

## Sorting Algorithm Coding Standards

Microsoft provides some coding standards for C# development, and following them is generally seen as the ideal way to produce C# software. I followed these standards in the production of this software.

### Layout Conventions

The C# standards specify that there should only be one statement or declaration per line, with four-space indentation, having at least one blank line between method definitions and property definitions, and finally separating clauses in an expression statement to make them more apparent (mairaw, et al., 2015). These conventions apply to layout and improve readability.

```
private void button1_Click(object sender, EventArgs e)
{
    NumberList.Clear();

    for (var i = 0; i < 10; i++) {
        NumberList.Add(Rnd.Next(0, 50000));
        progressBar2.Value = i + 1;
    }
}
```

```
for (j = i - 1; (j ≥ 0) && (flag ≠ 1);)
```

These snippets demonstrate these standards, with only one statement or declaration per line, a space between expressions and declarations, 4-space indents. Additionally, the separate clauses in the expression statement is used.

### Naming Conventions

For my project, I used camelCase for private variables and parameters, and PascalCase for global variables. Since I didn't use static variables, there wasn't a need to include them in the convention. As well as this, I tried to keep variable names as simple as possible, being descriptive with them. A good set of examples of these variables include tempList, a private variable used in insertion sort and bubble sort, swaps and comparisons, which are both used in the sorts and describe the amount of sorts and comparisons, and NumberList, which is the public list that contains all the numbers to be sorted.

Similarly, the methods are named using various conventions. The events are denoted by the object they're events for, an underscore, then a description of the event (button1\_Click), whereas the sorting and sort check methods use Pascal case, regardless of being public or private within the class. The class and namespace are both Pascal case additionally (mairaw, et al., 2008).

```
private void CheckSorted (List<int> list) {}  
private void BubbleSort() {}  
namespace SortingAlgorithmHNC {}  
public partial class Form1 : Form {}
```

## Type Conventions


The C# conventions document does specify to use var when variables are clearly defined as a specific type, along with using signed integers rather than unsigned integers. I didn't follow these conventions too closely, using explicit type definitions where possible, along with the usage of unsigned integers for swap and comparison counts.

The reasoning behind this was because most integers are initialised without a value, given a value later on, which is somewhat compliant with the conventions.

### Implicitly Typed Local Variables

- Use **implicit typing** for local variables when the type of the variable is obvious from the right side of the assignment, or when the precise type is not important.

C#

 Copy

```
// When the type of a variable is clear from the context, use var  
// in the declaration.  
var var1 = "This is clearly a string.";  
var var2 = 27;  
var var3 = Convert.ToInt32(Console.ReadLine());
```

This states that implicit typing should only be used when the type of the variable is obvious, which isn't true for the code I produced.

As well as this, I used unsigned integers specifically due to the nature of the comparison/swaps values only ever being positive. Given the size of the lists involved and the potential for the value to go past  $2^{31}$ , the upper limit of a signed integer, I changed it to be an unsigned integer, with the limits of 0 to  $2^{32}$ , fitting my project much better.

## Bibliography

mairaw, et al., 2015. *C# Coding Conventions (C# Programming Guide)*. [Online] Available at: <https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/inside-a-program/coding-conventions> [Accessed 9th January 2020].

mairaw, et al., 2008. *General Naming Conventions*. [Online] Available at: <https://docs.microsoft.com/en-us/dotnet/standard/design-guidelines/general-naming-conventions> [Accessed 9th January 2020].