OCR
Oxford Cambridge and RSA

A Level Computer Science

H446/02 Algorithms and programming

Practice paper – Set 2

Time allowed: 2 hours 30 minutes

Do not use:
- a calculator

First name										
Last name										
Centre number						Candidate number				

INSTRUCTIONS
- Use black ink.
- Complete the boxes above with your name, centre number and candidate number.
- Answer **all** the questions.
- Write your answer to each question in the space provided. Additional paper may be used if required but you must clearly show your candidate number, centre number and question number(s).
- Do **not** write in the barcodes.

INFORMATION
- The total mark for this paper is **140**.
- The marks for each question are shown in brackets **[ ]**.
- Quality of extended responses will be assessed in questions marked with an asterisk (*).
- This document consists of **28** pages.

Answer **all** the questions.

Section A

1 A binary search tree, `colour`, stores data about colours that are entered into a computer.

(a) A binary search tree is one example of a type of tree.

(i) State the main features of a tree.

.....

.....

.....

.....

.....

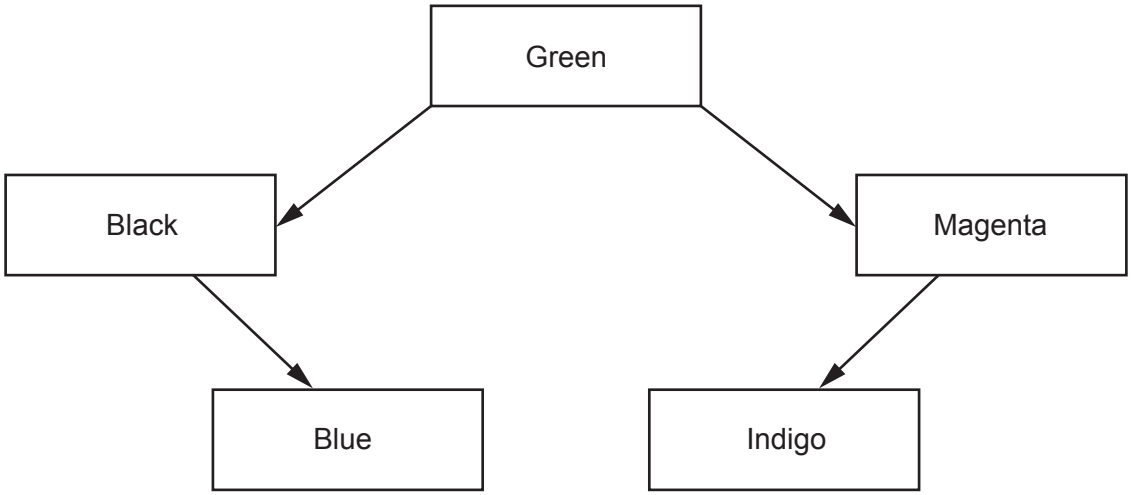
..... **[3]**

(ii) State the features that make a tree a binary search tree.

.....

..... **[1]**

(b) The current contents of `colour` are shown.

