

## Re: fastest way to parse a file; Most efficient way to store the data?

*Source:*

<http://www.tech-archive.net/Archive/DotNet/microsoft.public.dotnet.languages.csharp/2004-12/3487.html>

---

*From:* Bruce Wood ([brucewood\\_at\\_canada.com](mailto:brucewood_at_canada.com))

*Date:* 12/14/04

Date: 14 Dec 2004 14:49:17 -0800

First off, let me point out that in computing, you are usually facing a tradeoff between memory and speed. The question, "What is the fastest, most memory-efficient way to do this?" is like asking, "What is the quickest, cheapest way to get to Paris?" (Assuming, of course, that you're not already in Paris. :) Sometimes the fastest way is the cheapest way, but more often than not you have to make a tradeoff between speed and cost: airplanes cost more than freighters. So it is with speed and memory: sometimes the fastest way is also the most memory-efficient, but more often than not you have to trade one for the other.

That said, your solution depends very much upon whether the field you're sorting on is always also the field that you require be unique. If it is, then I suggest that you use some form of tree structure (research the already-available .NET collections), which will sort your items on the fly and give you some indication of uniqueness at the same time. Since you have to sort anyway, you might as well do that and your uniqueness check all at once.

However, if you could potentially be determining uniqueness on one field and sorting on a different field, then there's no value in determining uniqueness using anything other than a hash table. A hash table will give you lightning-fast lookup capabilities to determine if you've already seen a key. There's only one thing faster, which is a very, very big B-tree, but it uses up tons of memory so I wouldn't go that way. Hash tables are robust and fast.

As for sorting, you should either build a tree structure or use the quicksort algorithm. Both methods are reasonably quick. I wouldn't suggest using insertion sort, which is what Nicholas was suggesting (sorry) because with a million records you'll definitely notice a performance difference. The Array class contains a Sort method, but it doesn't mention which algorithm it uses, although I must suppose that if the MS people who wrote the Framework didn't use quicksort, or something even faster (yes, there are a few faster algorithms) then

microsoft.public.dotnet.languages.csharp: Re: fastest way to parse a file; Most efficient way to store the data?

they're not too sharp.

Finally, there's the problem of storage. Yes, you can parse each line and blow it out into an array of strings, but then if you have to write it out again. As well, if you're doing a quick sort, you have to shuffle (potentially) large records around in memory.

Another way to solve the problem is to create a small class containing a string, an offset, and a length. If you use short integers for the offset and the length you can pare this down to 64 bits. When you read in a line, and you want to represent field #15, for example, make a new one of these objects, set the string pointer to your line you read in, and the offset and the length to indicate where your field starts and how long it is.

Now if you write an IComparer for this structure:

```
public class FieldComparer : IComparer
{
    public int Compare(object x, object y)
    {
        MyField field1 = (MyField)x;
        MyField field2 = (MyField)y;

        if (field1.Length != field2.Length)
        {
            return field1.Length - field2.Length;
        }
        else
        {
            return String.Compare(field1.String, field1.Offset,
                field2.String, field2.Offset, field2.Length);
        }
    }
}
```

I'm using a class for Field rather than a struct to avoid boxing and unboxing in the standard Compare method.

So, assuming that you have to determine uniqueness on one field and sort on another field, here is how I would do it.

Make a Hashtable that will have Field objects as its keys. The values stored in the Hashtable don't matter, so you might as well use the lines you're reading from the file. This won't result in any extra storage for the lines, because if you're storing them as strings then the runtime will share points so long as the input lines themselves never change.

Make an ArrayList that will hold the Field objects for the fields you want to eventually sort on.

Re: fastest way to parse a file; Most efficient way to store the data?

microsoft.public.dotnet.languages.csharp: Re: fastest way to parse a file; Most efficient way to store the data?

Read each line, find the field you need to sort on, and the field you need to verify as unique. Create a Field object for each of them. Check to see if the Field object for your unique field is already stored in the Hashtable, and add it if it isn't. Add the Field object for your sort field to the ArrayList using the Add method. You didn't say whether you want the record in the output set only if the unique field is the first occurrence, but you can determine that here because you already tried to put in the hash table.

When you've read all the lines, use ArrayList.Sort to sort the array list using an instance of the IComparer class that you created above. This will take a while, but it's faster than insertion sort or any other sort method that you might roll yourself.

Run through the array list and feed the records one by one to some sort of output object, which will know which fields to pick out and display to the user. Since your Field class contains the original string pointer for the line, you can recover the input line and scan it for the output fields that you want.

The only extra overhead that this introduces is that you scan the input line twice: once to get your unique / sort fields, and once to get your output fields. However, I doubt that this will create a significant performance hit. Not after all of that sorting. Anyway, there's my solution! Good luck!