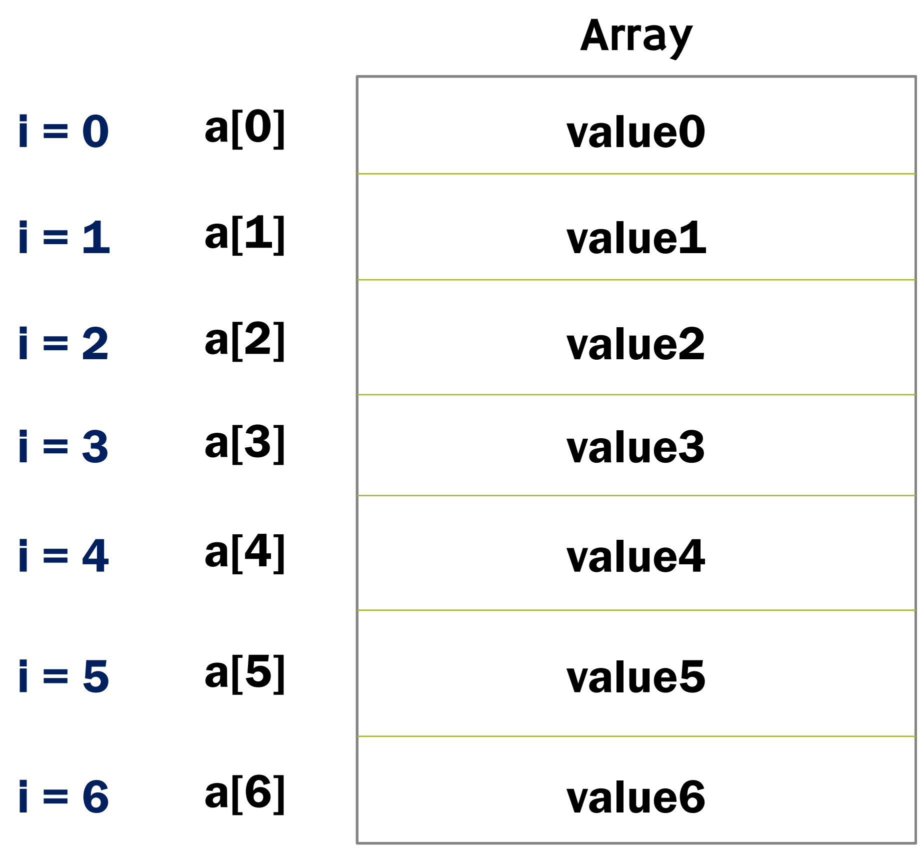
C#

System.Array class

What

- Every array is treated as an object for System.Array class.
- The System.Array class provides a set of properties and methods for every array.



