PART IIA 2006
3I1: Data structures and algorithms (CST)
Principal Assessor: Dr A Norman Trinity

Datasheet: None

ENGINEERING TRIPOS PART IIA

Monday 1 May 2006 2.30 to 4

Module 3I1

DATA STRUCTURES AND ALGORITHMS: CRIB

*Answer **all** of Section A (which consists of short questions), 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.*

STATIONERY REQUIREMENTS	SPECIAL REQUIREMENTS
Single-sided script paper	none

Version: CRIB

SECTION A

Answer all parts of this question. The question in this section will be marked out of the same total as each question in section B, and each part carries the same weight.

1 (a) Explain precisely what it means to say $f(n) = O(g(n))$. [10%]

There exist constants N and K such that $n < N$ implies $f(n) < Kg(n)$.

(b) In big-O notation, derive and justify a worst-case cost for Quicksort. [10%]

The worst case is when the pivot always turns out to be an extremal element in what remains. The cost recurrence is then $f(n) = n + f(n-1)$ where the linear cost n is the partition step. That has solution $O(n^2)$.

(c) Explain briefly what is meant by analysis in terms of *amortised* computing time. [10%]

Average of the cost-per operation over any long sequence of operations that are performed sequentially on the data structure we are concerned with.

(d) Given the recurrence formula $f(n) = 2f(n/2) + n$ with some suitable value for $f(1)$, find or quote and justify a closed-form solution for $f(n)$. [10%]

The solution is $n\log_2(n)$ which everybody should recognise, since this recurrence arises over and over again. Proof by substituting the result into the recurrence and simplifying to confirm.

(e) A hash table stores all data within itself. The table has space to hold up to 1000 values, and 500 of these are present. On average how many probes of the table are performed when a new key different from the 500 that are present is looked up? [10%]

With probability 0.5 the first probe will be in an empty gap and the cost is this 1. With probability 0.25 the cost is 2, etc so the total expected cost is

$$(1/2)+2*(1/4)+3*(1/8)+4*(1/16)+\ldots$$

This adds up to 2, eg as

$$(1/2)(1+1/2+1/4+1/8+...)^2$$

(f) What aspect of storage management that gives trouble for both first-fit and best-fit is addressed by Buddy allocation methods? [10%]

Part of the answer here is going to be fragmentation, and another part the complexity and cost of returning space to the free pool.

(g) Give a circumstance where a Buddy system of storage management might be less efficient than using Garbage Collection, and explain why this is so. [10%]

To do this I think I will want to produce a case that is as favourable as possible for garbage collection. So I will (say) make all items the same (small) size, will discard them randomly so that Buddy busily spends time merging and re-splitting, and I will keep total heap occupancy fairly low, after an initial surge that gets the Buddy system to split all blocks.

Then I can make the cost of allocation in buddy and gc both constant, but the buddy system costs log(heapsize) on every delete while GC can have what amortises to linear cost on each delete, so for big heapsize it outperforms buddy.

(h) Give a sequence of operations on a *splay tree* that will result in it holding *n* elements and being of height *n*. [10%]

Start with an empty tree and add items in increasing order. Easy! An no rotations at all get performed.

(i) Explain how to construct a string that Huffman encoding will not manage to shrink, but Lempel-Zif might. [10%]

(TURN OVER for continuation of Question 1

Having each symbol in the alphabet occurring an equal number of times means that Huffman will have to give them all codes of the same length. For L-Z to work you need repeats. So for an alphabet ABCD how about the string ABCDABCDABCDABCDABCD ?

(j) Explain what a minimum-cost spanning tree of a graph is. Does every graph have a minimum-cost spanning sub-tree? [10%]

A sub-graph that is (a) a tree and (b) connects every vertex in the original graph. A graph that is not connected to start with can not have a spanning sub-tree, but every connected graph does have one.

SECTION B

*Attempt **two** questions from this section. Each question has the same weight in marks.*

2 (a) In the context of Heapsort, explain how to take initially unordered data and rearrange it so that the Heap Property applies. [34%]

This is standard bookwork, and the $O(n)$ method should be known to them!