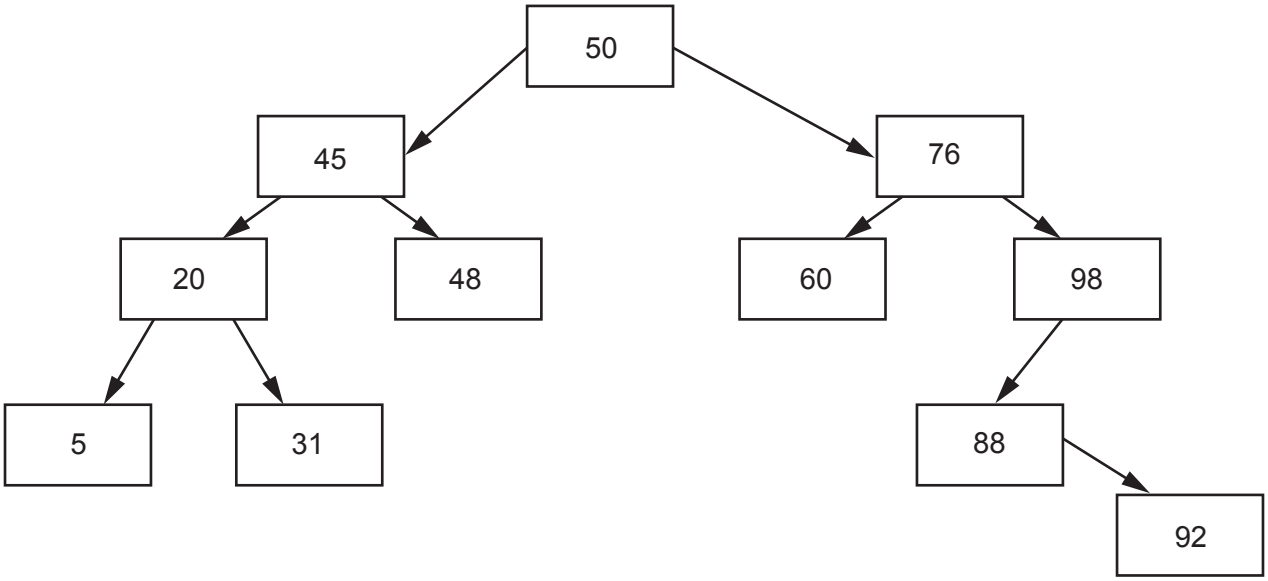


Add the following colours to the tree above in the order written:

Brown White Orange Purple

[4]

(c) A second binary search tree, `numbers`, stores numbers that are entered into a computer. The contents of the tree are shown below:



(i) Explain, using the binary search tree `numbers` as an example, how a depth-first (post-order) traversal is performed.

.....

.....

.....

.....

.....

.....

.....

.....

.....

..... **[5]**

(ii) Explain, using the binary search tree numbers as an example, how a breadth-first traversal is performed.

.....

.....

.....

.....

.....

.....

.....

..... **[5]**

(d) The binary search tree, `values`, is stored as an array of nodes. Each node has a left pointer, a right pointer and the data being stored. The following data is entered in the order shown below:

68 30 73 22 1 90 70

The following table shows the data stored in the array. The Root Pointer stores the node number of the first element in the tree.

Root Pointer

0

Free Pointer

7

Array Index	Left Pointer	Data	Right Pointer
0	1	68	2
1		30	
2		73	
3		22	
4		1	
5		90	
6		70	

Table 1.1

(i) Complete the remaining Left Pointer and Right Pointer values for the data entered in Table 1.1. Where the pointer is null, leave the space empty. **[3]**

(ii) State the purpose of the Free Pointer.

.....

..... **[1]**

(iii) The following data is added to the array in the given order:

6 100

Add the new nodes to Table 1.1 and update any relevant pointers. **[4]**

BLANK PAGE

2 Fig. 2.1 shows the flight paths between a country's airports. The value in bold beneath each node is the heuristic value from E.

