Part01

1. Why is it better to code against an interface rather than a concrete class?
   Coding against an interface makes your code flexible, maintainable, and adaptable to change.

2. When should you prefer an abstract class over an interface?
   When you want to share common implementation not just head method.

3. How does implementing IComparable improve flexibility in sorting?
   You can use an already implemented method rather than creating it from scratch.

4. What is the primary purpose of a copy constructor in C#?
   To make a deep copy of the data of an object to another one.

5. How does explicit interface implementation help in resolving naming conflicts?
   You can use more than one interface that have the same method names.

6. What is the key difference between encapsulation in structs and classes?
   Struct is public by default but Class is private by default.

7. what is abstraction as a guideline, what's its relation with encapsulation?
   Guideline: Show only the essential features and hide unnecessary details.
   Encapsulation hides *internal implementation details* from direct access.
   Both improve security.

8. How does constructor overloading improve class usability?
   You can create object in different ways as your needs.

Part02

1. What we mean by coding against interface rather than class ? and if u get it so What we mean by code against abstraction not concreteness ?
   - Interface: only creating method head not full implementation to reuse in several classes to avoid repeating code.
   - Abstract: the same thing but you can do partial implementations to the methods.