Best way to reverse a string

Ask Question

381

101

I've just had to write a string reverse function in C# 2.0 (i.e. LINQ not available) and came up with this:

```
public string Reverse(string text)
{
    char[] cArray = text.ToCharArray();
    string reverse = String.Empty;
    for (int i = cArray.Length - 1; i > -1; i--)
    {
        reverse += cArray[i];
    }
    return reverse;
}
```

Personally I'm not crazy about the function and am convinced that there's a better way to do it. Is there?

c# .net performance algorithm unicode

edited Mar 3 '13 at 22:18
R. Martinho Fernandes
165k 59 383 470

asked Oct 23 '08 at 0:31
Guy
30.9k 88 227 289

45 Surprisingly tricky if you want proper international support. Example: Croatian/Serbian have two-character letters lj, nj etc. Proper reverse of "ljudi" is "idulj", NOT "idujl". I'm sure

I wonder if it's slower to concat a string instead of initializing a temp array and storing the results in that, and then finally converting that to a string? – The Muffin Man Jun 1 '13 at 5:29

2 Much newer related thread: Reverse a string with accent chars? – Jeppe Stig Nielsen Jul 23 '13 at 15:15

3 This question could be improved by defining what you mean by "best". Fastest? Most readable? Most reliable across various edge cases (null checks, multiple languages, etc.)? Most maintainable across versions of C# and .NET? – hypehuman Apr 15 '16 at 14:23

42 Answers

1 2 next

506 ✓

```
public static string Reverse( string s )
{
    char[] charArray = s.ToCharArray();
    Array.Reverse( charArray );
    return new string( charArray );
}
```

edited Nov 4 '17 at 19:32
Ofer Zelig
13.8k 4 45 84

answered Oct 23 '08 at 0:40
PeteT
10.5k 24 83 128

14 sambo99: It doesn't need to mention unicode: chars in C# are unicode characters, not bytes. Xor may be faster, but apart from being far less readable, that may even be what Array.Reverse() uses internally. – Nick Johnson Oct 23 '08 at

takes two of them to represent a supplementary character. See jaggersoft.com/csharp_standard/9.4.1.htm. – Bradley Grainger Oct 23 '08 at 15:18

4 Yeah sambo99 I suppose you are correct but it's a pretty rare case to use UTF-32. And XOR is only faster for a very small range of values, the correct answer would be to implement different methods for different lengths I suppose. But this is clear and concise which is a benefit in my opinion. – PeteT Dec 8 '08 at 15:52

16 Unicode control characters makes this method useless for non latin character sets. See Jon Skeet explanation, using a sock puppet: codeblog.jonskeet.uk/2009/11/02/… (1/4 the way down), or the video: vimeo.com/7516539 – Callum Rogers Apr 19 '10 at 23:14

17 Hope you don't encounter any surrogates or combining characters. – dalle Oct 14 '10 at 19:04

160

Here a solution that properly reverses the string `"Les Mise\u0301rables"` as `"selbare\u0301siM seL"` . This should render just like `selbarésiM seL` , not `selbaŕesiM seL` (note the position of the accent), as would the result of most implementations based on code units (`Array.Reverse` , etc) or even code points (reversing with special care for surrogate pairs).

```
using System;
using System.Collections.Generic;
using System.Globalization;
using System.Linq;

public static class Test
{
    private static IEnumerable<string> GraphemeClusters(th
        var enumerator = StringInfo.GetTextElementEnumerat
        while(enumerator.MoveNext()) {
```

```
        private static string ReverseGraphemeClusters(this str
            return string.Join("", s.GraphemeClusters().Revers
        }

        public static void Main()
        {
            var s = "Les Mise\u0301rables";
            var r = s.ReverseGraphemeClusters();
            Console.WriteLine(r);
        }
    }
```

(And live running example here: https://ideone.com/DqAeMJ)

It simply uses the .NET API for grapheme cluster iteration, which has been there since ever, but a bit "hidden" from view, it seems.

edited Jul 24 '13 at 16:19

answered Feb 27 '13 at 12:06

R. Martinho Fernandes
165k 59 383 470

8 +1 One of the very few correct answers, and **a lot** more elegant and future proof than any of the others, IMO – sehe Feb 27 '13 at 12:29

This fails for some locale dependent stuff, though. – R. Martinho Fernandes Feb 27 '13 at 12:46

6 It's funny how most of the other answerers are trying to shave ms off of what are otherwise incorrect approaches. How representative. – G. Stoynev Dec 5 '13 at 19:38

2 It's actually significantly faster to instantiate StringInfo(s), then iterate through SubstringByTextElements(x, 1) and build a new string with a StringBuilder. – RobinHood70 Jul 9 '16 at 4:03

Misérables (though Jon didn't mention a solution, he just listed issues). Good that you came up with a solution. Maybe Jon skeet invented a time machine, went back to 2009 and posted the problem example that you used in your solution. – barlop Jul 10 '16 at 11:55

122

This is turning out to be a surprisingly tricky question.

