2.8 Recursion and Advanced Algorithms: 30-minute Pack

Total Marks

AQA 7517 · Calculators not allowed · about 30 minutes

Name: _____ Date: ____ / ____ / _____

Answer ALL questions. Show all your working.

1 An array called catalogue, indexed from 0, holds 10 sorted reference numbers:

catalogue = [2, 5, 8, 12, 16, 23, 38, 45, 56, 72] (indices 0 to 9)

A recursive binary search function is written as follows.

```
FUNCTION BinarySearch(list, target, low, high)
   IF low > high THEN
      RETURN -1
   ENDIF
   mid <- (low + high) DIV 2
   IF list[mid] = target THEN
      RETURN mid
   ELSE
      IF list[mid] < target THEN
         RETURN BinarySearch(list, target, mid + 1, high)
      ELSE
         RETURN BinarySearch(list, target, low, mid - 1)
      ENDIF
   ENDIF
ENDFUNCTION
```

(a) State one precondition that must hold for the array catalogue before binary search can be used on it correctly. **(1)**

(b) Complete a trace table for the call BinarySearch(catalogue, 23, 0, 9), showing the values of low, high, mid and list[mid] for each recursive call made, until the target is found. **(5)**

(c) State the value returned by the original call, and the total number of recursive calls made (including the first call). **(2)**

(d) State the time complexity of binary search in Big O notation, and explain why doubling the number of elements in a sorted array only adds, at most, one extra comparison to a binary search of it. **(2)**

(Total for Question 1 is 10 marks)

© Revision Library | revisionlibrary.co.uk

2 Merge sort is a recursive, divide-and-conquer sorting algorithm. It is applied to the list [8, 3, 5, 1, 9, 2].

(a) State the two main phases of the merge sort algorithm. **(2)**

(b) State the two sub-lists produced when [8, 3, 5, 1, 9, 2] is first split in half. **(2)**

(c) The sub-list [8, 3, 5] is recursively split down into single-element lists ([8], [3] and [5]) and then merged back together. State the result of merging [3] and [5] first, and then state the result of merging [8] with that result. **(2)**

(d) By the same process, [1, 9, 2] merges down to [1, 2, 9]. State the fully sorted list produced when [3, 5, 8] and [1, 2, 9] are finally merged together, and state the worst-case time complexity of merge sort in Big O notation. **(2)**

(e) State one disadvantage of merge sort compared with an in-place sorting algorithm such as insertion sort. **(1)**

(Total for Question 2 is 9 marks)

---

3 In the Towers of Hanoi puzzle, n disks of different sizes are stacked on a source peg, and must all be moved to a destination peg, using a third, spare peg, following two rules: only one disk may be moved at a time, and a larger disk may never be placed on top of a smaller one.

(a) State the recursive rule used to move n disks (n > 0) from a source peg to a destination peg, using a spare peg, and state the base case for n = 0. **(3)**

(b) Using AQA pseudocode conventions, write a recursive procedure Hanoi(n, source, destination, spare) that outputs the sequence of moves needed. **(4)**

(c) State the formula for the minimum number of moves needed to solve the puzzle with n disks, and use it to calculate the number of moves needed for 5 disks. **(2)**

(d) State the time complexity of the Hanoi procedure in Big O notation, in terms of n, and explain why. **(2)**

(Total for Question 3 is 11 marks)

© Revision Library | revisionlibrary.co.uk

Mark scheme · 2.8 Recursion and Advanced Algorithms: 30-minute Pack

Question 1

(a) the array must already be sorted (into ascending order, as given) oe **B1**

(a) Answer: The array must already be sorted into order.

(b) call 1: low = 0, high = 9, mid = 4, list[4] = 16 **B1**

(b) call 1 comparison: 16 < 23, so the next call has low = 5, high = 9 **B1**

(b) call 2: low = 5, high = 9, mid = 7, list[7] = 45 **B1**

(b) call 2 comparison: 45 > 23, so the next call has low = 5, high = 6 **B1**

(b) call 3: low = 5, high = 6, mid = 5, list[5] = 23, matches target cao **B1**

(b) Answer: Call 1: low=0, high=9, mid=4, list[4]=16 (16<23, so low<-5). Call 2: low=5, high=9, mid=7, list[7]=45 (45>23, so high<-6). Call 3: low=5, high=6, mid=5, list[5]=23 (match).