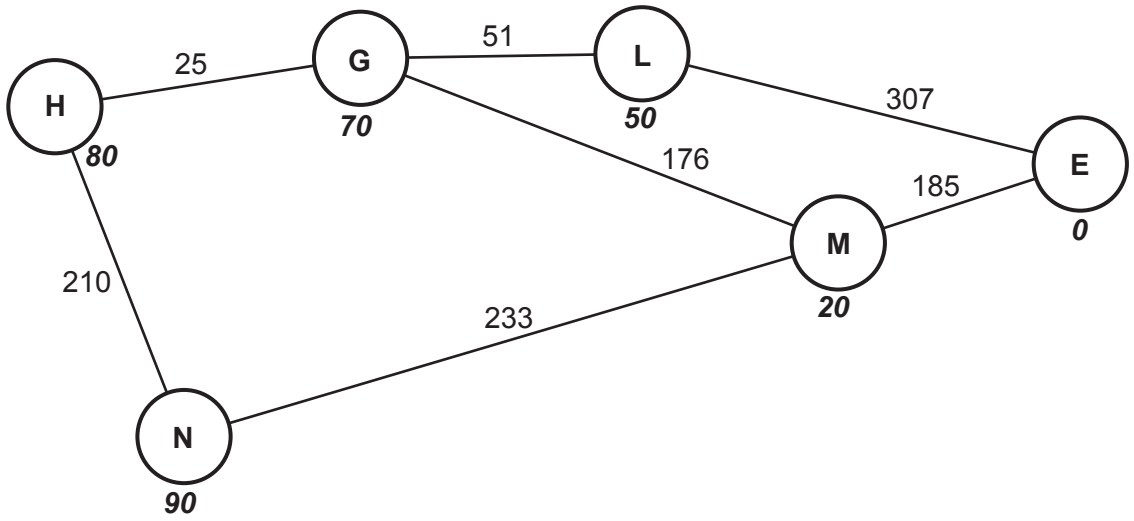


Fig. 2.1

(a) State the full name of the data structure shown in Fig. 2.1.

..... **[2]**

(b) The structure in Fig. 2.1 is searched using the A* algorithm making use of the heuristic values.

(i) State what the heuristic values could represent in Fig. 2.1.

.....

..... **[1]**

(ii) State the purpose of heuristic values in the A* algorithm.

.....

..... **[1]**

(iii) Perform an A* algorithm on the data structure in Fig. 2.1 to find the shortest distance between H and E. Show each step of the process, and the calculations performed for each node visited.

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

..... **[8]**

(iv) Give **one** decision that is made in the A* algorithm, and describe the effect of this decision on the next step(s) of the algorithm.

Decision .....

.....

Effect .....

.....

.....

.....

..... **[3]**

(c)* A programmer is interested in using concurrent processing to perform a searching algorithm.

Explain how concurrent processing could be used in searching algorithms, and evaluate the benefits and trade-offs from implementing concurrent processing in a searching algorithm.

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

..... **[9]**

3 Dexter is leading a programming team who are creating a computer program that will simulate an accident and emergency room to train hospital staff.

(a) Identify **two** features of the problem that make it solvable by computational methods.

.....

.....

.....

.....

..... **[2]**

(b)* Dexter has used decomposition and abstraction during the analysis of the problem.

Explain and evaluate the use of decomposition and abstraction in the creation of this simulation.

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

..... **[9]**

(c) Dexter has been told he should make use of caching in the simulation.

Describe what is meant by caching and explain how caching can be used within the simulation.

.....

.....

.....

.....

.....

.....

.....

.....

..... **[4]**

(d) Two of Dexter's programmers have developed different solutions to one part of the problem. Table 3.1 shows the Big O time complexity for each solution, where *n* = the number of data items.

	Solution A	**Solution B**
Time	O(n)	O(n)
Space	$O(k^n)$ (where k > 1)	O(log n)

Table 3.1

(i) The Big O time complexity for time of each solution is O(n).

Explain what is meant by time complexity, with reference to the solutions' Big O time complexity.

.....

.....

.....

.....

.....

..... **[3]**

(ii) Name the space complexity for each solution:

Solution A .....

Solution B .....

[2]

(iii) Explain, with reference to the Big O complexities of each solution, which solution you would suggest Dexter chooses.

.....

.....

.....

.....

.....

.....

.....

..... **[4]**

(e) Dexter's team is using an integrated development environment (IDE).

Describe how the programmers could make use of the following IDE tools:

Breakpoints .....

.....

.....

.....

Stepping .....

.....

.....

.....

[4]

4 A program needs to sort an array of lowercase strings into descending alphabetic order. An example of the data is shown in Fig. 4.1.

sheep	rabbit	dog	fox	cow	horse	cat	deer

Fig. 4.1

(a) Show how a bubble sort would sort the data in Fig. 4.1.