I would recommend using Array.Reverse for most cases as it is coded natively and it is very simple to maintain and understand.

It seems to outperform StringBuilder in all the cases I tested.

```
public string Reverse(string text)
{
   if (text == null) return null;

   // this was posted by petebob as well
   char[] array = text.ToCharArray();
   Array.Reverse(array);
   return new String(array);
}
```

There is a second approach that can be faster for certain string lengths which uses Xor.

```
    public static string ReverseXor(string s)
    {
        if (s == null) return null;
        char[] charArray = s.ToCharArray();
        int len = s.Length - 1;

        for (int i = 0; i < len; i++, len--)
        {
            charArray[i] ^= charArray[len];
        return new string(charArray);
    }
```

Note If you want to support the full Unicode UTF16 charset read this. And use the implementation there instead. It can be further optimized by using one of the above algorithms and running through the string to clean it up after the chars are reversed.

Here is a performance comparison between the StringBuilder, Array.Reverse and Xor method.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Diagnostics;

namespace ConsoleApplication4
{
    class Program
    {
        delegate string StringDelegate(string s);

        static void Benchmark(string description, StringDe
text)
        {
            Stopwatch sw = new Stopwatch();
            sw.Start();
            for (int j = 0; j < times; j++)
            {
                d(text);
            }
            sw.Stop();
            Console.WriteLine("{0} Ticks {1} : called {2}
description, times);
        }

        public static string ReverseXor(string s)
        {
            char[] charArray = s.ToCharArray();
            int len = s.Length - 1;
```

```
                charArray[i] ^= charArray[len];
                charArray[len] ^= charArray[i];
                charArray[i] ^= charArray[len];
            }

            return new string(charArray);
        }

        public static string ReverseSB(string text)
        {
            StringBuilder builder = new StringBuilder(text
            for (int i = text.Length - 1; i >= 0; i--)
            {
                builder.Append(text[i]);
            }
            return builder.ToString();
        }

        public static string ReverseArray(string text)
        {
            char[] array = text.ToCharArray();
            Array.Reverse(array);
            return (new string(array));
        }

        public static string StringOfLength(int length)
        {
            Random random = new Random();
            StringBuilder sb = new StringBuilder();
            for (int i = 0; i < length; i++)
            {
                sb.Append(Convert.ToChar(Convert.ToInt32(M
random.NextDouble() + 65))));
            }
            return sb.ToString();
        }

        static void Main(string[] args)
        {

            int[] lengths = new int[] {1,10,15,25,50,75,10

            foreach (int l in lengths)
            {
                int iterations = 10000;
                string text = StringOfLength(l);
iterations, text);
                Benchmark(String.Format("Xor (Length: {0})
text);

                Console.WriteLine();
            }

            Console.Read();
        }
    }
}
```

Here are the results:

```
26251 Ticks String Builder (Length: 1) : called 10000 time
33373 Ticks Array.Reverse (Length: 1) : called 10000 times
20162 Ticks Xor (Length: 1) : called 10000 times.

51321 Ticks String Builder (Length: 10) : called 10000 tim
37105 Ticks Array.Reverse (Length: 10) : called 10000 time
23974 Ticks Xor (Length: 10) : called 10000 times.

66570 Ticks String Builder (Length: 15) : called 10000 tim
26027 Ticks Array.Reverse (Length: 15) : called 10000 time
24017 Ticks Xor (Length: 15) : called 10000 times.

101609 Ticks String Builder (Length: 25) : called 10000 ti
28472 Ticks Array.Reverse (Length: 25) : called 10000 time
35355 Ticks Xor (Length: 25) : called 10000 times.

161601 Ticks String Builder (Length: 50) : called 10000 ti
35839 Ticks Array.Reverse (Length: 50) : called 10000 time
51185 Ticks Xor (Length: 50) : called 10000 times.

230898 Ticks String Builder (Length: 75) : called 10000 ti
40628 Ticks Array.Reverse (Length: 75) : called 10000 time
78906 Ticks Xor (Length: 75) : called 10000 times.

312017 Ticks String Builder (Length: 100) : called 10000 t
52225 Ticks Array.Reverse (Length: 100) : called 10000 tim
110195 Ticks Xor (Length: 100) : called 10000 times.

2970691 Ticks String Builder (Length: 1000) : called 10000
292094 Ticks Array.Reverse (Length: 1000) : called 10000 t
```

