PART IIA 2005
3I1: Data structures and algorithms (CST)
Principal Assessor: Dr A Norman Trinity

Datasheet: None

ENGINEERING TRIPOS PART IIA **CRIB**

Monday 2 May 2005 2.30 – 4.00

Module 3I1

DATA STRUCTURES AND ALGORITHMS

Answer **all** of Section A and **two** questions from Section B.

All questions carry the same number of marks.

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

There are no attachments..

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

(TURN OVER

SECTION A

Attempt all parts of this question

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) What is a splay-tree and how do you look up data in one? [10%]

A splay tree is a binary search tree that is managed in a self-adjusting manner. Whenever an item is accessed in any way a sequence of rotations is performed to move that item to the root of the tree. Lookup happens in two phases. The first is a simple search as in any binary search tree. The second is the rotations to move the item found to the top.

(b) Name a data-structure suitable for implementing a priority queue, and give an indication of how much time it should take to remove the top item from it. [10%]

A **heap**. If there are n items in the heap it should take O(log(n)) to remove the top item.

(c) Give an example of a useful algorithm whose computing time cost can be characterised as $O(n^{20})$. [10%]

Eg simple insertion sort is big-O of n to the power 20! We all know it is $O(n^2)$, but the nature of big-O notation means that any higher bound will also apply!

(d) Explain briefly what is meant by "amortised" computing cost estimates and comment why it may sometimes be more realistic than simple worst-case analysis.. [10%]

Worst case time-per operation when measured and avareged over a long sequence of operations. It can be useful to allow for algorithms where most individual operations are fast but occasionally a slow reorganisation is needed (eg splay trees) and because it captures the idea that successive operations on a data structure can not be viewed as independent.

(e) Give a set of operations and identities that you would expect to form an Abstract Data Type for a priority queue. [10%]

(CONT.

make empty queue
add item to queue
test if queue is empty
remove top item, returning its value

is_empty(make_empty()) = true
is_empty(add(..)) = false
top item removes is smallest item previously added but not yet removed, and this statement is messy to formalise!

(f) Suggest circumstances (for instance an application, programming language being used, performance demands or other constraints) where you would use free-store allocation based on a Buddy system rather than using Garbage Collection. [10%]

When I could not guarantee to find all root-pointers to data I could not use a garbage collector, so perhaps a case where my implementation language does not have enough discipline to let me access all roots

(g) Indicate circumstances where Garbage Collection would probably be better than use of the Buddy system. [10%]

GC has perhaps 2 key benefits over Buddy and when one of these truly matters I will prefer it. (a) a copying GC can guarantee to avoid fragmentation by recompacting memory and (b) a GC means that the user does not have to worry about freeing memory blocks, so the system is robust against user errors that could arise in any explicit return system. Also in the long term and with enough working memory GC will be faster. I think I will say "whenever it is technically possible I prefer GC" here.

(h) Perfection in a binary tree would see all leaves at the same height. Red-Black trees do not achieve perfection, even though they do avoid extremes of failure to balance the tree. How out of balance can a Red-Black tree get? [10%]

The 2-3-4 tree idea that underpins red-black trees has all leaves at the same height. This a red-black tree has at worst a factor of 2 difference in leaf height.

(i) Describe an algorithm that can find the minimum spanning tree of an arbitrary graph or show that no such tree exists. You do not have to prove its correctness of estimate its costs. [10%]

Select arbitrary vertex of your graph and make a set U with that as only element. While

(TURN OVER

possible identify shortest edge of graph that has one end in U and the other not. Add the vertex that that defines to U, and the edge to what will become your MST. If at end you have linked all vertices you have an MST.
Knuth reported that the sequence generated by $a_i = a_{i-24} + a_{i-55} \pmod{2^{32}}$ can give a reasonable sequence of pseudo-random values provided that the first 55 values in the sequence (which are needed to start things off) are not all even. In what way might the sequence fail to behave randomly if all the first 55 values are even? [10%]

All subsequent values will be even. If in a more extreme way the seed values are all zero then everything stays zero from then on.

(CONT.