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

..... **[6]**

(b) The algorithm will make use of a function, `contains`, that compares two strings and checks if the second string contains the first string. For example, calling the function with (`"fox"`, `"foxhound"`) this would return `true`.

The function needs to:

- Take two strings as parameters, `string1` and `string2`
- return `true` if `string1` is contained within `string2`, or both strings are identical
- return `false` if `string1` is not contained within `string2`

Write, using pseudocode, the function `contains`.

Annotate your pseudocode with comments to show how it solves the problem.

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

..... **[7]**

(c) **(i)** A merge sort could have been used instead of a bubble sort.

Describe how a merge sort differs from a bubble sort.

.....

.....

.....

.....

.....

.....

.....

.....

..... **[4]**

(ii) Name **two** sorting algorithms, other than a bubble sort and merge sort.

1 .....

2 .....

[2]

(d) Show how a binary search would be performed on the array shown in Fig. 4.2 to find the value 'duck'.

wolf	monkey	lion	iguana	goat	giraffe	frog	elephant	duck

Fig. 4.2

.....

.....

.....

.....

.....

.....

.....

.....

..... **[3]**

Section B

5 Kim is writing an object-oriented program for a four player board game. The board has 26 squares that players move around, as shown in Fig. 5.1.

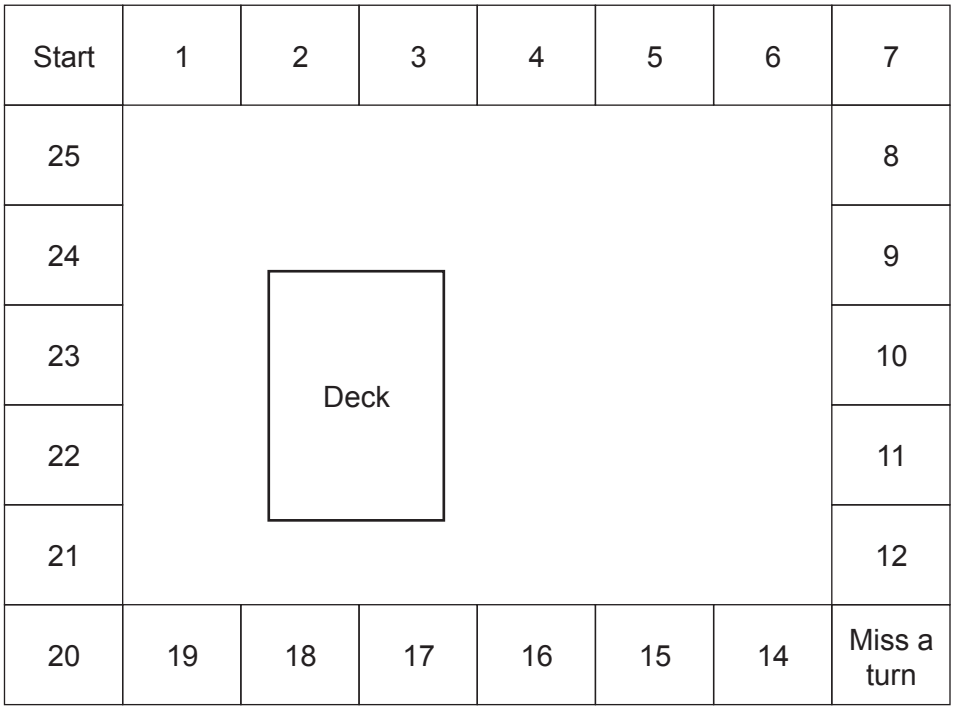


Fig. 5.1

Each player takes it in turn to roll two dice. They then move that number of spaces on the board. If they roll a double (both dice have the same value), they then take a card from the deck. The deck contains 40 cards that each include a sentence (such as “You have won the lottery”). The sentence on the card determines if money is given or taken away from the player.

