CSP2348 Data Structures

Assignment 2: A Mini Team Project

- Algorithm Design, Analysis & Java Implementation

Objectives from the Unit Outline

- Describe the general principles of algorithm complexity and performance;
- Describe the concept, application, and specification of an abstract data type (ADT) and employ Java classes to encapsulate ADTs.

General Information:

- This is a mini programming project, requiring up to two students to work as a team to complete it. It consists of three questions, all requiring implementation using Java programming language.
- The first question requests you design algorithms for a given application scenarios, and then implement them using array data structure. Part of the question also requires algorithm analysis using big-O notation.
- The second question requests you implement and test a particular Java ADT. This question requires you design and implement part of the application scenario using singly-linked list (SLL) to store the required information. While part of the work (e.g., Java codes) has been completed, you are requested to complete the rest.
- The third question requires you modify some Java code to implement specific application scenarios using binary tree data structure. Once again, part of the work (e.g., Java codes) has been completed, and you are requested to complete the rest.
- It is your responsibility to form your team/group. Please send the details of your team members (i.e., the student IDs and names) to your tutor by the end of week 9, if your team has more than one member (if you really prefer to work individually, you may do so but no workload is to be reduced).

Due: Friday 22nd May 2015 @ 5:00 pm
(i.e., 2nd last teaching week, see unit **Schedule**)

Value: 20%

Submission Instructions

- Submit your Assignment 2 via Blackboard electronic assessment submission facility. For a detailed submission procedure, please refer to "How to submit your Assignment online" item in the Assessment section of this unit website.
- Your submission should include the assignment main document (or a report) and your Java classes' source codes. The main assignment document must be in report style, in Word (or PDF) format (see detailed format requirement below). Pages must be numbered. Your Java source code file/s must compile and runnable.
- One submission per team - Your submission should be in a single compressed file (.zip), which contains your mini project report and Java source code file/s. Please name the zip file as
  <team-leader's Student ID>_< team-leader's full name>_A2_CSP2348.zip.
- Remember to keep the copy of your assignment. No hard copy is needed.
- Your attention is drawn to the University rules governing cheating and referencing. In general, cheating is the inclusion of the unacknowledged work of another person. Plagiarism (presenting other people's work and ideas as your own) will result in a zero mark.
- University rules/policies will apply for late submission of the assignment.

Main assignment document format requirement:

Must contain	**Cover page** Must show assignment title, student IDs and name/s of your team, due date etc.
	Executive Summary (optional) This should represent a snapshot of the entire report that your tutor will browse through and should contain the most vital information requested.
	Table of Contents (ToC) This must accurately reflect the content of your report and should be generated automatically in Microsoft Word with clear page numbers.
	I**ntroduction** Introduce the report, define its scope and state any assumption; Use in-text citation/reference where appropriate.
	Main body The report should contain (but not limited to): • Understanding of concepts/techniques involved in the report. • The questions being solved/answered, e.g., Q1: key algorithms and analysis (if applicable); a couple of screen snapshots of execution of the Java code (at least one snapshot per function); Q2: algorithms and analysis using O-notation; screen snapshots of execution of the Java code (at least) one snapshot per method); Q3: key screen snapshots of execution of the Java code (at least one snapshot for b), c) and d) part); • Any strategies used to solve the problems (e.g., provide a solution or an approach to develop a solution?) • Discussion or a critique of the solution developed /achieved, etc.
	Conclusion (optional) Outcomes or main works done in this assignment.
	References A list of end-text reference, if any, formatted according to the ECU requirements using the APA format (Web references are also OK).
Other requirement	The report should be no more than 6 pages (excluding references and diagrams) for individual assignment, and be no more than 10 pages for team work. The text must use font **Times New Roman** and **be not smaller than 12pt**.

Background Information

Arrays, linked lists and binary trees are typical data structures that are frequently used in many applications. While an array is a static data structure, linked lists and binary trees are among the most important yet simplest dynamic data structures. Many applications employ these data structures to model the applications scenarios.

An array is a random-access structure. An array component can be accessed using a unique index in constant time. This property makes array the most effective data structure in term of component access; therefore the most frequently used data structure.