SECTION B

2. How would you sort data in each of the following circumstances? Justify or comment on your choices of method:

(a) The data is a set of 10 entries in the high-table list for a video-game console, to be sorted by score. [14%]

For just 10 values I think that simple insertion will be good enough. A key point to make here is that since this is only a TINY amount of data performance does not matter much but simplicity does. It could be that the order in the list changes only slowly and that too suggests simple insertion.

(b) You have 10 million people to sort based on their age in years. If two people have the same age it does not matter which order they appear in the output list. [14%]

The key issue here is that there are only around 100 distinct keys. I would so one pass counting the number of people in each bucket, then a second pass could move them to the right place. Noting and exploiting the small number of distint keys is the issue here.

(c) You have 10 million values but rather than wanting them totally sorted you want a sorted list of the 101 values that rank at each percentile in the data. The item at percentile k will have k% of all the input values less than it, and k will be 0, 1, …, 100. [14%]

This is median-finding! But there is some fun in that when I partition in a quicksort-like manner my early partitions can be used for most of the percentiles. So: start quicksort and thus split the range up until you have chunks each with just one key value in. At that stage switch to the normal medial-finding variant.

A nice explanation given by one candidates was roughly "use quicksort as per normal, but keep track of the rank-range of each sub-range you have to recurse down onto, and if the sub-range will not contain any of the 101 key values you want do not bother to consider it further".

(d) You have just 1000 items to sort, but it takes 20 seconds or so to compare any pair of them. [14%]

So that I strictly minimise the number of comparisons I will use binary insertion and accept any excess data movement. I might use merge-sort. But in any case using ONLY n log n comparisons is what matters

(e) Every second you receive a large file of numbers over the network, and you must display the top 100 in descending order. If very occasionally you can not complete preparing this list of the sorted top 100 on time it does not matter. [14%]

Probablistic median finder based on quicksort to fine the item at rank n-100 was the answer I had expected, but heapsort is in fact very good if you just want the top K of the data (and it is guaranteed so solves (f) too)

(f) As (e) but you need the top 1000 and you will lose your job if the required data from one set of numbers can not be displayed before the next set of data arrives. [15%]

Here I think we need to use the guaranteed linear scheme with medians-of-5. Or Heapsort.

(g) Your data consists of a large file of names. The first (about) 90% of it is already believed to be in sorted order from last time (but you are not 100% confident about that). The remaining 10% is new data that has been written onto the end of the original file in a chaotic unordered state. You want a single file consisting of old and new data all neatly sorted. [15%]

This seems best solved by splitting the data where it stops being well sorted. Then use bubble sort on the almost sorted bit (linear cost since nearly sorted already) and quicksort (probably) on the rest, followed by a merge to put them together. Simple insertion on the whole lot does not seem sensible.

(CONT.

3. Consider the message "amanaplanacanalpanamaΩ", where the "Ω" signifies the end of the message. Observe for instance that the letter "a" occurs 10 times.

(a) Based on the frequencies of letters used in the message, show how to construct a Huffman code that will compress it well. In your code how many bits will be used to encode the letters "a" and "Ω"? [25%]

There are 22 chars in total

AAAAAAAAAA	10	v1	1
MM	2	s1	1100
NNNN	4	t1	111
PP	2	s2	1101
LL	2	r2	101
C	1	q1	1000
Eof	1	q2	1001

C/Eof	2	q	r1
q/L	4	r	u1
M/P	4	s	t2
N/s	8	t	u2
r/t	12	u	v2
A/u	22	v	

So "a" encodes as 1 bit and omega as 4 bits. On paper I would draw this as a decode tree. A proper answer should explain how this is obtained by at each stage combining the two symbols with lowest frequencies. Reducing the frequencies to fractions or decimals is not needed and not helpful!

(b) What information does a decoder need in order to expand a message based on your Huffman code? You do not need to discuss how it will obtain this information. [25%]

It needs the code-lengths associated with each symbol. It can then deduce bit-codes for them! This perhaps calls for a note that you do not need the initial frequencies of symbols. And given the code-length data there may be many different detailed codings available, but the two ends of the transmission can both sort out the exact coding based on lengths and thus make compatible choices.

