The University of Nottingham

SCHOOL OF COMPUTER SCIENCE AND INFORMATION TECHNOLOGY

A LEVEL B MODULE, AUTUMN SEMESTER 2003-2004

ALGORITHMS AND DATA STRUCTURES

(Course G5BADS)

Time allowed TWO Hours

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

Answer QUESTION ONE and THREE other questions

No calculators 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. No electronic devices capable of storing and retrieving text may be used.

DO NOT turn your examination paper over until instructed to do so.

1. This multiple choice question is compulsory. In each part, select one answer. For parts (b) to (i), you get 3 points if you select the right answer and 0 if you don't. For part (a) you get maximum of one point.

   (a) Binary search is only guaranteed to work if the collection to be searched is ordered: (1)

   i. yes
   ii. no

   (b) Which one of the following sorting algorithms has non-trivial (not $O(1)$) space complexity? (3)

   i. bubble sort
   ii. selection sort
   iii. insertion sort
   iv. merge sort
   v. quick sort

   (c) An algorithm's memory usage is described by the following function: $s(n) = 10n + log_2 n$. What is the algorithm's space complexity (choose the tightest upper bound): (3)

   i. $O(1)$
   ii. $O(log\ n)$
   iii. $O(n)$
   iv. $O(n\ log\ n)$
   v. $O(n^2)$

Question continued overleaf

(d) What is the tightest big O upper bound on the growth rate of running time for the following algorithm (assuming multiplication of integers is done in constant time): (3)

```
int power (int n, int k){
   int result = n;
   for (int i = 1; i < k; i++){
      result = result * n;
   }
}
```

i. $O(n)$
ii. $O(k)$
iii. $O(n^k)$
iv. $O(n * k)$
v. $O(1)$

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

```
int sum = 0;
for (int i = 0; i < array.length; i++){
   sum += array[i];
}
```

(3)

i. `sum = 0`
ii. `sum` = $\Sigma_{j<i}$`array[j]`
iii. `sum` = $\Sigma_{j\leq i}$`array[j]`
iv. `sum` = $\Sigma_{j\leq i+1}$`array[j]`
v. `sum` = $\Sigma_{j<n}$`array[j]` where `n = array.length`.

(f) Which data structure has reliably efficient (logarithmic) performance for search, insertion and deletion: (3)

i. ordered list
ii. ordered array
iii. binary search tree
iv. balanced binary search tree
v. unordered list

Question continued overleaf

(g) A sorting algorithm heapsort sorts a collection of items by creating a heap data structure, inserting items to be sorted into the heap, and then repeatedly removing the top of the heap and placing it into an ordered collection until the heap is empty. What is the big O upper bound on heapsort's *space usage*? (3)

i. $O(1)$
ii. $O(log\ n)$
iii. $O(n)$
iv. $O(n\ log\ n)$
v. $O(n^2)$

(h) Which of the following statements is true: (3)

i. Dijkstra's shortest path algorithm only works on directed acyclic graphs
ii. Dijkstra's algorithm shortest path algorithm does not work on graphs where all edges have the same weight
iii. If all edges have the same weight, Dijkstra's algorithm will visit them in the same order as breadth-first search
iv. If all edges have the same weight, Dijkstra's algorithm will visit them in the same order as depth-first search
v. If all edges have the same weight, Dijkstra's algorithm will return the same shortest path value for all pairs of nodes

(i) Suppose that running time of an algorithm has quadratic growth rate. On inputs of size 1000 it runs in 20 ms. What would be your estimate of its running time on inputs of size 10000? (3)

i. 40 ms
ii. 400 ms
iii. 4000 ms
iv. 200 ms
v. 2000 ms

2. (a) Give method signatures and pre- and postconditions for the methods of a Bounded Stack ADT. Bounded stacks are stacks which have a fixed maximal size and no items can be pushed onto the stack after it is filled to capacity. Assume that items to be pushed on the stack are of type Object. Give methods for pushing items on the stack, popping the stack, peeking at the top of the stack, checking whether the stack is empty or full. (10)

   (b) Write a Java implementation of this ADT using an array. You may also need to write classes for exception objects which are thrown when method preconditions do not hold. (15)

3. (a) Explain how quicksort algorithm works. Trace it on the following array: `6,3,2,5,4`, assuming the pivot is chosen randomly. (10)

   (b) Give an implementation of quicksort in Java, C, Pascal or Haskell. (10)

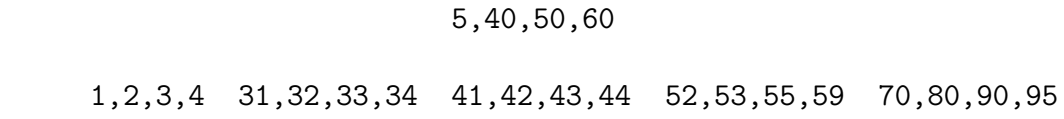
   (c) What is quicksort's worst case performance? When does it happen? (5)

4. (a) What is a B-tree? (6)

   (b) Where are B-trees used and why? (3)

   (c) Explain how search and insertion in B-trees works (in English or pseudocode, you may also use diagrams). (10)

   (d) The B-tree below holds maximum 4 items per node. Draw the result of inserting an item with key 45 in this tree: (6)



5. (a) Explain what are hash tables and hash functions. Explain what is meant by the terms 'separate chaining' and 'open addressing'. Briefly describe three strategies of open addressing. (11)

   (b) Suppose you need to hash Computer Science module codes; for simplicity assume that they all start with G5 followed by one of [1,2,3,A,B,C] followed by any three letters [A-Z], for example G5BADS or G51MCS. Design a perfect (no collisions) hash function for this task. How many possible values does it have and how large the corresponding data structure should be (you do not have to compute the exact number, just write a formula). (7)

   (c) Assume that Computer Science department never has more that 50 different modules on its books. Design a hash table which would have a load factor between 50% and 100%. How large will it be? Give a suitable hash function. (7)

6. (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? Your revised algorithm should not allow improper nestings like ''{(a})''. (5)