Array

```
type[ ]  arrayReferenceVariableName = new  type[ size ];
```

Properties

- Length

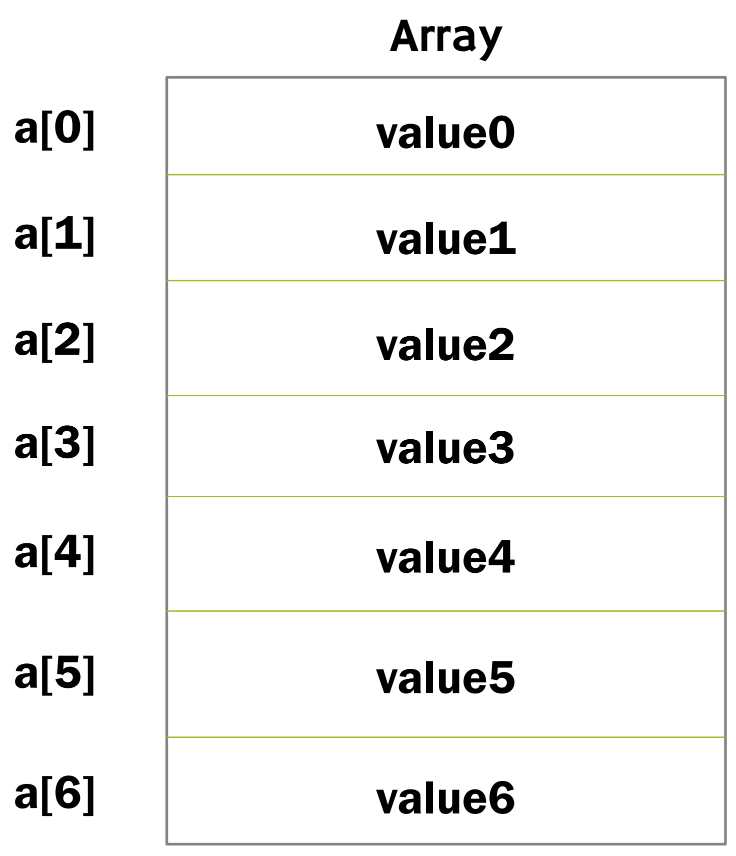
Methods

- IndexOf
- BinarySearch
- Clear
- Resize
- Sort
- Reverse
- CopyTo
- Clone

IndexOf()

IndexOf( )

- This method searches the array for the given value.
  - If the value is found, it returns its index.
  - If the value is not found, it returns -1.



Array - IndexOf() method

```
static int  Array.IndexOf( System.Array  array, object value)
```

Array - IndexOf() - Example

```
Array.IndexOf(array, value3) = 3
```

- The "IndexOf" method performs linear search. That means it searches all the elements of an array, until the search value is found. When the search value is found in the array, it stops searching and returns its index.
- The linear search has good performance, if the array is small. But if the array is larger, Binary search is recommended to improve the performance

Parameters

- **array:** This parameter represents the array, in which you want to search.
- **value:** This parameter represents the actual value that is to be searched.
- **startIndex:** This parameter represents the start index, from where the search should be started.

BinarySearch()

BinarySearch()

- This method searches the array for the given value.
  - If the value is found, it returns its index.
  - If the value is not found, it returns -1.

	Array
a[0]	value0
a[1]	value1
a[2]	value2
a[3]	value3
a[4]	value4
a[5]	value5
a[6]	value6

Array - BinarySearch() method
static int Array.BinarySearch(System.Array array, object value)

Array - BinarySearch() - Example
Array.BinarySearch(array, value3) = 3

- The "Binary Search" requires an array, which is already sorted.
  - On unsorted arrays, binary search is not possible.
- It directly goes to the middle of the array (array size / 2), and checks that item is less than / greater than the search value.
- If that item is greater than the search value, it searches only in the first half of the array.
- If that item is less than the search value, it searches only in the second half of the array.
- Thus it searches only half of the array. So in this way, it saves performance

Parameters

- **array:** This parameter represents the array, in which you want to search.
- **value:** This parameter represents the actual value that is to be searched

Clear()

Clear()

- This method starts with the given index and sets all the “length” no. of elements to zero (0).

	Array
a[0]	value0
a[1]	value1
a[2]	0
a[3]	0
a[4]	0
a[5]	value5
a[6]	value6

Array - Clear() method

```
static void  Array.Clear( System.Array  array, int index, int length)
```

