How many ways are there to sort GUIDs? How much time do you have?

devblogs.microsoft.com/oldnewthing/20190426-00

April 26, 2019

Raymond Chen

Suppose you want to sort GUIDs, say because they are a key in an ordered map. How many ways are there to order them?

Before we can even talk about how to order GUIDs, we need to figure out how we're going to represent them. You can take the view that a GUID is just an array of 16 bytes.[1]

```
 00  11  22  33  44  55  66  77  88  99  AA  BB  CC  DD  EE  FF
```

But the GUID structure itself groups them into four fields, one of which is an array.

```
typedef struct _GUID {
  DWORD Data1;
  WORD  Data2;
  WORD  Data3;
  BYTE  Data4[8];
} GUID;
```

This groups the bytes as follows:

```
 33221100  5544    7766    88  99  AA  BB  CC  DD  EE  FF
 ________  ____    ____    __  __  __  __  __  __  __  __
 Data1     Data2   Data3   Data4
```

The bytes in `Data1`, `Data2`, and `Data3` are flipped because Windows is little-endian.

And of course in Windows, it is common to represent GUIDs in their stringified form.

```
{  33221100  -  5544  -  7766  -  88  99  -  AA  BB  CC  DD  EE  FF  }
   ________     ____     ____     __  __     __  __  __  __  __  __
   Data1     -  Data2  -  Data3  -  Data4
```

Notice that the first three integer-sized groups are flipped, *but the fourth one isn't*. I'm always scared of that fourth group. (I'm not tempted to flip the last group because it's six bytes long, which is not the natural size of any integer type on Windows.)

Since the difference between the structured form and the string form is only in the placement of punctuation marks, and not in the byte ordering, I'll limit myself to byte-array representation and string representation.

Okay, now that we know how to represent GUIDs, we can start sorting them.

If you treat the GUID as an array of 16 bytes, then you can sort them with `memcmp`, which is a lexicographical sorting by bytes. The comparisons are made as unsigned values. (Thankfully, it never occurred to anyone to try to sort GUID components as signed integers!)[3] This means that the following list of GUIDs is sorted according to `memcmp`:

Byte array	String
`00 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF`	`{FFFFFF00-FFFF-FFFF-FFFF-FFFFFFFFFFFF}`
`FF 00 FF FF FF FF FF FF FF FF FF FF FF FF FF FF`	`{FFFF00FF-FFFF-FFFF-FFFF-FFFFFFFFFFFF}`
`FF FF 00 FF FF FF FF FF FF FF FF FF FF FF FF FF`	`{FF00FFFF-FFFF-FFFF-FFFF-FFFFFFFFFFFF}`
`FF FF FF 00 FF FF FF FF FF FF FF FF FF FF FF FF`	`{00FFFFFF-FFFF-FFFF-FFFF-FFFFFFFFFFFF}`
`FF FF FF FF 00 FF FF FF FF FF FF FF FF FF FF FF`	`{FFFFFFFF-FF00-FFFF-FFFF-FFFFFFFFFFFF}`
`FF FF FF FF FF 00 FF FF FF FF FF FF FF FF FF FF`	`{FFFFFFFF-00FF-FFFF-FFFF-FFFFFFFFFFFF}`
`FF FF FF FF FF FF 00 FF FF FF FF FF FF FF FF FF`	`{FFFFFFFF-FFFF-FF00-FFFF-FFFFFFFFFFFF}`
`FF FF FF FF FF FF FF 00 FF FF FF FF FF FF FF FF`	`{FFFFFFFF-FFFF-00FF-FFFF-FFFFFFFFFFFF}`
`FF FF FF FF FF FF FF FF 00 FF FF FF FF FF FF FF`	`{FFFFFFFF-FFFF-FFFF-00FF-FFFFFFFFFFFF}`
`FF FF FF FF FF FF FF FF FF 00 FF FF FF FF FF FF`	`{FFFFFFFF-FFFF-FFFF-FF00-FFFFFFFFFFFF}`
`FF FF FF FF FF FF FF FF FF FF 00 FF FF FF FF FF`	`{FFFFFFFF-FFFF-FFFF-FFFF-00FFFFFFFFFF}`
`FF FF FF FF FF FF FF FF FF FF FF 00 FF FF FF FF`	`{FFFFFFFF-FFFF-FFFF-FFFF-FF00FFFFFFFF}`

