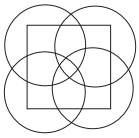
Problem 1:

The algorithm remains mostly the same. We divide the points into two sets of size $\lceil n/2 \rceil$ and $\lfloor n/2 \rfloor$, P_L and P_R, with P_L on or to the left of a vertical dividing line and P_R on or to the right of the vertical dividing line. We recursively find the smallest circle containing at least three points in each of P_L and in P_R. Let δ be the smaller of the diameters of these two circles. A circle of diameter $< \delta$ which contains points of both P_L and P_R must be completely contained in a vertical strip of width 2δ around the vertical dividing line, and within this strip it must be contained in a rectangle of height δ. Such a rectangle consists of two squares, one on the left of the dividing line and one on the right. To prove that the algorithm runs in $O(nlgn)$ time, we need only to prove that such a rectangle can contain at most a constant number of points of P_L or P_R.

This seems intuitively true. If we try to put a whole bunch of points into a $\delta \times \delta$ square, say 15, we notice that we are forced to have some three points contained in a circle of diameter less than δ. On the other hand, it is perfectly possible for two points to be very close together, so we can't just use the argument from the closest pair algorithm that we gave in class.



There are many correct ways to prove that a $\delta \times \delta$ square cannot contain more than a constant number of points if no three points are allowed to be in a circle of diameter $< \delta$. An easy one is illustrated in the picture above. Every point in the square is contained in at least one of the three circles, each of which has diameter $(1 - 1/100)\delta$. But each of the circles may contain at most two points. So the square can contain at most eight points.

Problem 2:

Show that the recurrence $T(n) = T(n/2) + a(n \lg n)$ solves to $O(n \lg^2 n)$. Assume that $T(1)$ is $O(1)$.

To prove that $T(n) = O(n \lg^2 n)$, we prove that there exist two constants $c > 0$ $n > n_0$, for all $n > n_0$, $T(n) \leq cn \lg^2 n$. We make the inductive assumption that this bound holds for $n/2$, that is, that $T(n/2) \leq c(n/2) \lg^2(n/2)$. Substituting this into the recurrence yields

$$\begin{aligned} T(n) &= 2T(n/2) + a(n \lg n) \\ &\leq 2c(n/2) \lg^2(n/2) + a(n \lg n) \\ &= cn(\lg n - 1)^2 + an \lg n \\ &= cn \lg^2 n + (an \lg n - 2cn \lg n + cn) \\ &\leq cn \lg^2 n + n \lg n(a - c) \qquad (1) \\ &\leq cn \lg^2 n, \qquad (2) \end{aligned}$$

where (1) holds when $n \geq 2$, and (2) holds as long as $c > a$. So we choose $c = a + 1$ and $n_0 = 2$.

Problem 3:

Show that $\log_k n = O(\log_2 n)$, for any constant $k > 1$.

We know that $k = 2^{\lg_2 k}$, so

$$2^{\lg_2 n} = n = k^{\lg_k n} = (2^{\lg_2 k})^{\lg_k n} = 2^{\lg_2 k \lg_k n}$$

So $\lg_2 n = \lg_2 k \lg_k n$, thus we can have $\lg_k n$ is $O(\lg_2)$ with constant $c = 1/\log_2 k$. By the way, $1/\lg_2 k = \lg_k 2$. Can you prove this?

Problem 4:

