📊 SECTION 3: Summary Table

Saved memory full ⓘ

Share

Structure	Access	Insert	Delete	Ordered?	Use When	Avoid When
Array	O(1)	O(n)	O(n)	❌	Fixed size, fast access	Need dynamic resizing
List<T>	O(1)	O(1)/O(n)	O(n)	❌	Dynamic size, index access	Frequent insert/delete in middle
LinkedList<T>	O(n)	O(1)*	O(1)*	❌	Frequent middle insert/delete (with ref)	Random access
Stack<T>	O(1)	O(1)	O(1)	❌	LIFO logic: recursion, undo	Need random access
Queue<T>	O(1)	O(1)	O(1)	❌	FIFO: BFS, task queues	Random access
Dictionary<TKey,V>	-	O(1)	O(1)	❌	Key-based access	Need order
HashSet<T>	-	O(1)	O(1)	❌	Unique values	Need duplicates or sorted order
SortedDictionary	-	O(log n)	O(log n)	✅	Sorted key-value lookups	Speed is more important than order
SortedList	O(log n)	O(n)	O(n)	✅	Few writes, sorted data	Frequent insert/delete
Tree (Custom)	O(log n)	O(log n)	O(log n)	Depends	Prefix/range problems	Basic lookups
Graph (Custom)	Depends	Depends	Depends	❌	Paths, cycles, networking problems	Non-relational tasks

◆ String Methods, Examples & Outputs

STRING

Operation	Example	Description	Code Snippet	Output
`Length`	`str.Length`	Gets number of characters	`string str = "hello";` `Console.WriteLine(str.Length);`	`5`
Access character	`str[0]`	Get character at index	`string str = "hello";` `Console.WriteLine(str[0]);`	`h`
`ToUpper / ToLower`	`str.ToUpper()`	Convert case	`string str = "Hello";` `Console.WriteLine(str.ToUpper());`	`HELLO`
	`str.ToLower()`		`Console.WriteLine(str.ToLower());`	`hello`
`Substring`	`str.Substring(1, 3)`	Extract part of string	`string str = "hello";` `Console.WriteLine(str.Substring(1, 3));`	`ell`
`IndexOf / LastIndexOf`	`str.IndexOf("l")`	Index of first match	`string str = "hello";` `Console.WriteLine(str.IndexOf('l'));`	`2`

	`str.LastIndexOf("l")`	Index of last match	`Console.WriteLine(str.LastIndexOf('l'));`	`3`

Saved memory full ⓘ

`Contains`	`str.Contains("ell")`	Checks if contains substring	`string str = "hello"; Console.WriteLine(str.Contains("ell"));`	`True`
`Replace`	`str.Replace("l", "x")`	Replace characters	`string str = "hello"; Console.WriteLine(str.Replace("l", "x"));`	`hexxo`
`Split`	`str.Split(',')`	Split into array	`string str = "a,b,c"; string[] parts = str.Split(','); Console.WriteLine(parts[1]);`	`b`
`Trim`	`str.Trim()`	Remove whitespace	`string str = " hello "; Console.WriteLine(str.Trim());`	`hello`
`StartsWith` / `EndsWith`	`str.StartsWith("he")`	Check prefix/suffix	`string str = "hello"; Console.WriteLine(str.StartsWith("he"));`	`True`
	`str.EndsWith("lo")`		`Console.WriteLine(str.EndsWith("lo"));`	`True`
`Equals`	`str.Equals("hello")`	Compare equality	`string str = "hello"; Console.WriteLine(str.Equals("hello"));`	`True`

```
int[] numbers = new int[5];            // declare array of size 5
int[] scores = new int[] { 10, 20, 30 }; // initialized array
string[] names = { "Alice", "Bob" };     // shorthand initialization
```

🔹 1.2 Common Methods & Properties of Arrays

Operation	Syntax / Example	Description
`Length`	`arr.Length`	Total number of elements
`Indexing`	`arr[0]`	Access element at index
`Update`	`arr[2] = 99;`	Change value at index
`Looping`	`foreach (int x in arr)`	Iterate over array
`Array.Sort`	`Array.Sort(arr)`	Sort in ascending order
`Array.Reverse`	`Array.Reverse(arr)`	Reverse elements