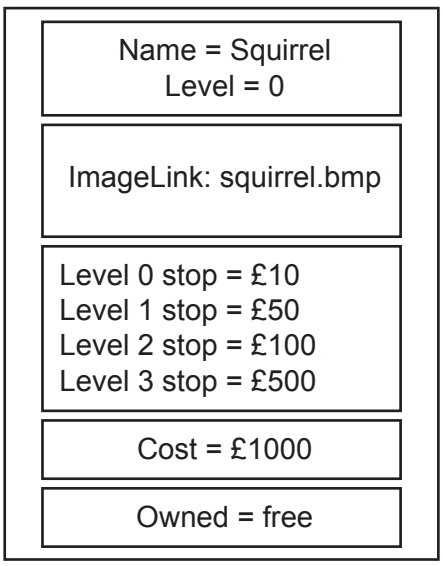


Fig. 5.2

Each square (apart from Start and Miss a turn) has an animal associated with it that the player can purchase, if it has not been purchased already, for example square 6 has a Squirrel. Fig. 5.2 shows an example of one of these animals. Once a player has purchased the animal, any opposing player which subsequently lands on the square/animal has to pay a fine.

Each animal can be upgraded, with each upgrade the game charges more each time a player stops on them. For example, with no upgrade the level 0 squirrel costs £10 when a player stops on it. If £1000 is paid to upgrade, the squirrel is then a level 1 animal and now charges £50 for a stop.

The cost to purchase and upgrade the animal is the same.

Each animal can be upgraded to a maximum of level 3.

When a player lands on, or passes the square ‘Start’ (position 0), they receive £500. If they land on ‘Miss a turn’ (position 13), they miss their next turn.

(a) (i) A class, `Player`, stores the player's ID (P1, P2, P3, P4), their current board position and the amount of money they have.

Fig. 5.3 shows a class diagram for `Player`. A class diagram describes a class. It contains the class name, followed by the attributes, then the methods.

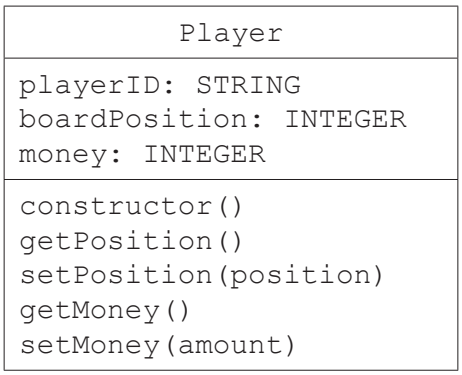


Fig. 5.3

The constructor creates a new instance of `Player`, taking the player's ID as a parameter. The board position is set to 0, and money to £2000.

Write, using pseudocode, the constructor method for the `Player` class.

.....

.....

.....

.....

..... **[3]**

(ii) A class, `Animal`, define the attributes and methods for the animals stored in each square.

Fig. 5.4 shows a class diagram for `Animal`.

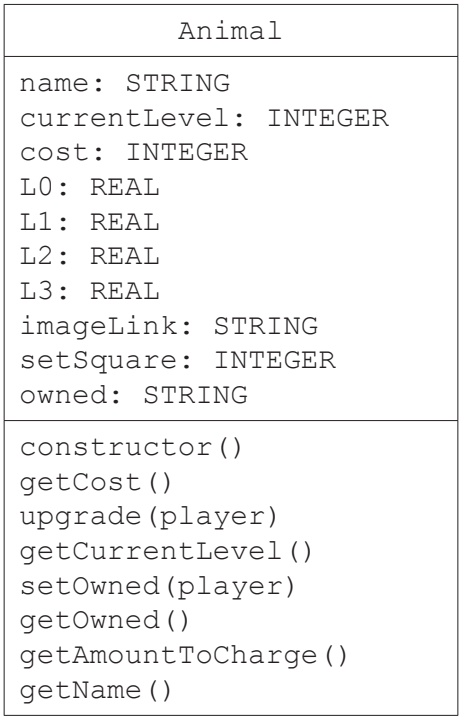


Fig. 5.4

The constructor takes the required data as parameters and then sets `currentLevel` to 0, and assigns the parameters as the remaining attributes for the new object.

