**public** **static** **int** BinarySearch(Array a, object v)

1. Returns the index of the first match.
2. If v is not found, returns a negative value.
3. The array must be sorted and one-dimensional.

**public** **static** **int** BinarySearch(Array a, object v, IComparer comp)

1. Searches the array a for v, using the comparison method comp.
2. Returns the index of the first match.
3. If v is not found, returns a negative value.
4. The array must be sorted and one-dimensional.

**public** **static** **int** BinarySearch(Array a, **int** start,**int** count, object v)

1. Searches a portion of the array a for the value v.
2. The search begins at the index 'start' and is restricted to 'count' elements.
3. Returns the index of the first match.
4. If v is not found, returns a negative value.
5. The array must be sorted and one-dimensional.

**public** **static** **int** BinarySearch(Array a, **int** start,**int** count, object v,IComparer comp)

1. Searches a portion of the array a for the value v, using the comparison method comp.
2. The search begins at the index start and is restricted to count elements.
3. Returns the index of the first match.
4. If v is not found, returns a negative value.
5. The array must be sorted and one-dimensional.

**public** **static** **void** Clear(Array a, **int** start, **int** count)

1. Sets the elements of a to zero.
2. The elements begin at the index specified by start and the length is count.

**public** virtual object Clone()

1. Clone this array.
2. The both copy refer to the same elements.
3. This is called a "shallow copy."
4. Changes to the elements affect both arrays.

**public** **static** **void** Copy(Array source,Array dest, **int** count)

1. Beginning at the start of each array, copies count elements from source to dest.
2. Copy() makes a "shallow copy".
3. Both arrays will refer to the same reference type elements.

**public** **static** **void** Copy(Array source,**int** srcStart,Array dest,**int** destStart, **int** count)

1. Copies count elements from source[srcStart] to dest[destStart].
2. Copy() makes a "shallow copy".
3. Both arrays will refer to the same reference type elements.

**public** virtual **void** CopyTo(Array dest, **int** start)

1. Copies the elements of the invoking array to dest, beginning at dest[start].

**public** **static** Array CreateInstance(Type t, **int** size)

1. Returns a one-dimensional array that contains size elements of type t.

**public** **static** Array CreateInstance(Type t, **int** size1, **int** size2)

1. Returns a size1-by-size2 two-dimensional array.
2. Each element is of type t.

**public** **static** Array CreateInstance(Type t, **int** size1,**int** size2, **int** size3)

1. Returns a size1-by-size2-by-size3 three-dimensional array.
2. Each element is of type t.

**public** **static** Array CreateInstance(Type t, **int**[ ] sizes)

1. Returns a multi-dimensional array.
2. The dimensions is specified in sizes.
3. Each element is of type t.

**public** **static** Array CreateInstance(Type t, **int**[ ] sizes,**int**[ ] startIndexes)

1. Returns a multi-dimensional array.
2. The dimensions is specified in sizes.
3. Each element is of type t.
4. The starting index of each dimension is startIndexes.
5. It is possible to create arrays that begin at some index other than zero.

**public** override bool Equals(object v)

Returns true if the value of the invoking object equals the value of v.

**public** virtual IEnumerator GetEnumerator()

1. Returns an enumerator object for the array.
2. An enumerator enables you to cycle through an array.

**public** **int** GetLength(**int** dim)

1. Returns the length at the specified dimension.
2. The dimension is zero-based.
3. To get the length of the first dimension, pass 0.
4. To obtain the length of the second dimension, pass 1.

**public** **int** GetLowerBound(**int** dim)

1. Returns the first index of the specified dimension, which is usually zero.
2. The parameter dim is zero-based.
3. To get the start index of the first dimension, pass 0.
4. To obtain the start index of the second dimension, pass 1.

**public** override **int** GetHashCode()

1. Returns the hash code for the invoking object.

**public** TypeCode GetTypeCode()

1. Returns the TypeCode enumeration value for Array, which is TypeCode.Array.

**public** **int** GetUpperBound(**int** dim)

1. Returns the last index of the specified dimension.
2. The parameter dim is zero-based.
3. To get the last index of the first dimension, pass 0.
4. To obtain the last index of the second dimension, pass 1.

**public** object GetValue(**int** idx)

1. Returns the value of the element at index idx.
2. The array must be one-dimensional.

**public** object GetValue(**int** idx1, **int** idx2)

1. Returns the value of the element at [idx1, idx2].
2. The array must be two-dimensional.

**public** object GetValue(**int** idx1, **int** idx2,**int** idx3)

1. Returns the value of the element at [idx1, idx2, idx3].
2. The array must be three- dimensional.

**public** object GetValue(**int**[ ] idxs)

1. Returns the value of the element at the specified indices.
2. The array must have as many dimensions as idxs has elements.

**public** **static** **int** IndexOf(Array a, object v)

1. Returns the index of the first element within the one-dimensional array a that has the value specified by v.
2. Returns -1 if the value is not found.

