Про .NET
Производительность

SearchValues<T>

```
private static readonly string s_input = new HttpClient()
    .GetStringAsync("https://gutenberg.org/cache/epub/2600/pg2600.txt").Result;

private static readonly SearchValues<string> s_warningError = SearchValues
    .Create(["warning", "error"], StringComparison.OrdinalIgnoreCase);

[Benchmark(Baseline = true)]
public bool TwoContains() =>
     s_input.Contains("warning", StringComparison.OrdinalIgnoreCase) ||
     s_input.Contains("error", StringComparison.OrdinalIgnoreCase);

[Benchmark]
public bool ContainsAny() =>
    s_input.AsSpan().ContainsAny(s_warningError);
```

Method	Mean	Ratio
TwoContains	70.03 us	**1.00**
ContainsAny	14.05 us	**0.20**

LINQ

```
IEnumerable<int> _arrayDistinct = Enumerable.Range(0, 1000).ToArray()
     .Distinct();
[Benchmark] public int DistinctFirst() => _arrayDistinct.First();
```

DistinctFirst	.NET 8.0	49.844 ns	**1.00**	328 B	1.00
DistinctFirst	.NET 9.0	7.928 ns	**0.16**	–	0.00

```
IEnumerable<int> _appendSelect = Enumerable.Range(0, 1000).ToArray()
     .Append(42).Select(i => i * 2);
[Benchmark] public int AppendSelectLast() => _appendSelect.Last();
```

AppendSelectLast	.NET 8.0	3,668.347 ns	**1.000**	144 B	1.00
AppendSelectLast	.NET 9.0	2.222 ns	**0.001**	–	0.00

```
IEnumerable<int> _listDefaultIfEmptySelect = Enumerable.Range(0, 1000).ToList()
     .DefaultIfEmpty().Select(i => i * 2);
[Benchmark] public int DefaultIfEmptySelectElementAt() =>
     _listDefaultIfEmptySelect.ElementAt(999);
```

DefaultIfEmptySelectElementAt	.NET 8.0	2,772.283 ns	**1.000**	144 B	1.00
DefaultIfEmptySelectElementAt	.NET 9.0	4.399 ns	**0.002**	–	0.00

```
IEnumerable<int> _rangeUnion = Enumerable.Range(0, 1000)
     .Union(Enumerable.Range(500, 1000));
[Benchmark] public int RangeUnionFirst() => _rangeUnion.First();
```

RangeUnionFirst	.NET 8.0	53.670 ns	**1.00**	344 B	1.00
RangeUnionFirst	.NET 9.0	5.181 ns	**0.10**	–	0.00

```
IEnumerable<int> _ints = new(Enumerable.Range(-8000, 8000 * 2));
new Random(42).Shuffle(CollectionsMarshal.AsSpan(_ints));

[Benchmark]
public int OrderByLast_Int32() => _ints.OrderBy(x => x).Last();
```

OrderByLast_Int32	.NET 8.0	34,715.6 ns	**1.00**	136 B	1.00
OrderByLast_Int32	.NET 9.0	25,001.1 ns	**0.72**	128 B	0.94

```
[Benchmark]
public int OrderLast_Int32() => _ints.Order().Last();
```

OrderLast_Int32	.NET 8.0	36,064.9 ns	**1.00**	112 B	1.00
OrderLast_Int32	.NET 9.0	693.8 ns	**0.02**	56 B	0.50

Dictionary<TKey, T>

```
private static readonly string s_input = new HttpClient()
    .GetStringAsync("https://gutenberg.org/cache/epub/2600/pg2600.txt").Result;

[GeneratedRegex(@"\b\w+\b")]
private static partial Regex WordParser();

[Benchmark(Baseline = true)]
public Dictionary<string, int> CountWords1()
{
    ReadOnlySpan<char> input = s_input;
    Dictionary<string, int> result = new(StringComparer.OrdinalIgnoreCase);
    foreach (ValueMatch match in WordParser().EnumerateMatches(input))
    {
        ReadOnlySpan<char> word = input.Slice(match.Index, match.Length);
        string key = word.ToString();
        result[key] = result.TryGetValue(key, out int count) ? count + 1 : 1;
    }
    return result;
}
```

