"},
                {17, "Delfín"},
                {18, "Pulpo"},
                {19, "Caballito de mar"},
                {20, "Coral"},
                {21, "Pingüinos"},
                {22, "Calamar"},
                {23, "Gaviota"},
                {24, "Foca"},
                {25, "Manatíes"},
                {26, "Ballena con barba"},
                {27, "Peces Guppy"},
                {28, "Orca"},
                {29, "Medusas"},
                {30, "Mejillones"},
                {31, "Caracoles"}
            };

            for (int cont = 1; cont <= 10; cont++) {
                int Llave = Azar.Next(11, Animales.Count + 11);
                Console.Write("Llave: " + Llave);
                Console.WriteLine(" cadena: " + Animales[Llave]);
            }
        }
    }
}
```

```
}  
}
```



The image shows a browser's developer console with a tab titled 'Consola de depuración de Mi'. The console contains ten lines of output, each representing an object with two properties: 'Llave' and 'cadena'. The values for 'Llave' are 17, 15, 19, 12, 16, 11, 30, 28, 24, and 15. The values for 'cadena' are Delfín, Hipocampo, Caballito de mar, Tortuga marina, Serpiente marina, Ballena, Mejillones, Orca, Foca, and Hipocampo.

```
Llave: 17 cadena: Delfín  
Llave: 15 cadena: Hipocampo  
Llave: 19 cadena: Caballito de mar  
Llave: 12 cadena: Tortuga marina  
Llave: 16 cadena: Serpiente marina  
Llave: 11 cadena: Ballena  
Llave: 30 cadena: Mejillones  
Llave: 28 cadena: Orca  
Llave: 24 cadena: Foca  
Llave: 15 cadena: Hipocampo
```

Ilustración 39: Dictionary

Llaves tipo string

También se puede usar una llave de tipo cadena. El diccionario tiene instrucciones de adicionar y borrar.

E/024.cs

```
namespace Ejemplo {
    class Program {
        static void Main() {
            //Se define un diccionario: llave, cadena
            //En este caso la llave es una cadena
            Dictionary<string, string> Extension = new() {
                {"exe", "Ejecutable"},
                {"com", "Ejecutable DOS"},
                {"vb", "Visual Basic .NET"},
                {"cs", "C#"},
                {"js", "JavaScript"},
                {"xlsx", "Excel"},
                {"docx", "Word"},
                {"html", "HTML 5"}
            };

            //Otra forma de adicionar
            Extension.Add("pptx", "PowerPoint");

            //Trae un elemento dada una llave
            string Llave = "cs";
            Console.Write("Llave: " + Llave);
            Console.WriteLine(" valor es: " + Extension[Llave]);

            //Tamaño del diccionario
            Console.WriteLine("Tamaño: " + Extension.Count);

            //Elimina un elemento
            Extension.Remove("docx");

            //Tamaño del diccionario
            Console.WriteLine("Después de eliminar: " + Extension.Count);
        }
    }
}
```

```
Llave: cs valor es: C#  
Tamaño: 9  
Después de eliminar: 8
```

Ilustración 40: Dictionary

Manejo de objetos en un Dictionary

Un "Dictionary" puede albergar objetos. Además, tiene una serie de métodos (adicionar, consultar, listar llaves, verificar si existe llave) que se ven a continuación:

E/025.cs

```
namespace Ejemplo {
    //Una clase con varios atributos
    class MiClase {
        public int Numero { get; set; }
        public double Valor { get; set; }
        public char Car { get; set; }
        public string Cad { get; set; }

        public MiClase(int Numero, double Valor, char Car, string Cad) {
            this.Numero = Numero;
            this.Valor = Valor;
            this.Car = Car;
            this.Cad = Cad;
        }
    }

    class Program {
        static void Main() {
            //Se define un diccionario: llave, objeto
            //En este caso la llave es una cadena
            var Objetos = new Dictionary<string, MiClase> {
                {"uno", new MiClase(1, 0.2, 'r', "Leafar") },
                {"dos", new MiClase(8, -7.1, 'a', "Otrebla")},
                {"tres", new MiClase(23, -13.6, 'm', "Onerom")},
                {"cuatro", new MiClase(49, 16.83, 'p', "Arrap")}
            };

            //Trae los datos del objeto guardado en el diccionario
            string Llave = "tres";
            Console.Write("Llave: " + Llave);
            Console.WriteLine(" atributo es: " + Objetos[Llave].Cad);