A linked list is a sequential data structure, consisting of a sequence of nodes connected by links. Each node contains a single element, together with links to one or both neighbouring nodes (depending on whether or not it is an SLL or DLL). To access a linked list node, you must search it through the header of the linked list. Linked list is one of the simplest yet the most important dynamic data structures.

A binary tree is a non-sequential dynamic data structure. It consists of a header, plus a number of nodes connected by links in a hierarchical way. The header contains a link to a node designated as the *root* node. Each node contains an element, plus links to at most two other nodes (called its *left* child and *right* child, respectively). Tree elements can only be accessed by way of the header.

This assignment focuses on algorithm design, analysis, and Java implementation using array, SLL and binary tree data structures. While all questions are of equal importance, pay more attention to Q2 and Q3 as these questions enable you practise using typical linked list and binary tree based algorithms (e.g., SLL creation and data/link manipulation, binary tree traversals and their applications, etc.).

Q1). ***Array Application Programming*:**

A tiny programming project mimicking Lotto's award checking system

A tiny Lotto system allows up to 1000 people play "lotto" with it. To play, each player is assigned an ID number (1≤ ID number ≤1000). Each player selects 6 distinct integer numbers (all integers are between 1 and 45, inclusive) as his /her *game-numbers*. All players' *game-numbers a*re stored in a tiny "*database*" which, in this case, is replaced by a Java array, lotto[1000][6]. That is, for each player with ID number *i*, his/her game-numbers are stored in array lotto[*i*-1][0…5].

The tiny lotto system has three functions:

(1) The system can generate a sequence of numbers, as winning numbers, consisting of 6 distinct integers. Each winning number is an integer between 1 and 45 (inclusive). The winning numbers can be randomly generated (or manually input) and then stored in the system;
(2) The system can check and calculate the total number of winners of each class, therefore generate a statistic table, as below:

Winners statistics:

Winner class	Total number of the winners
1st class	
2nd class	
3rd class	
4th class	

Notes:
A 1st class winner is one whose game-numbers match all 6 winning numbers;
A 2nd class winner is one whose game-numbers match 5 winning numbers;
A 3rd class winner is one whose game-numbers match 4 winning numbers;
A 4th class winner is one whose game-numbers match 3 winning numbers.

(3) The system allows individual player check whether or not he/she is a lotto winner. When the player input his ID number k ($1 \le k \le 1000$), the system checks whether or not the player wins the lotto. That is, if his/her game-numbers (stored in lotto[k-1][0…5]) match
   (a) all 6 winning numbers, the system prints his/her game-numbers in sequence, and then prints "you are a top class winner, congratulations!"
   (b) 5 winning numbers, the system prints his/her game-numbers in sequence, and then prints "you are a 2nd class winner, congratulations!"
   (c) 4 winning numbers, the system prints his/her game-numbers in sequence, and then prints "you are a 3rd class winner, congratulations!"
   (d) 3 winning numbers, the system prints his/her game-numbers in sequence, and then prints "you are a 4th class winner, congratulations!"
   (e) less than 3 winning numbers, the system prints his/her game-numbers in sequence, and then prints "Thanks for playing lotto. Good luck next time!"

Your Task:

Design algorithms and/or write Java program/s to implement (i.e., mimicking) the above tiny Lotto system. To improve the system's performance, you are requested to

(i) sort individual game-numbers for each player; and

(ii) use two different methods to implement the functions 2 and 3:

a) The first method uses normal array searching algorithms (e.g., linear or binary search algorithms) to implement the matching between game-numbers and winning-numbers; and

b) The second method implements the matching by sequentially comparing components of the two sorted arrays (i.e., one containing the player's game-numbers and the other contains the winning numbers), using a scanning technique similar to that of *arrays-merging-algorithm* (refer to Module 3).

(iii) Compare the complexities of the above two methods in the step (ii) using O-notations .

To reduce your workload, a Java method is given in Appendix A that demonstrates how to randomly generate all players lotto data, i.e., *game-numbers*, and store them in a Java array lotto[1000][6]. (You may modify the code and then include it as a part of your code/s)

Q2). ***Linked-list Programming:*** SLL deletion and reverse operations

A unit_list class has been given in Workshop 05. The class uses an SLL to represent a list of student assessment results of a unit. Each SLL node contains a record of one student's assessment results. That is, it stores a student ID followed by the marks of three assessment components (A1_result, A2_result, and exam_result), all are in integers. For convenience, student information stored in the SLL is in ascending order by the student_ID field. The class given shows the technique of traversing an SLL, searching an SLL and inserting a node into the SLL.