Method	Mean	Ratio	Allocated	Alloc Ratio
CountWords1	60.73 ms	**1.00**	20.67 MB	**1.00**

```
[Benchmark]
public Dictionary<string, int> CountWords2()
{
    ReadOnlySpan<char> input = s_input;
    Dictionary<string, int> result = new(StringComparer.OrdinalIgnoreCase);
    Dictionary<string, int>.AlternateLookup<ReadOnlySpan<char>> alternate = result
        .GetAlternateLookup<ReadOnlySpan<char>>();

    foreach (ValueMatch match in WordParser().EnumerateMatches(input))
    {
      ReadOnlySpan<char> word = input.Slice(match.Index, match.Length);
      alternate[word] = alternate.TryGetValue(word, out int count) ? count + 1 : 1;
    }
    return result;
}
```

Method	Mean	Ratio	Allocated	Alloc Ratio
CountWords1	60.73 ms	**1.00**	20.67 MB	**1.00**
CountWords2	54.01 ms	**0.89**	2.54 MB	**0.12**

Span<T>

```
using System;

public class C
{
    public void M()
    {
        Helpers.DoAwesomeStuff("Hello", "World");
    }
}

public static class Helpers
{
    public static void DoAwesomeStuff<T>(params T[] values) { }
    public static void DoAwesomeStuff<T>(params ReadOnlySpan<T> values) { }
}
```

```
using System;

public class C
{
    public void M()
    {
        Helpers.DoAwesomeStuff("Hello", "World");
    }
}

public static class Helpers
{
    public static void DoAwesomeStuff<T>(params T[] values) { }
    public static void DoAwesomeStuff<T>(params ReadOnlySpan<T> values) { }
}
```

```
<>y__InlineArray2<string> buffer = default;
<PrivateImplementationDetails>
    .InlineArrayElementRef<<>y__InlineArray2<string>, string>(ref buffer, 0) =
"Hello";
<PrivateImplementationDetails>
    .InlineArrayElementRef<<>y__InlineArray2<string>, string>(ref buffer, 1) =
"World";
Helpers.DoAwesomeStuff(<PrivateImplementationDetails>
    .InlineArrayAsReadOnlySpan<<>y__InlineArray2<string>, string>(ref buffer, 2));
```

```
[Benchmark]
public string Join() => Path.Join("a", "b", "c", "d", "e");
```

Method	Runtime	Mean	Ratio	Allocated	Alloc Ratio
Join	.NET 8.0	30.83 ns	**1.00**	104 B	**1.00**
Join	.NET 9.0	24.85 ns	**0.81**	40 B	**0.38**

```
namespace System.Threading
{
    public class Task
    {
        public static void WaitAll(params ReadOnlySpan<Task> tasks);
        public static Task WhenAll(params ReadOnlySpan<Task> tasks);
        public static Task<TResult[]> WhenAll<TResult>(
            params ReadOnlySpan<Task<TResult>> tasks
        );
        public static Task<Task> WhenAny(params ReadOnlySpan<Task> tasks);
        public static Task<Task<TResult>> WhenAny<TResult>(
            params ReadOnlySpan<Task<TResult>> tasks
        );
    }
}
```

LOOP

```
[Benchmark]
public int UpwardCounting()
{
    int count = 0;
    for (int i = 0; i < 100; i++)
    {
        count++;
    }
    return count;
}
```

```
M00_L00:
        inc         eax
        inc         ecx
        cmp         ecx,64
        jl          short M00_L00
```

```
[Benchmark]
public int DownwardCounting()
{
    int count = 0;
    for (int i = 99; i >= 0; i--)
    {
        count++;
    }
    return count;
}
```

```
M00_L00:
        inc         eax
        dec         ecx
        jns         short M00_L00
```