FF FF FF FF FF FF FF FF FF FF FF FF 00 FF FF FF	{FFFFFFFF-FFFF-FFFF-FFFF-FFFF00FFFFFF}
FF FF FF FF FF FF FF FF FF FF FF FF FF 00 FF FF	{FFFFFFFF-FFFF-FFFF-FFFF-FFFFFF00FFFF}
FF FF FF FF FF FF FF FF FF FF FF FF FF FF 00 FF	{FFFFFFFF-FFFF-FFFF-FFFF-FFFFFFFF00FF}
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF 00	{FFFFFFFF-FFFF-FFFF-FFFF-FFFFFFFFFF00}

The .NET Framework `System.Guid.CompareTo` method compares the structure members lexicographically, and the bytes of the array are also sorted lexicographically. The sorted array for `System.Guid` looks like this:

Byte array	String
FF FF FF 00 FF FF FF FF FF FF FF FF FF FF FF FF	{00FFFFFF-FFFF-FFFF-FFFF-FFFFFFFFFFFF}
FF FF 00 FF FF FF FF FF FF FF FF FF FF FF FF FF	{FF00FFFF-FFFF-FFFF-FFFF-FFFFFFFFFFFF}
FF 00 FF FF FF FF FF FF FF FF FF FF FF FF FF FF	{FFFF00FF-FFFF-FFFF-FFFF-FFFFFFFFFFFF}
00 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF	{FFFFFF00-FFFF-FFFF-FFFF-FFFFFFFFFFFF}
FF FF FF FF FF 00 FF FF FF FF FF FF FF FF FF FF	{FFFFFFFF-00FF-FFFF-FFFF-FFFFFFFFFFFF}
FF FF FF FF 00 FF FF FF FF FF FF FF FF FF FF FF	{FFFFFFFF-FF00-FFFF-FFFF-FFFFFFFFFFFF}
FF FF FF FF FF FF FF 00 FF FF FF FF FF FF FF FF	{FFFFFFFF-FFFF-00FF-FFFF-FFFFFFFFFFFF}
FF FF FF FF FF FF 00 FF FF FF FF FF FF FF FF FF	{FFFFFFFF-FFFF-FF00-FFFF-FFFFFFFFFFFF}
FF FF FF FF FF FF FF FF 00 FF FF FF FF FF FF FF	{FFFFFFFF-FFFF-FFFF-00FF-FFFFFFFFFFFF}
FF FF FF FF FF FF FF FF FF 00 FF FF FF FF FF FF	{FFFFFFFF-FFFF-FFFF-FF00-FFFFFFFFFFFF}
FF FF FF FF FF FF FF FF FF FF 00 FF FF FF FF FF	{FFFFFFFF-FFFF-FFFF-FFFF-00FFFFFFFFFF}
FF FF FF FF FF FF FF FF FF FF FF 00 FF FF FF FF	{FFFFFFFF-FFFF-FFFF-FFFF-FF00FFFFFFFF}

FF FF FF FF FF FF FF FF FF FF FF FF 00 FF FF FF	{FFFFFFFF-FFFF-FFFF-FFFF-FFFF00FFFFFF}
FF FF FF FF FF FF FF FF FF FF FF FF FF 00 FF FF	{FFFFFFFF-FFFF-FFFF-FFFF-FFFFFF00FFFF}
FF FF FF FF FF FF FF FF FF FF FF FF FF FF 00 FF	{FFFFFFFF-FFFF-FFFF-FFFF-FFFFFFFF00FF}
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF 00	{FFFFFFFF-FFFF-FFFF-FFFF-FFFFFFFFFF00}

Next is string sorting. You will use a case-insensitive sort if you want to preserve your sanity. A memberwise lexicographical sort of the structure is equivalent to sorting the strings because, as we noted earlier, the stringification is the same as the structure version, just with additional punctuation. So the above list is also sorted according to case-insensitive stringification. Hooray, two sorting algorithms agree on something!