Clear() - Example

```
Array.Clear(a, 2, 3)
```

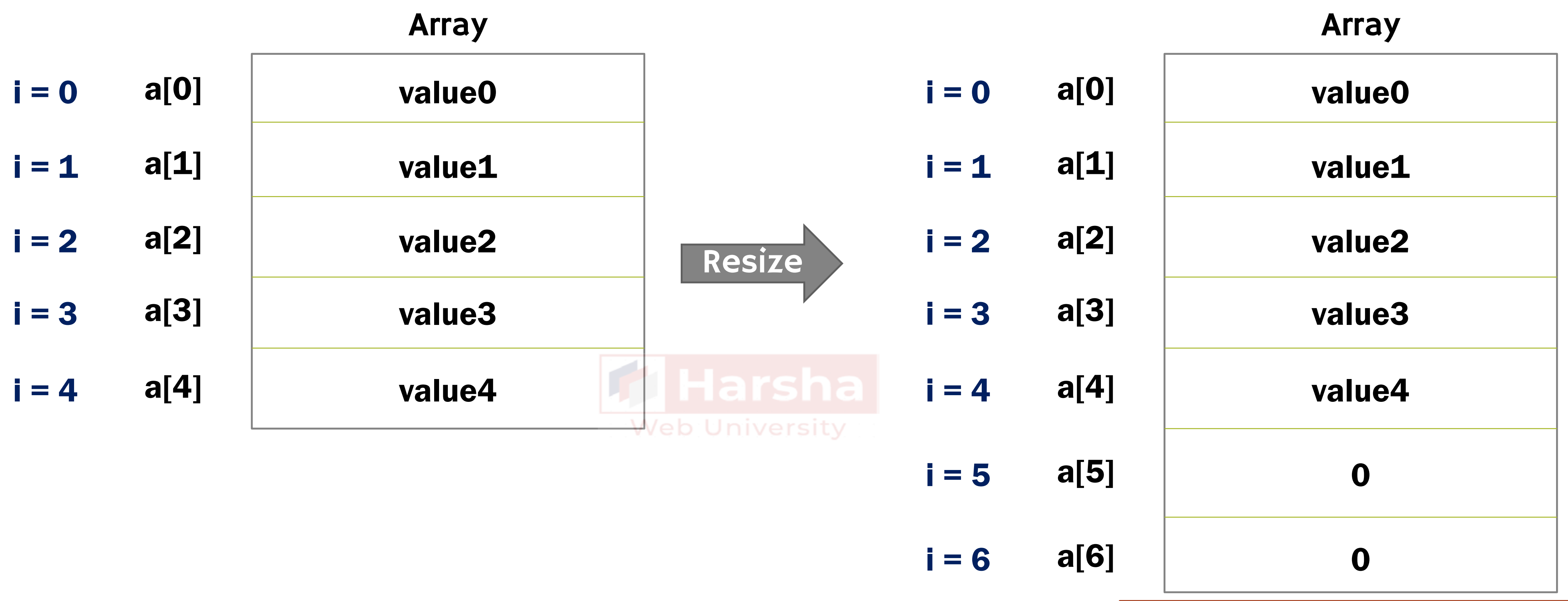
Parameters

- **array:** This parameter represents the array, in which you want to clear the elements.
- **index:** This parameter represents the index, from which clearing process is to be started.
- **length:** This parameter represents the no. of elements that are to be cleared.

Resize()

Resize()

› This method increases / decreases size of the array.



Array - Resize() method
static void Array.Resize(ref System.Array array, int newSize)

Array - Resize() - Example
Array.Resize(a, 6)

Parameters

› array: This parameter represents the array, which you want to resize.

› newSize: This parameter represents the new size of the array, how many elements you want to store in the array. It can be less than or greater than the current size.

Sort()

Sort() › This method sorts the array in ascending order.

		Array
i = 0	a[0]	100
i = 1	a[1]	950
i = 2	a[2]	345
i = 3	a[3]	778
i = 4	a[4]	20
i = 5	a[5]	800
i = 6	a[6]	80

Sort →

		Array
i = 0	a[0]	20
i = 1	a[1]	80
i = 2	a[2]	100
i = 3	a[3]	345
i = 4	a[4]	778
i = 5	a[5]	800
i = 6	a[6]	950

