# Assessment test Answers

1. (a) Give **three** essential properties of a recursive routine. [3]

 It must have a stopping condition / base case which when met means that the routine will not call itself and will start to unwind

 For input values other than the stopping condition the routine must call itself

 The stopping condition must be reached after a finite number of calls

 (b) Which of the following subroutines are recursive? [3]

 (i) SUB addNumbers(num,x)

 IF num > 1

 x = num + addNumbers(num – 1, x)

 ENDIF

 RETURN x

 ENDSUB

 (ii) SUB printNumbers(num)

 OUTPUT (num)

 RETURN num + printNumbers(num – 1)

 ENDIF

 ENDSUB

 (iii) SUB numbers(num)

 IF num <= 0 THEN

 RETURN 0

 ELSE

 RETURN numbers(num) - 1

 ENDIF

 ENDSUB

 (i) is the only recursive routine. (ii) has no base condition. (iii) never reaches the stopping condition as num never decreases.

 ***See Python program Unit assessment 1b recursive function.py***

 If the recursive routine is called with the statement

 x = addNumbers(6,0)

It will print

2

5

9

14

20

 (c) The following pseudocode subroutine merges two sorted lists of equal or unequal lengths. It is called with the two lists shown in the main program.

 NOTE: Given that list1 = [1, 2, 5, 10], then list1[1:] = [2, 5, 10].

 That is, it returns a sublist starting with list1[1].

1. SUB merge (list1,list2)
2. IF length(list1) = 0 and length(list2) = 0
3. return []
4. ENDIF
5. IF length (list1) = 0
6. return list2
7. ENDIF
8. IF length (list2) = 0
9. return list1
10. ENDIF
11. IF list1[0] <= list2[0]
12. return [list1[0]] + merge(list1[1:], list2)
13. ENDIF
14. IF list2[0] <= list1[0]
15. return [list2[0]] + merge(list1, list2[1:])
16. ENDIF
17. ENDSUB
18. #main program
19. list1 = [3, 7, 9]
20. list2 = [1, 2, 5, 10]
21. mergedList = merge(list1,list2)
22. print(mergedList)

 (i) What will be printed when the statement print(mergedList) at line 23 is executed? [1]

 [1, 2, 3, 5, 7, 9, 10] (nb only the numbers are important, not the commas or brackets)

 (ii) Give the statement numbers of the recursive calls in the subroutine. [2]

 12, 15

 (iii) What parameters will be passed to the subroutine the first, second and third times that a recursive call is made? [3]

 First time: [3, 7, 9], [2, 5, 10]

 Second time: [3, 7, 9], [5, 10]

 Third time: [7,9], [5, 10]

2. (a) The Big-O notation gives a measure of the time complexity of an algorithm relative to the size of the problem.

 Three different algorithms A, B, C to solve the same problem are:

 A = O(n2), B = O(log2n) and C = O(n!)

 (i) Which is the most efficient algorithm in terms of execution time? B [1]

 (ii) Which is the least efficient algorithm in terms of execution time? C [1]

 (b) Use the Big-O notation to express the complexity of the following pseudocode segments. In each case, show how you arrive at the answer.

 (i) i = n

 WHILE i > 0

 i = i DIV 2 # (DIV =integer division)

 ENDWHILE [2]

 Starting with say n = 16, i will go from 16 to 8, 4,2 ie 4 steps which is log2 16. This is a logarithmic function, O(log n)

 (ii) FOR i = 1 TO n

 FOR j = 1 TO n

 FOR k = 1 TO n

 listX[i, j, k] = 0

 ENDFOR

 ENDFOR

 ENDFOR [2]

 The assignment statement is performed n3 times, so this is O(n3)

(c) A subroutine big(aList) is given below.

 SUB bigO(aList)

 numberOfPrints = 0

 n = length(aList)

 FOR i = 0 TO n-1

 OUTPUT("In outer loop: ", aList[i])

 numberOfPrints = numberOfPrints + 1

 k = n/2 - 1

 FOR j = 0 TO k

 OUTPUT(" In inner loop: ",aList[j])

 numberOfPrints = numberOfPrints + 1

 ENDFOR

 ENDFOR

 OUTPUT("number Of Print statements executed: ", numberOfPrints)

 numberOfPrints = numberOfPrints + 1

 ENDSUB

 #main program

 listOfItems = [1,2,3,4,5,6]

 bigO(listOfItems)

 (i) Calculate the number of times an OUTPUT statement is executed in the subroutine **bigO(aList),** shown below, called from the main program and accepting as a parameter a list of 6 items. [4]

 The first outer loop is performed n times, and there is a single print statement in the loop. (1 mark) The inner loop is performed n/2 times and contains a single print statement.(1 mark) This gives a total of n(1 + n/2).(1 mark)

 In this case n = 6 so there are 6 + 36/2 = 24 print statements

 There is an extra print statement outside the FOR loop, so the total is 25.(1 mark)

(ii) Hence, express its time complexity using Big-O notation and explain your answer.

