PRINCE OF SONGKLA UNIVERSITY

FACULTY OF ENGINEERING

Department of Computer Engineering

Midterm Examination: Semester 2 **Academic Year**: 2013-2014

Date: 9th January, 2014 **Time**: 13:30 – **15:30 (2 hours)**

Subject Number: 241-423 **Room**: R200

Subject Title: Advanced Data Structures and Algorithms

Lecturer: Aj. Andrew Davison

Exam Duration: 2 hours

This paper has 4 pages.

Authorized Materials:

- Writing instruments (e.g. pens, pencils).
- Books (e.g. dictionaries) and calculators are **not** permitted.

Instructions to Students:

- *Answer questions in English.* Perfect English is **not** required.
- Attempt all questions.
- Write your answers in an answer book.
- Start your answer to each question on a new page
- Clearly number your answers.
- Any unreadable parts will be considered wrong.
- When writing programs, use good layout, and short comments; marks will not be deducted for minor syntax errors.
- The marks for each part of a question are given in brackets (...).

Question 1 (25 marks; 25 minutes)

You have an 100 element integer array that contain the integers 0, 1, 2, up to 99, in a random order. But there is a mistake in the data: one of the elements is a duplicate of another element.

a) Find the duplicate value using a $O(n^2)$ algorithm. (n is the size of the array) (5)

b) Find the duplicate value using a **O(n)** algorithm. (15)

Hint: my solution does **not** use sums of squares.

c) What are the drawbacks of the O(n) solution compared to the $O(n^2)$ solution? (5)

Question 2 (30 marks; 30 minutes)

a) Draw a diagram showing how mergesort() splits and then merges the following array when sorting it:

```
Integer[] arr = {25, 16, 7, 19, 3, 48};
```

Do **not** include any mergesort() code. (5)

b) Draw a diagram showing how quicksort() splits and then combines the following array when sorting it:

```
Integer[] arr = {25, 16, 7, 19, 3, 48};
```

Assume that the pivot is the integer midpoint position of the current sub-range (calculated using integer division).

Do **not** include any quicksort() code. (5)

c) Compare mergesort() and quicksort() in terms of their worst case running times **and** space requirements. Explain the comparisons in words. (5)

d) A sorting algorithm is *stable* if two items with the same value are not rearranged with respect to each other during the sorting. For instance, in the five-element array:

55 5_1 12 5_2 33

A stable sorting algorithm means that the final ordering is:

5_1 5_2 12 33 55

The first 5 (5_1) and the second 5 (5_2) remain in the same order with respect to each other.

Explain in words, with brief examples, if mergesort() and quicksort() are stable algorithms. (10)

e) Give an example of when a stable sorting algorithm is important. Do **not** include any code. (5)

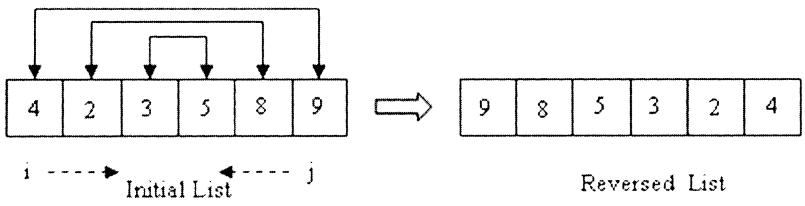
Question 3 (20 marks; 20 minutes)

In this question, assume the existence of an ArrayList class; you do **not** have to implement the ArrayList class.

a) Implement the method:

```
public static <T> void reverseByIndex(ArrayList<T> aList);
```

reverseByIndex() uses indicies to reverse the order of the elements in an ArrayList, as shown in the diagram below. (5)



b) Implement the method:

```
public static <T> ArrayList<T> reverseByCopy(ArrayList<T> aList);
```

reverseByCopy() copies the elements of an ArrayList in reverse order to a *new* ArrayList, which becomes the return value of the method. (5)

c) Compare the worst case running times **and** space requirements of the methods in part (a) and (b). Explain the comparisons in words. (10)

Question 4 (45 marks; 45 minutes)

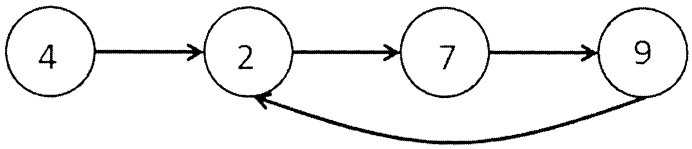
Assume the existence of the `Node<T>` class described in the Linked Lists notes. You do **not** have to implement the `Node<T>` class.

a) Write code that finds the middle element of a single-linked list built using `Node<T>` objects containing integers **in one pass**. One pass means that your code should contain a single loop. The list data is not ordered. (10)

b) Recode the algorithm to use a 'multi-pass' approach. Multi-pass means that your code can use as many loops as you want. (5)

c) Compare the running times of solutions (a) and (b) (10)

d) Now assume that the list contains a loop, like the one in the diagram below.



Write a `boolean hasLoop(Node<T> front)` function which returns true if the given Node is the front of a list with a loop, and false otherwise. `hasLoop()` should have running time O(n). (n is the number of nodes in the list.) (20)

Hint: I'm asking you to implement Floyd's cycle-finding algorithm, which is also knows as the *tortoise and hare* algorithm.

--- End of Examination ---