## Add(T) Complexity

Complexity is **O(n)** because on every add operation we make a new array and have to copy all old elements => thus n operations.

## Remove(index) Complexity – Worst Case

Complexity in all cases (worst, best and average) is **O(n)** as we’ll need to transfer n-1 elements regardless. Also the method doesn’t have validation for the given index, leaving possibility for IndexOutOfRangeExceptions.

## Remove(index) Complexity – Best Case

Complexity in all cases (worst, best and average) is **O(n)** as we’ll need to transfer n-1 elements regardless. Also the method doesn’t have validation for the given index, leaving possibility for IndexOutOfRangeExceptions.

## Remove(index) Complexity – Average Case

Complexity in all cases (worst, best and average) is **O(n)** as we’ll need to transfer n-1 elements regardless. Also the method doesn’t have validation for the given index, leaving possibility for IndexOutOfRangeExceptions.

## RemoveFirst(T) Complexity

Same as the above **O(n)** as the actual implementation actually calls the **Remove** method.

## RemoveLast(T) Complexity

Same as the above **O(n)** as the actual implementation actually calls the **Remove** method.

## Length Complexity

The complexity is **O(1)** as it calls the underlying C# implementation and the operation is simplified to a simple lookup with a constant value.

## This[index] Complexity

Same as the above – the method uses the underlying C# implementation which has a complexity of **O(1)**.

## First Complexity

Again uses the underlying C# index implementation which makes the complexity **O(1)**.

## Last Complexity

Again uses the underlying C# index implementation which makes the complexity **O(1)**, the in-built Length property also has a complexity of **O(1)** as stated above.