

Sorting Algorithm Comparison

Below is a screenshot of my executable that compares Bubble Sorts and Insertion Sorts, using a list of 50,000 integers. The two sorts are fairly similar in method and complexity, however there's a clear difference between the two using this tool, which measures the amount of swaps performed by each sort, which is the same in this case, the amount of comparisons performed, and the amount of time in milliseconds that each sort takes.

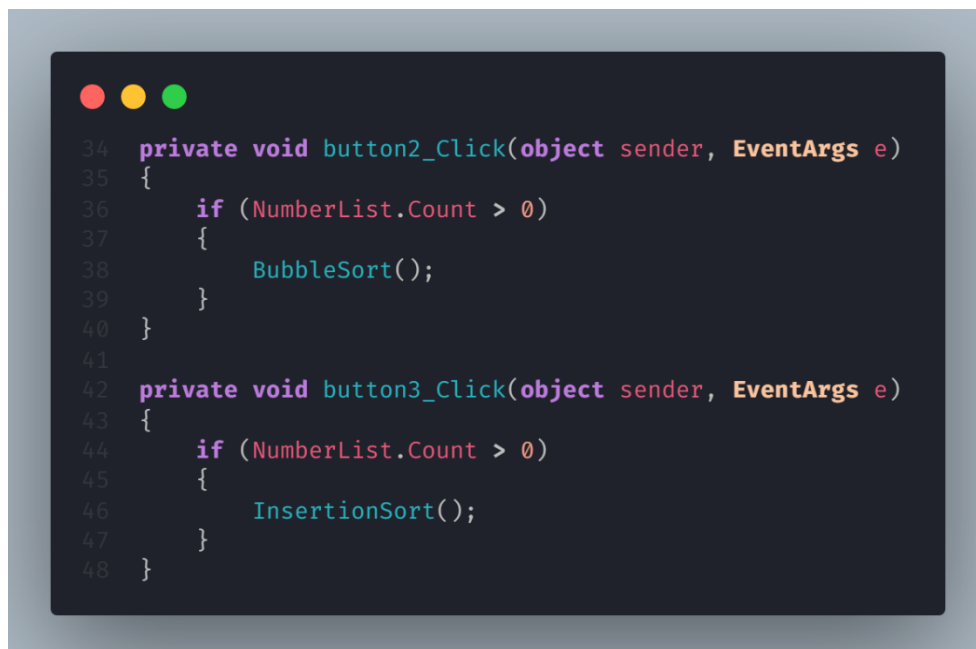
Performance Metrics	
Swaps	Swaps
627265166	627265166
Comparisons	Comparisons
1249975000	627315154
Time Taken (ms)	Time Taken (ms)
50330.3296	21746.4291

The sorts rely on having a list of integers to sort, so the button at the top (button1) along with the first few lines of the script focus on generating 50,000 integers using random number generation so that the sorting algorithms have a sample to sort. Each algorithm uses the same list of integers, which must be generated before the sorting functions can be called.

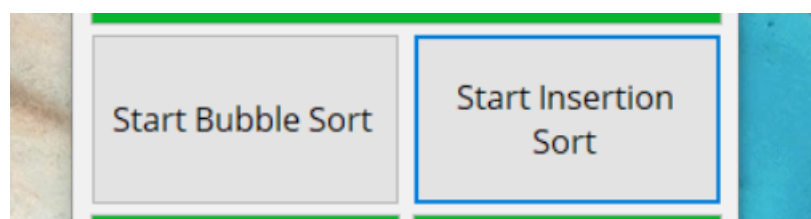
```
15 List<int> NumberList;  
16 Random rnd = new Random();  
17  
18 public Form1()  
19 {  
20     InitializeComponent();  
21     NumberList = new List<int>();  
22 }  
23  
24 private void button1_Click(object sender, EventArgs e)  
25 {  
26     NumberList.Clear();  
27     for (int i = 0; i < 50000; i++) {  
28         NumberList.Add(rnd.Next(0, 50000));  
29         progressBar2.Value = i + 1;  
30     }  
31     Console.WriteLine(NumberList.Count);  
32 }
```

There are two lines outside of methods here, declaring the integer list as well as declaring a new random object, which creates a pseudo-random seed for which later random numbers can be generated using the method `Random.Next`, which can be provided with arguments that effectively modulo and offset the answer, like what is required in C++ (cplusplus.com, n.d.) (Microsoft, n.d.). In this case, `rnd.Next(0, 50000)` is used, which produces a random number between 0 and 50,000. These values are arbitrary and the upper limit of 50,000 is only used to have an average amount of variance in the values in the list itself.

