2.1 Algorithms: 30-minute Pack

OCR J277 · Calculators not allowed · about 30 minutes

Total Marks

Name: _____ Date: ____ / ____ / _____

Answer ALL questions. Show all your working.

1 A programmer writes the following pseudocode, using the OCR Exam Reference Language.

```
total = 0
for i = 1 to 5
total = total + i
next i

print(total)
```

(a) Complete a trace table for this algorithm, showing the value of i and total at the end of each pass through the loop. **(4)**

(b) State the value output by the algorithm. **(1)**

(Total for Question 1 is 5 marks)

© Revision Library | revisionlibrary.co.uk

2 An array called scores contains the values [55, 62, 48, 71, 60, 39] at indices 0 to 5.

(a) Describe how a linear search algorithm would locate the value 71 in this array, stating the number of comparisons needed. **(3)**

(b) State one disadvantage of using a linear search, rather than a binary search, on a large, sorted dataset. **(1)**

(Total for Question 2 is 4 marks)

---

3 An array called scores has 10 elements, at indices 0 to 9.

(a) Write an algorithm, using pseudocode, that finds and outputs the largest value in the array scores. **(4)**

(Total for Question 3 is 4 marks)

---

4 An algorithm uses a FOR loop that runs from 0 to 15 inclusive, using a variable called cycle to count each iteration.

(a) State the 8-bit binary representation of the maximum value that cycle reaches. **(1)**

(b) Convert the maximum value of cycle to hexadecimal. **(1)**

(c) Explain why programmers often use hexadecimal, rather than binary, when reading values from trace tables or debugging output. **(2)**

(Total for Question 4 is 4 marks)

© Revision Library | revisionlibrary.co.uk

5 An array contains the values [8, 3, 6, 1, 9], which need to be sorted into ascending order using a bubble sort that compares and swaps adjacent items.

(a) Show the state of the array after each comparison in the first full pass of the bubble sort. **(4)**

(b) State the maximum number of full passes a bubble sort would need to fully sort a list of 5 items. **(1)**

(Total for Question 5 is 5 marks)

---

6 Merge sort is an example of a 'divide and conquer' algorithm.

(a) Describe how merge sort would sort a list of 8 unsorted numbers, referring to both the dividing stage and the merging stage. **(4)**

(b) State one advantage of merge sort over bubble sort when sorting a very large dataset. **(1)**

(Total for Question 6 is 5 marks)

© Revision Library | revisionlibrary.co.uk

Mark scheme · 2.1 Algorithms: 30-minute Pack

Question 1

(a) i = 1, total = 1 **B1**

(a) i = 2, total = 3 **B1**

(a) i = 3, total = 6; i = 4, total = 10 **B1**

(a) i = 5, total = 15 (all five rows correct, cao) **B1**

(a) Answer: i: 1,2,3,4,5 -> total: 1,3,6,10,15

(b) 15 (cao) **B1**

(b) Answer: 15

Question 2

(a) starts at index 0 and compares each item in turn to the target value (71) (oe) **B1**

(a) moves to the next index each time the current item does not match, continuing until a match is found (oe) **B1**

(a) correctly states 4 comparisons are needed (55, 62, 48, 71 - cao) **B1**

(a) Answer: The linear search checks index 0 (55), index 1 (62), index 2 (48), then index 3 (71), which matches. 4 comparisons are needed.

(b) a linear search can require checking every item in the worst case, so it is much slower/less efficient on large datasets than a binary search (oe) **B1**

(b) Answer: A linear search may need to check every item in the dataset, making it much slower than a binary search for large, sorted data.

Question 3

(a) initialises a variable to the first element of the array, e.g. max <- scores[0] **M1**

(a) uses a loop to iterate through the remaining elements of the array, e.g. FOR i <- 1 TO 9 **M1**

(a) compares each element to the current maximum and updates it if a larger value is found, e.g. IF scores[i] > max THEN max <- scores[i] **M1**

(a) outputs the maximum value after the loop has finished (any correct logic; ignore minor syntax errors) **A1**

(a) Answer:

```
max <- scores[0]
FOR i <- 1 TO 9
    IF scores[i] > max THEN
        max <- scores[i]
    ENDIF
NEXT i
OUTPUT max
```

Question 4

(a) 00001111 (cao) **B1**

(a) Answer: 00001111

(b) 0F (accept F, cao) **B1**

(b) Answer: 0F

(c) hexadecimal is much more compact/shorter than binary, so it is quicker and easier for a human to read (oe) **B1**

(c) each hexadecimal digit represents exactly 4 bits, so converting between binary and hexadecimal is quick and there is less chance of error than reading long binary strings (oe) **B1**

(c) Answer: Hexadecimal is far more compact than binary and easier for humans to read; each hex digit maps to exactly 4 bits, making conversion quick and reducing the chance of misreading long binary strings.

Question 5

(a) compare 8 and 3: swap, giving [3, 8, 6, 1, 9] **B1**

(a) compare 8 and 6: swap, giving [3, 6, 8, 1, 9] **B1**

(a) compare 8 and 1: swap, giving [3, 6, 1, 8, 9] **B1**

(a) compare 8 and 9: no swap needed, array remains [3, 6, 1, 8, 9] (cao) **B1**

(a) Answer: [3,8,6,1,9] -> [3,6,8,1,9] -> [3,6,1,8,9] -> [3,6,1,8,9] (end of pass 1)

(b) 4 (n - 1 passes for n = 5 items, cao) **B1**

(b) Answer: 4

Question 6

(a) the list is repeatedly split in half (divided) until each sub-list contains only one item (oe) **B1**

(a) a sub-list containing a single item is considered already sorted (oe) **B1**

(a) adjacent sub-lists are then merged back together, comparing the front item of each sub-list (oe) **B1**

(a) the smaller/next item is placed into the new merged list, and this repeats until all sub-lists have been merged into one fully sorted list (oe) **B1**

(a) Answer: The list is repeatedly divided in half until each sub-list has one item, then pairs of sub-lists are merged back together in order, comparing front items, until one fully sorted list remains.

(b) merge sort is more efficient/faster for large datasets, needing far fewer comparisons than bubble sort (oe) **B1**

(b) Answer: Merge sort is significantly more efficient than bubble sort for large datasets, requiring many fewer comparisons overall.

© Revision Library | revisionlibrary.co.uk