Method	Runtime	Mean	Ratio
UpwardCounting	.NET 8.0	30.27 ns	**1.00**
UpwardCounting	.NET 9.0	26.52 ns	**0.88**

```
private int[] _array = Enumerable
    .Range(0, 1000)
    .ToArray();

[Benchmark]
public int Sum()
{
    int[] array = _array;
    int sum = 0;
    for (int i = 0; i < array.Length; i++)
    {
        sum += array[i];
    }
    return sum;
}
```

```
; Tests.Sum()
push      rbp
mov       rbp,rsp
mov       rax,[rdi+8]
xor       ecx,ecx
xor       edx,edx
mov       edi,[rax+8]
test      edi,edi
jle       short M00_L01

M00_L00:
mov       esi,edx
add       ecx,[rax+rsi*4+10]
inc       edx
cmp       edi,edx
jg        short M00_L00
M00_L01:
mov       eax,ecx
pop       rbp
ret; Total bytes of code 35
```

```
private int[] _array = Enumerable
    .Range(0, 1000)
    .ToArray();

[Benchmark]
public int Sum()
{
    int[] array = _array;
    int sum = 0;
    for (int i = 0; i < array.Length; i++)
    {
        sum += array[i];
    }
    return sum;

}
```

```
; Tests.Sum()
push      rbp
mov       rbp,rsp
mov       rax,[rdi+8]
xor       ecx,ecx

mov       edx,[rax+8]
test      edx,edx
jle       short M00_L01
add       rax,10
M00_L00:
add       ecx,[rax]
add       rax,4
dec       edx
jne       short M00_L00

M00_L01:
mov       eax,ecx
pop       rbp
ret; Total bytes of code 35
```

```
; Tests.Sum()
push      rbp
mov       rbp,rsp
mov       rax,[rdi+8]
xor       ecx,ecx
xor       edx,edx
mov       edi,[rax+8]
test      edi,edi
jle       short M00_L01

M00_L00:
mov       esi,edx
add       ecx,[rax+rsi*4+10]
inc       edx
cmp       edi,edx
jg        short M00_L00
M00_L01:
mov       eax,ecx
pop       rbp
ret; Total bytes of code 35
```