            Console.Write("Llave: " + Llave);
            Console.WriteLine(" atributo es: " + Objetos[Llave].Numero);

            Console.Write("Llave: " + Llave);
            Console.WriteLine(" atributo es: " + Objetos[Llave].Valor);

            //Guarda las llaves en una lista
            Console.WriteLine("\r\nLista de Llaves:");
            var ListaLlaves = new List<string>(Objetos.Keys);
            foreach (string Llaves in ListaLlaves) {
```

```

        Console.WriteLine("Llave: " + Llaves);
    }

    //Verifica si existe una llave
    Console.WriteLine("\r\nVerifica si existe una llave:");
    if (Objetos.ContainsKey("cuatro")) {
        Console.WriteLine(Objetos["cuatro"].Cad);
    }
    else {
        Console.WriteLine("No existe esa llave");
    }
}
}
}

```

```

Llave: tres atributo es: Onerom
Llave: tres atributo es: 23
Llave: tres atributo es: -13.6

Lista de Llaves:
Llave: uno
Llave: dos
Llave: tres
Llave: cuatro

Verifica si existe una llave:
Arrap

```

Ilustración 41: Dictionary, manejo de objetos

Queue (Cola)

Una cola se parece a un ArrayList, la diferencia es que NO se puede acceder a los elementos por un índice, se respeta el orden de llegada, primero en entrar es primero en salir.

E/026.cs

```
using System.Collections;

namespace Ejemplo {
    class Program {
        static void Main() {
            //Se define una cola: Queue
            Queue Cola = new();

            //Se agregan elementos a la cola
            Cola.Enqueue("aaa");
            Cola.Enqueue("bbb");
            Cola.Enqueue("ccc");
            Cola.Enqueue("ddd");
            Cola.Enqueue("eee");
            Cola.Enqueue("fff");

            //Número de elementos en la cola
            Console.WriteLine("Número de elementos: " + Cola.Count);

            //Imprimir la cola
            Console.WriteLine("\r\nElementos: ");
            foreach (object elemento in Cola)
                Console.Write(elemento + ", ");

            //Quitar elemento de la cola.
            //Primero en llegar, primero en salir, luego quitaría a "aaa"
            Cola.Dequeue();
            Console.WriteLine("\r\nQuitar un elemento de la cola: ");
            foreach (object elemento in Cola)
                Console.Write(elemento + ", ");

            //Verificar si hay un elemento en la cola
            string Buscar = "ddd";
            if (Cola.Contains(Buscar) == true) {
                Console.WriteLine("\r\n\r\nLa cola contiene: " + Buscar);
            }
            else
                Console.WriteLine("\r\n\r\nLa cola NO contiene: " + Buscar);

            //Obtener el primer elemento de la cola
            //sin borrar ese elemento
        }
    }
}
```

```

string PrimerElemento = Convert.ToString(Cola.Peek());
Console.WriteLine("\r\nPrimer elemento: " + PrimerElemento);

//Leer y borrar la cola
Console.WriteLine("\r\nLee y borra la cola: ");
while (Cola.Count > 0)
    Console.Write(Cola.Dequeue() + "; ");
Console.WriteLine("\r\nNúmero de elementos: " + Cola.Count);
}
}
}

```

```

Número de elementos: 6

Elementos:
aaa, bbb, ccc, ddd, eee, fff,
Quitar un elemento de la cola:
bbb, ccc, ddd, eee, fff,

La cola contiene: ddd

Primer elemento: bbb

Lee y borra la cola:
bbb; ccc; ddd; eee; fff;
Número de elementos: 0

```

Ilustración 42: Queue

Dato definido en la cola

Los elementos de la cola pueden ser de tipo definido. Se modifica el programa anterior para que trabaje con el tipo string, obteniendo el mismo resultado.

E/027.cs

```
namespace Ejemplo {
    class Program {
        static void Main() {
            //Se define una cola de tipo string: Queue
            Queue<string> Cola = new();

            //Se agregan elementos a la cola
            Cola.Enqueue("aaa");
            Cola.Enqueue("bbb");
            Cola.Enqueue("ccc");
            Cola.Enqueue("ddd");
            Cola.Enqueue("eee");
            Cola.Enqueue("fff");

