Searching and Sorting Data Structure	Importance of searching in data structures 1)How fast we can retrieve data ? 2)how to reduce time complexity ? 3)Foundations of database and indexes
	Types of searching algorithms (Linear vs Binary) 1)linear searching: ● one by one searching ● data need not be sorted ● simple but inefficient for large data ● public Linearsearch(int [] arr , int key) 2) Binear Search ● Efficient searching ● Required sorted data ● Below are the steps ○ Validate sorted input ○ iterative binary search ○ return index or -1 ○ public Binarysearch(int [] arr , int key)
	Types of sorting algorithms (Comparison-based vs Non-comparison-based)
	Bubble Sort, Selection Sort , Insertion Sort ● repeatedly swaps adjacent element ● Large element moves to end ○ Nested loop ○ Swap counter ○ Early exit optimization ○ inefficient but easy to understand Selection Sort : ● Select minimum element ● Place it in correct position ● reduced no of swaps ● minimum index tracking Insertion Sort: ● Building sorted list incrementally ● Shift elements instead of swapping

	• Key based insertion • Efficient for nearly sorted small data

	Efficiency comparison among Bubble, Selection, and Insertion Sort
	Real-world scenarios for each sorting algorithm
	
	Non Comparison based algorithm:
	• They don't compare values directly.
	• it is implemented on the basis of properties of data.
	  ○ Value range
	  ○ Digit Position
	  ○ Frequency counts.
	1. Counting sort :
	  a. We count the frequency of each value.
	  b. It works when input values are non-negative numbers.
	  c. It should be known for its small range.
	
	ex. [4, 2, 2, 8, 3, 3, 1]
	
	Index: 0 1 2 3 4 5 6 7 8
	Count: 0 1 2 2 1 0 0 0 1
	
	Output will be : [1, 2, 2, 3, 3, 4, 8]
	
	--- 	
	
	2. Radix Sort :
	  • It sorts the number digit by digit.
	  • starting from the least significant digit(LSD) or Most significant digit(MSD).
	Step 1: Find the maximum number to know the number of digits.
	Step 2: Sort the element by 1st digit(ones Place)
	Step 3: Sort by 2nd Digit(tens place)
	Step 4: Continue until digit is processed
	
	Ex.
	[170, 45, 75, 90, 802, 24, 2, 66]

	Ones digit → `[170, 90, 802, 2, 24, 45, 75, 66]` Tens digit → `[802, 2, 24, 45, 66, 170, 75, 90]` Hundreds digit → `[2, 24, 45, 66, 75, 90, 170, 802]`
Indexers and Attributes in C#	Indexer Difference between index and properties Attributes

Delegates & Reflection	
Delegates in C#.NET	Delegates Single cast Multi cast
Standard C# Features	Anonymous method Lambda expression Func Action Predicate Events INtroduction to Reflection and its practical use Generics Generic class Generic field Generic method Advantage of generics Threading Ref and Out keyword Async & Await Understanding Http client

