University of Nottingham

DEPARTMENT OF COMPUTER SCIENCE

A LEVEL 2 MODULE, SEMESTER 1 1997–1998

ALGORITHMS AND DATA STRUCTURES
(Course G52ADS)

Time allowed TWO hours

Candidates must NOT start writing their answers until told to do so

Credit will be given for the best THREE answers

Marks available for sections of questions are shown in brackets in the right-hand margin

Only silent, self-contained calculators with a single-line display are permitted in this examination. Dictionaries are not allowed with one exception. Those whose first language is not English may use a dictionary to translate between that language and English provided that neither language is the subject of this examination.

1 The overall structure of the quicksort algorithm is

```
quicksort(vector↕)
DoQuickSort(vector↕, 1, N)

DoQuickSort(vector↕, first↓, last↓)
if last > first
   pivot := Partition(vector, first, last)
   DoQuickSort(vector, first, pivot-1)
   DoQuickSort(vector, pivot+1, last)
```

(a) Give a (pseudo-code) implementation of the `Partition` algorithm, and describe how quick sort works. (9)

(b) The best case time complexity of quick sort is $O(n \, lg \, n)$. Sketch a proof of this result. (8)

(c) What is the worst case time complexity of quick sort, and under what circumstances does it arise? (2)

(d) Given an unordered vector of integers, suggest how you could use the `Partition` algorithm to find the median value. (6)

2 This question concerns three different data structures and search algorithms, each appropriate for a different situation.

(a) You need to read in a (known length) sequence of integers presented in sorted order, and store them in some data structure. Once the integers have been read in, there will be no further insertions or deletions. The values of the integers have a uniform distribution.

(i) What data structure would you use to store the integers? (1)

(ii) What algorithm would you use to search for whether a given integer is present? Give pseudo-code for the search algorithm. (4)

(b) The integers are read in unsorted order, and stored in some data structure. Further insertions and deletions are possible.

(i) What data structure would you use to store the integers? (1)

(ii) Give pseudo-code for the algorithm to search for an integer. (4)

(iii) Describe the procedures for inserting and deleting new integers (code not required). (5)

(c) The integers are read in sorted order, and stored in some data structure. Further insertions are possible, but there are to be no deletions. New items to be inserted will be presented in roughly sorted order.

(i) What data structure would you use to store the integers, and what search algorithm would you use? (2)

(ii) Describe the operations required to insert a new integer (code not required). (8)

3 You are asked to design a program to count word frequencies. This reads in an ASCII text file, and builds a running total of how many times the words in it (re)occur: e.g. *the* 57 times; *is* 15 times; *refinement* 1 time. The program should print out a frequency count of the words in frequency order, with the least frequent words printed first: e.g.

```
refinement:  1
object:  1
...
is:  15
...
the:  57
```

Describe the data structures and algorithms you would employ in the program, explaining how they are to be used, and the reasons for your choices. Where appropriate, specify ADTs, pseudo-code for significant ADT methods employed, and an outline of the entire algorithm (25)

[Hints: (a) You can assume the existence of a procedure `nextWord` that reads the next word in the file and returns a string for the word, all in lower case. The procedure returns ``EOF'' at the end of the file.
(b) You may also assume a data structure for strings that supports comparison operators like `>=`, `==`, `<`.
(c) Word frequencies obey Zipf's law: There will be a large number of different words occurring at low frequencies, e.g. 1, 2 or 3 times in a text; and a small number of common words occurring at high frequencies, typically with large gaps between their frequencies. You may wish to take account of this when choosing a space efficient data structure to store the words and frequencies in frequency sorted order.]

4 Write notes on each of the following:

(a) The O-notation for complexity, and its pitfalls. (5)

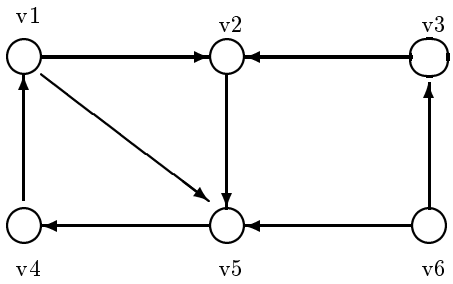
(b) Proving the correctness of algorithms. (5)

(c) The merits and demerits of vector and linked-list implementations of linear collections. (5)

(d) Eliminating recursion. (5)

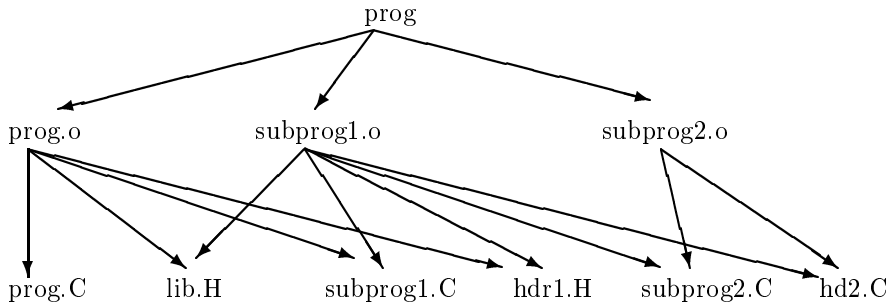
(e) Divide-and-conquer algorithms. (5)

5 (a) Show the adjacency matrix and adjacency list representations for the following graph: (3)



(b) Trace a depth-first search through the graph, showing how each vertex is marked with start and finish times. Assume this starts at vertex $v1$. (7)

(c) The `make` utility in UNIX constructs a precedence graph, showing how some files are dependent on others. A typical precedence graph would be:



If the date stamp on the file `prog` is older than any of the dates on `prog.o`, `subprog1.o` or `subprog2.o`, then the file `prog` must be rebuilt (via compilation or linking). The file `prog` is said to be dependent on `prog.o`, `subprog1.o` and `subprog2.o`.

Modify the algorithm for the depth first search of graphs to suggest how the `make` utility determines which files to rebuild, and does the rebuilding. (15)

[Hints: You may assume the existence of a function `outOfDate(file)`, which returns true if a file is older than any of its dependents, and hence needs rebuilding. You may also assume the existence of a function `rebuild(file)` to rebuild out-of-date files.

You may also assume a graph data type, whose vertices are identified by file names, and which provides a method `adjacent(v)` that returns the set of vertices adjacent to `v`.

Note that a file can only be rebuilt after any out of date files on which it depends have also been rebuilt. Your algorithm should thus process the precedence graph bottom up.]