```
74884495 Ticks Array.Reverse (Length: 100000) : called 100
125409674 Ticks Xor (Length: 100000) : called 10000 times.
```

It seems that Xor can be faster for short strings.

edited May 23 '17 at 12:34

community wiki
11 revs, 4 users 97%
Sam Saffron

2 That doesn't return a string - you need to wrap this in a call to "new String(...)" – Greg Beech Oct 23 '08 at 0:43

BTW .. I just had a look at the implementation of Array.Reverse, and its done naitively for chars ... it should be much faster than the StringBuilder option. – Sam Saffron Oct 23 '08 at 0:46

How nice of you, Greg, to halp Sambo arrive at a better solution instead of down-voting him. – DOK Oct 23 '08 at 0:47

@dok1 - don't mention it :) @sambo99 - now I'm intrigued, will have to whip out a code profiler tomorrow and have a look! – Greg Beech Oct 23 '08 at 0:50

9 These methods don't handle strings containing characters outside of the Base Multilingual Plane, i.e., Unicode characters >= U+10000 that are represented with two C# chars. I've posted an answer that handles such strings correctly. – Bradley Grainger Oct 23 '08 at 3:42

If the string contains Unicode data (strictly speaking, non-BMP characters) the other methods that have been posted will corrupt it, because you cannot swap the order of high

The following code sample will correctly reverse a string that contains non-BMP characters, e.g., "\U00010380\U00010381" (Ugaritic Letter Alpa, Ugaritic Letter Beta).

```
public static string Reverse(this string input)
{
    if (input == null)
        throw new ArgumentNullException("input");

    // allocate a buffer to hold the output
    char[] output = new char[input.Length];
    for (int outputIndex = 0, inputIndex = input.Length -
outputIndex++, inputIndex--)
    {
        // check for surrogate pair
        if (input[inputIndex] >= 0xDC00 && input[inputInde
            inputIndex > 0 && input[inputIndex - 1] >= 0xD
<= 0xDBFF)
        {
            // preserve the order of the surrogate pair co
            output[outputIndex + 1] = input[inputIndex];
            output[outputIndex] = input[inputIndex - 1];
            outputIndex++;
            inputIndex--;
        }
        else
        {
            output[outputIndex] = input[inputIndex];
        }
    }

    return new string(output);
}
```

edited Oct 23 '08 at 4:08

answered Oct 23 '08 at 3:40

Bradley Grainger
20k 4 66 88

supplementary character is encoded using two of them, so this is necessary, – Bradley Grainger Oct 23 '08 at 15:14

13 It seems like System.String really ought to expose a HereBeDragons property for strings that contain Unicode supplementary characters. – Robert Rossney Oct 23 '08 at 20:54

4 @SebastianNegraszus: That's correct: this method just reverses the codepoints in the string. Reversing the grapheme clusters would probably be more "useful" overall (but what's the "use" of reversing an arbitrary string in the first place?), but is not easy to implement with just the built-in methods in the .NET Framework. – Bradley Grainger Nov 6 '12 at 14:38

2 @Richard: The rules for breaking grapheme clusters are a little more complicated than just detecting combining code points; see the documentation on Grapheme Cluster Boundaries in UAX #29 for more information. – Bradley Grainger Feb 5 '13 at 16:56

1 Very good info! Does **ANYONE** have a failing test for the Array.Reverse test? And by test I mean a sample string not a whole unit test... It would really help me (and others) convince different persons about this issue.. – Andrei Rînea Jul 2 '13 at 15:44

45

From above 3.5 Framework

```
public string ReverseString(string srtVarable)
{
    return new string(srtVarable.Reverse().ToArray());
}
```

edited Apr 14 '16 at 8:51

527 5 6

That is around 5.7 times slower than the most upvoted version so i would not recommend using this! – Martin Niederl May 6 '17 at 13:42

1 Not the fastest solution, but useful as a one-liner. – adrianmp Oct 18 '17 at 14:19

23

Ok, in the interest of "don't repeat yourself," I offer the following solution:

```
public string Reverse(string text)
{
   return Microsoft.VisualBasic.Strings.StrReverse(text);
}
```

My understanding is that this implementation, available by default in VB.NET, properly handles Unicode characters.

answered Dec 7 '11 at 17:06

richardtallent
25.2k 13 72 108

11 This only handles surrogates properly. It messes up combining marks: ideone.com/yikdqX. – R. Martinho Fernandes Apr 3 '13 at 21:52

Greg Beech posted an `unsafe` option that is indeed as fast as it gets (it's an in-place reversal); but, as he indicated in

That said, I'm surprised there is so much of a consensus that `Array.Reverse` is the fastest method. There's still an `unsafe` approach that returns a reversed copy of a string (no in-place reversal shenanigans) **significantly faster than the `Array.Reverse` method** for small strings:

```
public static unsafe string Reverse(string text)
{
    int len = text.Length;

    // Why allocate a char[] array on the heap when you wo
    // outside of this method? Use the stack.
    char* reversed = stackalloc char[len];

    // Avoid bounds-checking performance penalties.
    fixed (char* str = text)
    {
        int i = 0;
        int j = i + len - 1;
        while (i < len)
        {
            reversed[i++] = str[j--];
        }
    }

    // Need to use this overload for the System.String con
    // as providing just the char* pointer could result in
    // at the end of the string (no guarantee of null term
    return new string(reversed, 0, len);
}
```

Here are some benchmark results.

You can see that the performance gain shrinks and then disappears against the `Array.Reverse` method as the strings get larger. For small- to medium-sized strings, though, it's tough to beat this method.

edited Apr 30 '13 at 0:19

Dan Tao
96.3k 44 255 412

2 StackOverflow on large strings. – Raz Megrelidze Jan 28 '14 at 19:11

@rezomegreldize: Yep, that'll happen ;) – Dan Tao Jan 28 '14 at 19:17

14

The easy and nice answer is using the Extension Method:

