A. NORMAN

ENGINEERING TRIPOS PART IIA

April 2003 **CRIB**

Paper 3I1

DATA STRUCTURES ANDALGORITHMS

Answer all of Section A (which consists of a number of short questions), and two questions from Section B.

The approximate percentage of marks allocated to each part of a question is indicated in the right margin.

You may not start to read the questions printed on the subsequent pages of this question paper until instructed that you may do so by the Invigilator

SECTION A

Attempt all parts of this question. A total of 20 marks will be awarded for the entire section, and each part carries the same weight.

1. You should answer each of the following parts with a short-answer, quoting relevant results rather than proving them. Give names for algorithms you need to refer to and sketch methods or justifications of claims you make. Credit will be given for the clarity and succinctness of answers as well as for basic factual accuracy.

a) If f(n) = O(g(n)) then is it true that f(n) = Θ(g(n))? Similarly if g(n)=Θ(f(n)) is it true that g(n)=O(f(n)), f(n)=O(g(n)) or neither? [10%]

f(n) = O(g(n)) means f(n) < K g(n) for large enough n. so it COULD be that f(n)=0 always, which is in rather great generality NOT big-theta(g(n).

If g(n) = Θ(f(n)) then g(n)= O(f(n)) since that is a subset of the definition of big-theta, and also f(n)=O(g(n)): just look at K f(n) < g(n) < M f(n) and thw two bits (one based on 1/K) give the results instantly.

b) Consider the function push(n,m) = $2^n(2m+1)$. Explain how an implementation of the abstract datatype STACK could be build on the basis of this, in particular indicating the value of m to be used to denote an empty stack and explaining how the POP operation could be implemented. [10%]

Observe one can count how many times 2 divides into push(n,m) and that will give you n, while the odd number eventually reached must be (2m+1) so recovering m is then trivial. If we let m=0 stand for an empty stack that would be tidy and easy to check for.

c) If f(n) = 3f(n/3) + 7n, what big-theta or big-O expressions can be used to characterize f(n)? [10%]

try f(n) = k n log n

k n log n = 3 k (n/3) (log n – log 3) + 7n
k n log n = k n log n + 7n – k n log 3
observe than k = 7/log 3 makes this hold, so f(n) = big-theta(n log n) gives at least A solution.

d) Suppose that because of a programming error hash-function always returns the value 1. In terms of whatever parameters are relevant estimate the cost of inserting a new item into an already partly-filled hash table.. [10%]

The cost will depend mostly on the number of items stored in the table so far, call it n. Then inserting a new item will involve checking against all the items currently present before an empty slot is found. Ie the cost is n, regardless of the size of the table.

e) Describe how Larsen's method of dynamic hashing can be used to fetch a record from disc using just one disc block transfer, at the cost of holding a table in memory that stores an integer for each disc block. You should only describe how to retrieve data: no discussion of how to add new records is required here. [10%]

You will have a succession of hash functions $h_i(k)$ and $s_i(k)$. For each I until success compute first n = h(k) mod disc size. n is then the preferred disc block. Compare s(k) with table[n], the limiting signature of records stored on that disc block. If s(k) is smaller then read that disc block – success. Otherwise just increment I and go on to try the next h/s pair.

f) Explain one scenario where a buddy system for storage allocation would out perform first-fit, and another pattern of allocation and release where it would not. [10%]

What buddy systems do is to arrange that merging adjacent blocks can be done reasonably efficiently when memory is released. For use over a long time with multiple block sizes first-fit can suffer badly from fragmentation so to get a case where buddy will behave better I just need to produce one where fragmentation kills. To find the other I can select a case where the way that buddy systems round allocation sizes up to a power of 2 is a waste and where fragmentation does not occur. Eg maybe if all blocks used are size 129, or perhaps when there are no "free" operations.

g) Give an upper and a lower bound for the number of nodes in a 2-3-4 tree of height h? Hence if a 2-3-4 tree holds n values find an upper bound on its height. [10%]

a 2-3-4 tree has nodes that branch 2,3 or 4 ways! So the sparsest one branches 2 at each stage for 2^h nodes at height h and the broadest might branch 4 at each stage for 4^n. Some people may consider whether as you build a 2-3-4 tree you can ever end up with all nodes 4... But my wording avoids this being fussed about! Thus the height for n nodes can be at most log<2>(n).

