1. What are the six combinations of access modifier keywords and what do they do?
public: A public type or member can be accessed from any code in the same assembly or in any other assembly that references the assembly where the public type or member is defined.

internal: An internal type or member can be accessed only from code in the same assembly as the internal type or member.

protected: A protected member can be accessed from code in the same class, a derived class, or any class in the same assembly.

protected internal: A protected internal member can be accessed from code in the same class, a derived class, any class in the same assembly, or any assembly that references the assembly where the protected internal member is defined.

private: A private member can be accessed only from code in the same class or struct where the private member is defined.

private protected: A private protected member can be accessed from code in the same class, a derived class in the same assembly

2.What is the difference between the static, const, and readonly keywords when applied to a type member?
static: A static member is associated with the type itself, rather than with any specific instance of the type.
const: A const member is a compile-time constant, which means its value is set at compile time and cannot be changed at runtime.
readonly: A readonly member is a runtime constant, which means its value is set at runtime and cannot be changed after it is set.

3. What does a constructor do?
Create an object and initialize class member.

4. Why is the partial keyword useful?
The partial keyword in C# is useful because it allows a class, struct, or interface to be split into multiple files, while still being treated as a single entity by the compiler.

5. What is a tuple?
A tuple is an ordered, immutable sequence of elements of different types in .NET languages such as C#.
In a tuple, each element has a unique index starting from 0. The elements can be of any type, and the type of each element can be different from the others.

6. What does the C# record keyword do?
The record keyword in C# is a feature introduced in C# 9.0 that provides a concise way to define classes that are primarily used for storing data. When you define a class with the record keyword, the C# compiler generates a lot of boilerplate code for you, such as property accessors, equality members, and a ToString() method.

7. What does overloading and overriding mean?
Overloading means multiple methods with the same name in a class, but different signature and functionality.
Override means refers to the ability to provide a new implementation for a method that is already defined in a super class or interface.

8. What is the difference between a field and a property?
A field is a variable that is declared inside a class or struct and directly holds a value. Fields are usually private or protected, meaning that they can only be accessed within the class or struct.
A property, on the other hand, is a method that is used to access and/or modify the value of a private field. Properties provide a way to expose the values of fields outside the class or struct, while still allowing the class to control how the values are accessed and modified.

9. How do you make a method parameter optional?
One can make a method parameter optional by specifying a default value for the parameter.

10. What is an interface and how is it different from abstract class?
Interface is a contract that defines a set of methods and/or properties that a class must implement.
An interface can only define abstract methods and properties, while an abstract class can define both abstract and non-abstract methods and properties.
A class can implement multiple interfaces but can only inherit from one abstract class.
An interface cannot contain any implementation code, while an abstract class can contain implementation code for non-abstract methods and properties.
An interface is used to define a set of methods and properties that a class must implement, while an abstract class is used to define a base set of functionality that derived classes can build upon.

11. What accessibility level are members of an interface?
All members of an interface are by default public, regardless of the access modifier used when declaring them.

12. **True**/False. Polymorphism allows derived classes to provide different implementations of the same method.

13. **True**/False. The override keyword is used to indicate that a method in a derived class is providing its own implementation of a method.
14. **True**/False. The new keyword is used to indicate that a method in a derived class is providing its own implementation of a method.
15. True/**False**. Abstract methods can be used in a normal (non-abstract) class.
16.**True**/False. Normal (non-abstract) methods can be used in an abstract class.
17. **True**/False. Derived classes can override methods that were virtual in the base class.
18. **True**/False. Derived classes can override methods that were abstract in the base class.
19. True/**False**. In a derived class, you can override a method that was neither virtual or abstract in the base class.
20. True/**False**. A class that implements an interface does not have to provide an implementation for all of the members of the interface.
21. **True**/False. A class that implements an interface is allowed to have other members that aren't defined in the interface.
22. True/**False**. A class can have more than one base class.
23. **True**/False. A class can implement more than one interface. What is meant by the terms managed resource and unmanaged resource in .NET
24. What's the purpose of Garbage Collector in .NET?
The purpose of the Garbage Collector (GC) in .NET is to automatically manage memory in a .NET application.

Leetcode 15. 3Sum

```
public class Solution {
    IList<IList<int>> res = new List<IList<int>>();
    public IList<IList<int>> ThreeSum(int[] nums) {
        Array.Sort(nums);
        for(int i = 0; i < nums.Length-1; i++) {
            if(i != 0 && nums[i] == nums[i-1]) continue;
            TwoSum(nums, i);
        }
        return res;
    }

    private void TwoSum(int[] nums, int index) {

        Dictionary<int, int> map = new Dictionary<int, int>();
        for(int i = index+1; i < nums.Length; i++) {
            if(map.ContainsKey(0 - nums[index] - nums[i])) {
                res.Add(new int[]{nums[index], 0 - nums[index] - nums[i], nums[i]});
                map.Remove(0 - nums[index] - nums[i]);
            }
            if(i > index+1 && nums[i] == nums[i-1]) {
                continue;
            }
            if(!map.ContainsKey(nums[i]))
                map.Add(nums[i], i);
        }
    }
}
```