```
static class ExtentionMethodCollection
{
    public static string Inverse(this string @base)
    {
        return new string(@base.Reverse().ToArray());
    }
}
```

and here's the output:

```
string Answer = "12345".Inverse(); // = "54321"
```

edited Jul 18 '16 at 0:06

answered Oct 18 '14 at 11:29
Mehdi Khademloo
1,596 1 10 29

`Reverse()` and `ToArray()` are in the wrong order in your code sample. – Chris Walsh Feb 24 '17 at 0:30

1 @user5389726598465 See this link: docs.microsoft.com/en-us/dotnet/csharp/language-reference/… Because 'base' is a keyword in C#, it must be prefixed with @ for the C# compiler to interpret it as an identifier. – Dyndrilliac May 10 at 8:02

12

If you want to play a really dangerous game, then this is by far the fastest way there is (around four times faster than the `Array.Reverse` method). It's an in-place reverse using pointers.

Note that I really do not recommend this for any use, ever (have a look here for some reasons why you should not use this method), but it's just interesting to see that it can be done, and that strings aren't really immutable once you turn on unsafe code.

```
public static unsafe string Reverse(string text)
{
    if (string.IsNullOrEmpty(text))
    {
        return text;
    }

    fixed (char* pText = text)
    {
        char* pStart = pText;
        char* pEnd = pText + text.Length - 1;
        for (int i = text.Length / 2; i >= 0; i--)
        {
            char temp = *pStart;
            *pStart++ = *pEnd;
            *pEnd-- = temp;
        }

        return text;
    }
}
```

Community ♦
1 1

answered Oct 23 '08 at 2:49
Greg Beech
103k 36 187 236

Im pretty sure this will return incorrect results for utf16 strings, it is really asking trouble :) – Sam Saffron Oct 23 '08 at 3:07

I never tested this but I like what you've done here. – Guy Oct 23 '08 at 3:43

Hi you should link to this post on this stackoverflow.com/questions/229346/… , as I said before this is really asking for trouble ... – Sam Saffron Oct 23 '08 at 11:53

This may be completely evil and ill-advised (as you yourself concede), but there's still a high-performance way to reverse a string using `unsafe` code that *isn't* evil and *still* beats `Array.Reverse` in many cases. Take a look at my answer. – Dan Tao Jun 15 '10 at 18:33

11

Have a look at the wikipedia entry here. They implement the String.Reverse extension method. This allows you to write code like this:

```
string s = "olleh";
s.Reverse();
```

They also use the ToCharArray/Reverse combination that other answers to this question suggest. The source code looks like this:

```
public static string Reverse(this string input)
```

```
        return new String(chars);
    }
```

edited Oct 23 '08 at 0:58

answered Oct 23 '08 at 0:37

Mike Thompson
5,583 3 26 38

That's wonderful, except extension methods weren't introduced in c# 2.0. – Kobi Jul 7 '09 at 12:40

11

Firstly you don't need to call `ToCharArray` as a string can already be indexed as a char array, so this will save you an allocation.

The next optimisation is to use a `StringBuilder` to prevent unnecessary allocations (as strings are immutable, concatenating them makes a copy of the string each time). To further optimise this we pre-set the length of the `StringBuilder` so it won't need to expand its buffer.

```
public string Reverse(string text)
{
    if (string.IsNullOrEmpty(text))
    {
        return text;
    }

    StringBuilder builder = new StringBuilder(text.Length)
    for (int i = text.Length - 1; i >= 0; i--)
    {
        builder.Append(text[i]);
    return builder.ToString();
}
```

Edit: Performance Data

I tested this function and the function using `Array.Reverse` with the following simple program, where `Reverse1` is one function and `Reverse2` is the other:

```
static void Main(string[] args)
{
    var text = "abcdefghijklmnopqrstuvwxyz";

    // pre-jit
    text = Reverse1(text);
    text = Reverse2(text);

    // test
    var timer1 = Stopwatch.StartNew();
    for (var i = 0; i < 10000000; i++)
    {
        text = Reverse1(text);
    }

    timer1.Stop();
    Console.WriteLine("First: {0}", timer1.ElapsedMillisec

    var timer2 = Stopwatch.StartNew();
    for (var i = 0; i < 10000000; i++)
    {
        text = Reverse2(text);
    }

    timer2.Stop();
    Console.WriteLine("Second: {0}", timer2.ElapsedMillise

    Console.ReadLine();
}
```

It turns out that for short strings the `Array.Reverse` method is around twice as quick as the one above, and for longer strings the difference is even more pronounced. So given this one up here just to show that it isn't the way you should do it (much to my surprise!)

edited Oct 23 '08 at 1:12

answered Oct 23 '08 at 0:38

Greg Beech
103k 36 187 236

Wouldn't storing text.Length in a variable give a little more speed as you are referencing this via an object? – David Robbins Oct 23 '08 at 1:59

10

Try using Array.Reverse