(c) 5 is returned ft **B1**

(c) 3 calls in total ft **B1**

(c) Answer: 5 is returned (the target 23 is stored at index 5); 3 calls are made in total.

(d) O(log n) oe **B1**

(d) each comparison eliminates half of the remaining elements from consideration, so doubling n only requires one more halving (one more comparison) to reduce the search space down to a single element (oe) **B1**

(d) Answer: O(log n). Each comparison halves the number of elements still being searched, so however large the array is, only one extra comparison is needed each time its size doubles, since one more halving is enough to bring it back down to the same remaining search space as before.

Question 2

(a) divide: the list is recursively split in half until every sub-list contains only one element (oe) **B1**

(a) merge: pairs of sorted sub-lists are then repeatedly combined back together, in order, until the whole list is a single sorted list again (oe) **B1**

(a) Answer: Divide: the list is recursively split into halves until every sub-list holds just one element. Merge: sorted sub-lists are then repeatedly combined back together in the correct order, until the whole list is fully sorted again.

(b) [8, 3, 5] **B1**

(b) [1, 9, 2] **B1**

(b) Answer: [8, 3, 5] and [1, 9, 2]

(c) merging [3] and [5] gives [3, 5] **B1**

(c) merging [8] with [3, 5] gives [3, 5, 8] cao **B1**

(c) Answer: Merging [3] and [5] gives [3, 5]. Merging [8] with [3, 5] gives [3, 5, 8].

(d) [1, 2, 3, 5, 8, 9] cao **B1**

(d) O(n log n) **B1**

(d) Answer: [1, 2, 3, 5, 8, 9]; worst-case time complexity O(n log n).

(e) merge sort needs extra (auxiliary) memory, proportional to the size of the list (O(n)), to hold the sub-lists during merging, since it is not an in-place algorithm (oe) **B1**

(e) Answer: Merge sort is not in-place: it needs additional memory, proportional to the size of the list, to hold the sub-lists while merging, unlike an in-place algorithm such as insertion sort, which sorts within the original array.

© Revision Library | revisionlibrary.co.uk

Question 3

(a) move the top n - 1 disks from source to spare, using destination as the temporary peg **B1**

(a) then move the single remaining (largest) disk directly from source to destination **B1**

(a) then move the n - 1 disks from spare to destination, using source as the temporary peg; base case: if n = 0, do nothing (there are no disks to move) **B1**

(a) Answer: Move the top n - 1 disks from source to spare (using destination as the temporary peg); move the remaining largest disk from source directly to destination; then move the n - 1 disks from spare to destination (using source as the temporary peg). Base case: if n = 0, there is nothing to move, so the procedure does nothing.

(b) procedure header PROCEDURE Hanoi(n, source, destination, spare) defined **M1**

(b) base case guard, e.g. IF n > 0 THEN (equivalent to stopping when n = 0) **M1**

(b) recursive calls in the correct order: CALL Hanoi(n - 1, source, spare, destination) before, and CALL Hanoi(n - 1, spare, destination, source) after, moving the largest disk **M1**

(b) correct OUTPUT statement for moving the single disk, placed between the two recursive calls cao **M1**

(b) Answer:

```
PROCEDURE Hanoi(n, source, destination, spare)
   IF n > 0 THEN
      CALL Hanoi(n - 1, source, spare, destination)
      OUTPUT "Move disk ", n, " from ", source, " to
      CALL Hanoi(n - 1, spare, destination, source)
   ENDIF
ENDPROCEDURE
```

(c) formula: $2^n - 1$ **B1**

(c) for n = 5: $2^5 - 1 = 31$ cao **B1**

(c) Answer: $2^n - 1$; for n = 5 this gives $2^5 - 1 = 31$ moves.

(d) $O(2^n)$ **B1**

(d) because each call to Hanoi(n, ...) makes two further recursive calls, each on n - 1, so the number of calls (and moves) roughly doubles for every extra disk, matching the exponential growth of the $2^n - 1$ moves formula from part (c) (oe) **B1**

(d) Answer: $O(2^n)$. Each call to Hanoi(n, ...) makes two further recursive calls on n - 1, so the total number of calls made (and moves output) roughly doubles every time n increases by 1, which is exactly the exponential growth shown by the $2^n - 1$ moves formula in part (c).

© Revision Library | revisionlibrary.co.uk