Next up is `System. Data. SqlTypes. SqlGuid`. Yes, SQL has its own GUID, because it's SQL. Not only does it have its own GUID, it has its own GUID sorting algorithm. Because it's SQL. And it doesn't call it a GUID, but rather calls it `uniqueidentifier`. Again, because it's SQL.

SQL sorts GUIDs by breaking the stringified version into groups, sorting groups right to left, and sorting bytewise within each group. I want to know what they were thinking when they came up with this.[2] You end up with this sorted array for `System. Data. SqlTypes. SqlGuid`:

Byte array	String
FF FF FF FF FF FF FF FF FF FF 00 FF FF FF FF FF	{FFFFFFFF-FFFF-FFFF-FFFF-00FFFFFFFFFF}
FF FF FF FF FF FF FF FF FF FF FF 00 FF FF FF FF	{FFFFFFFF-FFFF-FFFF-FFFF-FF00FFFFFFFF}
FF FF FF FF FF FF FF FF FF FF FF FF 00 FF FF FF	{FFFFFFFF-FFFF-FFFF-FFFF-FFFF00FFFFFF}
FF FF FF FF FF FF FF FF FF FF FF FF FF 00 FF FF	{FFFFFFFF-FFFF-FFFF-FFFF-FFFFFF00FFFF}
FF FF FF FF FF FF FF FF FF FF FF FF FF FF 00 FF	{FFFFFFFF-FFFF-FFFF-FFFF-FFFFFFFF00FF}
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF 00	{FFFFFFFF-FFFF-FFFF-FFFF-FFFFFFFFFF00}
FF FF FF FF FF FF FF FF 00 FF FF FF FF FF FF FF	{FFFFFFFF-FFFF-FFFF-00FF-FFFFFFFFFFFF}

FF FF FF FF FF FF FF FF FF 00 FF FF FF FF FF FF	{FFFFFFFF-FFFF-FFFF-FF00-FFFFFFFFFFFF}
FF FF FF FF FF FF 00 FF FF FF FF FF FF FF FF FF	{FFFFFFFF-FFFF-FF00-FFFF-FFFFFFFFFFFF}
FF FF FF FF FF FF FF 00 FF FF FF FF FF FF FF FF	{FFFFFFFF-FFFF-00FF-FFFF-FFFFFFFFFFFF}
FF FF FF FF 00 FF FF FF FF FF FF FF FF FF FF FF	{FFFFFFFF-FF00-FFFF-FFFF-FFFFFFFFFFFF}
FF FF FF FF FF 00 FF FF FF FF FF FF FF FF FF FF	{FFFFFFFF-00FF-FFFF-FFFF-FFFFFFFFFFFF}
00 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF	{FFFFFF00-FFFF-FFFF-FFFF-FFFFFFFFFFFF}
FF 00 FF FF FF FF FF FF FF FF FF FF FF FF FF FF	{FFFF00FF-FFFF-FFFF-FFFF-FFFFFFFFFFFF}
FF FF 00 FF FF FF FF FF FF FF FF FF FF FF FF FF	{FF00FFFF-FFFF-FFFF-FFFF-FFFFFFFFFFFF}
FF FF FF 00 FF FF FF FF FF FF FF FF FF FF FF FF	{00FFFFFF-FFFF-FFFF-FFFF-FFFFFFFFFFFF}

The last GUID sorting algorithm I could find is used by the `Platform::Guid` value class. Its `operator<` treats the GUID as if it were four 32-bit unsigned integers, and sorts them lexicographically. This sort order was designed for performance.