(b) In the content of Quicksort you are given an array of length n containing arbitrary data in arbitrary order and a selected value p taken from that data (*the pivot*). Explain how to rearrange the data so that all values less than the pivot end up to its left, and all values greater than it to its right. [33%]

Two strategies are reasonable. One would be to keep <=P and the start and >=P at the end and have undecided things between. The other orders segments of the array as <=P, >P, undecided and extends the regions upwards. In each case real care is needed in the description of stopping conditions – well particularly in the first strategy, which works best using sentinels. Any good testbook on algorithms discusses this in real detail and quotes the (quite short) code involved.

(c) In the context of binary insertion sort, which is expected to complete a sorting process using very close to the smallest possible number of comparisons (although it may use excessive data movement) explain how to find where an item must be inserted into that part of the data that has already been sorted. [33%]

Probe in the middle of the region and then home in on either above or below that point. A little care is needed with the end-condition, eg searching in a region of size 1 or 2.

In each case a full answer will give some explanation of context, will describe the procedure involved carefully (paying attention to any limiting or awkward cases) and will give and justify a cost estimate in big-O notation. Methods that are unnecessarily costly will gain less credit than those that maximise performance.

3 (a) What is a *red-black* tree? What are the key properties of red-black trees and what problem do they solve? [30%]

---

red-black trees are binary search trees that are managed in such a way that thet remain tolerably balanced, thereby avoiding the key problem that simple binary search trees suffer from. They are search trees here each node can be tagged "red" or "black" but red can only appear as a direct descendent of black. By so doing they mimic 2-3-4 trees.

(b) The lectures explained the structure of red-black trees and talked through the procedure for adding a new item into such a tree. Discuss the problem of looking up a value that is stored in a red-black tree and removing it from the tree. Show cases where deletion is easy and note cases where it is harder, and get as far towards inventing, describing and justifying a complete deletion algorithm as you can. [70%]

A leaf node descending from a 3 or 4-node is easy to delete. A harder case will be deleting a leaf that descends from a 2-node because the parent needs to merge with its neighbour or borrow an excess item from it. Deleting a node in the middle may also be nasty.

I do not expect any but the most keen students (who will have read the books properly) to come up with a complete solution: this is scope for more or less free-form discussion of what has to be done! I think I rather firmly believe that the clearest explanations will be in terms of 2-3-4 trees rather than red-black.

But the big books will explain suitable techniques so those in future years can check there and see what is possible, and what is involved then clearly applies to deletion from b-trees too. I find a quite decent enough explantion in `http://en.wikipedia.org/wiki/Red-black_trees#Removal`. Thai is enough to show clearly that in an exam students will not be able to produce a complete deletion algorithm, but that there are plenty of special cases (eg of leaf items) that are easy to cope with.

A good answer needs to be aware at all times of (a) preserving the sorted oredr in the tree and (b) preserving the fact you have a 2-3-4 (or equivalently red-black) tree.

4 (a) For arithmetic coding, explain how the recipient of a compressed string of bits can decode it to retrieve the original message. [50%]

Pretty well directly in the notes, though the lectures concentrated on working through the page about compression, leaving decompression for students to look at for themselves. But of course if you understand one the other is very similar. See the pictures in the lecture notes!

(b) When encoding, special treatment is required when the input is such that the compressed output will be either of the form 01111111 or 10000000 with either a zero followed by a long run of ones or a one followed by a long run of zeros. Is there any special treatment that your decoder has to take in these cases? [10%]

This gives a chance to remember the hard bit of compression, involving a "count step" markes as a hash-mark in the notes. In decompression this problem does not arise.

(c) What is the largest number of bits the decoder must read before it can output a symbol? [20%]

If the compression works using an n-bit integer buffer then around n bits can be needed.

(d) Arithmetic encoding is performed using limited precision integer arithmetic. For instance the examples in the notes used 3-bit arithmetic leading to a partition of the range [0 ... 1] into 8 regions. What effects on compression and on costs arise as that working precision changes? [20%]

If you use a higher precision you can model the probabilities for symbols to higher accuracy, and so get (slightly!) better compression. But you may find that tracking the way that the active interval gets divided up per character may become slightly more costly.

END OF PAPER