Array - Sort() method

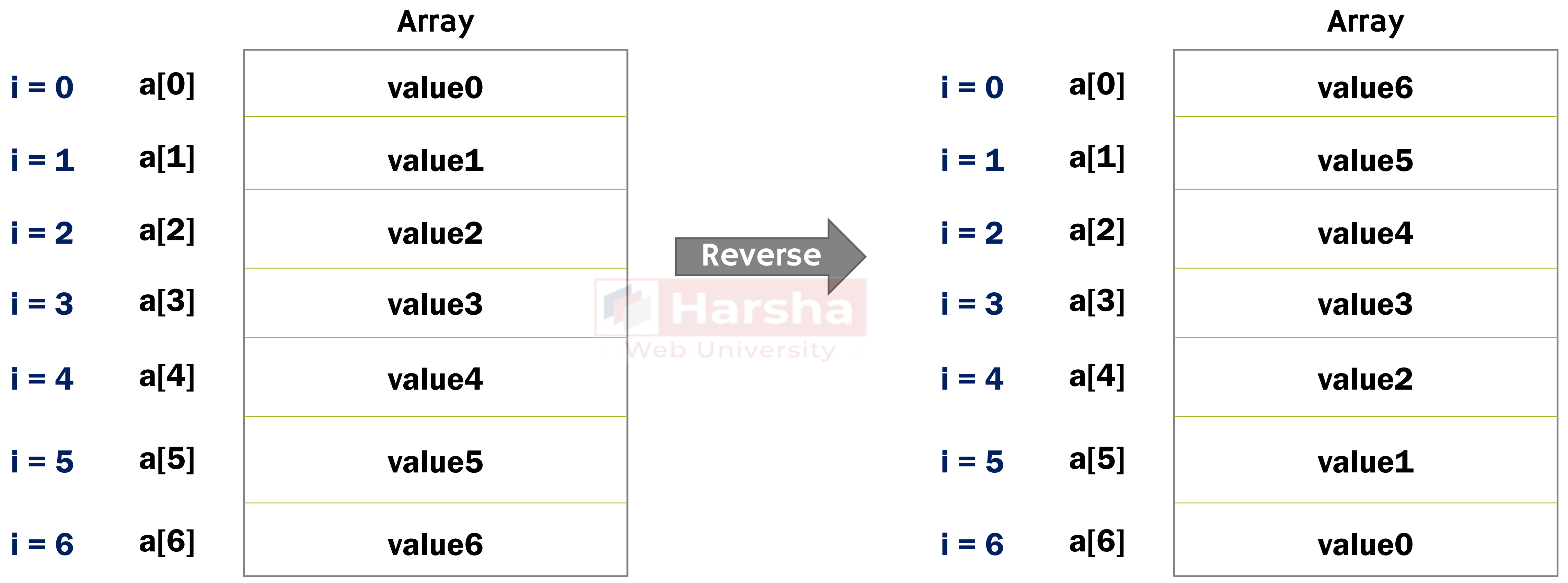
```
static void  System.Array.Sort( System.Array  array )
```

Array - Sort() - Example

```
Array.Sort(a)
```

Reverse()

Reverse() › This method reverses the array.



Array - Reverse() method

```
static void  System.Array.Reverse( System.Array  array )
```

