Introduction to algorithms (IT301)  
(July-November, 2010):  
Solutions to Assignment 3

1. (a) The strategy adopted by the bad chips is to always make a false declaration. Consequently, if two good chips are tested or two bad chips are tested we will get the outcome “Good, Good”. If we test a good chip and a bad chip with each other, we get the outcome “bad, bad”. In each case we can conclude nothing. In the first two cases all we know is that both chips are of the same type. In the last case, we know that at least one is bad. However, all future tests are going to give no further information to distinguish the good chips from the bad ones.

(b) In any comparison where both declare good, we know that either both are good or both are bad. For every such outcome we throw away exactly one such chip from the set. If we get any other outcome, we throw away both chips, since we know that at least one must be bad. Thus in any event we throw away either one or both chips and the reduced set has size at most half. And, we know that the fraction of good chips is at least as high as the original, since in any step we either throw away two bad chips or throw away one chip of the same type as one we retain. Eventually, when we reach a set of size two, by the fact that more than half of them are guaranteed to be good, we know that both of them must be good. We can use either of them to test all the other chips.

(c) The recurrence is $T(n) \leq T\left(\frac{n}{2}\right) + \frac{n}{2}$. This is to find a single good chip. A further $n - 1$ tests can be used to determine all the good chips. Thus the total time is $\theta(n)$.

2. (a) If $A[i, j] + A[i + 1, j + 1] > A[i, j + 1] + A[i + 1, j]$, then we can take $k = i + 1$ and $l = j + 1$ in the definition and see that the
given array is not monge.

Now, suppose $A[i, j] + A[i + 1, j + 1] \leq A[i, j + 1] + A[i + 1, j]$ for $1 \leq i < m$ and $1 \leq j < n$. Thus, we have $A[i, j + 1] + A[i + 1, j + 2] \leq A[i, j + 2] + A[i + 1, j + 1]$. Adding these two inequalities and cancelling common terms on both sides, we get, $A[i, j] + A[i + 1, j + 2] \leq A[i + 1, j] + A[i, j + 2]$. For an arbitrary difference in rows and columns, we can simply chain together the inequalities corresponding to submatrices of size 4 $(2 \times 2)$ first in the horizontal direction and then in the vertical direction to get the desired inequality. A formal proof by induction, mimicking exactly this procedure seals the result.

(b) Since it is given that we need to render the array monge by changing only one value, we know there is a violation which is restricted to one subarray. Moreover, from part (a), we know that the error is confined to a subarray of size 4 $(2 \times 2)$. Scanning systematically, we observe that the subarray of size $(2 \times 2)$ with left-top index $[1, 2]$ is non-monge. We need to reduce entries on the forward diagonal and/or increment entries on the backward diagonal, so as to satisfy the monge property for this subarray while not creating violations elsewhere. This is achieved by computing the maximum freedom available for each of the four elements with respect to other subarrays to which they belong and using that degree of freedom to satisfy the monge property for this subarray. The entry 23 can be reduced by at most 1, and still preserve other subarrays involving it as monge. We see that this change is insufficient to change the subarray into monge. Hence we turn to the other three elements. The element 7 can be reduced by up to 5. This is sufficient. Thus we reduce its value to 5 (a reduction by 2 is sufficient) and convert the input array into a monge one.

(c) Consider rows $i$ and $j$, with $i < j$. Let the indices of the leftmost minimum elements (since there might be repeat occurrences of the minimum element in a row) on these rows be, respectively, $f(i)$ and $f(j)$. Since they are minimum elements in their rows, we know that, $A[i, f(i)] \leq A[i, f(j)]$ and $A[j, f(j)] \leq A[j, f(i)]$. Adding these inequalities, we get, $A[i, f(i)] + A[j, f(j)] \leq A[i, f(j)] + A[j, f(i)]$. Since the array is monge, if $f(i) \neq f(j)$, we know that $f(i) < f(j)$. Thus we have proved the result $f(1) \leq \cdots \leq f(m)$.

(d) We know that for a given odd-numbered row the leftmost minimum element lies in between the leftmost minimum elements of
the preceding and succeeding even-numbered rows. Thus, we can scan only a subarray for each row, and the total dimensions of all these subarrays over all the rows is $O(n)$ and the number of entries to be computed is $O(m)$. Hence, these computations can be preformed by linear scans for each row at a total cost of $O(m + n)$.

(e) The recurrence is $T(m, n) = T(m/2, n) + O(m+n) = T(m/2, n) + c_1m + c_2n$. Thus, the solution is $T(m, n) = c_1(m + m/2 + m/4 + \cdots) + c_2n \log m = O(m + n \log m)$. Notice that the parameter $m$ decreases with each iteration, whereas the $n$ is invariant. That explains the asymmetry of the running time with respect to the two parameters.

3. We build a heap with the minimum elements from each of the $k$ lists. The overall minimum element is the minimum of these minimum elements. We also store in each node of the heap, the list number from which the element is drawn. We repeatedly perform an extract minimum and then replace the element with the next minimum element from the same list, and then update the minimum pointer in that sorted sublist. If a sublist becomes empty, we simply reduce the heap-size by 1. The time to build the initial heap is $O(k)$. The total number of calls to extract minimum is $O(n)$. Each such operation takes time $O(\log k)$. Thus the whole algorithm takes time $O(n \log k)$. 