`Form1()` initialises the form itself, as well as creating the new list itself, so that `NumberList.Clear()` and `Add()` can be used in the next function. `button1_Click()` is an event called by the button at the top of the form, and in this case it clears the list, then adds 50,000 integers using the random function between 0 and 50,000, then updates the progressbar below the button. The `Console.WriteLine` at the bottom is for debug purposes, outputting 50000 into the debug console.



```
34 private void button2_Click(object sender, EventArgs e)
35 {
36     if (NumberList.Count > 0)
37     {
38         BubbleSort();
39     }
40 }
41
42 private void button3_Click(object sender, EventArgs e)
43 {
44     if (NumberList.Count > 0)
45     {
46         InsertionSort();
47     }
48 }
```



In the code, these sorts are modular; having two buttons (as shown) that check if the list has any numbers to sort in the first place, then calls the respective sort function depending on which button is clicked; in the form, the left button is `button2`, and the right button is `button3`. The labels that these buttons have reflect which function is called. This section could be improved by passing by reference, using a dropdown for which sorting algorithm is used or compared, however in this case it's simpler to have fixed values for the buttons.

Sorting Algorithm Implementation

```
public void BubbleSort()
{
    DateTime startTime = DateTime.Now;
    int temp = 0, swaps = 0, comparisons = 0;
    List<int> bTempList = NumberList.Cast<int>().ToList();

    for (int j = 0; j <= bTempList.Count - 2; j++)
    {
        for (int i = 0; i <= bTempList.Length - 2; i++)
        {
            if (bTempList[i] > bTempList[i + 1])
            {
                bTempList = bTempList[i + 1];
                bTempList[i + 1] = bTempList[i];
                bTempList[i] = temp;
                swaps++;
            }
            comparisons++;
        }
        progressBarLeft.Value = j;
    }

    DateTime endTime = DateTime.Now;
    progressBarLeft.Value = 50000;
    textBox1.Text = swaps.ToString();
    textBox3.Text = comparisons.ToString();
    textBox6.Text = (endTime - startTime).TotalMilliseconds.ToString();
}
```

(Boyini, 2018)

```
public void InsertionSort()
{
    DateTime startTime = DateTime.Now;
    int swaps = 0, comparisons = 0, temp, j, flag;
    List<int> iTempList = NumberList.Cast<int>().ToList();

    for (int i = 1; i < iTempList.Count; i++)
    {
        temp = iTempList[i];
        flag = 0;
        for (j = i - 1; j >= 0 && flag != 1;)
        {
            if (temp < iTempList[j])
            {
                iTempList[j + 1] = iTempList[j];
                j--;
                iTempList[j + 1] = temp;
                swaps++;
            }
            else flag = 1;
            comparisons++;
        }
        progressBarRight.Value = i;
    }

    DateTime endTime = DateTime.Now;
    progressBarRight.Value = 50000;
    textBox2.Text = swaps.ToString();
    textBox4.Text = comparisons.ToString();
    textBox5.Text = (endTime - startTime).TotalMilliseconds.ToString();
}
```

(Thakur, 2018)

Above are code snippets of both the sorting algorithms used in this project, Bubble sort on the left, and Insertion sort on the right. Interestingly, these algorithms are similar in process, however they differ in application. Both algorithms have a sorted pool and an unsorted pool, with bubble sort having the bubbles gravitate towards the top or bottom of the list, whereas insertion sort has a separate list. This diagram shows the difference, with the green section being sorted, and the red section being unsorted. You can see that insertion sort involves having two lists, the green one being temporary and the red one being the original list, whereas bubble sort keeps everything in one list. It's worth noting that the numbers in this list may be incorrect in a real application, but the concept is the important part.

```
- insertion sort
[ 1, 2, 3 ] [ 6, 3, 9, 10, 29, 4 ]

- bubble sort
[ 1, 2, 3, 6, 3, 9, 10, 29, 4 ]
```

The number of swaps between Bubble sort and Insertion sort should be very similar due to this similarity in the process, however there would likely be differences in the amount of comparisons and the time taken for the process to be completed. To account for these values, the functions for each algorithm have a swap counter, a comparison counter, and a simple timer that checks how long they take to execute. Upon the completion of each sort, some disabled text boxes (which are disabled to prevent user input) update showing these three values.