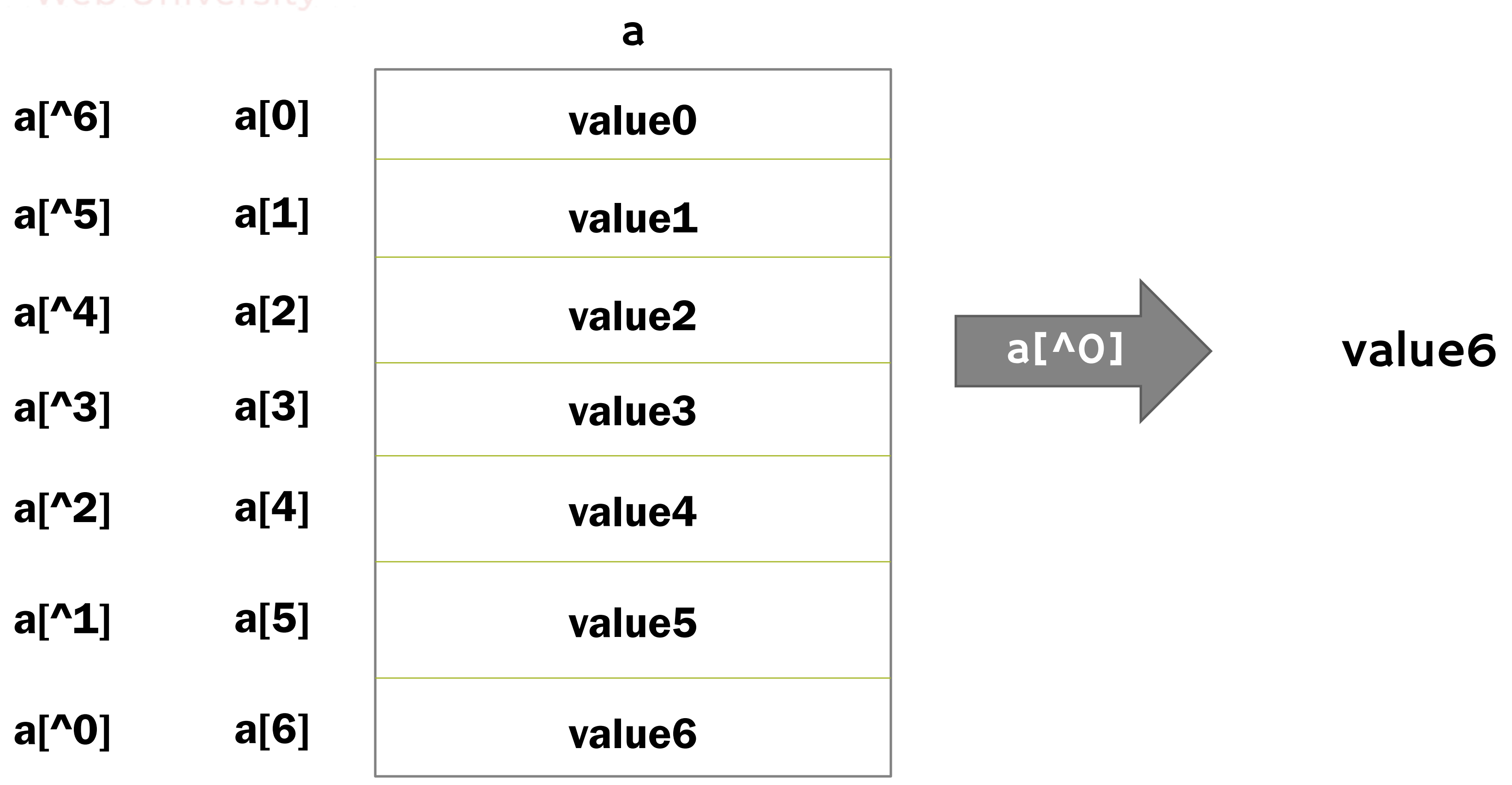
Array - Reverse() - Example

```
Array.Reverse(a)
```

Index-From-End operator and Ranges

^

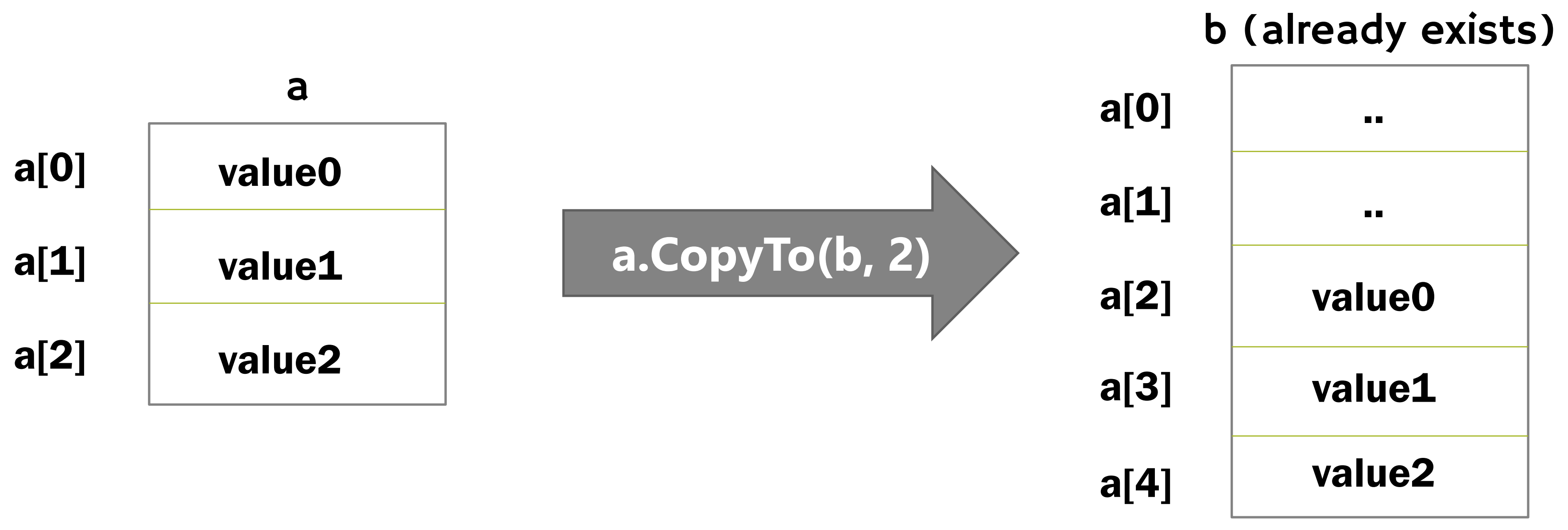
- This operator returns the index from end of the array (last element is treated as index '0').