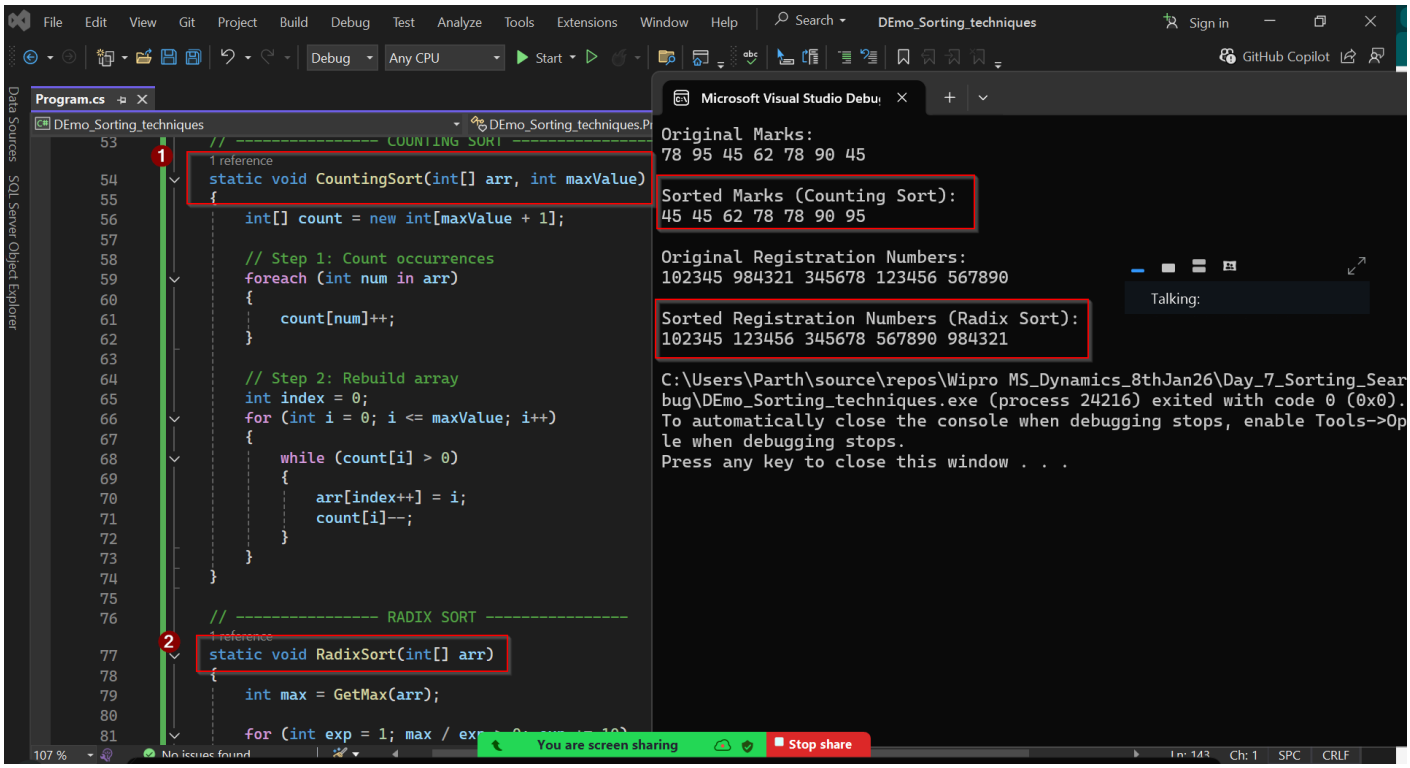
Algorithm	Best	Average	Worst	Stable

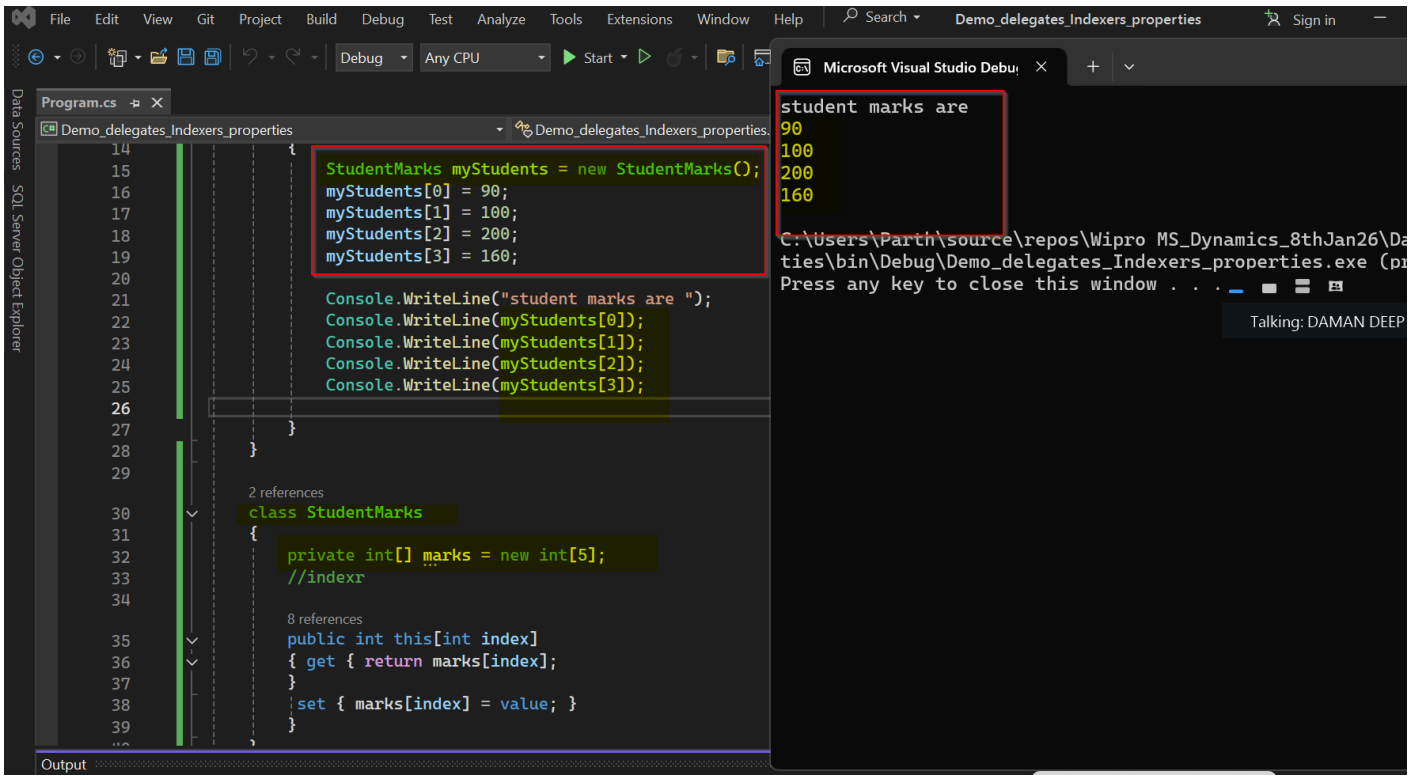
Bubble Sort	O(n)	$O(n^2)$	$O(n^2)$	Yes
Selection Sort	$O(n^2)$	$O(n^2)$	$O(n^2)$	No
Insertion Sort	O(n)	$O(n^2)$	$O(n^2)$	Yes

Algorithm	Real-world Scenario
Bubble Sort	Teaching concepts, very small lists
Selection Sort	Memory-constrained environments
Insertion Sort	Sorting live data streams

Algorithm	Best Case Time	Average Case Time	Worst Case Time	Space Complexity	Stable	In-Place
Counting Sort	**O(n + k)**	**O(n + k)**	**O(n + k)**	**O(k)**	✅ Yes	❌ No
Radix Sort	**O(d × (n + k))**	**O(d × (n + k))**	**O(d × (n + k))**	**O(n + k)**	✅ Yes	❌ No

Scenario	Better Choice	Reason
Student marks (0–100)	Counting Sort	Small fixed range
Employee IDs (6 digits)	Radix Sort	Large values, limited digits
Sparse large numbers	Radix Sort	Avoids huge count array
Frequency analysis	Counting Sort	Direct count usage





Event-Driven Order Processing System (C#)

User Story

As a backend developer,
I want to process orders using flexible, reusable functions and event-driven notifications,
So that business rules can change without rewriting core logic.