```
public string Reverse(string str)
{
    char[] array = str.ToCharArray();
    Array.Reverse(array);
    return new string(array);
}
```

answered Oct 23 '08 at 0:48

Mike Two
35.4k 7 71 94

This is incredibly fast. – Michael Stum ♦ Oct 23 '08 at 1:03

Why the down vote? Not arguing it, but I'd rather learn from my mistakes. – Mike Two Jul 27 '11 at 12:38

@MooingDuck - thanks for explaining, but I don't know what you mean by code points. Also could you elaborate on "many other things". – Mike Two Mar 2 '13 at 1:21

@MooingDuck I looked up code points. Yes. You are correct. It does not handle code points. It is tough to determine all of the requirements for such a simple looking question. Thanks for the feedback – Mike Two Mar 2 '13 at 1:28

10

```
public static string Reverse(string input)
{
    return string.Concat(Enumerable.Reverse(input));
}
```

Of course you can extend string class with Reverse method

```
public static class StringExtensions
{
    public static string Reverse(this string input)
    {
        return string.Concat(Enumerable.Reverse(input));
    }
}
```

answered Apr 9 '13 at 17:47

Vlad Bezden
32.4k 11 139 115

`Enumerable.Reverse(input)` is equal to `input.Reverse()` – fubo May 17 '18 at 11:41

6

Immediate window of some VS versions.

```
string s = "Blah";
s = new string(s.ToCharArray().Reverse().ToArray());
```

answered Apr 2 '13 at 0:45

B H
922 12 18

No comment explaining reasoning for a down vote? – B H Apr 29 '13 at 18:44

1 Some guy took the time to downvote every answer (mine included) without explaining why. – Marcel Valdez Orozco May 15 '13 at 15:54

This is not really in place, since you are creating a `new string` – mbadawi23 Sep 8 '18 at 16:52

5

Had to submit a recursive example:

```
private static string Reverse(string str)
{
    if (str.IsNullOrEmpty(str) || str.Length == 1)
        return str;
    else
        return str[str.Length - 1] + Reverse(str.Substring
}
```

edited May 6 '15 at 10:46

Abel
43.2k 17 114 199

answered Oct 23 '08 at 0:53

1 string of Length 0 are not handled – bohdan_trotsenko May 15 '13 at 14:07

5

"Best" can depend on many things, but here are few more short alternatives ordered from fast to slow:

```
string s = "z̆ăl̈ĭg̈ȯ😀😆", pattern = @"(?s).(?<=(?:.(?=.*$(?
\p{M}*)\1?))))*)";

string s1 = string.Concat(s.Reverse());

string s2 = Microsoft.VisualBasic.Strings.StrReverse(s);

string s3 = string.Concat(StringInfo.ParseCombiningChract
    .Select(i => StringInfo.GetNextTextElement(s, i)));

string s4 = Regex.Replace(s, pattern, "$2").Remove(s.Lengt
```

answered Oct 1 '17 at 18:38
Slai
15.8k 3 25 37

4

Sorry for long post, but this might be interesting

```
using System;
using System.Collections.Generic;
using System.Diagnostics;
using System.Text;

namespace ConsoleApplication1
{
    class Program
```

```
        Array.Reverse(chars);
        return new string(chars);
    }

    public static string ReverseUsingCharacterBuffer(s
    {
        char[] charArray = new char[text.Length];
        int inputStrLength = text.Length - 1;
        for (int idx = 0; idx <= inputStrLength; idx++
        {
            charArray[idx] = text[inputStrLength - idx
        }
        return new string(charArray);
    }

    public static string ReverseUsingStringBuilder(str
    {
        if (string.IsNullOrEmpty(text))
        {
            return text;
        }

        StringBuilder builder = new StringBuilder(text
        for (int i = text.Length - 1; i >= 0; i--)
        {
            builder.Append(text[i]);
        }

        return builder.ToString();
    }

    private static string ReverseUsingStack(string inpu
    {
        Stack<char> resultStack = new Stack<char>();
        foreach (char c in input)
        {
            resultStack.Push(c);
        }

        StringBuilder sb = new StringBuilder();
        while (resultStack.Count > 0)
        {
            sb.Append(resultStack.Pop());
        }
        return sb.ToString();
    }
```

```
            char[] charArray = text.ToCharArray();
            int length = text.Length - 1;
            for (int i = 0; i < length; i++, length--)
            {
                charArray[i] ^= charArray[length];
                charArray[length] ^= charArray[i];
                charArray[i] ^= charArray[length];
            }

            return new string(charArray);
        }


        static void Main(string[] args)
        {
            string testString = string.Join(";", new strin
                new string('a', 100),
                new string('b', 101),
                new string('c', 102),
                new string('d', 103),
            });
            int cycleCount = 100000;

