COURSEWORK ASSIGNMENT **UNIVERSITY OF EAST ANGLIA**

School of Computing Sciences

UNIT : CMP-5014Y Data Structures and Algorithms

ASSIGNMENT TITLE : Module Assessment Exercises 2

DATE SET **:** 01/02/17

DATE OF SUBMISSION **:** 3pm Weds 15 March (Week 9)

RETURN DATE **:** Week 13

ASSIGNMENT VALUE **:** 35%

SET BY **:** Tony Bagnall **SIGNED:**

CHECKED BY **:** Tony Bagnall / Jason Lines **SIGNED:**

Aim:

Subject specific

To test your ability to design, analyse and implement algorithms.

Transferable skills

Describing algorithms using pseudo-code; use of mathematical notation when analysing algorithms; computer programming; use of drawing tools.

Learning outcomes:

Subject specific

Increased experience of problem-solving and algorithm design and analysis; further experience of programming in Java; increased awareness of the importance of algorithm complexity.

Assessment criteria:

Part marks are shown on the exercise sheet.

Description of the Assignment

1 Algorithm Design and Implementation (40%)

Let D denote the domain of two-dimensional square matrices of positive integers. The same integer may be stored in more than one element of $A \in D$.

Given $A \in D$ and a positive integer, p, the *FindElement* problem is to determine whether or not p is contained in A.

1. ***For each of the following, you must submit pseudo-code algorithms using the notation and approach described in previous lectures. It is NOT sufficient to give program listings. In each case, derive the worst-case run-time complexity of your algorithm, giving an example of a worst-case instance. In each case, try to derive an algorithm which is as efficient as possible for such worst-case instances and describe a way of generating worst-case instances for*** $n \times n$ ***matrices.***

   (a) Design an algorithm FindElementD(A, n, p). (*Note that here no assumptions may be made about the order of the elements in A.*)

   [4 marks]

   (b) Design an algorithm, FindElementD1(A, n, p), to solve the *FindElement* problem when the domain of the arrays is restricted to those for which each row of integers is in non-decreasing order.

   Thus, $A = [A_{i,j}] \in D1$ if and only if:

   $$A_{i,1} \leq A_{i,2} \leq \cdots \leq A_{i,n} \qquad i = 1, ..., n,$$

   where n denotes the size of the square matrix. [6 marks]

   (c) Finally, design an algorithm, FindElementD2(A, n, p), to solve the *FindElement* problem when the domain of the matrices is restricted to those for which both rows and columns are held in non-decreasing order. For example,

$$\begin{bmatrix} 1 & 3 & 7 & 8 & 8 & 9 & 12 \\ 2 & 4 & 8 & 9 & 10 & 30 & 38 \\ 4 & 5 & 10 & 20 & 29 & 50 & 60 \\ 8 & 10 & 11 & 30 & 50 & 60 & 61 \\ 11 & 12 & 40 & 80 & 90 & 100 & 111 \\ 13 & 15 & 50 & 100 & 110 & 112 & 120 \\ 22 & 27 & 61 & 112 & 119 & 138 & 153 \end{bmatrix}.$$

   [9 marks]

2. Implement each of your algorithms as a static method in a Java program. Test your implementations using the matrix given above and for each of the following values of p:

$$4, 12, 110, 5, 6, 111.$$

[9 marks]

3. Design appropriate experiments to test the worst-case performance of your algorithms. For each case, you should output a table demonstrating the worst-case run-time complexity of the algorithm. Each table should have entries for matrix size

$$n = 10, 20, \ldots, 100, 200, \ldots, 1000, 2000, 3000, 4000$$

Using Microsoft Excel or otherwise, plot a graph from each set of results and discuss what you can infer from the graphs in relation to your analysis. Use the following approach for performing your timing experiments

```
// Record mean and std deviation of performing an operation
// reps times
     double sum=0,s=0;
     double sumSquared=0;
     for(int i=0;i<reps;i++){
         long t1=System.nanoTime();
            //call FindElement method for an instance of size n
         long t2=System.nanoTime()-t1;
 //Recording it in milli seconds to make it more interpretable
         sum+=(double)t2/1000000.0;
         sumSquared+=(t2/1000000.0)*(t2/1000000.0);
     }
     double mean=sum/reps;
     double variance=sumSquared/reps-(mean*mean);
     double stdDev=Math.sqrt(variance);
```

[12 marks]

2 Data Structure Design and Implementation (40%)

For this question you will implement a hash table with chaining that uses arrays to store the chains rather than linked lists. This is not necessarily a sensible approach, but it is included to test your ability to manipulate array based data structures. You must implement the hash table with arrays, not with the `ArrayList` class. Your data structure need not be generic, it can work with just objects, but you will not be penalised for using any of the Collections features (generics, iterable etc) taught in programming 2. You can include class variables and methods not specified. You may, if you wish, adapt the abstract class, but please keep the methods that are already in there. **Please note I do not want you to implement rehashing for this hash table**. The capacity should remain constant.