brAnchIng

```
[Benchmark]
[Arguments(3)]
public string? Test() => M(["123"]);

[MethodImpl(MethodImplOptions.NoInlining)]
private static string? M(ReadOnlySpan<string> values)
{
    if (values.Length <= 1)
    {
        return values.Length == 0 ?
            string.Empty :
            values[0];
    }
    return null;
}
```

```
; Tests.M(System.ReadOnly...
push      rbp
mov       rbp,rsp
cmp       esi,1
jg        short M01_L01
test      esi,esi
je        short M01_L00
test      esi,esi
je        short M01_L02
mov       rax,[rdi]
pop       rbp         ret
M01_L00:
mov       rax,7FB62147C008
pop       rbp
retM01_L01:
xor       eax,eax
pop       rbp
ret
M01_L02:
call
CORINFO_HELP_RNGCHKFAIL
int       3;
Total bytes of code 44
```

```
[Benchmark]
[Arguments(3)]
public string? Test() => M(["123"]);

[MethodImpl(MethodImplOptions.NoInlining)]
private static string? M(
    ReadOnlySpan<string> values
) {
    if (values.Length <= 1)
    {
        return values.Length == 0 ?
            string.Empty :
            values[0];
    }
    return null;
}
```

```
; Tests.M(System.ReadOnly...
push      rbp
mov       rbp,rsp
cmp       esi,1
jg        short M01_L01
test      esi,esi
je        short M01_L00
mov       rax,[rdi]
pop       rbp
ret
M01_L00:
mov       rax,7F5700FB1008
pop       rbp
ret
M01_L01:
xor       eax,eax
pop       rbp
ret;
Total bytes of code 34
```

```
; Tests.M(System.ReadOnly...
push      rbp
mov       rbp,rsp
cmp       esi,1
jg        short M01_L01
test      esi,esi
je        short M01_L00
test      esi,esi
je        short M01_L02
mov       rax,[rdi]
pop       rbp          ret
M01_L00:
mov       rax,7FB62147C008
pop       rbp
retM01_L01:
xor       eax,eax
pop       rbp
ret
M01_L02:
call
CORINFO_HELP_RNGCHKFAIL
int       3;
Total bytes of code 44
```

```
private static ReadOnlySpan<int> Lookup
    => [1, 2, 3, 5, 8, 13, 21];

[Benchmark]
[Arguments(3)]
public int Test1(int i)
    => (uint)i < 7 ? Lookup[i] : -1;

[Benchmark]
[Arguments(3)]
public int Test2(int i)
    => (uint)i <= 6 ? Lookup[i] : -1;
```

```
Tests.Test2(Int32)
push     rbp
mov      rbp,rsp
cmp      esi,6
ja       short M00_L00
cmp      esi,7
jae      short M00_L01
mov      eax,esi
mov      rcx,7F8D11621030
mov      eax,[rcx+rax*4]
pop      rbp
ret
M00_L00:
mov      eax,0FFFFFFFF
pop      rbp
ret
M00_L01:
call     CORINFO_HELP_RNGCHKFAIL
int      3
; Total bytes of code 44
```

```
private static ReadOnlySpan<int> Lookup
    => [1, 2, 3, 5, 8, 13, 21];

[Benchmark]
[Arguments(3)]
public int Test1(int i)
    => (uint)i < 7 ? Lookup[i] : -1;

[Benchmark]
[Arguments(3)]
public int Test2(int i)
    => (uint)i <= 6 ? Lookup[i] : -1;
```

```
; Tests.Test1(Int32)
cmp       esi,7
jae       short M00_L00
mov       eax,esi
mov       rcx,7F5B9DC5E030
mov       eax,[rcx+rax*4]
ret
M00_L00:
mov       eax,0FFFFFFFF
ret
; Total bytes of code 27;

Tests.Test2(Int32)
cmp       esi,6
ja        short M00_L00
mov       eax,esi
mov       rcx,7F7FDE2C9030
mov       eax,[rcx+rax*4]
ret
M00_L00:
mov       eax,0FFFFFFFF
ret
; Total bytes of code 27
```

```
Tests.Test2(Int32)
push      rbp
mov       rbp,rsp
cmp       esi,6
ja        short M00_L00
cmp       esi,7
jae       short M00_L01
mov       eax,esi
mov       rcx,7F8D11621030
mov       eax,[rcx+rax*4]
pop       rbp
ret
M00_L00:
mov       eax,0FFFFFFFF
pop       rbp
ret
M00_L01:
call
CORINFO_HELP_RNGCHKFAIL
int       3
; Total bytes of code 44
```

```
private readonly int[] _x = new int[10];

[Benchmark]
[Arguments(2)]
public int Add(int y) => _x[y] + (y % 8);
```

```
; Tests.Add(Int32)
push      rax
mov       rax,[rdi+8]
cmp       esi,[rax+8]
jae       short M00_L00
mov       ecx,esi
mov       edx,esi
sar       edx,1F
and       edx,7
add       edx,esi
and       edx,0FFFFFFF8
mov       edi,esi
sub       edi,edx
add       edi,[rax+rcx*4+10]
mov       eax,edi
add       rsp,8
ret
M00_L00:
call CORINFO_HELP_RNGCHKFAIL
int  3
; Total bytes of code 46
```

```
private readonly int[] _x = new int[10];

[Benchmark]
[Arguments(2)]
public int Add(int y) => _x[y] + (y % 8);
```

```
; Tests.Add(Int32)
push      rax
mov       rax,[rdi+8]
cmp       esi,[rax+8]
jae       short M00_L00
mov       ecx,esi
and       esi,7
add       esi,[rax+rcx*4+10]
mov       eax,esi
add       rsp,8
ret
M00_L00:
call CORINFO_HELP_RNGCHKFAIL
int  3
; Total bytes of code 32
```

```
; Tests.Add(Int32)
push      rax
mov       rax,[rdi+8]
cmp       esi,[rax+8]
jae       short M00_L00
mov       ecx,esi
mov       edx,esi
sar       edx,1F
and       edx,7
add       edx,esi
and       edx,0FFFFFFF8
mov       edi,esi
sub       edi,edx
add       edi,[rax+rcx*4+10]
mov       eax,edi
add       rsp,8
ret
M00_L00:
call CORINFO_HELP_RNGCHKFAIL
int  3
; Total bytes of code 46
```

