Ensures that a given array can hold up to minCapacity elements.

```
/*
Copyright 1999 CERN – European Organization for Nuclear Research.
Permission to use, copy, modify, distribute and sell this software and its documentation for any purpose
is hereby granted without fee, provided that the above copyright notice appear in all copies and
that both that copyright notice and this permission notice appear in supporting documentation.
CERN makes no representations about the suitability of this software for any purpose.
It is provided “as is” without expressed or implied warranty.
*/
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace DiscoveryLogic.Common.Numeric
{
public class Arrays : System.Object
{
///

Ensures that a given array can hold up to minCapacity elements.
///
/// Returns the identical array if it can hold at least the number of elements specified.
/// Otherwise, returns a new array with increased capacity containing the same elements, ensuring
```

```
/// that it can hold at least the number of elements specified
by
/// the minimum capacity argument.
///
///
///  the desired minimum capacity.
///  public static sbyte[] ensureCapacity(sbyte[] array, int
minCapacity)
{
int oldCapacity = array.Length;
sbyte[] newArray;
if (minCapacity > oldCapacity)
{
int newCapacity = (oldCapacity * 3) / 2 + 1;
if (newCapacity < minCapacity) { newCapacity = minCapacity; }
newArray = new sbyte[newCapacity]; Array.Copy(array, 0,
newArray, 0, oldCapacity); } else { newArray = array; } return
newArray; } ///
Ensures that a given array can hold up to minCapacity
elements.
///
/// Returns the identical array if it can hold at least the
number of elements specified.
/// Otherwise, returns a new array with increased capacity
containing the same elements, ensuring
/// that it can hold at least the number of elements specified
by
/// the minimum capacity argument.
///
///

///  the desired minimum capacity.
///  public static short[] ensureCapacity(short[] array, int
minCapacity)
{
int oldCapacity = array.Length;
short[] newArray;
if (minCapacity > oldCapacity)
```

```
{
int newCapacity = (oldCapacity * 3) / 2 + 1;
if (newCapacity < minCapacity) { newCapacity = minCapacity; }
newArray = new short[newCapacity]; Array.Copy(array, 0,
newArray, 0, oldCapacity); } else { newArray = array; } return
newArray; } ///
Ensures that a given array can hold up to minCapacity
elements.
///
/// Returns the identical array if it can hold at least the
number of elements specified.
/// Otherwise, returns a new array with increased capacity
containing the same elements, ensuring
/// that it can hold at least the number of elements specified
by
/// the minimum capacity argument.
///
///

///  the desired minimum capacity.
///  public static bool[] ensureCapacity(bool[] array, int
minCapacity)
{
int oldCapacity = array.Length;
bool[] newArray;
if (minCapacity > oldCapacity)
{
int newCapacity = (oldCapacity * 3) / 2 + 1;
if (newCapacity < minCapacity) { newCapacity = minCapacity; }
newArray = new bool[newCapacity]; Array.Copy(array, 0,
newArray, 0, oldCapacity); } else { newArray = array; } return
newArray; } ///
Ensures that a given array can hold up to minCapacity
elements.
///
/// Returns the identical array if it can hold at least the
number of elements specified.
```

```
/// Otherwise, returns a new array with increased capacity
containing the same elements, ensuring
/// that it can hold at least the number of elements specified
by
/// the minimum capacity argument.
///
///

///  the desired minimum capacity.
///   public static char[] ensureCapacity(char[] array, int
minCapacity)
{
int oldCapacity = array.Length;
char[] newArray;
if (minCapacity > oldCapacity)
{
int newCapacity = (oldCapacity * 3) / 2 + 1;
if (newCapacity < minCapacity) { newCapacity = minCapacity; }
newArray = new char[newCapacity]; Array.Copy(array, 0,
newArray, 0, oldCapacity); } else { newArray = array; } return
newArray; } ///
Ensures that a given array can hold up to minCapacity
elements.
///
/// Returns the identical array if it can hold at least the
number of elements specified.
