# Worksheet 3 Writing and interpreting algorithms Answers

**Task 1**

1. An Internet site has the following login procedure. To access their account, a customer has to enter

* a 10-character user ID (3 attempts allowed)
* a 4-digit PIN (1 attempt allowed)
* three random characters from their password (3 attempts allowed)

 Once the user enters their user ID, a subroutine is called which looks up their record and reads the stored userID, PIN and password.

 If the user fails after 3 attempts to enter an ID which is held in the record, a message is displayed “Incorrect ID – access denied”

 Similar messages are displayed if the user fails to enter a correct PIN or password

 If all details are correct, access to the account is permitted.

 (a) Draw a hierarchy chart to show the tasks and subtasks involved in the login procedure.

Login procedure

User ID

PIN

Look up customer details and check ID exists

Enter ID

Verify password characters

Generate 3 random numbers

Password

Students will come up with many variations on this, which may well be correct – the main thing to remember is that a hierarchy chart does not show the detail involved in any of the routines. It just shows the tasks and subtasks.

 (b) Write pseudocode for a subroutine to check that the user ID exists on Customer file.

 SUB userIDEntry

 attempts 🡨 0

 validID 🡨 FALSE

 OUTPUT “Please enter ID”

 WHILE validID = FALSE or attempts < 3

 userID = USERINPUT

 READ (CUSTOMER file, userID)

 IF userID exists THEN

 validID = TRUE

 ELSE

 attempts = attempts + 1

 OUTPUT “Invalid ID – please re-enter”

 ENDIF

 ENDWHILE
ENDSUB

 (c) What are the weaknesses in this login procedure? What improvements would you suggest?

 **Ease of use:**

 The user should be offered some way of receiving a forgotten user ID, PIN or password.
There could be a “Remember password” option so that they don’t have to enter details each time

 **Security:**

 If the user ID, PIN and password are stored on a file in unencrypted form, it would be easy for someone to hack in and find them out. They should be stored in encrypted form and the encryption algorithm also applied to whatever the user enters, so the two can be compared.

**Task 2**

2. The following numbers are to be sorted into ascending order using an insertion sort:

15, 73, 29, 66, 35, 11, 43, 21

 (a) Show the sequence of the numbers after each pass through the insertion sort algorithm

 userNumber = [15, 73, 29, 66, 35, 11, 43, 21]

 #assume first element of array is userNumber[0]

 FOR j = 1 to LEN(userNumber)

 nextNum = userNumber[j]

 i = j – 1

 WHILE i >= 0 and userNumber [i] > nextNum

 userNumber[i + 1] = userNumber [i]

 i = i - 1

 ENDWHILE

 userNumber[i + 1] = nextNum

 ENDFOR

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **15** | **73** | **29** | **66** | **35** | **11** | **43** | **21** |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **15** | **29** | **73** | **66** | **35** | **11** | **43** | **21** |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **15** | **29** | **66** | **73** | **35** | **11** | **43** | **21** |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **15** | **29** | **35** | **66** | **73** | **11** | **43** | **21** |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **11** | **15** | **29** | **35** | **66** | **73** | **43** | **21** |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **11** | **15** | **29** | **35** | **43** | **66** | **73** | **21** |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **11** | **15** | **21** | **29** | **35** | **43** | **66** | **73** |

 (b) How many passes are made through the data for a dataset of n numbers? n-1

 See Python program insertion sort.py

3. Write an algorithm which will compare the length of time to sort a list of n random numbers using a bubble sort and an insertion sort.

 If you have time, write the program in a language of your choice and experiment with different values of n. Be prepared to wait a long time if you choose a number greater than 10,000!

 You can find an algorithm for a **merge sort** on the Internet and time that one too.

 Which is fastest for 10 numbers?

 Which is fastest for 100 numbers? 1000 numbers?

 See Python program timing sorts.py

 The algorithm has a main program which asks the user how many numbers (n) they want to sort, generates n random numbers and puts them in 3 identical arrays numbers1, numbers2 and numbers3.

 The main program then gets the clock time, calls the first sort, gets the clock time again and prints the elapsed time. It does this again for the other two sorts.

 For a very small array of 10 numbers, the bubble sort and insertion sort are faster than the merge sort. For 10,000 numbers, however, the merge sort takes only a small fraction of the time taken by the other sorts.

 If you run the program with the same number of values each time, the timing will be different because the computer is multiprocessing and it depends what other tasks it is carrying out. A smaller effect is how well sorted the initial list of numbers is, but as these are random numbers, this will be negligible.