- **Benefit of generic sorting over non-generic:**

  It avoids code duplication, increases reusability, and ensures type safety without casting.

- **Lambda expressions in sorting:**

  They make code shorter, more readable, and allow defining custom sort logic inline.

- **Dynamic comparer function importance:**

  It allows flexibility to sort objects of different types or criteria without rewriting sorting logic.

- **IComparable<T> in derived classes:**

  Enables custom sorting rules specific to the class, allowing objects to be compared naturally.

- **Advantage of built-in delegates (Func):**

  Reduces boilerplate code, improves readability, and integrates seamlessly with LINQ and generic methods.

- **Anonymous vs lambda functions:**

  Anonymous functions are more verbose; lambdas are more concise and efficient for inline logic.

- **Benefit of generic Swap<T> method:**

  Increases code reusability and works with any type without rewriting swapping logic.

- **Multi-criteria sorting challenges & benefits:**

  Challenge: complexity in logic. Benefit: provides precise, flexible sorting for real-world scenarios.

- **Importance of default(T):**

  Ensures safe initialization of generics; returns 0 for value types and null for reference types.

- **Generic constraints (e.g., ICloneable):**

  Ensure only valid types are used, increasing safety, reliability, and preventing runtime errors.

- **Benefit of delegates for string transformations:**

  Enable reusable, flexible, and modular transformations in a functional programming style.

- **Delegates in mathematical operations:**

  Promote reusability by decoupling logic from implementation, allowing dynamic operation changes.

- **Advantages of generic delegates:**

  Provide flexibility to transform data structures without type restrictions, maximizing reusability.

- **Func simplification:**

  Removes the need for custom delegate declarations, making delegate usage quick and clean.

- **Why Action is preferred:**

  It clearly indicates that the method performs an operation but does not return a value.

- **Role of predicates:**

  Simplify filtering logic, enhance readability, and integrate well with LINQ and collection methods.

- **Anonymous functions in modularity:**

  Allow quick, localized customization without polluting code with extra named methods.

- **When to use anonymous functions:**

  When the logic is short, used only once, and doesn't justify creating a named method.

- **Importance of lambda expressions:**

  They enable concise, expressive, and functional-style programming in C#, widely used with LINQ.

- **Lambda in mathematical computations:**

  Makes operations more expressive, readable, and closer to mathematical notation.