PRIFYSGOL CYMRU; UNIVERSITY OF WALES

DEGREE EXAMINATIONS MAY/JUNE 2002

SWANSEA

Computer Science

CS 332 Designing Algorithms

Attempt 2 questions out of 3

Time allowed: 2 hours

Students are permitted to use the dictionaries provided by the University

Students are NOT permitted to use calculators

CS_332
DESIGNING ALGORITHMS
(*Attempt 2 questions out of 3*)

Question 1

(a) State the (Simplified) Master Theorem, and use it to give asymptotically tight bounds for each of the following recurrences.

(i) $T(n) = 4T(\lfloor n/3 \rfloor) + 2n$.

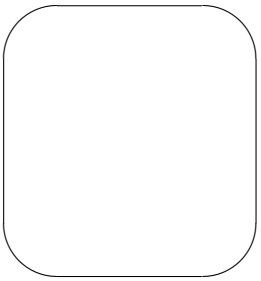
(ii) $T(n) = T(\lfloor n/3 \rfloor) + 2$.

(iii) $T(n) = 9T(n/3) + n^2$. **[8 marks]**

(b) You are in the dungeon of the castle of the evil CS_332 Professor, and you find a pot containing n coins ($n \geq 3$). You know from CS_332 legend that only one of these coins is an authentic (and priceless) gold coin, while the remaining $(n-1)$ coins are made from a cheap gold-coloured alloy. The only way to distinguish the authentic coin from the others is by its weight: CS_332 legend has it that—while the authentic coin appears to be identical to the cointerfeit coins—it actually has a slightly different weight than the counterfeit coins, all of which are themselves identical in weight. (The legend does not stipulate whether or not the real coin is heavier or lighter than the cointerfeit coins.)

You have come to steal the real coin from the evil CS_332 Professor, but you want to leave all of the other coins behind so that the evil CS_332 Professor does not realize that the real coin is gone. (Otherwise, in his fury he would fail all of his students, including yourself; and in their fury your fellow students would inform on you.)

To discover the real coin, you have brought with you a balance scale (like the one pictured) with which you can compare the weights of two sets of coins: you can place piles of coins on both arms of this scale and discover if one pile of coins is heavier than the other. You also have a clever idea involving divide-and-conquer which you learned from the CS_332 Professor. (He's not all bad after all!) You need to find the real coin and get out before he discovers you.



(i) Describe a recursive Divide-and-Conquer algorithm for finding the real coin. Justify that your algorithm is correct. (Hint: Start by dividing the coins into thirds.)
[6 marks]

(ii) Write down a recurrence relation for $T(n)$ which represents the maximum number of weighings your algorithm needs to make in order to find the real coin.
[3 marks]

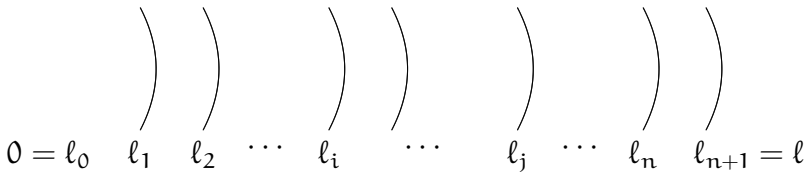
(iii) Give an asymptotically tight bound for $T(n)$.
[2 marks]

(c) Explain the information theoretic lower bound of $\Omega(\lg n)$ for the problem in part (b).

[6 marks]

Question 2

You bring an ℓ-foot log of wood to your local sawmill. You want it cut in n specific places: $\ell_1, \ell_2, \ldots, \ell_n$ feet from the left end. The sawmill charges £x to cut an x-foot log any place you like.



For example, suppose we wish to cut a log ℓ=12-feet long at lengths ℓ_1=2, ℓ_2=5, and ℓ_3=8 feet from the left end. The first cut (regardless of where it is made) will cost £12, but the cost for the two subsequent cuts will depend on the order in which the cuts are made (as the lengths of the subsequent sublogs to be cut will be different). For example, cutting in the order ℓ_1, ℓ_2, ℓ_3 would cost $12 + 10 + 7 =$ £29, while cutting in the order ℓ_2, ℓ_1, ℓ_3 would cost $12 + 5 + 7 =$ £24. In this example, it makes sense to start by cutting in the most central spot, to minimize the length of the two remaining pieces.

(a) Consider a greedy algorithm that cuts the log so that the maximum length of the resulting two pieces is always as small as possible; that is, it cuts it in the most central spot. Show that this algorithm does not necessarily achieve the minimal cost, by giving an example in which it fails to do so. (Hint: Consider making three cuts all close to the midpoint.)

[3 marks]

(b) Argue why Dynamic Programming is appropriate for this problem.

[4 marks]

Let $c[i, j]$ (for $0 \leq i < j \leq n+1$) be the optimal (i.e., least) cost of completely cutting the sublog whose left endpoint is at ℓ_i and whose right endpoint is at ℓ_j. We thus wish to compute $c[0, n+1]$.

(c) Give a recursive definition for $c[i, j]$. (Hint: Start by defining $c[i, i+1]$, and $c[i, i+2]$.)

[7 marks]

(d) Give pseudocode for a dynamic programming algorithm which computes $c[0, n+1]$.
(Note: You do *not* need to compute the optimal order itself, just the optimal cost.)

[7 marks]

(e) Analyze the run time and space requirement of this algorithm.

[4 marks]

Question 3

In the ***Majority Problem***, we are given an array $A[1..n]$ of n integers, and we must find the element which appears more than $n/2$ times in the array, or report that no such majority element exists. For example,

- the array $[3, 5, 5, 2, 5]$ contains the majority element 5;
- the array $[3, 5, 5, 2, 5, 4]$ does not contain a majority element.

(a) Describe a $O(n \lg n)$ time algorithm which starts by sorting the array with, e.g., mergesort. (You do ***not*** need to describe mergesort; and you needn't give your solution in pseudocode.)

[3 marks]

(b) Describe a linear-time algorithm which makes use of the linear-time algorithm we devised for selecting the kth smallest from an array of n elements. (Hint: Think *median*.)

[3 marks]

(c) The algorithm to the right determines, with some error probability p, whether or not an array $A[1..n]$ contains a majority element.

Explain when this algorithm gives a guaranteed correct result, and give a bound on the error probability p of this algorithm in the instances in which the result is not guaranteed to be correct. Explain your reasoning.

```
RANDOM-MAJORITY(A[1..n])
1  x ← A[random(1..n)]
2  k ← 0
3  for i ← 1 to n do
4        if A[i] = x then k ← k+1
5  if k ≥ ⌈n/2⌉ then return x
6  else return "no majority exists"
```

[3 marks]

(d) Here, we derive another linear-time algorithm which only relies on comparing elements for equality: we can test if two elements are equal, but not if one is less than another. Thus it can be used on arrays containing arbitrary incomparable elements, not just integers.

(i) Prove that if $A[i] \neq A[j]$ then removing $A[i]$ and $A[j]$ from the array preserves the majority element, if it exists. That is, if x is a majority element of the original array, then x must still be a majority element of the shorter array with the two unequal elements removed.

[5 marks]

(ii) Prove, using a suitable counter-example, that the converse of the above fact is false in general. That is, give an array which has no majority element but the removal of two unequal members produces a shorter array which does have a majority element.

[3 marks]

(iii) Using the above observations, design an algorithm for the majority problem which runs in linear time. Present your algorithm in pseudocode, and justify its correctness and running time.

[8 marks]