DEPARTMENT OF COMPUTER SCIENCE

COURSEWORK ASSESSMENT SPECIFICATION

MODULE DETAILS:

Module Number:	08112	Semester:	2
Module Title:	Software Engineering and HCI		
Lecturer:	Dr JD Rayner		

COURSEWORK DETAILS:

Coursework Assessment Number:	1	of	2		
Title of Assignment:	Algorithm Analysis				
Format:	Report	Program			
Method of Working:	Individual				
Workload Guidance:	Typically, you should expect to spend between	10	and	14	hours on this assessment

PUBLICATION:

Date of issue:	04/03/2008

SUBMISSION:

ONE copy of this assignment should be handed in via:	White Box	Other (please state method)	
Time and date for submission:	**9.30am**	Friday 11 April 2008	

The assignment should be handed in no later than the time and date shown above, unless an extension has been authorised on a *Request for an Extension for an Assessment* form which is available from the Office or http://www.student-admin.hull.ac.uk/downloads/Mitcircs.doc. The extension form, once authorised by the lecturer concerned, should be attached to the assignment on submission (or given to the lecturer in the case of electronic submission).

MARKING:

Marking will be by:	Student Number

BEFORE submission, each student must complete the **correct** departmental coursework cover sheet dependant upon whether the assignment is being marked by student number, student name, group number or group name. This is obtainable from the departmental student intranet at http://intra.net.dcs.hull.ac.uk/sites/home/student/ACW%20Cover%20Sheets/Forms/AllItems.aspx

ASSESSMENT:

The assignment is marked out of:	100	and is worth	10	% of the module marks

ASSESSMENT STRATEGY AND LEARNING OUTCOMES:
The overall assessment strategy is designed to evaluate the student's achievement of the module learning outcomes, and is subdivided as follows:

LO	Learning Outcome	Method of Assessment *{e.g. report, demo}*
2 ***4*** ***6***	*Analyse algorithms and compare efficiency* *Implement algorithms for volumes of data* *Communicate alternatives and evaluate*	Report results Report code Report analysis

Assessment Criteria	Contributes to Learning Outcome	Mark
Implemented code for search method(s) Tabulated experimental results Discussion of results	4, 6 2, 6 2, 6	25% 25% 50%

FEEDBACK

Feedback will be given via:	In Class	Other (please state method)	
Other feedback (if appropriate) will be given via:	Individual annotation of scripts plus summary email		
Feedback will be given no later than:	end of week 12		

Questions
If you have any questions regarding this assessment you **MUST** speak to the lecturer as soon as possible.

You are advised to read the **NOTES** regarding Late Penalties, Use of Unfair means and Quality Assurance on the department's student intranet at:
http://intra.net.dcs.hull.ac.uk/sites/home/student/ACW%20Cover%20Sheets/Forms/AllItems.aspx

In case of any subsequent dispute, query, or appeal regarding your coursework, you are reminded that it is your responsibility, not the Department's, to produce the assignment in question.

Department of Computer Science

08112 Software Engineering

Algorithm Analysis Practical work

Algorithm Analysis – Trading Profits

Mick Donald's café/restaurant has been trading erratically in recent years. Mick has recorded data about the profit made by the café/restaurant each week, some weeks being profitable while others showed a loss (a negative profit value). All in all there is four years' worth of data (4 x 52 weeks). The data is stored in a set of plain text files, with each week's profit value on a separate line. The data files can be downloaded from the module website.

Mick would like to have a program which can analyse his data to see if the most profitable weeks relate to the times when he ran advertising and promotional offers. Effectively he needs a program which can look through the complete set of data to find the subsequence of weekly values which add up to the highest total profit value. There are several alternative algorithms available for this analysis, but which is the most effective one to use? The objective of this practical is to experiment with each of the algorithms and so obtain some measurements of their performance.

Exercise Assessment

This laboratory exercise and an associated workshop discussion contribute to the module assessment. Your attendance at your scheduled workshop tutorial in week-8 (April 7-9) is essential, and you should bring your tabulated data output results and Search() method listings to your workshop. For final assessment, you are required to submit a Formal Report containing your tabulated results, your code for the three alternative search methods, and your discussion of the results leading to appropriate classification of the three methods in terms of "big O".

Program Structure

The method which performs the algorithm is implemented as a static member of a class. It is fed the data array and outputs the start month number, end month number and total sales by means of out parameters:

```
using System;
using System.IO;

namespace ProfitCalculator
{

   public class RangeFinder
   {
      /// <summary>
      /// Performs a search to find the start and end of the "best" period and the
      /// total amount of sales over that period.
      /// </summary>
      /// <param name="data">the data to be examined</param>
      /// <param name="bestStart">the start point found by the search</param>
      /// <param name="bestEnd">the end point found by the search</param>
      /// <param name="bestTotal">the total sales over that period</param>
      /// <param name="loops">the number of executions of the inner loop</param>
      public static void Search(double [] data, out int bestStart,
            out int bestEnd, out double bestTotal, out int loops)
      {
         bestTotal = 0 ;
         bestStart = 0;
         bestEnd = 0;
         loops=0;
```

```
            /// TODO - put your search code here
        }
    }

    /// <summary>
    /// Tests the Profits Calculator
    /// </summary>
    class Test
    {
        /// <summary>
        /// The main entry point for the application.
        /// </summary>
        static void Main()
        {
            double [] data;
            int bestStart, bestEnd;
            double bestTotal;
            int loops;

            /// name of the file and the number of readings
            string filename = "week52.txt";
            int items=52;

            data = new double [items];

            try
            {
                TextReader textIn = new StreamReader(filename);
                for( int i=0 ; i < items ; i++ )
                {
                    string line = textIn.ReadLine();
                    data[i] = double.Parse(line);
                }
                textIn.Close();
            }
            catch
            {
                Console.WriteLine ("File Read Failed");
                return;
            }

            RangeFinder.Search(
                data, out bestStart, out bestEnd, out bestTotal, out loops);

            Console.WriteLine ( "Start : {0} End : {1} Total {2} Loops {3}",
                bestStart, bestEnd, bestTotal, loops);
        }
    }
}
```