code moVINg

```
using System.Diagnostics;

new Thread(() => {
     var a = new object[1000];
     while (true) a.AsSpan().Fill(a);
})
{ IsBackground = true }.Start();

var sw = new Stopwatch();
while (true) {
    sw.Restart();
    for (int i = 0; i < 10; i++)
    {
        GC.Collect();
        Thread.Sleep(15);
    }
    Console.WriteLine(sw.Elapsed.TotalSeconds);
}
```

```
using System.Diagnostics;

new Thread(() => {
     var a = new object[1000];
     while (true) a.AsSpan().Fill(a);
})
{ IsBackground = true }.Start();

var sw = new Stopwatch();
while (true) {
    sw.Restart();
    for (int i = 0; i < 10; i++)
    {
        GC.Collect();
        Thread.Sleep(15);
    }
    Console.WriteLine(sw.Elapsed.TotalSeconds);
}
```

```
1.0683524
0.8884759
0.8420748
1.1101804
1.2730635
```

```
using System.Diagnostics;

new Thread(() => {
     var a = new object[1000];
     while (true) a.AsSpan().Fill(a);
})
{ IsBackground = true }.Start();

var sw = new Stopwatch();
while (true) {
    sw.Restart();
    for (int i = 0; i < 10; i++)
    {
        GC.Collect();
        Thread.Sleep(15);
    }
    Console.WriteLine(sw.Elapsed.TotalSeconds);
}
```

```
1.0683524
0.8884759
0.8420748
1.1101804
1.2730635
```

```
0.1638237
0.2129748
0.2859566
0.3020449
0.2871952
```

i ndex of

String.Contains(string) vs. String.Contains(char)

```
[Benchmark]
public bool Contains_Char()
{
    return _inputString.Contains('X');
}

[Benchmark]
public bool Contains_String()
{
    return _inputString.Contains("X");
}
```

class System.String
Represents text as a sequence of UTF-16 code units.

CA1847: Use 'string.Contains(char)' instead of 'string.Contains(string)' when searching for a single character

Method	StringLength	Mean	Error	StdDev	Median
Contains_String	10	5.644 ns	0.1490 ns	0.2137 ns	5.674 ns
Contains_Char	10	2.032 ns	0.0661 ns	0.1546 ns	1.996 ns
Contains_String	100	7.260 ns	0.1822 ns	0.3842 ns	7.122 ns
Contains_Char	100	3.427 ns	0.2424 ns	0.6915 ns	3.186 ns
Contains_String	1000	20.426 ns	0.4371 ns	0.9776 ns	20.540 ns
Contains_Char	1000	17.509 ns	0.5083 ns	1.4908 ns	17.292 ns

```
public bool Contains(string value)
{
    if (value == null)
        ThrowHelper.ThrowArgumentNullException(ExceptionArgument.value);

    return SpanHelpers.IndexOf(
        ref _firstChar,
        Length,
        ref value._firstChar,
        value.Length) >= 0;
}
```

```
public static int IndexOf(
    ref char searchSpace, int searchSpaceLength, ref char value, int valueLength
) {
    if (valueLength == 0)
        return 0;

    int valueTailLength = valueLength - 1;
    if (valueTailLength == 0)
    {
        // for single-char values use plain IndexOf
        return (ref searchSpace, value, searchSpaceLength);
    }
    ...
```

```
public bool Contains(string value)
{
    if (value == null)
        ThrowHelper.ThrowArgumentNullException(ExceptionArgument.value);

    // PR added this if block
    if (RuntimeHelpers.IsKnownConstant(value) && value.Length == 1)
        return Contains(value[0]);

    return SpanHelpers.IndexOf(
        ref _firstChar,
        Length,
        ref value._firstChar,
        value.Length) >= 0;
}
```

```
public bool Contains(string value)
{
    if (value == null)
        ThrowHelper.ThrowArgumentNullException(ExceptionArgument.value);

    return SpanHelpers.IndexOf(
        ref _firstChar,
        Length,
        ref value._firstChar,
        value.Length) >= 0;
}
```

```
public bool Contains(string value)
{
    if (value == null)
        ThrowHelper.ThrowArgumentNullException(ExceptionArgument.value);

    return Contains(value[0]);
}
```

```
readonly string _inputString =
    "Permission is hereby granted, free of charge, to any person obtaining " +
    "a copy of the Unicode data files and any associated documentation"

[Benchmark]
[Arguments(2)]
public bool Contains_string() => _inputString.Contains("X");
```

Method	Runtime	Mean	Ratio
Contains_string	.NET 8.0	4.050 ns	**1.00**
Contains_string	.NET 9.0	2.361 ns	**0.58**