h) Experience with the "zip" compression program suggests that the contents of very many computer files can be compressed to less than half their original size using Lempel Ziv. Is it reasonable to expect that by using Lempel Ziv twice they can be compressed further? [10%]

No it is not – LZ works if the input data has repeated sets of byte adjacencies and its output is not very likely to have that – eg because it has tried to remove redundancy. If you could ALWAYS compress you could compress every string down to a very short length, which becomes ridiculous since there will then be more strings than compressed forms for them.

i) What is a strongly-connected component of a graph G? [10%]

A sub-set of the graph's vertices such that from any one in it you can reach any other.

j) What is a "winding number" and how can you use one to test if a point is inside or outside a given polygon? [10%]

Take your point. Measure the angle from it to each vertex of the polygon (in order) and add up the (signed) differences you get as you move from one vertex to the next. The eventual sum will be a multiple of 2pi since you end up pointing the same way you started. The multiple is the "winding number". If zero you are outside the poly.

SECTION B

Attempt TWO of the following questions. Each question is worth an equal number of marks.

2. (a) One method of finding the median of a set of values starts by sorting them. Binary insertion sort is known to keep the number of comparisons that are performed low, even though may involve excessive data movement. Explain how this sorting method works by showing what sequence of comparisons it could involve when sorting exactly five items. Work through examples so that you can show both the best-case and the worst-case number of comparisons involved in finding the median of five distinct values this way. [20%]

Put 1st item in list. 0 comparisons.
To put 2nd one in need to do 1 comparison.
To put 3rd in with existing 2 compare with lowest. In GOOD case the new one is smaller and we do 1 comp. IN worst case it is bigger and we need 2.
To put 4th in cf mid item then cf one of others for 2 more
For final one do 2 comps in good case,3 in bad one to sort, but if we are just finding MEDIAN it is 1-3 comparisons.
TOTAL 5 to 8 comparisons.

(b) Another median finding algorithm involves selecting a pivot and partitioning the data. For the case of just five values (again all known to be distinct) show what the best and worst-case number of comparisons are. [20%]

To partition the data about our pivot takes 4 comparisons. In a happy case we find 2 lower, 2 higher so just report we have the median. IN bad case our pivot is say smallest value so we now have 4 to find 2nd from. Do 3 comps to partition. A bad case is now if pivot was again smallest. So that leaves wanting smallest of 3. Do 2 comps to partition and suppose this time pivot ends up biggest value – need yet 1 more. That is 10 in all so we have 4 to 10 comparisons in all.

(c) A standard guaranteed linear-cost medium finding algorithm organises its input data into groups of five and continues work with the median values from each of these sets of five. Establish a recurrence formula whose solution will confirm that the cost of this method is linear. [30%]

f(n) = K (n/5) **cost of finding medians of the groups of 5. K from above!**
+ f(n/5) **recurse to find medians of medians**

+ f(7n/10) worst-case split is 70-30

(d) Suppose now that instead of selecting a pivot by using the median of medians-of-five the algorithm is adjusted to form groups of seven. Find a recurrence formula that predicts the costs of the new method. Is its cost still linear? [15%]

f(n) = K (n/7) medians of the groups of 7
+ f(n/7) median of medians via recursion
+ f(5n/7)
I argue that I have n/14 medians lower than my final pivot, and each of those has 3 values smaller – so I have 4/14 = 2/7 values (at least) below me and also of course at least 2/7 above. So the biggest possible thing to have to recurse down onto is 5n/7. Because n/7+5n/7 <1 this still gives linear cost.

In a similar style one might try to simplify by forming groups of three rather than five. How do the expected costs grow with *n* in this case? [15%]

This time it is
f(n) = K(n/3)
+ f(n/3)
+ f(2n/3)
There are n/6 medians below my median of medians. Each carries 1 item with it so I have at least n/3 items below my pivot. But this means the worst recursion is onto a sub-list of size 2n/3. This recurrence ends up with nlog(n) cost.