Business Scenario

An **E-Commerce Order Processing System** must:

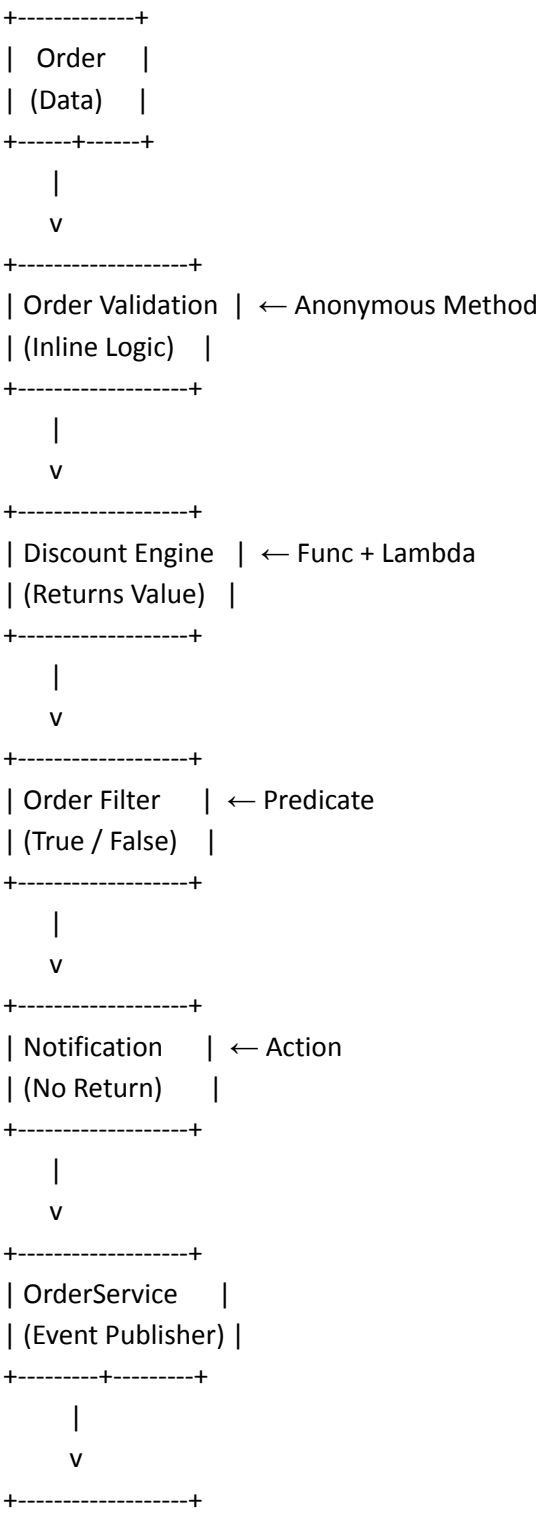
- Validate orders
- Apply dynamic discounts
- Notify stakeholders when orders are placed
- Filter orders based on business rules
- Execute different behaviors without creating multiple classes

---

Technical Mapping (Topics → Usage)

Topic	Application
Anonymous Method	Inline validation logic
Lambda Expression	Discount calculation
Func	Return computed order value
Action	Logging and notifications
Predicate	Order filtering

Events	Order placed notification



| Subscribers | ← Events
| (Email / Log) |
+-------------------+

Case Study: Generic Async Order Processing System

User Story

As a backend system developer,
I want to process different types of orders safely and efficiently using generics,
handle parallel operations using threading,
share and return values using ref & out,
and perform non-blocking operations using async & await,
so that the system is **scalable, type-safe, and high-performance**.

Business Scenario

An **Order Processing Platform** must:

- Handle **multiple data types** (int orders, string orders, custom objects)
- Process orders **in parallel**
- Modify shared values and return computed results
- Perform **I/O-like operations asynchronously**

Topic	Applied As
Generic Class	Generic order processor
Generic Field	Stores generic data

Generic Method	Processes any data type
Advantage of Generics	Type safety, reusability
Threading	Parallel order processing
ref	Modify existing value
out	Return additional computed value
async & await	Non-blocking operations