            //Número de elementos en la cola
            Console.WriteLine("Número de elementos: " + Cola.Count);

            //Imprimir la cola
            Console.WriteLine("\r\nElementos: ");
            foreach (object elemento in Cola)
                Console.Write(elemento + ", ");

            //Quitar elemento de la cola.
            //Primero en llegar, primero en salir, luego quitaría a "aaa"
            Cola.Dequeue();
            Console.WriteLine("\r\nQuitar un elemento de la cola: ");
            foreach (object elemento in Cola)
                Console.Write(elemento + ", ");

            //Verificar si hay un elemento en la cola
            string Buscar = "ddd";
            if (Cola.Contains(Buscar) == true) {
                Console.WriteLine("\r\n\r\nLa cola contiene: " + Buscar);
            }
            else
                Console.WriteLine("\r\n\r\nLa cola NO contiene: " + Buscar);

            //Obtener el primer elemento de la cola
            //sin borrar ese elemento
            string PrimerElemento = Convert.ToString(Cola.Peek());
            Console.WriteLine("\r\nPrimer elemento: " + PrimerElemento);
        }
    }
}
```

```

//Leer y borrar la cola
Console.WriteLine("\r\nLee y borra la cola: ");
while (Cola.Count > 0)
    Console.Write(Cola.Dequeue() + "; ");
Console.WriteLine("\r\nNúmero de elementos: " + Cola.Count);
}
}
}

```

```

Número de elementos: 6

Elementos:
aaa, bbb, ccc, ddd, eee, fff,
Quitar un elemento de la cola:
bbb, ccc, ddd, eee, fff,

La cola contiene: ddd

Primer elemento: bbb

Lee y borra la cola:
bbb; ccc; ddd; eee; fff;
Número de elementos: 0

```

Ilustración 43: Dato definido en la cola

Objetos en la cola

Una cola puede tener objetos personalizados.

E/028.cs

```
namespace Ejemplo {
    //Una clase con varios atributos
    class MiClase {
        public int Numero { get; set; }
        public double Valor { get; set; }
        public char Car { get; set; }
        public string Cad { get; set; }

        public MiClase(int Numero, double Valor, char Car, string Cad) {
            this.Numero = Numero;
            this.Valor = Valor;
            this.Car = Car;
            this.Cad = Cad;
        }
    }

    class Program {
        static void Main() {
            //Se define una cola de tipo objeto personalizado
            Queue<MiClase> Cola = new();

            //Se agregan elementos a la cola
            Cola.Enqueue(new MiClase(1, 0.2, 'r', "Leafar"));
            Cola.Enqueue(new MiClase(8, -7.1, 'a', "Otrebla"));
            Cola.Enqueue(new MiClase(23, -13.6, 'm', "Onerom"));
            Cola.Enqueue(new MiClase(49, 16.83, 'p', "Arrap"));

            //Número de elementos en la cola
            Console.WriteLine("Número de elementos: " + Cola.Count);

            //Imprimir la cola
            Console.WriteLine("\r\nElementos: ");
            foreach (MiClase elemento in Cola)
                Console.Write(elemento.Cad + ", ");

            //Quitar elemento de la cola
            //Primero en llegar, primero en salir,
            //luego quitaría a "aaa"
            Cola.Dequeue();
            Console.WriteLine("\r\nAl quitar un elemento de la cola: ");
            foreach (MiClase elemento in Cola)
                Console.Write(elemento.Cad + ", ");
        }
    }
}
```

```

//Obtener el primer elemento de la cola
//sin borrar ese elemento
MiClase PrimerElemento = Cola.Peek();
Console.WriteLine("\r\n\r\nPrimer: " + PrimerElemento.Cad);

//Leer y borrar la cola
Console.WriteLine("\r\nLee y borra la cola: ");
while (Cola.Count > 0)
    Console.Write(Cola.Dequeue().Cad + "; ");
Console.WriteLine("\r\nNúmero de elementos: " + Cola.Count);

//Agrega elementos a la cola y luego la borra
Cola.Enqueue(new MiClase(7, 6.5, 'z', "qwerty"));
Cola.Enqueue(new MiClase(4, -3.2, 'y', "asdfg"));
Console.WriteLine("\r\nElementos: " + Cola.Count);
Cola.Clear();
Console.WriteLine("Después de borrar: " + Cola.Count);
}
}
}

```