            Stopwatch stopwatch = new Stopwatch();
            stopwatch.Start();
            for (int i = 0; i < cycleCount; i++)
            {
                ReverseUsingCharacterBuffer(testString);
            }
            stopwatch.Stop();
            Console.WriteLine("ReverseUsingCharacterBuffer
stopwatch.ElapsedMilliseconds + "ms");

            stopwatch.Reset();
            stopwatch.Start();
            for (int i = 0; i < cycleCount; i++)
            {
                ReverseUsingArrayClass(testString);
            }
            stopwatch.Stop();
            Console.WriteLine("ReverseUsingArrayClass: " +
+ "ms");

            stopwatch.Reset();
            stopwatch.Start();
            for (int i = 0; i < cycleCount; i++)
            stopwatch.Stop();
            Console.WriteLine("ReverseUsingStringBuilder: 
stopwatch.ElapsedMilliseconds + "ms");

            stopwatch.Reset();
            stopwatch.Start();
            for (int i = 0; i < cycleCount; i++)
            {
                ReverseUsingStack(testString);
            }
            stopwatch.Stop();
            Console.WriteLine("ReverseUsingStack: " + stop
"ms");

            stopwatch.Reset();
            stopwatch.Start();
            for (int i = 0; i < cycleCount; i++)
            {
                ReverseUsingXOR(testString);
            }
            stopwatch.Stop();
            Console.WriteLine("ReverseUsingXOR: " + stopwa
"ms");
        }
    }
}
```

Results:

- ReverseUsingCharacterBuffer: 346ms
- ReverseUsingArrayClass: 87ms
- ReverseUsingStringBuilder: 824ms
- ReverseUsingStack: 2086ms
- ReverseUsingXOR: 319ms

answered Oct 23 '08 at 1:17

aku
101k 30 158 199

on the length of the string as well as the algorithm, it would be interesting to graph it. I still think Array.Reverse will be fastest in all cases ... – Sam Saffron Oct 23 '08 at 1:43

"will be fastest in all cases" when magical TrySZReverse function (it's used in Reverse implementation) fails, Array.Reverse fallbacks to simple implementation involving boxing, so my method will win. However I don't know what is a condition to make TrySZReverse fail. – aku Oct 23 '08 at 2:03

Turns out its not fastest in all cases :), I updated my post. This still needs to be tested with unicode for both correctness and speed. – Sam Saffron Oct 23 '08 at 2:49

4

```
public string Reverse(string input)
{
    char[] output = new char[input.Length];

    int forwards = 0;
    int backwards = input.Length - 1;

    do
    {
        output[forwards] = input[backwards];
        output[backwards] = input[forwards];
    }while(++forwards <= --backwards);

    return new String(output);
}

public string DotNetReverse(string input)
{
    char[] toReverse = input.ToCharArray();
    Array.Reverse(toReverse);
    return new String(toReverse);
}

public string NaiveReverse(string input)
{
    char[] outputArray = new char[input.Length];
```

```
        }

        return new String(outputArray);
    }

    public string RecursiveReverse(string input)
    {
        return RecursiveReverseHelper(input, 0, input.Length -
    }

    public string RecursiveReverseHelper(string input, int sta
    {
        if (startIndex == endIndex)
        {
            return "" + input[startIndex];
        }

        if (endIndex - startIndex == 1)
        {
            return "" + input[endIndex] + input[startIndex];
        }

        return input[endIndex] + RecursiveReverseHelper(input,
    + input[startIndex];
    }


    void Main()
    {
        int[] sizes = new int[] { 10, 100, 1000, 10000 };
        for(int sizeIndex = 0; sizeIndex < sizes.Length; sizeI
        {
            string holaMundo  = "";
            for(int i = 0; i < sizes[sizeIndex]; i+= 5)
            {
                holaMundo += "ABCDE";
            }

            string.Format("\n**** For size: {0} ****\n", sizes

            string odnuMaloh = DotNetReverse(holaMundo);

            var stopWatch = Stopwatch.StartNew();
            string result = NaiveReverse(holaMundo);
            ("Naive Ticks: " + stopWatch.ElapsedTicks).Dump();
            stopWatch.Restart();
            result = RecursiveReverse(holaMundo);
            ("Recursive Ticks: " + stopWatch.ElapsedTicks).Dum

            stopWatch.Restart();
            result = DotNetReverse(holaMundo);
            ("DotNet Reverse Ticks: " + stopWatch.ElapsedTicks
        }
    }
```

Output

For size: 10

```
Naive Ticks: 1
Efficient linear Ticks: 0
Recursive Ticks: 2
DotNet Reverse Ticks: 1
```

For size: 100

```
Naive Ticks: 2
Efficient linear Ticks: 1
Recursive Ticks: 12
DotNet Reverse Ticks: 1
```

For size: 1000

```
Naive Ticks: 5
Efficient linear Ticks: 2
Recursive Ticks: 358
DotNet Reverse Ticks: 9
```

For size: 10000

```
Naive Ticks: 32
Efficient linear Ticks: 28
Recursive Ticks: 84808
```

edited May 15 '13 at 15:52

answered Sep 8 '12 at 6:03

Marcel Valdez Orozco
2,680 19 23

4

Stack-based solution.