Byte array	String
FF FF FF 00 FF FF FF FF FF FF FF FF FF FF FF FF	{00FFFFFF-FFFF-FFFF-FFFF-FFFFFFFFFFFF}
FF FF 00 FF FF FF FF FF FF FF FF FF FF FF FF FF	{FF00FFFF-FFFF-FFFF-FFFF-FFFFFFFFFFFF}
FF 00 FF FF FF FF FF FF FF FF FF FF FF FF FF FF	{FFFF00FF-FFFF-FFFF-FFFF-FFFFFFFFFFFF}
00 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF	{FFFFFF00-FFFF-FFFF-FFFF-FFFFFFFFFFFF}
FF FF FF FF FF FF FF 00 FF FF FF FF FF FF FF FF	{FFFFFFFF-FFFF-00FF-FFFF-FFFFFFFFFFFF}
FF FF FF FF FF FF 00 FF FF FF FF FF FF FF FF FF	{FFFFFFFF-FFFF-FF00-FFFF-FFFFFFFFFFFF}
FF FF FF FF FF 00 FF FF FF FF FF FF FF FF FF FF	{FFFFFFFF-00FF-FFFF-FFFF-FFFFFFFFFFFF}

FF FF FF FF 00 FF FF FF FF FF FF FF FF FF FF	{FFFFFFFF-FF00-FFFF-FFFF-FFFFFFFFFFFF}
FF FF FF FF FF FF FF FF FF FF FF 00 FF FF FF	{FFFFFFFF-FFFF-FFFF-FFFF-FF00FFFFFFFF}
FF FF FF FF FF FF FF FF FF FF FF 00 FF FF FF	{FFFFFFFF-FFFF-FFFF-FFFF-00FFFFFFFFFF}
FF FF FF FF FF FF FF FF FF 00 FF FF FF FF FF	{FFFFFFFF-FFFF-FFFF-FF00-FFFFFFFFFFFF}
FF FF FF FF FF FF FF FF 00 FF FF FF FF FF FF	{FFFFFFFF-FFFF-FFFF-00FF-FFFFFFFFFFFF}
FF FF FF FF FF FF FF FF FF FF FF FF FF FF 00	{FFFFFFFF-FFFF-FFFF-FFFF-FFFFFFFFFF00}
FF FF FF FF FF FF FF FF FF FF FF FF FF 00 FF	{FFFFFFFF-FFFF-FFFF-FFFF-FFFFFFFF00FF}
FF FF FF FF FF FF FF FF FF FF FF FF 00 FF FF	{FFFFFFFF-FFFF-FFFF-FFFF-FFFFFF00FFFF}
FF FF FF FF FF FF FF FF FF FF FF 00 FF FF FF	{FFFFFFFF-FFFF-FFFF-FFFF-FFFF00FFFFFF}

Okay, let's try to summarize all these results. I'm going to number the bytes of the GUID in the order they are compared, where `00` is the byte compared first (most significant), and `FF` is the byte compared last (least significant).

Algorithm	Byte array	String
`memcmp`	00 11 22 33 44 55 66 77 88 99 AA BB CC DD EE FF	{33221100-5544-7766-8899-AABBCCDDEEFF}
`System.Guid` string	33 22 11 00 55 44 77 66 88 99 AA BB CC DD EE FF	{00112233-4455-6677-8899-AABBCCDDEEFF}
`SqlGuid`	CC DD EE FF AA BB 88 99 66 77 00 11 22 33 44 55	{FFEEDDCC-BBAA-9988-6677-001122334455}
`Platform::Guid`	33 22 11 00 77 66 55 44 BB AA 99 88 FF EE DD CC	{00112233-6677-4455-BBAA-9988FFEEDDCC}

If you find another GUID sorting algorithm in common use, let me know. Or maybe I'm better off not knowing.

Bonus chatter: The result of sorting GUIDs is not generally meaningful, but some algorithms and data structures require keys to be sortable. For example, binary search and `std::map` require that the key space be totally-ordered.

[1] Although that isn't quite right because GUIDs must be 4-byte aligned, and bytes don't come with that restriction.

[2] Maybe they wanted to group together GUIDs from the same system? In type 1 GUIDs, the final six bytes identify the machine that generated the GUID.

[3] **Update**: Turns out I was wrong. There exist people who really are that crazy.

Raymond Chen

Follow