```

Número de elementos: 4

Elementos:
Leafar, Otrebla, Onerom, Arrap,
Al quitar un elemento de la cola:
Otrebla, Onerom, Arrap,

Primer: Otrebla

Lee y borra la cola:
Otrebla; Onerom; Arrap;
Número de elementos: 0

Elementos: 2
Después de borrar: 0

```

Ilustración 44: Objetos en la cola

Stack (Pila)

La pila es una estructura que análogo a una pila de platos, cuando se adicionan elementos, estos van quedando encima, por lo que el último en entrar es el primero en salir. Es muy similar a la cola, sólo cambian algunos métodos como el Push (poner) y Pop (retirar).

E/029.cs

```
using System.Collections;

namespace Ejemplo {
    class Program {
        static void Main() {
            //Se define una pila: Queue
            Stack Pila = new();

            //Se agregan elementos a la pila
            Pila.Push("aaa");
            Pila.Push("bbb");
            Pila.Push("ccc");
            Pila.Push("ddd");
            Pila.Push("eee");
            Pila.Push("fff");

            //Número de elementos en la pila
            Console.WriteLine("Número de elementos: " + Pila.Count);

            //Imprimir la pila
            Console.WriteLine("\r\nElementos: ");
            foreach (object elemento in Pila)
                Console.Write(elemento + ", ");

            //Quitar elemento de la pila
            //Último en llegar, primero en salir,
            //luego quitaría a "fff"
            Pila.Pop();
            Console.WriteLine("\r\nAl quitar un elemento de la pila: ");
            foreach (object elemento in Pila)
                Console.Write(elemento + ", ");

            //Verificar si hay un elemento en la pila
            string Buscar = "ddd";
            if (Pila.Contains(Buscar) == true) {
                Console.WriteLine("\r\n\r\nLa pila contiene: " + Buscar);
            }
            else
                Console.WriteLine("\r\n\r\nLa pila NO contiene: " + Buscar);
        }
    }
}
```

```

//Obtener el primer elemento de la pila sin borrar ese elemento
string PrimerElemento = Convert.ToString(Pila.Peek());
Console.WriteLine("\r\nPrimer elemento: " + PrimerElemento);

//Leer y borrar la pila
Console.WriteLine("\r\nLee y borra la pila: ");
while (Pila.Count > 0)
    Console.Write(Pila.Pop() + "; ");
Console.WriteLine("\r\nElementos: " + Pila.Count);
}
}
}

```

Número de elementos: 6

Elementos:

fff, eee, ddd, ccc, bbb, aaa,

Al quitar un elemento de la pila:

eee, ddd, ccc, bbb, aaa,

La pila contiene: ddd

Primer elemento: eee

Lee y borra la pila:

eee; ddd; ccc; bbb; aaa;

Elementos: 0

Ilustración 45: Stack

Dato definido en la pila

Los elementos de la pila pueden ser de tipo definido. Se modifica el programa anterior para que trabaje con el tipo string, obteniendo el mismo resultado.

E/030.cs

```
namespace Ejemplo {
    class Program {
        static void Main() {
            //Se define una pila: Queue
            Stack<string> Pila = new();

            //Se agregan elementos a la pila
            Pila.Push("aaa");
            Pila.Push("bbb");
            Pila.Push("ccc");
            Pila.Push("ddd");
            Pila.Push("eee");
            Pila.Push("fff");

            //Número de elementos en la pila
            Console.WriteLine("Número de elementos: " + Pila.Count);

            //Imprimir la pila
            Console.WriteLine("\r\nElementos: ");
            foreach (object elemento in Pila)
                Console.Write(elemento + ", ");

            //Quitar elemento de la pila
            //Último en llegar, primero en salir,
            //luego quitaría a "fff"
            Pila.Pop();
            Console.WriteLine("\r\nAl quitar un elemento de la pila: ");
            foreach (string elemento in Pila)
                Console.Write(elemento + ", ");

            //Verificar si hay un elemento en la pila
            string Buscar = "ddd";
            if (Pila.Contains(Buscar) == true) {
                Console.WriteLine("\r\n\r\nLa pila contiene: " + Buscar);
            }
            else
                Console.WriteLine("\r\n\r\nLa pila NO contiene: " + Buscar);

            //Obtener el primer elemento de
            //la pila sin borrar ese elemento
            string PrimerElemento = Pila.Peek();
        }
    }
}
```