3. The SET datatype can be represented by keeping the items that are members of a set in a linked list. In such a list there will be no repeated items, but the order in which values appear can be arbitrary, and in particular at this stage you should consider the case where you can not assume that the list has been sorted.

For sets with n elements, represented this way, what are the expected costs of

a) Checking if an item is in the set; [10%]

Need to scan all n items to see that it is NOT present. Expected cost is n/2 if it is present.

b) Adding a new item, given the knowledge that it is not already present; [10%]

If you KNOW it is not present you can add it at the front of the list at cost O(1).

c) Forming the intersection and union of two sets using direct algorithms based on the above two operations. [20%]

Both cases lead to east quadratic-cost methods (assuming both lists are the same length) since for each item in list A you need to scan list B to see if it is present.

Devise intersection and union algorithms that start by inserting all items from one set into a hash table and continue by looking up all members of the second set in the same table. Assuming that each hash table operation has cost O(1) what overall cost predictions can you make? Your methods may assume that at the start of the procedure the hash table concerned is empty, but must leave it empty at the end and your cost estimates must include allowing for this and for forming the result as a linked list. [30%]

Insert list A into hash table. Cost is n for list of length n. Take second list and look up each item in hash table and set flag there when item was already present. Cost is again m. Now can scan each list again checking the "was it present in the other" mark in the hash table and generating the output list, while at the same time removing items from hash table. Total cost is linear in sum of lengths of the two lists.

Alternatively suppose that the items in the list are objects that have a spare bit somewhere in them. And that as far as set operations are concerned items are only equal if they are actually represented as the same object. Devise intersection and union algorithms that start by scanning the first set and setting the spare bit in each object present in it, then scanning the second list checking which items have their mark bit set. You may again suppose that all the relevant bits are set to zero to start with but must have reset them to zero by the time

your algorithm terminates. Again give cost estimates. [30%]

Scan list A setting the bit. For intersection, say scan list B. If bit is set then copy item to output list, if not ignore. Re-scan A to clear all bits. Not very different for union. Cost is LINEAR and does not depend on the nice behaviour of hashing.

Compare the merits of these two schemes, identifying any circumstances where you might want or need to use one rather than the other. [10%]

Hash method is good if the items involved are not "objects with a spare bit". But hashing may be bad if you want a guarantee of performance.

4. Suppose that a statistical model for a stream of characters indicates that only the digits 0 to 9 will appear, that each digit is equally probable and that there is no need to allow for an end-of-data marker. Then the idea behind arithmetic coding suggests that a string such as "752904..." would be associated with the number z=0.752904... and would be transmitted as a sequence of bits that were the binary representation of z.

(a) What is the binary representation for the value (1/3) that has decimal representation 0.33333...? [10%]

0.1010101010101010... Consider summing the GP sum(4^(-i)).

(b) Supposing that the arithmetic coding algorithm from the notes and lectures were implemented using a 5-bit register (ie the integer arithmetic it uses is based on the values 0,1,...,31) find what the first few bits of output would be if you started to code the string "33333...". Explain the algorithm, your working and any choices you have to make along the way. . [70%]

Split the range 0-31 into 10 chunks each with the same number of cells. Well you in fact have to use a couple of 4s and mostly 3s – you can not get an EXACT reflection of the desired probabilities (and that fact is the essence of this whole question). Input a digit (in this case always 3!). That gives you a sub-range. Double its size outputting bits as you go until you have a range > 16. Then divide it up and input another digit

.

(c) Arithmetic coding using integer arithmetic should not generate exactly the bit-string you have as your answer to part (a). Explain why not. Discuss whether the bit-string that will be generated is liable to be periodic and whether changing the size of the integer register used makes any substantial changes to behaviour. [20%]

Mainly because the finite arithmetic does not let you partition the space of numbers EXACTLY as per the desired probabilities – the widths used get rounded. If you round consistently you should eventually repeat a situation and so your output bit-stream will recur. Using a bigger register (eg 32 bits) will let the FIRST few bits of output be 101010 as in perfection but rounding will still cut in eventually and the final repeat pattern may be longer. The better statistical modelling with a bigger register

should SLIGHTLY improve compression performance.

END OF PAPER