3.2 Common Methods & Properties of List

Operation	Example	Description	Code Snippet	Output
`Add`	`list.Add(5)`	Add element to end	`var list = new List<int>();` `list.Add(5);` `Console.WriteLine(list[0]);`	`5`
`AddRange`	`list.AddRange(new[]{1,2})`	Add multiple elements	`list.AddRange(new[]{1, 2});`	`[1, 2]`
`Insert`	`list.Insert(1, 100)`	Insert at index	`list.Insert(1, 100);`	`[5, 100, 1, 2]`
`Remove`	`list.Remove(100)`	Remove first occurrence	`list.Remove(100);`	`[5, 1, 2]`
`RemoveAt`	`list.RemoveAt(0)`	Remove at index	`list.RemoveAt(0);`	`[1, 2]`
`Contains`	`list.Contains(2)`	Check if list contains element	`Console.WriteLine(list.Contains(2));`	`True`
`IndexOf`	`list.IndexOf(2)`	Get index of element	`Console.WriteLine(list.IndexOf(2));`	`1`
`Count`	`list.Count`	Get number of elements	`Console.WriteLine(list.Count);`	`2`
`Sort`	`list.Sort()`	Sort list ascending	`list.Sort();`	`[1, 2, 5]`
`Reverse`	`list.Reverse()`	Reverse the list	`list.Reverse();`	`[5, 2, 1]`
`Clear`	`list.Clear()`	Remove all elements	`list.Clear();`	`[]` (empty list)
`ToArray`	`list.ToArray()`	Convert to array	`int[] arr = list.ToArray();`	`Same elements as list`

Saved memory full

🔹 4.2 Common Methods & Properties of Dictionary

Dictionary<string, int> ageMap = new Dictionary<string, int>();

Operation	Example	Description	Code Snippet	Output
`Add`	`dict.Add("Alice", 25)`	Add key-value pair	`var dict = new Dictionary<string, int>(); dict.Add("Alice", 25);`	`{"Alice": 25}`
`[]` indexer	`dict["Bob"] = 30`	Add/update value by key	`dict["Bob"] = 30;`	`{"Alice":25, "Bob":30}`
`Remove`	`dict.Remove("Alice")`	Remove key-value by key	`dict.Remove("Alice");`	`{"Bob":30}`
`ContainsKey`	`dict.ContainsKey("Alice")`	Check if key exists	`Console.WriteLine(dict.ContainsKey("Alice"));`	`False`
`ContainsValue`	`dict.ContainsValue(30)`	Check if value exists	`Console.WriteLine(dict.ContainsValue(30));`	`True`
`Count`	`dict.Count`	Number of key-value pairs	`Console.WriteLine(dict.Count);`	`1`
`Keys`	`dict.Keys`	All keys	`foreach (var key in dict.Keys) Console.WriteLine(key);`	`Bob`
`Values`	`dict.Values`	All values	`foreach (var val in dict.Values) Console.WriteLine(val);`	`30`
`TryGetValue`	`dict.TryGetValue("Alice", out var age)`	Safe get without exception	`bool found = dict.TryGetValue("Alice", out var age); Console.WriteLine(found);`	`False`
`Clear`	`dict.Clear()`	Remove all key-value pairs	`dict.Clear();`	`{}` (empty dictionary)

🔹 7.2 Common HashSet Methods & Properties of Hashset

Operation	Example	Description	Code Snippet	Output
`Add`	`set.Add(10)`	Adds a value if not present	`set.Add(10);` `set.Add(10);`	Set: `{10}` (no duplicates)
`Contains`	`set.Contains(10)`	Checks if value exists	`Console.WriteLine(set.Contains(10));`	`True`
`Remove`	`set.Remove(10)`	Removes a value if exists	`set.Remove(10);`	Removes 10 from set
`Count`	`set.Count`	Number of unique items in set	`Console.WriteLine(set.Count);`	`0` or more
`Clear`	`set.Clear()`	Removes all elements	`set.Clear();`	Empty set
`SetEquals`	`set1.SetEquals(set2)`	Check if two sets have same values	`Console.WriteLine(set1.SetEquals(set2));`	`True` or `False`
`UnionWith`	`set1.UnionWith(set2)`	Merge sets (no duplicates)	`set1.UnionWith(set2);`	Combined unique set
`IntersectWith`	`set1.IntersectWith(s2)`	Keep only common elements	`set1.IntersectWith(set2);`	Set with common values
`ExceptWith`	`set1.ExceptWith(set2)`	Remove values that exist in another set	`set1.ExceptWith(set2);`	Only unique to set1
`ToArray`	`set.ToArray()`	Convert set to array	`var arr = set.ToArray();`	Array from set

