08112 Software Engineering ACW - Algorithm Analysis Report

Introduction

In this report I will evaluate the efficiency of three different algorithms. I will calculate the relationship between the data volume and the number of loops the algorithm has to perform.

Algorithm 1

This algorithm uses a brute force method to analyse the data. It uses three loops which manage the start position, end position and the subsequence profit values. This algorithm is highly inefficient due to that it calculates similar data again through each loop. E.g. During the first loop it will calculate a subtotal for weeks 1-5 and then for the next loop it will reset the subtotal and calculate it for weeks 1-6.

```
//subseq info
            bestTotal = 0;
            bestStart = 0;
            bestEnd = 0;
            loops = 0;

// Algorithm 1. The Obvious Approach
            for (int i = 0; i < data.Length; i++)
            {
                for (int j = i; j < data.Length; j++)
                {
                    double subTotal = 0;
                    for (int k = i; k <= j; k++)
                    {
                        subTotal += data[k];
                        loops++;
                    }

                    if (subTotal > bestTotal)
                    {
                        bestTotal = subTotal;
                        bestStart = i;
                        bestEnd = j;
                    }
                }
            }
```

Algorithm 2

This algorithm is slightly more efficient than the first algorithm because it saves the data from the last loop and simply adds onto it to calculate a subtotal. E.g. During the first loop it will calculate a subtotal for weeks 1-5 and then for the second loop it will add the profit for week 6 onto the subtotal.

```
//subseq info
            bestTotal = 0;
            bestStart = 0;
            bestEnd = 0;
            loops = 0;

// Algorithm 2. A More Efficient Alternative
            for (int i = 0; i < data.Length; i++)
            {
                double subTotal = 0;
                for (int j = i; j < data.Length; j++)
                {
                    subTotal += data[j];
                    loops++;

                    if (subTotal > bestTotal)
                    {
                        bestTotal = subTotal;
                        bestStart = i;
                        bestEnd = j;
                    }
                }
            }
```

Algorithm 3

This algorithm is the most efficient. It only has one loop and if the subtotal falls below 0 it will reset it to 0. This is done because any profits added onto the subtotal is worth more on their own.

```
//subseq info
            bestTotal = 0;
            bestStart = 0;
            bestEnd = 0;
            loops = 0;

// Algorithm 3. A Further Optimisation

            int startPos = 0; double subTotal = 0;

            for (int i = 0; i < data.Length; i++)
            {
                subTotal += data[i];

                if (subTotal > bestTotal)
                {
                    bestTotal = subTotal;
                    bestStart = startPos;
                    bestEnd = i;
                }

                if (subTotal < 0)
                {
                    startPos = i + 1;
                    subTotal = 0;
                }

                loops++;
            }
```

Analysis of Results

The following tables show the application output from each algorithm for each data set (26,52,104,208 weeks):

Data: 26	**Best Start**	**Best End**	**Best Total**
Algorithm 1	17	20	99.72
Algorithm 2	17	20	99.72
Algorithm 3	17	20	99.72

Data: 52	**Best Start**	**Best End**	**Best Total**
Algorithm 1	17	20	99.72
Algorithm 2	17	20	99.72
Algorithm 3	17	20	99.72

Data: 104	**Best Start**	**Best End**	**Best Total**
Algorithm 1	77	98	167.01
Algorithm 2	77	98	167.01
Algorithm 3	77	98	167.01

Data: 208	**Best Start**	**Best End**	**Best Total**
Algorithm 1	122	207	336.43
Algorithm 2	122	207	336.43
Algorithm 3	122	207	336.43

These tables prove that all the algorithms are equivalent in their application output and working correctly.

The following table show the number of operations (loops) each algorithm does for each of the data set:

	Algorithm 1	**Algorithm 2**	**Algorithm 3**
Data: 26	3276	351	26
Data: 52	24804	1378	52
Data: 104	192920	5460	104
Data: 208	152150	21736	208

The following table shows the relationship between the change (ratio) of operational workload that algorithm 1 does for each change (ratio) of input data volume:

Data Vol	**Ratio**	**Alg 1**	**Ratio 1**	**Alg 2**	**Ratio 2**	**Alg 3**	**Ratio 3**
26		3276		351		26	
52	2	24804	7.57	1378	3.93	52	2
104	2	192920	7.78	5460	3.97	104	2
208	2	1521520	7.89	21736	3.99	208	2
Limit	2		8		4		2

Big "O" Notation

Big "O" Notation represents how much time (or space) required by an algorithm varies with a growing volume of data.

To calculate the relationship between the data volume and the number of operations:

Algorithm 1: 2:8 ratio.

2x2x2 = 8 … therefore the relationship is O(n3).

Algorithm 1 has a ratio of 8 and a relationship of O(n3).

Algorithm 2 has a ratio of 4 and a relationship of O(n2).

Algorithm 3 has a ratio of 2 and a relationship of O(n).

Conclusion

Algorithms 1 & 2 are too inefficient and waste time due to calculating similar data during each loop. Algorithm 3 avoids this and therefore proves to be the most efficient by completing analysing the data and producing an output in the shortest amount of operations (loops). This is backed up by the data recorded – Algorithm 3 has a relationship between the data volume and number of operations of O(n) compared to Algorithms 1 & 2 O(n3) & O(n2) respectively.