Computer Science (H046, H446)

searching and sorting - Time complexity
Mr. Montgomery

Please note that you may see slight differences between this paper and the original.

Candidates answer on the Question paper.

OCR supplied materials:
Additional resources may be supplied with this paper.

Other materials required:
- Pencil
- Ruler (cm/mm)

35/40

Duration: Not set

Candidate forename	Daillen	Candidate surname	Lui

Centre number						Candidate number				

INSTRUCTIONS TO CANDIDATES

- Write your name, centre number and candidate number in the boxes above. Please write clearly and in capital letters.
- Use black ink. HB pencil may be used for graphs and diagrams only.
- Answer **all** the questions, unless your teacher tells you otherwise.
- Read each question carefully. Make sure you know what you have to do before starting your answer.
- Where space is provided below the question, please write your answer there.
- You may use additional paper, or a specific Answer sheet if one is provided, but you must clearly show your candidate number, centre number and question number(s).

INFORMATION FOR CANDIDATES

- The quality of written communication is assessed in questions marked with either a pencil or an asterisk. In History and Geography a *Quality of extended response* question is marked with an asterisk, while a pencil is used for questions in which *Spelling, punctuation and grammar and the use of specialist terminology* is assessed.
- The number of marks is given in brackets **[ ]** at the end of each question or part question.
- The total number of marks for this paper is 40.
- The total number of marks may take into account some 'either/or' question choices.

1 A 1-dimensional array stores a set of numbered cards from 0 to 7. An example of this data is shown in Fig in 4.1

2	0	1	7	4	3	5	6

Fig 4.1

The programmer wants to search for a specific card in the array.

State whether a binary search or a linear search would be the most appropriate method to search for a specific card, and justify your answer.

Search method Linear search

Justificatio
n binary search only works for ordered ~~lists~~ data, and since the ~~list~~ data is not numerically ordered since 2 is before 0 even though 2 is larger, binary will not work. linear works since it ~~as~~ simply runs through every item in the array starting from The number 2. [3]

2(a) Linear search and binary search are two different algorithms which can be used for searching arrays.

When comparing linear and binary search it is possible to look at the best, worst and average number of items in the array that need to be checked to find the item being searched for. Assume every item in the array is equally likely to be searched for.

Complete the table below

	Worst Case number of searches	Average Case	Best Case
Binary Search	~~$O(\log_2 n)$~~ $\log_2 n$	$\text{Log}_2(n)$-1	~~O(1)~~ 1
Linear Search	~~O(n)~~ n	n/2	~~to. O(1)~~ 1

[4]

(b) As the size of an array increases the average number of checks grows logarithmically. State what is meant by logarithmic growth.

The increase amount halves each time, B.O.D

not necessarilly halves

[1]

(c) Assuming an array is sorted give a situation when a linear search would perform better than a binary search.

If the item you are looking for is the first item in the array (it is the smallest it is first).

[1]

return

3(a) A programmer has been tasked with writing a function that uses a binary search to return a Boolean value. The function should return `true` if the target integer is found in a list of integers. Using pseudocode, write an algorithm for the function.

```
dataItem = integer (input ("enter number to search for"))
~~requestedItem~~
~~procedure~~
function binarySearch (array, requestedItem)
        startPointer = 0
        endPointer = length(array) -1                              ✓
        middlePointer = (startPointer + endPointer) DIV 2          ✓
        while startPointer <= endPointer
              if array[~~start~~ middlePointer] == requestedItem then   ✓
                    return ~~True~~ true
              else
                    if requestedItem > array[middlePointer] then
                          startPointer = middlePointer +1              ✓
                          middlePointer = (startPointer + EndPointer) DIV 2
                    else
                          endPointer = middlePointer -1                ✓
                          middlepointer = (startPointer + endPointer) DIV 2
                    endif
              endif
        endwhile
        return -1   # only runs if item not in list
end function
```

[8]

False

B.O.D

(b) The target integer 8 exists in a list of integers 1, 4, 6, 9, 8, 12, 15 but is not found during a binary search. There are no errors in the code.

unsorted

(i) Give the reason why the target integer 8 is **not** found.