Prove that an array is Monge if and only if for all $i = 1, 2, ..., m-1$ and $j = 1, 2, ..., n-1$, we have $A_{i,j} + A_{i+1,j+1} \leq A_{i,j+1} + A_{i+1,j}$. Base case: if a 2×2 array is Monge, $A_{i,j} + A_{i+1,j+1} \leq A_{i,j+1} + A_{i+1,j}$. This is obvious by inspection.
Now we'll show that it's true for an $m \times 2$, for all $m > 2$, by induction. Our inductive assumption is that it's true for an $m \times 2$ array. Thus for any row i, $A_{i,1} + A_{m,2} \leq A_{m,1} + A_{i,2}$. If we add a new row $m+1$ which is Monge with respect to row m, $A_{m,1} + A_{m+1,2} \leq A_{m+1,1} + A_{m,2}$.
Then $A_{i,1} - A_{i,2} \leq A_{m,1} - A_{m,2}$, and $A_{m+1,1} - A_{m+1,2} \geq A_{m,1} - A_{m,2}$.
Thus $A_{i,1} - A_{i,2} \leq A_{m,1} - A_{m,2} \leq A_{m+1,1} - A_{m+1,2}$, and $A_{i,1} + A_{m+1,2} \leq A_{i,2} + A_{m+1,1}$.
Therefore for it is true for all $(m+1) \times 2$ arrays.
The same argument can be applied for $2 \times n$ arrays.
To show it's true for $m \times n$ arrays, we follow a similar inductive proof:
Base case: A $m \times 2$ array is Monge, for all $i = 1, 2, ..., m-1$, if we have $A_{i,1} + A_{i+1,2} \leq A_{i,2} + A_{i+1,2}$, as we showed above.
Our inductive assumption is that a $m \times n$ array is Monge. We need to show that if for all $i = 1, 2, ..., m-1$ and $j = 1, 2, ..., n$, $A_{i,j} + A_{i+1,j+1} \leq A_{i,j+1} + A_{i+1,j}$, the $m \times n+1$ array is also Monge:
We know that for any $i \in 1, 2, ..., m-1$ and for any $j \in 1, 2, ..., n-1$, $k \in j+1, j+2, ..., n-1$, $A_{i,j} + A_{m,k} \leq A_{m,j} + A_{i,k}$.
Add a column $n+1$, ensuring that it is Monge with respect to column n:
$A_{j,n} + A_{k,n+1} \leq A_{j,n+1} + A_{k,n}$.
Then $A_{i,n} - A_{k,n} \leq A_{j,n+1} - A_{k,n+1}$.
By the inductive assumption, we know that $A_{j,i} + A_{k,n} \leq A_{k,i} - A_{j,n}$, so we have $A_{j,i} - A_{k,i} \leq A_{j,n} - A_{k,n}$.
Thus $A_{j,i} - A_{k,i} \leq A_{j,n+1} - A_{k,n+1}$,
and $A_{j,i} + A_{k,n+1} \leq A_{j,n+1} + A_{k,i}$.

4.b.

We know that the 2×2 subarray $\begin{vmatrix} 23 & 22 \\ 6 & 7 \end{vmatrix}$ is not Monge, because $23+7 > 6+22$. Changing the upper-right element will do, without breaking any constraints. Thus, for example, the following array is Monge:

$$\begin{vmatrix} 37 & 23 & 24 & 32 \\ 21 & 6 & 7 & 10 \\ 53 & 34 & 30 & 31 \\ 32 & 13 & 9 & 6 \\ 43 & 21 & 15 & 8 \end{vmatrix}.$$

4.c. Prove that $f(1) \leq f(2) \leq ... \leq f(m)$ for any $m \times n$ Monge array.

Proof: Assume that for some $m \times n$ Monge array, there exists some row i such that $f(i)$ is not less than $f(i+1)$. Let $j = f(i+1)$, and $k = f(i)$. Since $f(i)$ is the index of the minimum value of row i, and $f(i+1)$ is the index of the minimum value of row $i+1$, there can be no pair of values in rows $i, i+1$ whose sum is less than or equal to $A_{i+1,j} + A_{i,k}$. But due to the definition of Monge, we know $A_{i,j} + A_{i+1,k+1} \leq A_{i+1,j} + A_{i,k}$. Therefore there cannot be a Monge array such that $f(i) > f(i+1)$.

4.d. Explain how to compute the leftmost minimum in the odd-numbered rows of A in $O(m+n)$ time.

We know from section 4.c that the leftmost minimum of row i is less than or equal to the leftmost minimum of $i+1$. We further know the leftmost minimum of the even rows. Given $f(2)$, we find $f(1)$ as follows: begin with column $f(2)$, and scan leftwards in row 1 until we find its minimum. Similarly, for row 3, we scan leftwards until we reach index $f(1)$.
We must scan all n columns, and at worst we must scan all m rows. Thus this algorithm takes $O(n+m)$ time.

4.e. Show the running time of the divide-and-conquer leftmost minimum element algorithm is $O(m + n \log m)$.

$T(n) = T(n/2) + O(m + n)$
Assume $T(n) = O(m + n \log m)$. Then
$T(n/2) = O(m + n/2 \log m) = O(m + n \log m)$, and
$T(n) = O(m + n \log m) + O(m + n)$
$\leq O(m + n \log m)$.