LiNKs

https://devblogs.microsoft.com/dotnet/performance-improvements-in-net-9

https://source.dot.net/

https://github.com/dotnet/runtime/pulls

https://youtu.be/6Ixv-XmK8YY

https://youtu.be/hJIrbWzoeP0

https://youtu.be/aLQpnpSxosg

https://dev.to/leandroveiga/boosting-net-9-application-performance-proven-tips-and-techniques-n0m

https://medium.com/c-sharp-programming/net-9-linq-performance-improvements-457ee3481e0d

https://ricomariani.medium.com/performance-improvements-in-net-9-d32afb4febca

https://abp.io/community/articles/.net-9-performance-improvements-summary-gmww3gl8

https://www.reddit.com/r/dotnet/comments/1ff00p7/performance_improvements_in_net_9

diagnostiCS

```
private MetricsEventListener _listener = new MetricsEventListener();
private Meter _meter = new Meter("Example");
private Counter<int> _counter;

[GlobalSetup]
public void Setup() => _counter = _meter.CreateCounter<int>("counter");

[GlobalCleanup]
public void Cleanup()
{
    _meter.Dispose();
}

[Benchmark]
public void Counter_Parallel()
{
    Parallel.For(0, 1_000_000, i =>
    {
        _counter.Add(1);
        _counter.Add(1);
    });
}
```

Method	Runtime	Mean	Ratio
Counter_Parallel	.NET 8.0	137.90 ms	**1.00**
Counter_Parallel	.NET 9.0	30.65 ms	**0.22**

```
private MetricsEventListener _listener = new MetricsEventListener();
private Meter _meter = new Meter("Example");
private Counter<int> _counter;

[GlobalSetup]
public void Setup() => _counter = _meter.CreateCounter<int>("counter");

[GlobalCleanup]
public void Cleanup()
{
    _meter.Dispose();
}

[Benchmark]
public void Counter_Parallel()
{
    Parallel.For(0, 1_000_000, i =>
    {
        _counter.Add(1);
        _counter.Add(1);
    });
}
```

```
[StructLayout(LayoutKind.Explicit, Size = 64)]
private struct PaddedDouble
{
    [FieldOffset(0)]
    public double Value;
}
```

Method	Runtime	Mean	Ratio
Counter_Parallel	.NET 8.0	137.90 ms	1.00
Counter_Parallel	.NET 9.0	30.65 ms	0.22

```
private int[] _values = new int[32];
[Params(1, 31)]
public int Index { get; set; }
[Benchmark]
public void ParallelIncrement()
{
    Parallel.Invoke(
        () => IncrementLoop(ref _values[0]),
        () => IncrementLoop(ref _values[Index]));

    static void IncrementLoop(ref int value)
    {
        for (int i = 0; i < 100_000_000; i++)
        {
            Interlocked.Increment(ref value);
        }
    }
}
```

Method	Index	Mean	InstructionRetired/Op	CacheMisses/Op
ParallelIncrement	1	**1,846.2 ms**	804,300,000	**177,889**
ParallelIncrement	31	**442.5 ms**	824,333,333	**52,429**

aDDiTionals

```
private static readonly FrozenSet<string> s_words = Regex.Matches("""
        Let me not to the marriage of true minds
        Admit impediments; love is not love
        Which alters when it alteration finds,
        Or bends with the remover to remove.
        O no, it is an ever-fixed mark
        That looks on tempests and is never shaken;
        It is the star to every wand'ring bark
        Whose worth's unknown, although his height be taken.
        Love's not time's fool, though rosy lips and cheeks
        Within his bending sickle's compass come.
        Love alters not with his brief hours and weeks,
        But bears it out even to the edge of doom:
        If this be error and upon me proved,
        I never writ, nor no man ever loved.
        """, @"\b\w+\b")
    .Cast<Match>()
    .Select(w => w.Value)
    .ToFrozenSet();

private string _word = "quickness";

[Benchmark]
public bool Contains() => s_words.Contains(_word);
```