```
    public static string Reverse(string text)
    {
        var stack = new Stack<char>(text);
        var array = new char[stack.Count];

        int i = 0;
        while (stack.Count != 0)
        {
            array[i++] = stack.Pop();
        }

        return new string(array);
    }
```

Or

```
    public static string Reverse(string text)
    {
        var stack = new Stack<char>(text);
        return string.Join("", stack);
    }
```

edited Jun 17 '14 at 22:35

1,836 1 18 23

How about:

3

```
private string Reverse(string stringToReverse)
{
    char[] rev = stringToReverse.Reverse().ToArray();
    return new string(rev);
}
```

edited Feb 4 '11 at 1:03

Juliet
60.9k 40 184 222

answered Feb 3 '11 at 19:57

Zamir
67 3

Has the same codepoint issues as other methods above and will perform much slower than when doing a `ToCharArray` first. The LINQ enumerator is also way slower than `Array.Reverse()` . – Abel May 6 '15 at 10:44

3

I've made a C# port from Microsoft.VisualBasic.Strings. I'm not sure why they keep such useful functions (from VB) outside the System.String in Framework, but still under Microsoft.VisualBasic. Same scenario for financial functions (e.g. `Microsoft.VisualBasic.Financial.Pmt()`).

```
public static string StrReverse(this string expression)
```

```
        int srcIndex;

        var length = expression.Length;
        if (length == 0)
            return "";

        //CONSIDER: Get System.String to add a surrogate aware

        //Detect if there are any graphemes that need special
        for (srcIndex = 0; srcIndex <= length - 1; srcIndex++)
        {
            var ch = expression[srcIndex];
            var uc = char.GetUnicodeCategory(ch);
            if (uc == UnicodeCategory.Surrogate || uc == Unico
    uc == UnicodeCategory.SpacingCombiningMark || uc == Unicod
            {
                //Need to use special handling
                return InternalStrReverse(expression, srcIndex
            }
        }

        var chars = expression.ToCharArray();
        Array.Reverse(chars);
        return new string(chars);
    }

    ///<remarks>This routine handles reversing Strings contain
    /// GRAPHEME: a text element that is displayed as a single
    private static string InternalStrReverse(string expression
    {
        //This code can only be hit one time
        var sb = new StringBuilder(length) { Length = length }

        var textEnum = StringInfo.GetTextElementEnumerator(exp

        //Init enumerator position
        if (!textEnum.MoveNext())
        {
            return "";
        }

        var lastSrcIndex = 0;
        var destIndex = length - 1;

        //Copy up the first surrogate found
```

```
            destIndex -= 1;
            lastSrcIndex += 1;
        }

        //Now iterate through the text elements and copy them
        var nextSrcIndex = textEnum.ElementIndex;

        while (destIndex >= 0)
        {
            srcIndex = nextSrcIndex;

            //Move to next element
            nextSrcIndex = (textEnum.MoveNext()) ? textEnum.El
            lastSrcIndex = nextSrcIndex - 1;

            while (lastSrcIndex >= srcIndex)
            {
                sb[destIndex] = expression[lastSrcIndex];
                destIndex -= 1;
                lastSrcIndex -= 1;
            }
        }

        return sb.ToString();
    }
```

answered Jun 30 '14 at 21:31

natenho
2,105 13 32

+1, a nice addition! I just tried it with `string s = "abo\u0327\u0307\u035d\U0001d166cd"` , which contains the letter `o` followed by 3 combining diacritical marks in the BMP and one combining mark (MUSICAL SYMBOL COMBINING STEM) from the astral plane (non-BMP) and it keeps them intact. But the method is slow if such characters only appear at the end of a long string, as it has to go twice over the whole array. – Abel May 6 '15 at 12:44

3

This was what I came up with for C#. My first version before refactoring was horrible.

```
static String Reverse2(string str)
{
    int strLen = str.Length, elem = strLen - 1;
    char[] charA = new char[strLen];

    for (int i = 0; i < strLen; i++)
    {
        charA[elem] = str[i];
        elem--;
    }

    return new String(charA);
}
```

In Contrast to the `Array.Reverse` method below, it appears faster with 12 characters or less in the string. After 13 characters, the `Array.Reverse` starts to get faster, and it eventually dominates pretty heavily on speed. I just wanted to point out approximately where the speed starts to change.