CopyTo()

CopyTo()

- This method copies (shallow copy) all the elements from source array to destination array, starting from the specified 'startIndex'.



Array - CopyTo() method
void Array.CopyTo(System.Array destinationArray, int startIndex)

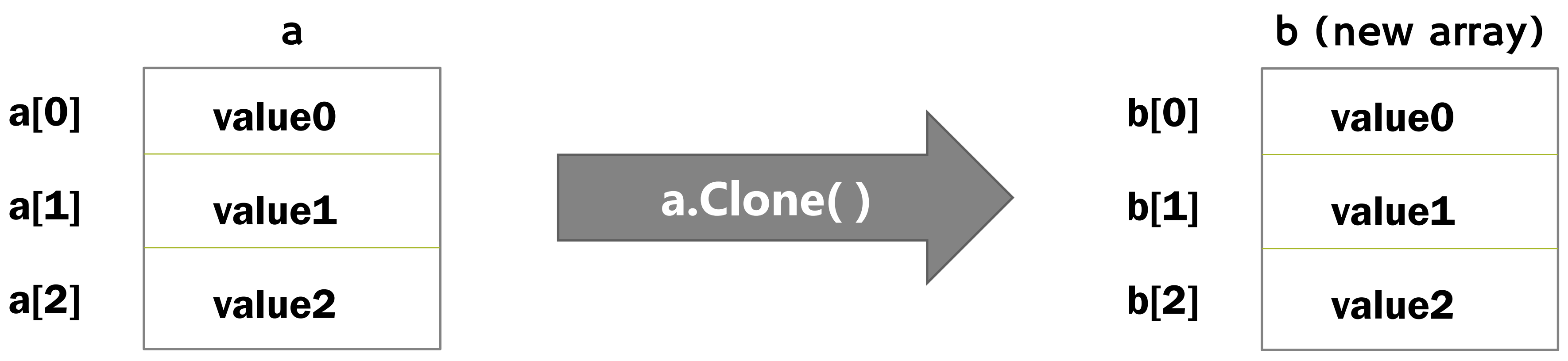
Parameters

- **sourceArray:** This parameter represents the array, which array you want to copy.
- **destinationArray:** This parameter represents the array, into which you want to copy the elements of source array. The destination array must exist already and should be large enough to hold new values.
- **startIndex:** This parameter represents the index of the element, from which you want to start copying.

Clone()

Clone()

› This method creates a new array & copies (shallow copy) all the elements from source array to the new destination array.



Array - Clone() method

```
object System.Array.Clone()
```

Array.CopyTo()

CopyTo() equires to have an existing destination array; and the destination array should be large enough to hold all elements from the source array, starting from the specified startIndex.

CopyTo() allows you to specify the startIndex at destination array.

The result array need not be type-casted explicitly.

Array.Clone()

Clone() creates a new destination array; you need not have an existing array.

Clone() doesn't allow you to specify the startIndex at destination array.

The result array will be returned as 'object' type; so need to be type-casted to array type.