Method	Runtime	Mean	Ratio
Contains	.NET 8.0	4.373 ns	1.00
Contains	.NET 9.0	1.154 ns	0.26

```
[Benchmark]
[Arguments(5)]
public uint DivideBy4_UInt32(uint value)
    => value / 4;

[Benchmark]
[Arguments(5)]
public int DivideBy4_Int32(int value)
    => value / 4;


[Benchmark]
[Arguments(5)]
public int DivideBy4_Int32(int value)
    => value < 4 ? 0 : value / 4;
```

```
; Tests.DivideBy4_UInt32(UInt32)
mov       eax,esi
shr       eax,2
ret
; Total bytes of code 6;

Tests.DivideBy4_Int32(Int32)
mov       eax,esi
sar       eax,1F
and       eax,3
add       eax,esi
sar       eax,2
ret
; Total bytes of code 14

; Tests.DivideBy4_Int32(Int32)
cmp       esi,4
jl        short M00_L00
mov       eax,esi
shr       eax,2
ret
M00_L00:
xor       eax,eax
ret
; Total bytes of code 14
```

```
[Benchmark]
[Arguments(1)]
[Arguments(8)]
[Arguments(500)]
public string[] IteratorToArray(int count) =>
GetItems(count).ToArray();

private IEnumerable<string> GetItems(int count)
{
    for (int i = 0; i < count; i++)
    {
        yield return ".NET 9";
    }
}
```

Method	Runtime	count	Mean	Ratio	Allocated	Alloc Ratio
IteratorToArray	.NET 8.0	1	65.51 ns	1.00	136 B	1.00
IteratorToArray	.NET 9.0	1	41.39 ns	0.63	80 B	0.59
IteratorToArray	.NET 8.0	8	103.30 ns	1.00	192 B	1.00
IteratorToArray	.NET 9.0	8	74.66 ns	0.72	136 B	0.71
IteratorToArray	.NET 8.0	500	3,100.69 ns	1.00	8536 B	1.00
IteratorToArray	.NET 9.0	500	3,080.31 ns	0.99	4072 B	0.48

```
private char[][] _values = new char[10_000][];
[GlobalSetup]
public void Setup()
{
    var rng = new Random(42);
    for (int i = 0; i < _values.Length; i++) {
        _values[i] = new char[rng.Next(0, 128)];
         rng.NextBytes(MemoryMarshal.AsBytes(_values[i].AsSpan()));
    }
}

[Benchmark]
public int Count()
{
    int count = 0;
    foreach (char[] numbers in _values)
    {
        count += numbers.AsSpan().Count('a');
    }
    return count;
}
```

Method	Runtime	Mean	Ratio
Count	.NET 8.0	133.25 us	1.00
Count	.NET 9.0	74.30 us	0.56

Conversation 6 | Commits 3 | Checks 111 | Files changed 1 | +5 −1

Changes from all commits | File filter | Conversations | Jump to

6 src/libraries/System.Runtime.Numerics/src/System/Number.BigInteger.cs

```
@@ -662,13 +662,17 @@ internal static ParsingStatus TryParseBigIntegerBinaryNumberStyle(ReadOnlySpan<c
662 662            // algorithm with a running time of O(N^2). And if it is greater than the threshold, use
663 663            // a divide-and-conquer algorithm with a running time of O(NlogN).
664 664            //
    665 +          // `1233`, which is approx the upper bound of most RSA key lengths, covers the majority
    666 +          // of most common inputs and allows for the less naive algorithm to be used for
    667 +          // large/uncommon inputs.
    668 +          //
665 669    #if DEBUG
666 670            // Mutable for unit testing...
667 671            internal static
668 672    #else
669 673            internal const
670 674    #endif
671     -          int s_naiveThreshold = 20000;
    675 +          int s_naiveThreshold = 1233;
672 676            private static ParsingStatus NumberToBigInteger(ref NumberBuffer number, out BigInteger result)
673 677            {
674 678                int currentBufferSize = 0;
```

```
private string _digits = string.Create(
    2000,
    0,
    (dest, _) => new Random(42).GetItems<char>("0123456789", dest)
);

[Benchmark]
public BigInteger ParseDecimal() => BigInteger.Parse(_digits);
```

Method	Runtime	Mean	Ratio	Allocated	Alloc Ratio
ParseDecimal	.NET 8.0	24.60 us	1.00	5528 B	1.00
ParseDecimal	.NET 9.0	18.95 us	0.77	856 B	0.15