C# .NET Compiler Limits
Maximum int value : Int32.MaxValue = 2^31
Maximum path value: OS path = 260 characters
Multi-classes inherit: *interface* keyword for *class,struct*
Maximum recursion: 100K calls = int.MaxValue
Tuple size: 7 different types
Types (no GC): StringBuilder/TcpClient/Most Streams and OS funct.
Minimum larg Obj size: SOH size = 85KB
Maximum .NET6+: Json serialization = maximum plaintext 166MB/maximum base64 125MB
Minimum CPU threads: 1 thread = 1 CPU core (incl. logical cores)
Maximum non-float types: 9 types = SByte / Int16 / Int32 / Int64 / Byte / UInt16 / UInt32 / UInt64 / BigInteger
Maximum float types: 3 types = float / double / Decimal
Max string usages: 6 C# string creations = interpolation / formatting / operator concat / string method concat / append / join method
Max types support T: Generics, Arrays, All IEnumerables Max types support TSource: Entities, All IQueryables

C# .NET Environment Limits
Heap size: 2bn bytes = 2GB Max object heap size
CLR size: Win32 max = C:\...256chars...<NUL>
Multi-classes inherit: *abstract* or *interface* keywords
Heap size: 100K fn calls = 2GB Max limit
Tuple size: TRest = 7 different types as generics
Types (with GC): SqlConnection/SqlCommand/ All IDisposables
Minimum small obj type: Gen0/Gen1 = 85KB
System.Text.Json limit: Utf8JsonWriter serial. = max tokenized text 166MB/max base64 125MB
.NET ThreadPool size: ThreadPool.SetMaxThreads, SetMinThreads = CPU default values
.NET max-min values: 9 ranges = -128..127 / 0..255 / -32768..32767 / 0..65535 / -2147483648..2147483647 / 0..4294967295 / -9,223,372,036,854,775,808..-9,223,372,036,854,775,807 / 0..18,446,744,073,709,551,615 / or...runtime computed size
String definitions: $"{}" / string.Format() / + += / string.Concat()/StringBuilder.Append()/string.Join()
.NET Allocation of T: Generics, Indexed Structures, All IEnumerables

Maximum Objects Gen: 4 Gens = Gen0,Gen1 - small objects
Gen2,Gen3 - large objects

Inappropriate usages of *yield return*:

- **Inside** *foreach(var i in items) {...}*
- **When sequentially accessing elements** *index[0..N]*
- **When allocating a very long sequence of elements at once and not separate elements.**
- **When non-defering execution state (***try-catch* **block with yield)**

.NET Allocation of TSource: Entities, All IQueryables

.NET Heap Allocat: 2 Sizes = SOH (small heap) - Gen0,Gen1
LOH (large heap) - Gen2,Gen3

Results of bad *yield return* **use:**

- **Causes** *MoveNext()* **per element inside the** *foreach* **section.**
- **Causes non-sequential alloc. of elements on lazy evaluated long sequences.**
- **Causes very large datasets allocated at once when not necessary.**
- **On exception before** *yield return*, **there is noticeable delay on allocating the yielded part.**