 [2]

 Time complexity = O(n2) (1 mark)

 since the dominant term is n2/2 and the coefficient of n2 and the term in n can be ignored.(1 mark)

3. A procedure to process an array of numbers is defined as follows.

 SUB p(listA)

 i 🡨 length(listA)-1

 flag 🡨 True

 WHILE i > 0 AND flag = True

 flag 🡨 False

 FOR j = 0 to i - 1

 IF listA[j] > listA[j + 1] THEN temp 🡨 listA(x)

 temp = listA[j]

 listA[j] 🡨 listA[j + 1]

 listA[j + 1] 🡨 temp

 flag 🡨 True

 ENDIF

 ENDFOR

 i 🡨 i - 1

 ENDWHILE

 ENDSUB

The array listA[19,10,21,7,26,16] is to be processed by this procedure.

 (a) List the array after the WHILE loop has been executed once. [2]

 10 19 7 21 16 26

 (b) What algorithm does the procedure p describe? [1]

 Bubble sort

 (c) What is the purpose of flag in this procedure? [2]

 If no swaps are made in the FOR loop, the flag remains False,
indicating that the list is sorted, and the procedure ends.

 This makes the procedure more efficient for a partially sorted list.

 (d) (i) Trace the execution of the subroutine p if

 listA = [2, 5, 8, 12, 16] [5]

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **i** | **Flag** | **j** | **listA[j]** | **listA[j + 1]** |
| 4 | True |  |  |  |
|  | False | 0 | 2 | 5 |
|  |  | 1 | 5 | 8 |
|  |  | 2 | 8 | 12 |
|  |  | 3 | 12 | 16 |
| 3 |  |  |  |  |

 (ii) How many times is the WHILE loop executed? [1]

 Once, since the list is already sorted and the flag is False after the first execution of the outer loop

4. Djikstra’s algorithm finds the shortest path between a given start node and every other node in a weighted graph.

 (a) Describe briefly **two** applications of Djikstra’s algorithm. In each case state what the weights on the edges represent. [4]

 Sight-seeing map of Paris. The nodes represent the major attractions, such as Notre Dame and the Eiffel Tower. The edges represent the distance between each attraction. A route can be planned that will involve the shortest amount of walking.

 In project management, the nodes could represent milestones. The edges could represent the number of hours required to move from milestone to milestone. As long as there were no precedence rules, the shortest path would represent the fewest hours of work.

 Shown below is a weighted graph.



 (b) In the algorithm below to find the shortest distances from A to the other nodes, a tentative distance from A to every other node is initially assigned.

 (i) In the table below, show what these tentative distances are. [2]

 (ii) In the second row of the table, show the tentative distances after A and the next node have been visited. [4]

 (D is visited next as it is closest to A. D’s neighbours are C and F)

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | **A** | **B** | **C** | **D** | **E** | **F** |
| Initial tentative distance  | 0 | ∞ | ∞ | ∞ | ∞ | ∞ |
| tentative distance after 2 nodes visited | 0 | 10 | 11 | 6 | ∞ | 18 |

 Assign a tentative distance value to every node

Add all the vertices to a priority queue, sorted by current distance

 WHILE priority queue is not empty

 remove the vertex u from the front of the queue

 FOR each unvisited neighbour w of the current vertex u

 newDistance 🡨 distanceAtU + distanceFromUtoW

 IF newDistance < distanceAtW THEN

 distanceAtW 🡨 newDistance

change position of w in priority queue to reflect new distance to w

 ENDIF

 ENDFOR

 ENDWHILE

5. (a) What is meant when a problem is said to be **intractable**? [2]

 An intractable problem cannot be solved with a known algorithm with a time complexity of polynomial or better. For example, problems that can be solved with algorithms of O(n), O(n2), O(nk), O(log n), O(n log n) are tractable, whereas those solved with algorithms of O(n!), or O(2n) are considered intractable. Intractable problems are not soluble in an efficient amount of time. They are not, however, insoluble.

 (b) Give an example of an intractable problem. [1]

 The travelling salesman problem, whereby a number of cities are to be visited exactly once each. For small numbers of cities, the solution can be found. For larger numbers of cities, the solution cannot be computed in adequate time, if ever.

 (c) Name and describe briefly another approach to solving a problem if no algorithm can be found which will find a solution in a reasonable time. [2]

 A heuristic algorithm can be used. This is a general rule of thumb approach and will often generate a solution that is ‘good enough’ in the circumstances. It may not be an optimal solution, but one which will suffice.

 Total 50 marks