Swaps	Swaps
627646236	627646236
Comparisons	Comparisons
2499900001	627696230
Time Taken (ms)	Time Taken (ms)
27343.8593	7726.9611

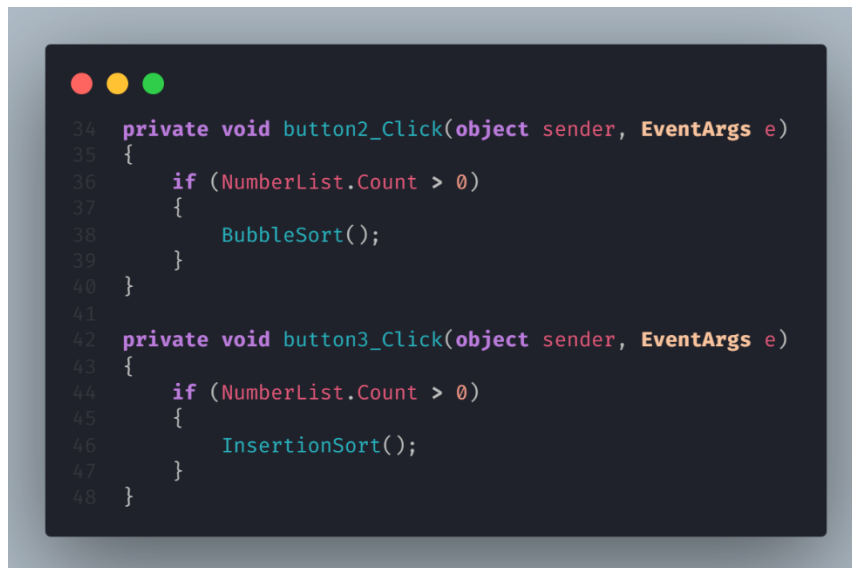
The values on the left side represent the Bubble sort algorithm, whereas the values on the right side represent the Insertion sort algorithm. This shows that, for an array size of 50,000 digits, 627,265,166 swaps take place for both algorithms, reflecting the similarities between the two. Importantly, the comparison count for the bubble sort is almost 4x higher than that for insertion sort, and the time for completion is 3.5x as long additionally. This reflects the efficiency of the two algorithms, showing that Insertion sort is a better implementation of a similar method.

This behaviour was unexpected at first, which could show that there's an error in the script that executes both sorts, however the comparison/swap increments are in the correct place, the variables are private to each method (BubbleSort() and InsertionSort()), and they both interact with different textboxes (1&3 for Bubble sort, 2&4 for Insertion sort). This shows that the similarities between the two are not caused by errors in the methods themselves.

In order to make sure that these numbers are accurate, both sorts require a method to check that the lists are sorted, which is included here. This snippet scans through the list and, should any value be bigger than the next value in the list, a popup window will show up, demonstrating that the lists are not sorted. This doesn't happen with either of these sorting algorithms, so it's clear that the sorting is applied correctly.

```
bool notSorted = false;
var formPopup = new Form();
for (int i=0; i < iTempList.Count - 1; i++) {
    if (iTempList[i] > iTempList[i+1]) {
        notSorted = true;
    }
}
if (notSorted) {
    formPopup.Show(this);
}
```

The modularity of the sorting algorithms used allows for the easy replacement of algorithms for each button. Referring back to the button event calls, it's easy to swap out the sorting method that's called by adding a new method entirely, such as `CocktailSort()`, or by modifying the method directly to adapt the bubble sort or insertion sort algorithms. There's no pass by reference in these button calls, but due to the very small scale of this project I feel that would be unnecessary.

A screenshot of a code editor with a dark background and light-colored text. The code is C# and shows two event handler methods. The first method, `button2_Click`, calls `BubbleSort()` if `NumberList.Count > 0`. The second method, `button3_Click`, calls `InsertionSort()` if `NumberList.Count > 0`. The code is line-numbered from 34 to 48.

```
34 private void button2_Click(object sender, EventArgs e)
35 {
36     if (NumberList.Count > 0)
37     {
38         BubbleSort();
39     }
40 }
41
42 private void button3_Click(object sender, EventArgs e)
43 {
44     if (NumberList.Count > 0)
45     {
46         InsertionSort();
47     }
48 }
```

References

Boyini, K., 2018. *Bubble Sort program in C#*. [Online]
Available at: <https://www.tutorialspoint.com/Bubble-Sort-program-in-Chash>
[Accessed 30th December 2019].

cplusplus.com, n.d.. *C++ Function: rand*. [Online]
Available at: <http://www.cplusplus.com/reference/cstdlib/rand/>
[Accessed 19th December 2019].

Microsoft, n.d.. *Random.Next Method*. [Online]
Available at: <https://docs.microsoft.com/en-us/dotnet/api/system.random.next?view=netframework-4.8>
[Accessed 19th December 2019].

Thakur, A., 2018. *Insertion Sort in C#*. [Online]
Available at: <https://www.tutorialspoint.com/insertion-sort-in-chash>
[Accessed 30th December 2019].