BU CLA CS 530: Analysis of Algorithms
Solutions for End-of-Term Test (12 May 1999)

Problem 1

First, sort $A[1\,..,n]$ in $\mathcal{O}(n\log n)$ time. Second, if the entries in the sorted array are $x_1, x_2, \ldots, x_n$, we pair x_1 with x_n, x_2 with x_{n-1}, x_3 with x_{n-2}, and so on. The total time for the two phases is $\mathcal{O}(n\log n) + \mathcal{O}(n) = \mathcal{O}(n\log n)$. (5 points)

We have to prove this procedure is correct (5 points). Suppose otherwise and we will get a contradiction. Suppose there is an optimal partition, i.e. a partition that minimizes the maximum sum, which does *not* match the smallest integer x_1 with the largest x_n, but instead matches x_1 with x_i and x_n with x_j, for some $i, j \in \{2, \ldots, n-1\}$. If $x_i \leq x_j$, then

$$x_i = x_1 + k, \qquad x_j = x_i + k + \ell, \qquad x_n = x_i + k + \ell + m$$

for some $k, \ell, m \geq 1$, so that $x_1 + x_i = 2x_i + k$ and $x_j + x_n = 2x_i + 2k + 2\ell + m$. But in this case we can get a better bound by matching x_1 with x_j and x_n with x_i: $x_1 + x_j = 2x_i + k + \ell$ and $x_i + x_n = 2x_i + 2k + \ell + m$ — which contradicts the optimality of the partition. On the other hand, if $x_j \leq x_i$, then

$$x_j = x_1 + k, \qquad x_i = x_i + k + \ell, \qquad x_n = x_i + k + \ell + m$$

for some $k, \ell, m \geq 1$, and $x_1 + x_i = 2x_i + k + \ell$ and $x_j + x_n = 2x_i + 2k + \ell + m$. But we can get a better bound by matching x_1 with x_n and x_i with x_j: $x_1 + x_n = 2x_1 + k + \ell + m$ and $x_i + x_j = 2x_1 + 2k + \ell$ — another contradiction.

Problem 2

(a) (5 points) We assign one processor to each item in the list, for a total of n processors. The behavior of each processor is determined by whether or not its item is X. If a processor's item is X, then this processor checks whether it has the highest index among the items in the list which are equal to X, by examining the next item in the list; the processor with the highest index whose item is X (there can be at most one such processor) outputs its index. If a processor's item is not X, then this processor attempts to detect whether X does not appear in the list at all, by checking if its own item is smaller than X and the next one is larger than X; if it determines that this is indeed the case (there can be at most one such processor), it outputs *no*.

(b) (5 points) This is a parallel constant-time algorithm that uses n processors. The running time is therefore $\mathcal{O}(1)$. The *work* performed by this algorithm is $\mathcal{O}(n)$, which is the product of its running time and the number of processors it uses. This algorithm is therefore *not* work-efficient, because there are sequential algorithms which solve the problem in time $\mathcal{O}(\log n)$.

(There are other parallel algorithms for this problem, which are more efficient, i.e. perform less *work*, but are also more complicated to design. If you are interested, an appropriate reference is Chapter 5 of the book [A].)

Problem 3

The first reduction is trivial (worth 1 point): **clique** $\leq_P$ **maximum-clique**. Details omitted. The third reduction is also easy (3 points): **maximum-clique-size** $\leq_P$ **clique**, if we are allowed to call an algorithm CLIQUE for **clique** a polynomial number of times (instead of just one call, as in [CLR]). Details omitted.

The only non-trivial reduction (6 points) is the second one:

maximum-clique $\leq_P$ **maximum-clique-size**.

For this reduction we write an algorithm MAXIMUM-CLIQUE that calls an algorithm MAXIMUM-CLIQUE-SIZE a polynomial number of times (actually, at most $\mathcal{O}(n)$ times, where n is the number of vertices in the input graph G). First, we introduce some convenient notation. If $G = (V, E)$ is an undirected graph and $v \in V$, let $G_v = (V', E')$ be the subgraph of G consisting of v and all its adjacent vertices — formally: $V' = \{v\} \cup \{ w \mid (v, w) \in E \}$ and $E' = E \cap (V' \times V')$.