```

        Console.WriteLine("\r\nPrimer elemento: " + PrimerElemento);

        //Leer y borrar la pila
        Console.WriteLine("\r\nLee y borra la pila: ");
        while (Pila.Count > 0)
        {
            Console.Write(Pila.Pop() + "; ");
            Console.WriteLine("\r\nNúmero de elementos: " + Pila.Count);
        }
    }
}

```

```

Número de elementos: 6

Elementos:
fff, eee, ddd, ccc, bbb, aaa,
Al quitar un elemento de la pila:
eee, ddd, ccc, bbb, aaa,

La pila contiene: ddd

Primer elemento: eee

Lee y borra la pila:
eee; ddd; ccc; bbb; aaa;
Número de elementos: 0

```

Ilustración 46: Dato definido en la pila

```
namespace Ejemplo {
    //Una clase con varios atributos
    class MiClase {
        public int Numero { get; set; }
        public double Valor { get; set; }
        public char Car { get; set; }
        public string Cad { get; set; }

        public MiClase(int Numero, double Valor, char Car, string Cad) {
            this.Numero = Numero;
            this.Valor = Valor;
            this.Car = Car;
            this.Cad = Cad;
        }
    }

    class Program {
        static void Main() {
            //Se define una pila de tipo objeto personalizado
            Stack<MiClase> Pila = new();

            //Se agregan elementos a la pila
            Pila.Push(new MiClase(1, 0.2, 'r', "Leafar"));
            Pila.Push(new MiClase(8, -7.1, 'a', "Otrebla"));
            Pila.Push(new MiClase(23, -13.6, 'm', "Onerom"));
            Pila.Push(new MiClase(49, 16.83, 'p', "Arrap"));

            //Número de elementos en la pila
            Console.WriteLine("Número de elementos: " + Pila.Count);

            //Imprimir la pila
            Console.WriteLine("\r\nElementos: ");
            foreach (MiClase elemento in Pila)
                Console.Write(elemento.Cad + ", ");

            //Quitar elemento de la pila
            Pila.Pop(); //Último en llegar, primero en salir
            Console.WriteLine("\r\nAl quitar un elemento de la pila: ");
            foreach (MiClase elemento in Pila)
                Console.Write(elemento.Cad + ", ");

            //Obtener el primer elemento de la pila
            //sin borrar ese elemento
            MiClase Primer = Pila.Peek();
            Console.WriteLine("\r\n\r\nElemento más arriba: " + Primer.Cad);
        }
    }
}
```

```

//Leer y borrar la pila
Console.WriteLine("\r\nLee y borra la pila: ");
while (Pila.Count > 0)
    Console.Write(Pila.Pop().Cad + "; ");
Console.WriteLine("\r\nNúmero de elementos: " + Pila.Count);

//Agrega elementos a la pila y luego la borra
Pila.Push(new MiClase(7, 6.5, 'z', "qwerty"));
Pila.Push(new MiClase(4, -3.2, 'y', "asdfg"));
Console.WriteLine("\r\nWlementos: " + Pila.Count);
Pila.Clear();
Console.WriteLine("Después de borrar: " + Pila.Count);
}
}
}

```

```

Número de elementos: 4

Elementos:
Arrap, Onerom, Otrebla, Leafar,
Al quitar un elemento de la pila:
Onerom, Otrebla, Leafar,

Elemento más arriba: Onerom

Lee y borra la pila:
Onerom; Otrebla; Leafar;
Número de elementos: 0

Wlementos: 2
Después de borrar: 0

```

Ilustración 47: Objetos en la pila

Hashtable

Hashtable funciona similar a Dictionary. Estas son sus diferencias:

Hashtable	Dictionary
Es seguro ser accedido por múltiples hilos ("thread safe").	Sólo miembros públicos estáticos son seguros para ser accedidos por hilos.
Retorna "null" si se intenta acceder a un dato por una llave inexistente.	Genera un error si intenta acceder por una llave inexistente. Requiere usar try catch.
La recuperación de datos es más lenta.	La recuperación de datos es más rápida.
No requiere definir el tipo de dato de la llave y el valor.	Requiere definir el tipo de datos de la llave y el valor.

E/032.cs

```
using System.Collections;

namespace Ejemplo {
    class Program {
        static void Main() {
            Random Azar = new();