**public** **static** **int** IndexOf(Array a, object v,**int** start)

1. Returns the index of the first element within the one-dimensional array a that has the value specified by v.
2. The search begins at a[start].
3. Returns -1 if the value is not found.

**public** **static** **int** IndexOf(Array a, object v,**int** start, **int** count)

1. Returns the index of the first element within the one-dimensional array a that has the value specified by v.
2. The search begins at a[start].
3. The search runs for count elements.
4. Returns -1 if the value is not found.

**public** **void** Initialize()

1. Initializes each element in the invoking array.
2. It calls the element's default constructor.
3. This method can be used only on arrays of value types.

**public** **static** **int** LastIndexOf(Array a, object v)

1. Returns the index of the last element within the one-dimensional array a that has the value specified by v.
2. Returns -1 if the value is not found.

**public** **static** **int** LastIndexOf(Array a, object v,**int** start)

1. The search proceeds in reverse order, beginning at a[start] and stopping at a[0].
2. Returns -1 if the value is not found.

**public** **static** **int** LastIndexOf(Array a, object v,**int** start, **int** count)

1. The search proceeds in reverse order, beginning at a[start] and running for count elements.
2. Returns -1 if the value is not found within the specified range.

**public** **static** **void** Reverse(Array a)

1. Reverses the elements in a.

**public** **static** **void** Reverse(Array a, **int** start,**int** count)

1. Reverses a range of elements in a.
2. The range reversed begins at a[start] and runs for count elements.

**public** **void** SetValue(object v, **int** idx)

1. The array must be one-dimensional.

**public** **void** SetValue(object v, **int** idx1, **int** idx2)

1. Sets the value at indices [idx1, idx2].
2. The array must be two-dimensional.

**public** **void** SetValue(object v, **int** idx1,**int** idx2, **int** idx3)

1. Sets the value at indices [idx1, idx2, idx3].
2. The array must be three-dimensional.

**public** **void** SetValue(object v, **int**[ ] idxs)

1. Sets the value of the element at the specified indices within the invoking array to v.
2. The array must have as many dimensions as idxs has elements.

**public** **static** **void** Sort(Array a)

1. Sorts a into ascending order.
2. The array must be one-dimensional.

**public** **static** **void** Sort(Array a,IComparer comp)

1. The array must be one-dimensional.

**public** **static** **void** Sort(Array k, Array v)

1. Sorts a pair of one-dimensional arrays into ascending order.
2. The k array contains the sort keys.
3. The v array contains the values linked to those keys.
4. The two arrays contain key/value pairs.
5. After the sort, both arrays are in ascending key order.

**public** **static** **void** Sort(Array k, Array v,IComparer comp)

1. Sorts a pair of one-dimensional arrays into ascending order using the comparison method specified by comp.
2. The k array contains the sort keys.
3. The v array contains the values linked to those keys.
4. The two arrays contain key/value pairs.
5. After the sort, both arrays are in ascending key order.

**public** **static** **void** Sort(Array a, **int** start,**int** count)

1. Sorts a range of a into ascending order.
2. The range begins at a[start] and runs for count elements.
3. The array must be one-dimensional.

**public** **static** **void** Sort(Array a, **int** start,**int** count,IComparer comp)

1. Sorts a range of a into ascending order using the comparison method specified by comp.
2. The range begins at a[start] and runs for count elements.
3. The array must be one-dimensional.

**public** **static** **void** Sort(Array k, Array v,**int** start, **int** count)

1. Sorts a range within a pair of one-dimensional arrays into ascending order.
2. Within both arrays, the range to sort begins at the index passed in start and runs for count elements.
3. The k array contains the sort keys.
4. The v array contains the values linked to those keys.
5. The two arrays contain key/value pairs.
6. After the sort, both ranges are in ascending-key order.

**public** **static** **void** Sort(Array k, Array v,**int** start, **int** count,IComparer comp)

1. Sorts a range within a pair of one- dimensional arrays into ascending order using the comparison method specified by comp.
2. Within both arrays, the range to sort begins at the index passed in start and runs for count elements.
3. The k array contains the sort keys.
4. The v array contains the values linked to those keys.
5. The two arrays contain key/value pairs.
6. After the sort, both ranges are in ascending-key order.

|  |  |
| --- | --- |
| **Property** | **Meaning** |
| public virtual bool IsFixedSize | Is the array fixed size or dynamic.(read-only property) |
| public virtual bool IsReadOnly | Is the array read-only. (read-only property). |
| public virtual bool IsSynchronized | Is the array safe for use in a multithreaded environment. (read-only property) |
| public int Length | The number of elements in the array. (read-only property) |
| public int Rank | The dimensions in the array. (read-only property) |
| public virtual object SyncRoot | A read-only property that contains the object that must be used to synchronize access to the array. |