MAXIMUM-CLIQUE(G)
$k \leftarrow$ MAXIMUM-CLIQUE-SIZE(G)
find first $v \in V$ such that MAXIMUM-CLIQUE-SIZE(G_v) $= k$
return vertices of G_v

Assuming we have an "oracle" for **maximum-clique-size**, i.e., every call to MAXIMUM-CLIQUE-SIZE is assumed to be free (no cost), the total cost of running MAXIMUM-CLIQUE must include the cost of constructing G_v for every $v \in V$ from the given G. But the cost of this construction is "cheap" (a low-degree polynomial).

Problem 4

In any tiling with dominoes, we can place the dominoes so that their black and white squares coincide with the black and white squares of the chessboard. Therefore, any successful tiling by dominoes will have strictly equal numbers of black and white squares. The solution of this problem is based on 4 easy facts, proved together by induction on $n \geqslant 1$ (induction omitted):

1. If n is even then C_n without missing squares can *always* be tiled with dominoes.

2. If n is odd then C_n without missing squares can *never* be tiled with dominoes.

3. If n is even then C_n has one diagonal all black and one diagonal all white. Moreover, C_n has the same number of black squares and white squares.

4. If n is odd then C_n has both of its diagonals with the same color, i.e., either both are black or both are white. Moreover, if both diagonals of C_n are black (resp. white), then the black (resp. white) squares exceed the white (resp. black) squares by exactly one.

<u>Case 1</u>: *Yes* if n is even, *No* if n is odd.
<u>Case 2</u>: *No* if n is even, *Yes* if n is odd.
<u>Case 3</u>: *No* if n is even, *No* if n is odd.

Correctness of the algorithm for each case is based on facts 1, 2, 3 and 4 above (5 points). If n is given as part of the input, in each case the algorithm runs in constant time (it is just a matter of testing whether n is even or odd). If n is not given as part of the input, in each case the algorithm runs in time linear in n (the algorithm must measure the size n of C_n first, before it can find out whether n is even or od). (5 points for the run time)

Problem 5

The argument below applies to the whole cartesian plane, i.e., there is no reason to restrict R to be a finite rectangle. Put differently, imagine that R extends in all directions to cover the whole plane. Moreover, the argument below applies to an arbitrary set L of n straight lines, i.e., there is no reason to restrict each of the n lines to be parallel to one of the two axes.

Part 1 of the requested output is always *Yes*, i.e., it is always possible to color with black and white the regions defined by the n line (5 points). The proof is by induction on $n \geqslant 1$. This induction can also be used to define a low-polynomial time (in fact, linear in n) algorithm to construct a valid coloring, for part 2 of the requested output (5 points). I omit the details of this algorithm, which requires a way (i.e., a data structure) to specify the n lines and the colors of the regions; one efficient approach is, first, to place all vertical lines (sequentially from left to right) and, second, to place all horizontal lines (sequentially from bottom to top).

For the basis of the induction, $n = 1$, it is clear that there is always a valid coloring: Just color one side black and one side white. Suppose there is always a valid coloring with $n \geqslant 1$ lines (this is the induction hypothesis), and we want to prove there is a valid coloring with $n + 1$ lines, say $L = \{z_1, z_2, \ldots, z_{n+1}\}$. Let $L' = \{z_1, z_2, \ldots, z_n\}$, so that $L = L' \cup \{z_{n+1}\}$. By the induction hypothesis, there is a valid coloring with all the lines of L' placed on the plane. Now place the additional line z_{n+1}, reversing all the colors (from black to white and from white to black) on one side of z_{n+1}, and preserving all the colors on the other side of z_{n+1}.

We claim that the resulting coloring is valid. Consider two regions that have a line in common. If that line is not z_{n+1}, then by the induction hypothesis, the two regions have different colors (either the same as before placing z_{n+1} or reversed after placing z_{n+1}). If that line is z_{n+1}, then the two regions formed a single region before z_{n+1} was placed. Because we reversed colors on only one side of z_{n+1}, the two regions now have different colors.

Assaf Kfoury
Created: 99.05.12