            //Se define un Hashtable
            //En este caso la llave es un número entero
            Hashtable Animales = new();
            Animales.Add(11, "Ballena");
            Animales.Add(12, "Tortuga marina");
            Animales.Add(13, "Tiburón");
            Animales.Add(14, "Estrella de mar");
            Animales.Add(15, "Hipocampo");
            Animales.Add(16, "Serpiente marina");
            Animales.Add(17, "Delfín");
            Animales.Add(18, "Pulpo");
            Animales.Add(19, "Caballito de mar");
            Animales.Add(20, "Coral");
            Animales.Add(21, "Pingüinos");
            Animales.Add(22, "Calamar");
            Animales.Add(23, "Gaviota");
            Animales.Add(24, "Foca");
            Animales.Add(25, "Manaties");
            Animales.Add(26, "Ballena con barba");
            Animales.Add(27, "Peces Guppy");
            Animales.Add(28, "Orca");
            Animales.Add(29, "Medusas");
            Animales.Add(30, "Mejillones");
            Animales.Add(31, "Caracoles");
```

```

for (int cont = 1; cont <= 10; cont++) {
    //Busque al azar un número entre mínimo y máximo
    //valor de llave.
    //Hay que sumarle +1 al máximo valor de llave para
    //que quede dentro del rango de los números aleatorios
    int Llave = Azar.Next(11, 31 + 1);

    //Muestre el registro según la llave
    Console.Write("Llave: " + Llave);
    Console.WriteLine(" cadena: " + Animales[Llave]);
}
}
}
}

```

```

Llave: 24 cadena: Foca
Llave: 29 cadena: Medusas
Llave: 18 cadena: Pulpo
Llave: 27 cadena: Peces Guppy
Llave: 13 cadena: Tiburón
Llave: 29 cadena: Medusas
Llave: 18 cadena: Pulpo
Llave: 21 cadena: Pingüinos
Llave: 22 cadena: Calamar
Llave: 11 cadena: Ballena

```

Ilustración 48: Hashtable

Manejo de objetos en un Hashtable

Cabe recordar que hay que hacer la conversión para acceder a los atributos del objeto almacenado así:

(objeto as clase).atributo

E/033.cs

```
using System.Collections;

namespace Ejemplo {
    class MiClase {
        public int Numero { get; set; }
        public double Valor { get; set; }
        public char Car { get; set; }
        public string Cad { get; set; }

        public MiClase(int Numero, double Valor, char Car, string Cad) {
            this.Numero = Numero;
            this.Valor = Valor;
            this.Car = Car;
            this.Cad = Cad;
        }
    }

    class Program {
        static void Main() {
            //Se define un Hashtable
            Hashtable Tablahash = [];

            //Agrega registros
            Tablahash.Add("uno", new MiClase(1, 0.2, 'r', "Leafar"));
            Tablahash.Add("dos", new MiClase(8, -7.1, 'a', "Otrebla"));
            Tablahash.Add("tres", new MiClase(23, -13.6, 'm', "Onerom"));
            Tablahash.Add("cuatro", new MiClase(49, 16.83, 'p', "Arrap"));

            //Trae los datos del objeto guardado en el diccionario
            string Llave = "tres";
            Console.Write("Llave: " + Llave);
            Console.WriteLine(" atributo: " + (Tablahash[Llave] as
MiClase).Cad);

            Console.Write("Llave: " + Llave);
            Console.WriteLine(" atributo: " + (Tablahash[Llave] as
MiClase).Numero);
        }
    }
}
```

```

        Console.WriteLine("Llave: " + Llave);
        Console.WriteLine(" atributo: " + (Tablahash[Llave] as
MiClase).Valor);

        //Guarda las llaves en una variable de colección
        Console.WriteLine("\r\nLista de Llaves:");
        var ListaLlaves = Tablahash.Keys;
        foreach (string Llaves in ListaLlaves) {
            Console.WriteLine("Llave: " + Llaves);
        }

        //Verifica si existe una llave
        Console.WriteLine("\r\nVerifica si existe una llave:");
        if (Tablahash.ContainsKey("cuatro"))
            Console.WriteLine((Tablahash["cuatro"] as MiClase).Cad);
        else
            Console.WriteLine("No existe esa llave");
    }
}
}

```

```

Llave: tres atributo: Onerom
Llave: tres atributo: 23
Llave: tres atributo: -13.6

```

