The University of Nottingham

SCHOOL OF COMPUTER SCIENCE AND INFORMATION TECHNOLOGY

A LEVEL B MODULE, AUTUMN 2004–2005

ALGORITHMS AND DATA STRUCTURES

Time allowed TWO hours

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

Answer QUESTION ONE and THREE other questions. *Marks available for sections of questions are shown in brackets in the right-hand margin*

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, including electronic dictionaries, may be used.

DO NOT turn examination paper over until instructed to do so

1 This multiple choice question is compulsory. In each part, select one answer.

(a) Which of the two statements is correct: [1]

(i) a stack is a 'last in, first out data structure'

(ii) a stack is a 'first in, first out data structure'

(b) What is the worst case time complexity of insertion in a balanced binary tree (n is the number of nodes in the tree): [3]

(i) $O(1)$

(ii) $O(log_2 n)$

(iii) $O(n)$

(iv) $O(n * log_2 n)$

(v) $O(n^2)$

(c) An algorithm's memory usage is described by the following function: $s(n) = 10n + 20n * log_2 n$. 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) $O(n^2)$

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

```
int divisions(int n){
   int result = 0;
   while (n > 1){
      n = n/10;
      result++;
   }
}
```

(i) $O(1)$

(ii) $O(log_2 n)$

(iii) $O(n)$

(iv) $O(n * log_2 n)$

(v) $O(n^2)$

(e) Which one of the following is an invariant of the loop: [3]

```
for (int i = 0; i < array.length; i++)
   array[i] = 100;
```

(i) for all indices j with $0 \leq j \leq$ `array.length` $- 1$, `array`[j] $= 100$;

(ii) for all indices j with $0 \leq j \leq i$, `array`[j] $= 100$;

(iii) for all indices j with $0 \leq j < i$, `array`[j] $= 100$;

(iv) for all indices j such that $j > i$, `array`[j] $\neq 100$;

(v) `array`[0] $= 100$

(f) Which of the algorithms below is using a divide-and-conquer strategy: [3]

(i) bubble sort

(ii) insertion sort

(iii) selection sort

(iv) heap sort

(v) quick sort

(g) Suppose your program is going to use a large fixed size collection of data which you need to be able to search very fast. Which data structure would you find most appropriate to store the data in this case: [3]

(i) unordered linked list

(ii) ordered linked list

(iii) (2,4)-tree

(iv) hash table

(v) unordered array

(h) Suppose your program is going to maintain a large dynamic collection of data. You do not know the maximal size of the collection and you also don't know whether the data is going to arrive in the order of keys or not. You want to be able to make insertion, deletion and search as fast as possible. Which data structure seems most suitable: [3]

(i) unordered linked list

(ii) ordered linked list

(iii) (2,4)-tree

(iv) hash table

(v) unordered array

(i) Suppose that the running time of an algorithm grows quadratically. On inputs of size 100 it runs in 20 ms. What would be your estimate of its running time on inputs of size 300? [3]

(i) 40 ms

(ii) 60 ms

(iii) 90 ms

(iv) 120 ms

(v) 180 ms

2 (a) Write the following algorithm (in Java or in pseudocode):

Algorithm `int[] intersect(int[] x, int[] y)`

Input: two ordered arrays of integers which do not contain duplicates (same number more than once).

Output: an ordered array which contains all integers which occur both in `x` and in `y`.

For full marks, the algorithm should have worst case time complexity $O(n + m)$ where n is the size of array `x` and m is the size of array `y`. [15]

(b) What is the time complexity of your algorithm? Justify your answer. [5]

(c) What is the space complexity of your algorithm? Justify your answer. [5]

3 (a) Write the following algorithm (in Java or in pseudocode): [10]

Algorithm `int[] evenodd(int[] x)`

Input: an array of integers

Output: an array which lists all even integers from `x` followed by all odd integers from `x`.

Recall that you can check whether an integer i is even by testing whether $i\%2 == 0$.

(b) Prove correctness of your algorithm using loop invariants. [15]

4 (a) What is a proper, or perfectly balanced, binary tree? [3]

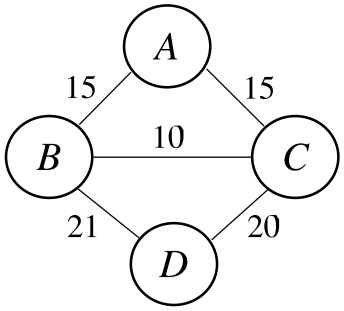
(b) Assume the root of a tree is at level 0, its children at level 1, and so on. How many nodes are at level k in a proper binary tree? Give an inductive proof of your answer. [5]

(c) How many nodes does a proper binary tree of height h contain? (Assume that height is the number of levels minus 1.) Give an inductive proof of your answer. [10]

(d) Give pseudocode for a search method for a binary search tree. Show that in a proper, or perfectly balanced, binary search tree, time complexity of search in the worst case is $O(log_2\ n)$, where n is the number of nodes in the tree. [7]

5 (a) What is a minimal spanning tree of a graph? Describe an application which requires computing a minimal spanning tree. [5]

(b) Give a minimal spanning tree of the following graph: [5]



(c) Describe an algorithm for computing a minimal spanning tree. [10]

(d) What is a greedy algorithm? Is the algorithm you described greedy? Justify your answer. [5]

6 (a) Using the following auxiliary `Node` class

```
class Node {
   Object value;
   Node next;
   Node(Object v, Node n) {
      this.value = v;
      this.next = n; }}
```

write a Java implementation of a single ended, single linked `List` class with the following methods: [15]

- `public List() // constructor`
  Postcondition: creates an empty list
- `public void insert(Object o)`
  Postcondition: inserts `o` at the head of the list
- `public void remove()`
  Postcondition: removes the head of the list. If the list is empty, does nothing.
- `public Object head()`
  Postcondition: returns the object stored at the head of the list, without updating the list; if the list is empty, returns `null`.

(b) Using only the `List` class methods defined above, give an implementation of the following Stack ADT: [10]

- `public Stack() // constructor`
  Postcondition: creates an empty stack
- `public void push(Object o)`
  Postcondition: pushes `o` on top of the stack
- `public Object pop()`
  Postcondition: pops the stack and returns the value on top. If the stack is empty, returns `null`.
- `public Object top()`
  Postcondition: returns the Object on top without updating the stack; if the stack is empty, returns `null`.