(c) Huffman coding maps each character of its input onto as few bits as it can. How many bits will it take to pack the first four characters of the sample message here, and what will be the exact sequence of bits that it uses? [25%]

(TURN OVER

1.1110.1.111. where the dots show where symbols end but are not part of what would be transmitted. The credit for this really belongs somewhat with the previous parts where you have derived what is needed to do it!

(d) How many bits would Lempel-Zif compression need to send the first four characters of this message, supposing that it knew in advance that it was having to deal with messages based on an alphabet of 6 characters plus an end of file marker? Explain why Huffman that reduces the number of bits per symbol sent and L-Z that increases it can both eventually lead to good compression. [25%]

LZ would start with 3-bit codes for the first char or 2 but would then need to switch to 4-bit ones, so it will use around 3+3+4+4 =14 bits for these 4 chars rather than the 8 used by Huffman. LZ eventually starts to pay back because one symbol that it sends may correspond to many input symbols, so the fact that it needs more bits per item that it transmits is not a problem – whatever it seems for the first few chars when its compression scheme has not built up enough info to start paying off. Note also that Huffman needs a priori frequency data which must be transmitted to the recipient somehow, LZ does not need any additional data so you should expect the first few chars of a LZ message not to compress or even to expand since it uses them to build up data that let later bits compress well.

(CONT.

A DVD can store around 4 Gbytes of information. You are charged with designing a way of storing a large dictionary on a DVD. There will be around 500,000 words included and each word has an associated description that may be several thousand characters long. You may suppose that data can be read from the DVD at around 5 Mbytes per second, but a "seek" to a new location on the DVD takes around a fifth of a second.

(a) Explain what benefit Larsen's Dynamic Hashing could have for organising the data on your disc. Describe how you would organise the DVD into blocks, how you would create the data in the form to be written to it and how the program to read the DVD would locate desired dictionary entries. Estimate the time it would take to look up a given word on your dictionary-disc. [35%]

With Larsen you only do exactly one disc transfer per fetch. With the parameters given I might view the DVD as having say 2 Mbyte blocks (reading a block then has most of its time in seek which is unavoidable). If I am using most of the disc there are then around 2000 such blocks.

I will describe lookup first. Read from a fixed place on the disc a 2000-entry table of signatures. Take the word to lookup and produce a a hash and signature values, H and S. Reduce H to 1-2000. Check the signature table at that location, and if S < sigtable[H] then just read block H and the data will be there. Otherwise compute a subsequent pair of (H,S) values and keep going until success. You may need many hash calculations but only one disc access.

To add data keep a hard disc image of what will become the DVD. To add new data lookup as before and if there is space in the block so identified just put the data there. Else eject the item in that block with extremal signature and adjust sigtable so that next time it will be detected as not there. Now try to re-insert the ejected data. Unless the disc is very nearly full you should succeed after a few tries.

(b) As an alternative to Larsen's method, discuss B-trees. Explain and estimate the same things you did in section (a). [35%]

B-trees are trees with huge but variable fan-out such that the height of the tree is uniform. In this case if I use 2 Mbyte blocks and so have only 2000 blocks I think I can make the top-level block have fan-out 2000 and hence my B-tree will only have 2 levels! I can then keep the top block in memory always and what had started by looking as if it was going to be a complicated scheme with greater cost than Larsen in this case seems simple and as fast!!!!!

(c) Although in the short term your dictionary will always be accessed directly off its DVD, you imagine that in the not too medium-term future your users will have enough disc space that they copy all the data from your DVD onto their hard disc to get faster access to it. At that stage you will offer a service of network-delivered corrections and updates. In the light of this and considering any effect that might arise as your dictionary grows so that it almost totally fills the DVD, discuss the relative merits of the two approaches. [30%]

For the parameters given I think that B-trees on the DVD seem good and that would make updating on hard disc much easier. But on hard disc I might want to move to a much smaller block size and hence eg B-trees of depth 2 not 1 – that will be easy. The scatterdness of Larsen will not be good for use on a hard disc shared with anything else.

END OF PAPER