```

Lista de Llaves:
Llave: tres
Llave: dos
Llave: uno
Llave: cuatro

```

```

Verifica si existe una llave:
Arrap

```

Ilustración 49: Manejo de objetos en un Hashtable

SortedList

SortedList es muy similar a Dictionary, en este caso la lista es ordenada automáticamente por las llaves. Eso es visible al imprimirla.

E/034.cs

```
namespace Ejemplo {
    class Program {
        static void Main() {
            //Se define una lista ordenada: llave, cadena
            //En este caso la llave es una cadena
            SortedList<string, string> Extensiones = new() {
                { "exe", "Ejecutable" },
                { "com", "Ejecutable DOS" },
                { "vb", "Visual Basic .NET" },
                { "cs", "C#" },
                { "js", "JavaScript" },
                { "xlsx", "Excel" },
                { "docx", "Word" },
                { "pptx", "PowerPoint" }
            };

            //Imprime la lista ordenada
            foreach (object elemento in Extensiones)
                Console.WriteLine(elemento);

            //Otra forma de adicionar
            Extensiones.Add("html", "HTML 5");

            //Imprime llave y valor
            var ListaLlaves = Extensiones.Keys;
            Console.WriteLine("\r\nImprime llave y valor en separado");
            foreach (string Llave in ListaLlaves) {
                Console.WriteLine("Llave: " + Llave);
                Console.WriteLine(" Valor: " + Extensiones[Llave]);
            }
        }
    }
}
```

```
[com, Ejecutable DOS]
[cs, C#]
[docx, Word]
[exe, Ejecutable]
[html, HTML 5]
[js, JavaScript]
[pptx, PowerPoint]
[vb, Visual Basic .NET]
[xlsx, Excel]
```

Imprime llave y valor en separado

Llave: com Valor: Ejecutable DOS

Llave: cs Valor: C#

Llave: docx Valor: Word

Llave: exe Valor: Ejecutable

Llave: html Valor: HTML 5

Llave: js Valor: JavaScript

Llave: pptx Valor: PowerPoint

Llave: vb Valor: Visual Basic .NET

Llave: xlsx Valor: Excel

Ilustración 50: SortedList

LinkedList

Lista enlazada, no se accede directamente por un índice.

E/035.cs

```
namespace Ejemplo {
    class Program {
        static void Main() {
            //Se define una lista enlazada
            LinkedList<string> Lenguajes = new();

            //Agrega al final
            Console.WriteLine("Agregando con AddLast");
            Lenguajes.AddLast("Visual Basic .NET");
            Lenguajes.AddLast("F#");
            Lenguajes.AddLast("C#");
            Lenguajes.AddLast("TypeScript");

            //Imprime esa lista
            foreach (string elemento in Lenguajes)
                Console.Write(elemento + "; ");

            //Agrega al inicio
            Console.WriteLine("\r\n\r\nAgregando con AddFirst");
            Lenguajes.AddFirst("C++");
            Lenguajes.AddFirst("C");

            //Imprime esa lista
            foreach (string elemento in Lenguajes)
                Console.Write(elemento + "; ");

            //Agrega al final
            Lenguajes.AddLast("Python");
            Console.WriteLine("\r\n\r\nAgregando con AddLast");
            foreach (string elemento in Lenguajes)
                Console.Write(elemento + "; ");

            //Cantidad
            Console.WriteLine("\r\n\r\nCantidad es: " + Lenguajes.Count);

            //Elimina primer elemento
            Lenguajes.RemoveFirst();
            Console.WriteLine("\r\nEliminado el primer elemento");
            foreach (string elemento in Lenguajes)
                Console.Write(elemento + "; ");
        }
    }
}
```

```

//Elimina último elemento
Lenguajes.RemoveLast();
Console.WriteLine("\r\n\r\nEliminado el último elemento");
foreach (string elemento in Lenguajes)
    Console.Write(elemento + "; ");

//Elimina determinado elemento
Lenguajes.Remove("F#");
Console.WriteLine("\r\n\r\nEliminado F#");
foreach (string elemento in Lenguajes)
    Console.Write(elemento + "; ");

//Adiciona antes de C#
//Busca el nodo que tiene C#
LinkedListNode<string> nodoPosiciona = Lenguajes.Find("C#");
Lenguajes.AddBefore(nodoPosiciona, "Assembler");
Console.WriteLine("\r\n\r\nAdiciona antes de C#");
foreach (string elemento in Lenguajes)
    Console.Write(elemento + "; ");

