CS_M32 (2004-05)

ALGORITHM DESIGN AND ANALYSIS

(*Attempt 2 questions out of 3*)

Question 1

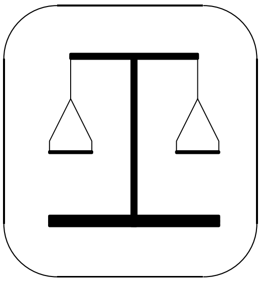
(a) State the (Simplified) Master Theorem, and use it to give asymptotically tight bounds for each of the following recurrences.

- (i) $T(n) = 4T(\lfloor n/3 \rfloor) + 2n$.
- (ii) $T(n) = T(\lfloor n/3 \rfloor) + 2$.
- (iii) $T(n) = 9T(\lceil n/3 \rceil) + 4n^2$. **[9 marks]**

(b) You are in the dungeon of the castle of the evil CS_M32 Lecturer, and you find a pot containing n coins ($n \geq 3$). You know from CS_M32 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_M32 legend has it that—while the authentic coin appears to be identical to the counterfeit 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 the real coin is heavier or lighter than the counterfeit coins.)

You have come to steal the real coin from the evil CS_M32 Lecturer, but you want to leave all of the other coins behind so that the evil CS_M32 Lecturer does not realize that the real coin is gone. (Otherwise, in her fury the Lecturer would fail all of the 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 the arms of this scale to see 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_M32 Lecturer. (The Lecturer's not all bad after all!) You need to find the real coin and get out before being discovered.



- (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 three piles.)* **[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). **[5 marks]**

Question 2

One day, instead of going to your CS_M32 lecture, you break into your Lecturer's house. There are n items which you can steal, but you cannot carry them all. Of course you want to maximize the value of the items that you do steal.

More specifically, the n items weigh $w_1, w_2, \ldots, w_n$ and have values $v_1, v_2, \ldots, v_n$. You can only carry a total weight of W. Hence you want to find a set $I \subseteq \{1, 2, \ldots, n\}$ which maximizes $M = \sum_{i \in I} v_i$ under the constraint that $\sum_{i \in I} w_i \leq W$.

For example, you might only be able to carry W=17kg, and there may be n=3 items to be stolen (as shown in the table). In this case, the optimal solution is $I = \{2, 3\}$, giving a maximal value of $M = 18$.

Item	value	weight
1. TV	v_1=12	w_1=10
2. Stereo	v_2=10	w_2=9
3. VCR	v_3=8	w_3=8

(a) Since you are missing the lecture on Dynamic Programming, you may think it best to use one of the following greedy strategies. Starting with $I = \emptyset$,

- (i) repeatedly try adding to I the items in order of decreasing value.
- (ii) repeatedly try adding to I the items in order of increasing weight.
- (iii) repeatedly try adding to I the items in order of decreasing $(\frac{v_i}{w_i})$.

Demonstrate that none of these strategies is guaranteed to give an optimal solution. *(**Hint**: Two of these strategies will fail on the above example.)*

[8 marks]

(b) Having realized that no greedy algorithm will work, you recall the Lecturer saying something about Dynamic Programming. Fortunately you have your *Algorithms* textbook with you (you don't go anywhere without it), and you quickly read the chapter on Dynamic Programming. "Aha!", you say to yourself.

You decide that you need to devise a Dynamic Programming algorithm based on a recursive equation for the maximum value $m[i, w]$ of a set of items taken from $\{1, 2, \ldots, i\}$ with total weight bounded by w (with $1 \leq i \leq n$ and $0 \leq w \leq W$).

- (i) Give such a recursive definition for $m[i, w]$, and explain it carefully.
  *(**Hint**: First, what is $m[0, w]$? Second, what is $m[i, w]$ if $w_i > w$?)*

[8 marks]

- (ii) Give pseudocode for the Dynamic Programming algorithm which computes the values of $m[i, w]$.

[5 marks]

(c) Of course, you only have so much time before the police arrive in response to the silent burglar alarm. Also, you don't have much paper to write on.

Analyze the running time and space requirement of your algorithm.

[4 marks]

(***Postscript***: In your haste, you left your textbook on the kitchen table, open at the chapter on Dynamic Programming, so you were arrested.

Moral: Don't leave your studying to the last minute!)

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 for solving the Majority Problem 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; a short explanation will suffice.)

[4 marks]

(b) Describe a linear-time algorithm which makes use of the linear-time algorithm we devised for selecting the k^{th} smallest from an array of n elements. (Again, a short explanation will suffice.) *(**Hint:** Think median, ie, the* $\lceil n/2 \rceil^{th}$ *smallest.)*

[4 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($A[1..n]$)
  $x \leftarrow A[\text{random}(1..n)]$
  $k \leftarrow 0$
  for $i \leftarrow 1$ to $n$ do
      if $A[i] = x$ then $k \leftarrow k+1$
  if $k > n/2$ then return $x$
  else return "no majority exists"
```

[4 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.

[4 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.

[4 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.

[5 marks]