HashSet<int> set = new HashSet<int>();

🔹 5.2 Common Stack Methods & Properties

Operation	Example	Description	Code Snippet	Output
`Push`	`stack.Push(10)`	Add item to the top of the stack	`stack.Push(10);`	Stack: [10]
`Pop`	`stack.Pop()`	Remove and return top item	`int x = stack.Pop();` `Console.WriteLine(x);`	`10`
`Peek`	`stack.Peek()`	View top item without removing	`stack.Push(5);` `Console.WriteLine(stack.Peek());`	`5`
`Count`	`stack.Count`	Number of elements in the stack	`Console.WriteLine(stack.Count);`	`1`
`Contains`	`stack.Contains(5)`	Check if value exists in stack	`Console.WriteLine(stack.Contains(5));`	`True`
`Clear`	`stack.Clear()`	Remove all elements	`stack.Clear();`	Stack is empty
`ToArray`	`stack.ToArray()`	Convert to array	`var arr = stack.ToArray();` `Console.WriteLine(arr[0]);`	Top of stack

🔹 6.2 Common Queue Methods & Properties

Operation	Example	Description	Code Snippet	Output
`Enqueue`	`queue.Enqueue("A")`	Add to end of queue	`queue.Enqueue("A");`	Queue: ["A"]
`Dequeue`	`queue.Dequeue()`	Remove and return front item	`Console.WriteLine(queue.Dequeue());`	`"A"`
`Peek`	`queue.Peek()`	View front item without removing	`Console.WriteLine(queue.Peek());`	`"B"`
`Count`	`queue.Count`	Number of elements in queue	`Console.WriteLine(queue.Count);`	`1`
`Contains`	`queue.Contains("X")`	Check if item exists in queue	`Console.WriteLine(queue.Contains("X"));`	`True` or `False`
`Clear`	`queue.Clear()`	Removes all elements	`queue.Clear();`	Queue is empty
`ToArray`	`queue.ToArray()`	Converts queue to array	`var arr = queue.ToArray();` `Console.WriteLine(arr[0]);`	First item in queue

```
LinkedList<int> list = new LinkedList<int>();
```

> Namespace needed: `using System.Collections.Generic;`

🔹 8.2 LinkedList Methods & Properties

Operation	Example	Description	Code Snippet	Output
`AddFirst`	`list.AddFirst(10)`	Add item to beginning	`list.AddFirst(10);`	List: 10 →
`AddLast`	`list.AddLast(20)`	Add item to end	`list.AddLast(20);`	List: 10 → 20
`RemoveFirst`	`list.RemoveFirst()`	Remove first item	`list.RemoveFirst();`	List: 20
`RemoveLast`	`list.RemoveLast()`	Remove last item	`list.RemoveLast();`	List becomes empty
`First`	`list.First`	Reference to first node	`Console.WriteLine(list.First.Value);`	Prints value of first node
`Last`	`list.Last`	Reference to last node	`Console.WriteLine(list.Last.Value);`	Prints value of last node

`AddBefore`	`list.AddBefore(node, value)`	Insert before specific node	`list.AddBefore(list.First, 5);`	List: 5 → 10
`AddAfter`	`list.AddAfter(node, value)`	Insert after specific node	`list.AddAfter(list.First, 15);`	List: 10 → 15
`Remove(value)`	`list.Remove(10)`	Remove specific value	`list.Remove(10);`	Removes first occurrence of 10
`Contains(value)`	`list.Contains(10)`	Checks if value exists	`Console.WriteLine(list.Contains(10));`	`True` or `False`
`Count`	`list.Count`	Number of items in list	`Console.WriteLine(list.Count);`	`0` or more