1. **Basic structure and constructor.**

   Implement a class `ArrayHashTable` that extends the abstract class `HashTable` and uses chaining. The data should be stored in a two dimensional array called `table`. In addition to the inherited variables, the class should have an integer variable `chainSize` (the initial size of each array to store the chain, default to 5). These variables should be updated when items are added or removed from the table. The default `capacity` should be 10 and the constructor should initialise table to an array of arrays size `capacity`, with each value set to null.

   ```
   table=new Object[capacity][];
   ```

   the number of elements at each location of the hash table should be stored in an array of integers called `counts`, all initialised to zero. [5 marks]

2. **Implement the method `add`**.

   Use the method `hashCode` to find the hash value for the location in the hash table. If the entry is empty (e.g. if the hash value is 5 and `table[5] == null`), create a new array of size `chainSize`. If the entry is not empty, insert the item at the correct location, if it is not already present. If this means the current array is now full, double the size of the current array and copy over the values. This method should return **`true`** if the item is added to the table, **`false`** if not (i.e. if it is already present). [10 marks]

3. **Implement the method `contains`.** This should be trivial once you have implemented `add`.

   [5 marks]

4. **Implement the method `remove`.**

   Use the method `hashCode` to find the hash value for the location in the hash table. If the entry is empty, or the array for that entry does not include the object passed, return **`false`**. Otherwise, remove the item, adjusting the hash table variables and remaining hash table objects accordingly. [10 marks]

5. Design an experiment to test the run time of your hash table and of the Java class `HashSet` for inserting n randomly generated integers into a hash table with initial capacity 10, then removing the same n integers. The timing experiment code is similar to that for question 1. Use $n =< 1000, 2000, \ldots, 10000, 15000, 20000, 25000, \ldots, 50000 >$, although you can reduce the maximum if it is too slow to run. You can generate an array of numbers to add then remove with the following code (or use your own technique).

```
int[] numbers=new int[n];
Random r=new Random();
for(int j=0;j<n;j++){
    numbers[j]=Math.abs(r.nextInt());
}
```

Using Microsoft Excel or otherwise, plot a graph of these results and explain what the graph demonstrates and why there are any observed differences. [10 marks]

3 Sorting Practice (20%)

Sort the following numbers by hand

18, 33, 30, 18, 38, 16, 11, 43, 16, 18

using:

1. Merge Sort; [5 marks]
2. Quicksort using
   (a) a random pivot, [4 marks]
   (b) the median of three pivot. [4 marks]

In each case, show all working and count the number of comparisons performed.

Repeat 2(b) but this time draw diagrams to illustrate the steps in the partitioning algorithm. [7 marks]

Assessment

Marks will be awarded as follows

40% Question 1

40% Question 2

20% Question 3

 Submission instructions overleaf.

Submission Procedure

Via evision: I will attempt to get PASS working with this to allow electronic pdf submission. The pdf document (generated by PASS) should contain

1. Question 1
    - a discussion of the ideas underlying your algorithms;
    - a pseudo-code description of your algorithms;
    - an analysis of the run-time complexity of your algorithm, including a description of what constitutes a worst-case instance;
    - a description of how to generate worst-case instances.
    - graph(s) of experimental results for question 1 and discussion.
2. Question 2 graph(s) of experimental results for question 2 and discussion.
3. Question 3 A solution to the sorting question. This can be done by hand if you wish, there will be no penalty.

via blackboard: a zipped Netbeans project with your code questions 1 and 2. Put the code for each question in a different package named question1 and question2

Written coursework should be submitted by following the standard CMP practice. Students are advised to refer to the Guidelines and Hints on Written Work in CMP (`https://intranet.uea.ac.uk/computing/Links/Reports?_ga=1.7481330.1383599.1413214592`).

Plagiarism: Plagiarism is the copying of close paraphrasing of published or unpublished work, including the work of another student; without due acknowledgement. Plagiarism is regarded a serious offence by the University, and all cases will be investigated. Possible consequences of plagiarism include deduction of marks and disciplinary action, as detailed by UEA's Policy on Plagiarism and Collusion.