CSE 101 - ALGORITHMS - SUMMER 2000
Homework 1 - Solutions

Problem 1: (Merge Sort)

(10 points) The following algorithm for merge sort was presented in the first lecture:

MERGE-SORT(A, i, k)

1 **if** $i \geq k$ **then return**
2 $j \leftarrow [\frac{i+k}{2}]$
3 MERGE-SORT(A, i, j)
4 MERGE-SORT$(A, j+1, k)$
5 MERGE(A, i, j, k)

where MERGE(A, i, j, k) assumes that the subarrays $A[i, ..., j]$ and $A[j+1, ..., k]$ are sorted and returns all the $k-i+1$ elements in a single sorted subarray that replaces the current subarray $A[i, ..., k]$. Write and analyze pseudocode for MERGE(A, i, j, k).

Proof: There may be many pseudocode algorithms for the same problem, but we expect all of them to use auxiliary memory. Notice that our RAM computation model allows us to use as much memory as we need. Then the following is a possible pseudocode:

MERGE(A, i, j, k)

1 $i' \leftarrow i;\ j' \leftarrow j+1;\ counter \leftarrow 0$
2 **while** $i' \leq j$ or $j' \leq k$ **do**
3 **if** $(i' > j)$ or $(j' \leq k$ and $A[i'] \geq A[j'])$ **then**
4 $counter \leftarrow counter + 1;\ A'[counter] = A[j'];\ j' \leftarrow j' + 1$
5 **if** $(j' > k)$ or $(i' \leq j$ and $A[i'] \leq A[j'])$ **then**
6 $counter \leftarrow counter + 1;\ A'[counter] = A[i'];\ i' \leftarrow i' + 1$
7 **for** $i' \leftarrow 1$ **to** $counter$ **do**
8 $A[i + i' - 1] \leftarrow A'[i]$

where $A'[1, ..., k-i+1]$ is an auxiliarry array.

Therefore, each iteration of the while loop increments $counter$ and either i' or j' by one. Since there are $k-i+1$ elements in total to be merged and since the while loop is executed until both $i' \geq j$ and $j' \geq k$, we deduce that the while loop is executed exactly $k-i+1$ times and that this is the value of $counter$. If we let n denote the total number of elements in the input, that is, $n = k-i+1$, then the running time of MERGE is $\Theta(n)$.

Problem 2: (Function Order - Skiena, Exercise 1-7)

(10 points) Give a proof or counter-example to the following claim: for all functions f and g, either $f \in O(g(n))$ or $g \in O(f(n))$.

Proof: We give a counter-example: let $f(n) = 1 + (-1)^n$ and $g(n) = 1 - (-1)^n$ for all $n \geq 0$, so f takes the values $2, 0, 2, 0, 2, 0, ...$ and g takes the values $0, 2, 0, 2, 0, 2, ...$ For the sake of contradiction, suppose that $f \in O(g(n))$. Then there are $c > 0$ and $n_0 > 0$ such that $f(n) \leq cg(n)$ for all $n \geq n_0$. But that means that $f(2k) \leq cg(2k)$ for all $k > n_0/2$, that is, that $2 \leq 0$. Contradiction. It can be similarly shown that $g \notin O(f(n))$.

Problem 3: (Recurrence)

(10 points) This problem has three parts:

1. (2 points) Solve the recurrence $T(n) = T(n/2) + n$.

2. (5 points) You are a young scientist who just got a new job in a large team of 100 people (you the 101-st). A friend of yours who you believe told you that you have more honest colleagues than liars, and that that's all what he can tell you, where a *liar* is a person who can either lie or tell the truth, while an *honest* person is one who always tells the truth. Of course, you'd like to know exactly your honest colleagues and the liars, so that you decide to start an investigation, consisting of a series of questions you are going to ask your colleagues. Since you don't wish to look suspicious, you decide to ask only questions of the form "Is Mary an honest person?" and of course, to ask as few questions as possible. Can you sort out all your honest colleagues? What's the minimum number of questions you'd ask in the worst case? You can assume that your colleagues know each other well enough to say if another person is a liar or not. (Hint: Group people in pairs (X,Y) and ask X the question "Is Y honest?" and Y the question "Is X honest?". Analyze all the four possible answers. Once you find an honest person, you can easily find all the others. Challenge: can you solve this enigma asking less than 280 questions in total?)

3. (3 points) Generalize the strategy above and show that given n people such that less then half are liars, you can sort them out in honest persons and liars by asking $O(n)$ questions.

Proof: The role of 1 is to suggest you that you should reduce the problem to a related subproblem of size half the size of the original problem. Additionally, the recurrence in 1 is closely related to the recurence obtained in 3.