You can cut this program out and paste it into Notepad. It will compile and run. The **`Test`** class **`Main`** method reads in a data array from the file and then calls the **`Search`** method of the **`RangeFinder`** class. This sets the **`bestStart`**, **`bestEnd`**, **`bestTotal`** and **`loops`** variables with the values that it has calculated. At the moment it just sets them all to zero. You need to add code which implements the algorithms as described below.

The Search Method

You need to complete the search method. For this exercise, we present three alternative algorithms for the search process – implement each one as a separate static method in the **`RangeFinder`** class.

Algorithm 1. The Obvious Approach

The most obvious implementation of the search method is a simple brute force algorithm described by the following pseudo code:

```
For every possible start position      // every array index
      For every possible end position  // rest of array from current start
      {
            Set subtotal to 0
            For every value in subseq  // between current start and end
                  Add profit value to subtotal
            Update subseq info when subtotal exceeds current best total
      }
```

Initially you can ignore the loops variable shown as an output parameter of the **`Search`** method, and just concentrate on creating a program which returns the **`bestStart`**, **`bestEnd`** and **`bestTotal`** values. Note that you can use the **`Length`** property of the data array to determine how large it is, eg:

```
for ( int i = 0 ; i < data.Length ; i++ )
{
   Console.WriteLine ( "Value {0}", data[i] );
}
```

would print out the contents of the array.

Algorithm 2. A More Efficient Alternative

The above algorithm will work but it is inefficient. One reason is that it takes many operations to calculate totals for very similar subsequences. For example: suppose the algorithm works out a total for the subsequence of weeks 1 – 5. This requires a loop which counts through the 5 weeks and keeps a running total. The algorithm will then quite separately calculate a total for weeks 1 – 6, re-using the loop to count through all the 6 weeks and build a fresh running total. This is a waste when we can obtain the subsequence for 1-6 by incrementing the total of the subsequence for 1-5; we just have to add on the profit value for week 6. Add a new static method called **`Search2`** to the **`RangeFinder`** class which implements the improved search algorithm shown here:

```
For every possible start position...
      Set subtotal to 0
      For every possible end position...
      {
            Add end position's profit value to subtotal
            Update subseq if subtotal exceeds current best total
      }
```

Algorithm 3. A Further Optimisation

A further optimisation uses the observation that, if ever the subtotal is less than zero, it can be ignored – further values added on will be worth more on their own! Make a further method called **`Search3`** which implements the algorithm shown below.

```
Set start position to 0, subtotal to 0
For every profit value...  // index from 0 to end of array as end position
{
      Add value to subtotal
      Keep subseq info (start, end, total) if total exceeds current best
      If total is less than 0,
            set start position to next index and set total to 0
}
```

Initial Tests

If you run each method on the data file **`week52.txt`** you can test that your implementations work. This data should give the same results in every case, as follows:

```
Start : 17
End   : 20
Total : 99.72
```

Once you have all three algorithms correctly implemented, you can begin your experimental measurements, as described below. *Your experimental results will be discussed during the Workshop Tutorial classes in week 8.*

Performance Analysis

Now that you have the three algorithms as experimental implementations in C#, you need to measure their behaviour. We need to count the number of critical operations each algorithm involves, to process a given data set, and to measure this for each algorithm across several different volumes of data.

First of all:

- identify, in each version of the **`search`**() method, the loop body which is executed the most times – where the subtotal value is incremented;
- Add one further statement to this loop, to increment a counter: **`loops = loops +1;`** (or **`loops++;`**);
- Now run your three test programs again with the 52-weeks data. Each program should still report the best subsequence information, but you will also see how many loop cycles have been involved. Note the four output values in each case.

Each of the data files has a different number of weeks' profit values, as indicated in the file names **`week52.txt`**, **`week104.txt`**; and **`week208.txt`**, relating to one, two and four years of operations. Run each of your programs with each of the data files provided on the website, and make a note of all the output values in each case.

Note, to gather your experimental data correctly, you need to adjust the array size value in each program for each data set (from 52 to 104 and then 208), or augment the programs to request input of the data file size before instantiating the array. For the purposes of this simple experimental context, it's probably simpler to change the code and recompile for each of the data sets. You could augment the program to request user input of the file name, but again the experimental context *does not require* this level of refinement.

If you have time, you should run one further set of tests, using the 52-week data but with the array limit constant set to 26, so only *half* the data values are processed. The same week range and profit total as for 52 weeks should be reported, but what are the results for the loop cycle counts?

Discussion of Results

Bring your observations, suitably tabulated*, and code listings of every **`search()`** method, to your workshop tutorial in week 8. In the workshop, the interpretation of all the results will be discussed, and advice given as to the presentation of your Report for assessment.

* Hint: For each alternative search algorithm, make a table with a row for each data file (volume of data) and a column for each observation measure. This will let you see at a glance how the number of operations changes with data volume for each algorithm separately, and so help you to calculate the algorithm's classification. You will also be able to show that each algorithm gives the same output results as the others for each data sequence.

JDR 4/03/08