//Adiciona después de C#
Lenguajes.AddAfter(nodoPosiciona, "Ada");
Console.WriteLine("\r\n\r\nAdiciona después de C#");
foreach (string elemento in Lenguajes)
    Console.Write(elemento + "; ");
    }
}
}

```

```
Agregando con AddLast
Visual Basic .NET; F#; C#; TypeScript;

Agregando con AddFirst
C; C++; Visual Basic .NET; F#; C#; TypeScript;

Agregando con AddLast
C; C++; Visual Basic .NET; F#; C#; TypeScript; Python;

Cantidad es: 7

Eliminado el primer elemento
C++; Visual Basic .NET; F#; C#; TypeScript; Python;

Eliminado el último elemento
C++; Visual Basic .NET; F#; C#; TypeScript;

Eliminado F#
C++; Visual Basic .NET; C#; TypeScript;

Adiciona antes de C#
C++; Visual Basic .NET; Assembler; C#; TypeScript;

Adiciona después de C#
C++; Visual Basic .NET; Assembler; C#; Ada; TypeScript;
```

Ilustración 51: LinkedList

```
namespace Ejemplo {

    //Una clase con varios atributos
    class MiClase {
        public int Numero { get; set; }
        public double Valor { get; set; }
        public char Car { get; set; }
        public string Cad { get; set; }

        public MiClase(int Numero, double Valor, char Car, string Cad) {
            this.Numero = Numero;
            this.Valor = Valor;
            this.Car = Car;
            this.Cad = Cad;
        }
    }

    class Program {
        static void Main() {
            //Se define una lista enlazada
            LinkedList<MiClase> Lenguajes = new();

            //Agrega al final
            Console.WriteLine("Agregando con AddLast");
            Lenguajes.AddLast(new MiClase(16, 83.29, 'R', "Lenguaje R"));
            Lenguajes.AddLast(new MiClase(29, 89.7, 'A', "ADA"));
            Lenguajes.AddLast(new MiClase(2, 80.19, 'M', "Máquina"));
            Lenguajes.AddLast(new MiClase(95, 7.21, 'P', "PHP"));

            //Imprime esa lista
            foreach (MiClase elemento in Lenguajes)
                Console.Write(elemento.Cad + "; ");

            //Agrega al inicio
            Lenguajes.AddFirst(new MiClase(78, 12.32, 'S', "C#"));
            Lenguajes.AddFirst(new MiClase(5, -3.1, 'V', "J#"));

            //Imprime esa lista
            Console.WriteLine("\r\n\r\nAgregando con AddFirst");
            foreach (MiClase elemento in Lenguajes)
                Console.Write(elemento.Cad + "; ");

            //Agrega al final
            Lenguajes.AddLast(new MiClase(16, 83.29, 'C', "C++"));
        }
    }
}
```



```

        Console.WriteLine("\r\n\r\nAgregando con AddLast");
        foreach (MiClase elemento in Lenguajes)
            Console.Write(elemento.Cad + "; ");

        //Cantidad
        Console.WriteLine("\r\n\r\nCantidad es: " + Lenguajes.Count);

        //Elimina primer elemento
        Lenguajes.RemoveFirst();
        Console.WriteLine("\r\n\r\nEliminado el primer elemento");
        foreach (MiClase elemento in Lenguajes)
            Console.Write(elemento.Cad + "; ");

        //Elimina último elemento
        Lenguajes.RemoveLast();
        Console.WriteLine("\r\n\r\n\r\nEliminado el último elemento");
        foreach (MiClase elemento in Lenguajes)
            Console.Write(elemento.Cad + "; ");
    }
}

```

```

Agregando con AddLast
Lenguaje R; ADA; Máquina; PHP;

Agregando con AddFirst
J#; C#; Lenguaje R; ADA; Máquina; PHP;

Agregando con AddLast
J#; C#; Lenguaje R; ADA; Máquina; PHP; C++;

Cantidad es: 7

Eliminado el primer elemento
C#; Lenguaje R; ADA; Máquina; PHP; C++;

Eliminado el último elemento
C#; Lenguaje R; ADA; Máquina; PHP;

```

Ilustración 52: Objetos en LinkedList