1. Since $T(n) \geq n$, we immeditely get that $T(n) \in \Omega(n)$. On the other hand, since

$$1 + \frac{1}{2} + \frac{1}{2^2} + \cdots \frac{1}{2^k} \leq 2,$$

unwinding the recurence we obtain:

$$\begin{aligned}
T(n) &= T(\frac{n}{2}) + n \\
&= T(\frac{n}{2^2}) + n(1 + \frac{1}{2}) \\
&\vdots \\
&= T(\frac{n}{2^k}) + n(1 + \frac{1}{2} + \frac{1}{2^2} + \cdots + \frac{1}{2^{k-1}}) \\
&\vdots \\
&\leq T(1) + n(1 + \frac{1}{2} + \frac{1}{2^2} + \cdots + \frac{1}{2^{\log n - 1}}) \\
&\leq T(1) + 2n \\
&\in O(n).
\end{aligned}$$

Therefore, $T(n) \in \Theta(n)$.

2. Our first goal is to find a honest person. Group your colleagues in pairs (X, Y) and ask X the question "Is Y honest?" and Y the question "Is X honest?". A first important observation is that at least one of X, Y is a liar whenever you get at least a NO answer; then you can remove both X and Y from the set of potential honests, because the remaining $n - 2$ people still verufy the property that more than half are honest. Therefore, we'll keep only those pairs whose answers are YES, noticing that either both people in each group are honest or both are liars.

   Let $T(n)$ be the number of questions (the running time :) you need to ask a group of n people with more honests than liars, in order to find a honest person. We are looking for a recurence for $T(n)$. There can be distinguished three cases, depending on the number of pairs answering (YES,YES) and the parity of people:

   (a) Even number of people grouped in k pairs. Then we can remove one person from each pair, the remaining k people still having the property that more than half are honest. Thus, we obtain the recurrence

$$T(2k) \leq T(k) + 2k,$$

   because we asked $2k$ questions to reduce the problem by half. The worst case is of course when $n = 2k$;

(b) Odd number of people, grouped in even number of pairs $2k$ and one person singled out. Then we can remove one person from each pair, obtaining the recurrence

$$T(4k+1) \leq T(2k+1) + 4k,$$

since we asked questions only the $4k$ people grouped in pairs;

(c) Odd number of people grouped in odd number of pairs $2k+1$ and one person singled out. Then we can safely remove one person from each group together with the singled out one, obtaining the recurrence

$$T(4k+3) \leq T(2k+1) + 4k + 2.$$

Applying the divide and conquer technique described above in which about half the people are removed each step, we can calculate the number of questions needed in the worst case to detect an honest out of 100:

$$\begin{array}{rcrcr} T(100) & \leq & T(50) & + & 100 \\ T(50) & \leq & T(25) & + & 50 \\ T(25) & \leq & T(13) & + & 24 \\ T(13) & \leq & T(7) & + & 12 \\ T(7) & \leq & T(3) & + & 6 \\ T(3) & \leq & T(1) & + & 2 \\ T(1) & = & & & 0 \\ \hline T(100) & \leq & & & 194 \end{array}$$

Once an honest person is found, say X, then we might ask X 99 questions and thus completely correctly divide the 100 people in honests and liars. Therefore, we apparantly need 293 questions in total.

However, it can be done better! We do not need to ask X 99 questions, but to reuse the information we have from previous questions we asked X. For example, during the division of the problem into subproblems, we asked X 6 questions about other 6 people and we got answers YES, so we already knwo that those 6 people are honest. Moreover, each of the known 6 people were asked questions about other people who were previously removed, so those people are also honest. This procedure can be iterated and reduce the number of questions for X by 63! Thus, the total number of questions needed to find all the honest people is 230. I do not see how to do it better.

3. We can upper bound the recurrence above by another more compact recurrence $T(n) \leq T(3n/4) + n$, whose solution is in $O(n)$.

Problem 4: (Sorting)

(10 points) Given a fixed $k > 0$, show how to sort any n integers in the range 1 to n^d in $O(n)$ time. (Hint: do it for $d = 2$ first; you get 8 points if you solve it for $d = 2$.)

Proof: We present the general solution. The idea is to represent the numbers in base n and then use radix sort.

Let's substract 1 from each number (it can be done in linear time), so that the numbers are in the range 1 to $n^d - 1$. Each such number m can be uniquely decomposed as $m = m_1 + m_2 \cdot n + m_3 \cdot n^2 + \cdots m_d \cdot n^{d-1}$, where m_1 is the rest of the division of m by n, m_2 is the rest of the division of $m - (m - m_1)/n$ by n, etc. All the numbers $m_1, m_2, ..., m_d$ are in the range 0 to $n-1$; we can add 1 to each in linear time. Thus each of the n numbers in the range 1 to n^d can be uniquely represented as a tuple of d numbers in the range 1 to n, $(m_d, m_{d-1}, ..., m_2, m_1)$.

But this is exactly the hypothesis of radix sort! Therefore, we can simply use radix sort for d "digits", with counting sort applied d times to sort the numbers by each digit. Since the digits range between 1 and n, counting sort runs in $\Theta(n+n) = \Theta(n)$ time, so the whole alogithms runs in $\Theta(d \cdot n)$, that is, $\Theta(n)$.