Your Task:

Starting from the unit_list class given, you are requested to expand the class by implementing the following two Java methods:

(1) Deletion of an student's information:

```
private static void delete_unit_result(unit_list u_list, int ID)
{ // your work here…….
}
```

When being given a student ID (as input), the method searches the SLL to see if the student with that ID existed in the SLL or not. If it is existed, the method deletes the student information (i.e., the node in the SLL) and keeps all other student information in the SLL unchanged.

Note that the method requires no return (i.e., *void* return). As such, you should make sure that the first node of the SLL is not physically deleted from the SLL, even if it might be logically deleted (why?).

(2) Display/print the student information in descending order on the Student ID field:

```
private static void reverse_print_unit_result(unit_list u_list)
{ // your work here…….
 }
```

This method displays/prints all student information in descending order on student ID field. Please note that, to keep the unit_class work properly you should not destroy the original SLL (why?). To this, you may create a new SLL which is the reverse of the original SLL (or if you have to destroy the original SLL, make sure your recover it afterwards).

(3) Analyse the above methods using O-notation.

Q3). <u>Binary tree traversal</u>: Print leaf and non-leaf node and tree-height

A Java code is given in TreeTest.java in Module 6 (see Task 3 of Workshop 6) It prints the *Pre-order, In-order* and *Post-order* traversal sequences of a given binary tree.

<u>Your Task:</u>

Analyze this code and modify it so that it would:

a) print the *Pre-order, In-order* and *Post-order* traversal sequences of the given binary tree (*already implemented*);
b) print all leaf nodes only (of the tree);
c) print non-leaf nodes only (of the tree);
d) print the height of the tree.

Test your code by creating a more complicated BST, e.g., using the following array to replace the array in the third line of the main method:

```
int [] a = {49, 76, 67, 29, 75, 18, 4, 83, 87, 40, 80, 46, 42, 43, 45, 41};
```

Indicative Marking Guide:

		Allocated Marks (%)	Marks achieved & Comments
Q1	**Algorithm design & Analysis** (as in step (ii)) Algorithm / Pseudocode Analysis using O-notation	**40**	
	Java codes Key methods: explanation of design Full implementation		
	Compile & Running outputs		
	Programming style: code readability		
	Documentation		
	Over all		
Q2	**Algorithm design & Analysis** First method: Algorithm Pseudocode Analysis using O-notation 2nd method: Algorithm Pseudocode Analysis using O-notation	**30**	
	Java codes Key methods: explanation of design Full implementation		
	Compile? Running? Outputs? & others		
Q3	**Algorithm design & Analysis** Algorithm / Pseudocode for method b) method c) method d)	**20**	
	Java code: Compile? Running outputs? Others		
Report	Presented as per format requirement?	**10**	
	Submitted as per submission requirement?		
Total mark of 100 which is then converted to 20% of unit mark		**100 (/20)**	

Appendix A: A Java code demonstrating generation of lotto[1000][6]

```
import java.util.Random;
public class A2Q1_lotto_data {
   public static void creat_randomArray(int[] records,int n, int range)
   {int x=0, k=0, j=0, y=0;
      Random randomGenerator = new Random();
      records[0]=randomGenerator.nextInt(range)+1;
      for(int i=1;i<n;i++)
      {  y=0;
         while (y==0)
         { x=randomGenerator.nextInt(range)+1;
           k=0;
           for(j=0;j<i;j++)
                 if (x != records[j]) k++;
           if (k == i) {records[i]=x; y++;}
         }
      }
   }
  public static void main(String[] args)
   {   int n=1000;
      int [][] lotto = new int[n][6];
       for(int i=0;i<n;i++)
       creat_randomArray(lotto[i], 6, 45);
      for(int i=0;i<n;i++)
        { for(int j=0;j<6;j++) System.out.print(lotto[i][j]+" ");
          System.out.println();
        }
    }
}
```

(The END of the Assignment Description)