8 comes after 9, which is the first item, since list is not ordered, the program 'cuts' off the right side of list, including 8, so it never gets to examine another 8. [1]

(ii) Identify and describe an alternative search algorithm that could be used.

Since list is unordered, Linear search would be fine. Linear search starts at the first item (1), examines it, if it is 8, or the item you are looking for, it returns true and the index ~~position~~ of the item, if this item is not the one you are looking for, move onto the next item in list which is right after the one just examined. Repeat until you reach end of list. If item not found, ~~return~~ tell the user that item isn't in list or return false [3]

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

objective: sheep , rabbit , horse , fox , dog , deer , cow , cat

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

pass 1: sheep, rabbit, dog, fox, cow, horse, cat, deer

~~→ sheep, rabbit, fox, dog, cow, horse, cat, deer~~

~~→ sheep, rabbit, fox, dog, horse, deer,~~

→ sheep, rabbit, fox, dog, horse, cow, deer, cat

pass 2: sheep, rabbit, fox, dog, horse, cow, deer, cat

→ sheep, rabbit, fox, horse, dog, deer, cow, cat

pass 3: sheep, rabbit, fox, horse, dog, deer, cow, cat

→ sheep, rabbit, horse, fox, dog, deer, cow, cat

pass 5: sheep, rabbit, horse, fox, dog, deer, cow, cat

→ sheep, rabbit, horse, fox, dog, deer, cow, cat.

since no items need to be swapped in pass 5, the list is now sorted

: Bubble sort runs through every index each pass once, by the end **[6]**
of each pass, at least 1 item will be in it's final and correct position.

(3)

You need to show / identify
every swap individually
when showing how bubble
sort works.

5(a) A programmer needs to sort an array of numeric data using an insertion sort.

(i) The following, incomplete, algorithm performs an insertion sort.

Complete the algorithm.

```
procedure sortit(dataArray, lastIndex)
  for x = 1 to lastIndex
    currentData = dataArray[x-1]  ✗
    position = x
    while (position > 0 AND dataArray[position-1] > currentData)
      dataArray[position]  = dataArray[position-1]  ✓
      position = position - 1
    endwhile

    dataArray[position] = currentData  ✓
  next x
endprocedure
```

[3]

2

(ii) Show how an insertion sort would sort the following data:

6	1	15	12	5	6	9

6, 1, 15, 12, 5, 6, 9

1. 6, 1, 15, 12, 5, 6, 9 ✓ nothing happens yet, this is just the list of integers before sorting

2. 1, 6, 15, 12, 5, 6, 9 ✓ 1 swaps with ~~15~~ 6 since it is smaller

3. 1, 6, 15, 12, 5, 6, 9 ✓ 15 stays where it is since its the largest, the other items will soon be sorted ~~to~~ such that they are ~~ate~~ all before 15

4. 1, 6, ~~15~~ 12, 15, 5, 6, 9 ✓

5. 1, 5, 6, 12, 15, 6, 9 ✓

6. 1, 5, 6, 6, 12, 15, 9 ✓

7. 1, 5, 6, 6, 9, 12, 15 ✓

8. 1, 5, 6, 6, 9, 12, 15, this checks and makes sure there is no more ~~to~~ items that need sorting.

[6] 6

(b)

(i) Using Big-O notation state the best case complexity of insertion sort.

~~O(n)~~ $\Omega(n)$ 1 [1]

(ii) Explain what your answer to part **(b)(i)** means.

The best case is if the list is already sorted, meaning the algorith only has to run through the list which is n operations where n is the length of list, to make sure the list is sorted and that no more items need to be swapped. Ω means Omega which stands for best case in big-O notation.

Explain what linear time complexity means. [3]

2

END OF QUESTION PAPER