/// Otherwise, returns a new array with increased capacity
containing the same elements, ensuring
/// that it can hold at least the number of elements specified
by
/// the minimum capacity argument.
///
///

///  the desired minimum capacity.
///  public static double[] ensureCapacity(double[] array, int
minCapacity)
```

```
{
int oldCapacity = array.Length;
double[] newArray;
if (minCapacity > oldCapacity)
{
int newCapacity = (oldCapacity * 3) / 2 + 1;
if (newCapacity < minCapacity) { newCapacity = minCapacity; }
newArray = new double[newCapacity]; //for (int i =
oldCapacity; --i >= 0; ) newArray[i] = array[i];
Array.Copy(array, 0, newArray, 0, oldCapacity);
}
else
{
newArray = array;
}
return newArray;
}

///

Ensures that a given array can hold up to minCapacity
elements.
///
/// Returns the identical array if it can hold at least the
number of elements specified.
/// Otherwise, returns a new array with increased capacity
containing the same elements, ensuring
/// that it can hold at least the number of elements specified
by
/// the minimum capacity argument.
///
///
///  the desired minimum capacity.
///  public static float[] ensureCapacity(float[] array, int
minCapacity)
{
int oldCapacity = array.Length;
float[] newArray;
if (minCapacity > oldCapacity)
```

```
{
int newCapacity = (oldCapacity * 3) / 2 + 1;
if (newCapacity < minCapacity) { newCapacity = minCapacity; }
newArray = new float[newCapacity]; Array.Copy(array, 0,
newArray, 0, oldCapacity); } else { newArray = array; } return
newArray; } ///
Ensures that a given array can hold up to minCapacity
elements.
///
/// Returns the identical array if it can hold at least the
number of elements specified.
/// Otherwise, returns a new array with increased capacity
containing the same elements, ensuring
/// that it can hold at least the number of elements specified
by
/// the minimum capacity argument.
///
///

///  the desired minimum capacity.
///   public static int[] ensureCapacity(int[] array, int
minCapacity)
{
int oldCapacity = array.Length;
int[] newArray;
if (minCapacity > oldCapacity)
{
int newCapacity = (oldCapacity * 3) / 2 + 1;
if (newCapacity < minCapacity) { newCapacity = minCapacity; }
newArray = new int[newCapacity]; Array.Copy(array, 0,
newArray, 0, oldCapacity); } else { newArray = array; } return
newArray; } ///
Ensures that a given array can hold up to minCapacity
elements.
///
/// Returns the identical array if it can hold at least the
number of elements specified.
```

```
/// Otherwise, returns a new array with increased capacity
containing the same elements, ensuring
/// that it can hold at least the number of elements specified
by
/// the minimum capacity argument.
///
///

///  the desired minimum capacity.
///   public static long[] ensureCapacity(long[] array, int
minCapacity)
{
int oldCapacity = array.Length;
long[] newArray;
if (minCapacity > oldCapacity)
{
int newCapacity = (oldCapacity * 3) / 2 + 1;
if (newCapacity < minCapacity) { newCapacity = minCapacity; }
newArray = new long[newCapacity]; Array.Copy(array, 0,
newArray, 0, oldCapacity); } else { newArray = array; } return
newArray; } ///
Ensures that a given array can hold up to minCapacity
elements.
///
/// Returns the identical array if it can hold at least the
number of elements specified.
/// Otherwise, returns a new array with increased capacity
containing the same elements, ensuring
/// that it can hold at least the number of elements specified
by
/// the minimum capacity argument.
///
///

///  the desired minimum capacity.
///            public      static      System.Object[]
ensureCapacity(System.Object[] array, int minCapacity)
```

```
{
int oldCapacity = array.Length;
System.Object[] newArray;
if (minCapacity > oldCapacity)
{
int newCapacity = (oldCapacity * 3) / 2 + 1;
if (newCapacity < minCapacity) { newCapacity = minCapacity; }
newArray = new System.Object[newCapacity]; Array.Copy(array,
0, newArray, 0, oldCapacity); } else { newArray = array; }
return newArray; } } } [/csharp]
```