University of Nottingham

SCHOOL OF COMPUTER SCIENCE AND INFORMATION TECHNOLOGY

A LEVEL B MODULE, AUTUMN SEMESTER 2000–2001

ALGORITHMS AND DATA STRUCTURES
(Course G5BADS)

Time allowed TWO hours

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

Candidates should attempt FOUR out of six questions. Question 1 is compulsory.

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 multiple choice question is compulsory. You don't have to copy questions into the book; just state the most appropriate answer for every question. Please read the questions carefully.

(a) An abstract data type is (4)

1. A description of a data type in terms of a domain of values and a set of operations which can be performed on them.
2. A description of a data type in terms of how it is built from the basic data types and a set of operations which can be performed on the data type.
3. An abstract description of how data is organised in computer memory, without concrete implementation details.
4. A set of Java methods, with parameters and return types fixed but without implementation.
5. The same as a data structure.

The question is continued overleaf

(b) An algorithm's memory usage is described by the following function: $S(N) = \frac{1}{2}N * c + 2log_2N * k$, where c and k are constants. What is the algorithm's space complexity: (3)

1. $O(N\ log_2N)$
2. $O(N^2)$
3. $O(log_2N)$
4. $O(1)$
5. $O(N)$

(c) Which one of the following is an invariant of the loop:

```
i=0;
while (i < k){
   array[i]=0;
   i++;
}
```

(3)

1. for all j, if $0 \leq$ j $\leq$ i then array[j] = 0
2. for all j, if $0 \leq$ j $<$ k then array[j] = 0
3. for all j, if $0 \leq$ j $<$ i then array[j] = 0
4. for all j, if $0 \leq$ j $\leq$ k then array[j] = 0
5. i = 0

(d) Which data structure has the fastest insertion procedure: (3)

1. binary search tree
2. ordered array
3. heap
4. unordered linked list
5. ordered linked list

(e) Which data structure has *the slowest* search procedure (search for an item given a key): (3)

1. binary search tree
2. ordered array
3. hash table
4. unordered linked list
5. height balanced binary search tree

The question is continued overleaf

(f) What is the advantage of height-balanced search trees as compared to ordinary binary search trees: (3)

1. Height-balanced trees take less space
2. Height-balanced trees are easier to implement
3. Insertion, deletion and search are $log_2 N$ times faster for height balanced trees
4. Insertion, deletion and search are faster on average
5. Only insertion is faster

(g) A perfect hashing function (avoiding collisions): (3)

1. does not exist
2. is only possible to construct if there are as many slots in the hash table as there are values in the key range
3. can be constructed if the set of records to be hashed is fixed in advance and is not going to change
4. is possible, but involves checking for collisions and therefore cannot be computed in constant time
5. only possible if incoming data is well-distributed across the key range

(h) Space complexity of breadth-first traversal of a graph using a queue is: (3)

1. $O(1)$
2. $O(|E|)$ where $|E|$ is the number of edges
3. $O(|V|)$ where $|V|$ is the number of vertices
4. $O(|V| + |E|)$
5. $O(2^{|V|})$

2 (a) Describe in English how merge sort works. (5)

(b) The following program is supposed to be an implementation of merge sort in Java. What is its space complexity? Use big O notation and justify your answer. (5)

```
public static int[] msort(int[] arr) {
    int len = arr.length;
    int[] sorted = new int[len];
    if (len <= 1){
        sorted = arr;
    return sorted;
    }
    int[] first = new int[len/2];
    int[] second = new int[len - len/2];
    for( int i = 0; i  < len/2; i++) {
        first[i] = arr[i];
    }
    for (int i = 0; i < len - (len/2); i++) {
        second[i] = arr[i + len/2];
    }
    int[] sorted1 = new int[len/2];
    int[] sorted2 = new int[len - (len/2)];
    sorted1 = msort(first);
    sorted2 = msort(second);
    merge(sorted1, sorted2, sorted);
    return sorted;
}
void merge(int sorted1[], int sorted2[], int sorted[]) {
    int len1 = sorted1.length;
    int len2 = sorted2.length;
    int i1, i2, j;
    while(i1 < len1 && i2 < len2) {
     if( sorted1[i1] < sorted2[i2] )
            sorted[j++] = sorted1[i1++];
         else
            sorted[j++] = sorted2[i2++];
    }
    while(i1 < len1) sorted[j++] = sorted1[i1++];
    while(i2 < len2) sorted[j++] = sorted2[i2++];
}
```

(c) Is the given implementation correct (Do not worry about the Java syntax, only the logic of the program)? If not give a counterexample and correct it. (5)

(d) Give an implementation which has a lower space complexity. (10)

3 Write a clear and well structured essay on how differences between main memory and peripheral storage affect the design of algorithms and data structures. (25)

4 (a) Write an algorithm (in Java or pseudocode) which given an expression (a string) containing parentheses returns true if the parentheses are balanced and properly nested, and false otherwise. Explain how your algorithm works and why it is correct. (10)

(b) What is the time complexity of your algorithm (use big O notation and justify your answer). (5)

(c) What is the space complexity of your algorithm (use big O notation and justify your answer). (5)

(d) How would you extend your algorithm to cope with more than one kind of brackets (not allowing improper nestings like "{(*a*})")? (5)

5 (a) Describe how topological sort algorithm works. You can use Java, pseudocode or English so long as your description is clear and unambiguous. Give an example of a task for which topological sort can be used. (15)

(b) Modify topological sort algorithm so that it can be used to detect cycles in directed graphs. Explain how your algorithm works and why it is correct. (10)

6 Imagine that you want to implement an automatic reminder program, which tells you every day the things you have to do, in order of importance. Describe the data structures and algorithms you would employ in the program, and the reasons for your choices. Where appropriate, specify ADTs, pseudocode for significant ADT methods employed, and an outline of the entire program. (25)