Write, using pseudocode, the constructor method for the `Animal` class. **[4]**

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

(iii) Write, using pseudocode, the code to create an instance of `Animal` for the Squirrel shown in Fig. 5.2, positioned on square number 6, for the constructor function you wrote in part **(a)(ii)**.

.....

.....

..... **[2]**

(b) The board is stored as a 1D array, `board`, of data type `Animal`. The spaces at 0, and 13, are left as empty elements that are checked using separate functions.

(i) Complete, using pseudocode, the function to:

- Roll both dice
- Move the player, the dice number of spaces
- If a double is rolled, calls the procedure `pickDeck`
- Adds £500 if they have passed or landed on Start
- Calls the procedure `missAGo` if they land on space 13 or
- Calls the procedure `checkAnimal`
- Return the new position

```
function playerMove(currentPlayer)

    dice1 = random(1,6)

    dice2 = random(1,6)

    boardPosition = ........................... + dice1 + dice2

    if ..............  == dice2 then

        pickDeck(currentPlayer)

    endif

    if position > 25 then

        currentPlayer.setMoney(currentPlayer.getMoney() +  .............)

        position = position -  ..............................

    endif

    if position == ................................ then

        missAGo(currentPlayer)

    elseif position != 0 then

        checkAnimal(currentPlayer)

    endif

    ..............................................................

endfunction
```

[6]

(ii)* The parameter `currentPlayer` from part **(b)(i)** can be passed by value or by reference.

Explain the difference, benefits and drawbacks between passing by value and by reference. Recommend which should be used for `currentPlayer`, justifying your decision.

.... **[9]**

(c) The deck is stored as a zero-indexed 1D array, named `deck`, of type `Card`.

The class diagram for `Card` is shown in Fig. 5.5.

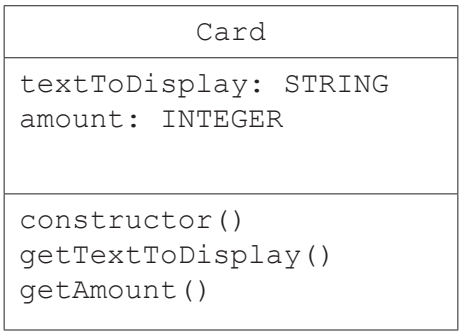


Fig. 5.5

The array, `deck`, is treated as a queue, with a variable, `headPointer`, identifying the first card in the deck. When a card has been used, the head pointer increases to move to the next position. If the end of the deck is reached, the head pointer returns to 0 and starts again.

The procedure `pickDeck`:

- takes the current player as a parameter
- outputs the text to be displayed from the first card in the queue
- adds or subtracts the amount to/from the current player's money
- increases the head pointer

Write, using pseudocode, the procedure `pickDeck`.

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

..... **[6]**

(d) The procedure `checkAnimal`:

- Takes the current player as a parameter
- Accesses the data for the animal at the player's position in the array `board`
- If the animal is free, asks the player if they would like to purchase the animal and outputs its `name` and `cost`, if they choose to buy the animal, it calls the procedure `purchase()` with the player and animal as parameters
- If that player owns the animal, and it is not at level 3, it asks if they would like to upgrade the animal
- If they would like to upgrade, it calls the method `upgrade` for that animal with the current player as a parameter
- If a different player owns the animal, it calls the method `getAmountToCharge()` for that animal, sending this value and the current player as parameters to the procedure `chargeStay()`

Write, using pseudocode, the procedure `checkAnimal`. **[10]**

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

END OF QUESTION PAPER

BLANK PAGE

BLANK PAGE