```
static String Reverse(string str)
{
    char[] charA = str.ToCharArray();

    Array.Reverse(charA);

    return new String(charA);
}
```

At 100 characters in the string, it is faster than my version x 4. However, if I knew that the strings would always be less than 13 characters, I would use the one I made.

Testing was done with `Stopwatch` and 5000000 iterations. Also, I'm not sure if my version handles Surrogates or

Jason Ausborn
319 2 8

2

"Better way" depends on what is more important to you in your situation, performance, elegance, maintainability etc.

Anyway, here's an approach using Array.Reverse:

```
string inputString="The quick brown fox jumps over the laz
char[] charArray = inputString.ToCharArray();
Array.Reverse(charArray);

string reversed = new string(charArray);
```

answered Oct 23 '08 at 0:41

Ash
47.8k 29 140 165

2

If it ever came up in an interview and you were told you can't use Array.Reverse, i think this might be one of the fastest. It does not create new strings and iterates only over half of the array (i.e O(n/2) iterations)

```
    public static string ReverseString(string stringToReve
    {
        char[] charArray = stringToReverse.ToCharArray();
        int len = charArray.Length-1;
        int mid = len / 2;

        for (int i = 0; i < mid; i++)
        {
            char tmp = charArray[i];
        return new string(charArray);
    }
```

answered Feb 16 '12 at 7:21

mike01010

1,434 4 26 57

2 I'm pretty certain stringToReverse.ToCharArray() call will produce a O(N) execution time. – Marcel Valdez Orozco Sep 8 '12 at 5:35

In Big-O notation, the factor not dependent on `x`, or in your case, `n`, is not used. Your algorithm has performance `f(x) = x + ½x + C`, where C is some constant. Since both `C` and the factor `1½` are not dependent on `x`, your algorithm is `O(x)`. That does not mean that it won't be faster for any input of length `x`, but its performance is linearly dependent on the input length. To answer @MarcelValdezOrozco, yes, it is also `O(n)`, though it copies per 16-byte chunks to improve speed (it does not use a straight `memcpy` on the total length). – Abel May 6 '15 at 12:11

2

If you have a string that only contains ASCII characters, you can use this method.

```
public static string ASCIIReverse(string s)
{
    byte[] reversed = new byte[s.Length];

    int k = 0;
    for (int i = s.Length - 1; i >= 0; i--)
    {
        reversed[k++] = (byte)s[i];
    }

    return Encoding.ASCII.GetString(reversed);
}
```

Raz Megrelidze
1,836 1 18 23

2

```
public static string reverse(string s)
{
    string r = "";
    for (int i = s.Length; i > 0; i--) r += s[i - 1];
    return r;
}
```

answered Oct 29 '15 at 11:58
ddagsan
1,178 13 13

1

```
public static string Reverse2(string x)
        {
            char[] charArray = new char[x.Length];
            int len = x.Length - 1;
            for (int i = 0; i <= len; i++)
                charArray[i] = x[len - i];
            return new string(charArray);
        }
```

edited Apr 27 '12 at 7:46
antyrat
24.3k 9 59 71

answered Apr 27 '12 at 6:21
Shrini
19 1

1

```
private static string Reverse(string str)
        {
            string revStr = string.Empty;
            for (int i = str.Length - 1; i >= 0; i--)
            {
                revStr += str[i].ToString();
            }
            return revStr;
        }
```

Faster than above method

```
private static string ReverseEx(string str)
        {
            char[] chrArray = str.ToCharArray();
            int len = chrArray.Length - 1;
            char rev = 'n';
            for (int i = 0; i <= len/2; i++)
            {
                rev = chrArray[i];
                chrArray[i] = chrArray[len - i];
                chrArray[len - i] = rev;
            }
            return new string(chrArray);
        }
```

edited Feb 22 '13 at 2:09

answered Feb 21 '13 at 16:09

vikas
1,792 3 19 32

First of all what you have to understand is that str+= will resize your string memory to make space for 1 extra char.

The solution that some people might suggest is using StringBuilder. What string builder does when you perform a += is that it allocates much larger chunks of memory to hold the new character so that it does not need to do a reallocation every time you add a char.

If you really want a fast and minimal solution I'd suggest the following:

```
char[] chars = new char[str.Length];
for (int i = str.Length - 1, j = 0; i >= 0; --
{
    chars[j] = str[i];
}
str = new String(chars);
```

In this solution there is one initial memory allocation when the char[] is initialized and one allocation when the string constructor builds the string from the char array.

On my system I ran a test for you that reverses a string of 2 750 000 characters. Here are the results for 10 executions:

StringBuilder: 190K - 200K ticks

Char Array: 130K - 160K ticks

I also ran a test for normal String += but I abandoned it after 10 minutes with no output.

However, I also noticed that for smaller strings the StringBuilder is faster, so you will have to decide on the implementation based on the input.

Cheers

answered Oct 9 '14 at 5:51

1

As simple as this:

```
string x = "your string";
string x1 = "";
for(int i = x.Length-1 ; i >= 0; i--)
    x1 += x[i];
Console.WriteLine("The reverse of the string is:\n {0}", x
```

See the output.

answered Sep 21 '16 at 17:05
Raktim Biswas
3,649 5 18 28

2 Please be aware, that using this method you're creating `x.Length` times a new `string` object `x1` as you're not taking into account the inherent immutability of `string`. – Wim Ombelets Jul 31 '17 at 10:11

1 2 next

protected by Mureinik Feb 7 '15 at 8:47

Thank you for your interest in this question. Because it has attracted low-quality or spam answers that had to be removed, posting an answer now requires 10 reputation on this site (the association bonus does not count).

Would you like to answer one of these unanswered questions instead?