NOTE: The proofs here are intended to provide the rough idea, and not necessarily give the full details.

1: Recurrences, recurrences

(a) In the recursion tree, we have $k := \log_4 n$ levels, and in level i, we have 4^{i-1} terms that are $n/4^{i-1}$, along with 4^i terms $T(n/4^i)$ terms. Thus the contribution of each level to the sum is n, and the contribution of the *last* level due to the $T()$ terms is $4^k T(1) = O(n)$. This gives an overall bound of $O(n \log n)$.

(b) We consider the recursion tree as before. Now, in level i, we have 4^{i-1} terms that are 1, along with the $T()$ terms as above. The contribution to the last level due to the $T()$ terms is $O(n)$ once again. Thus, the overall bound is

$$\left(\sum_{i=1}^{\log_4 n} 4^{i-1}\right) + O(n) = O(n).$$

(c) $T(n) = T(n-1) + n = T(n-2) + (n-1) + n = \cdots = T(1) + 2 + \cdots + n = \frac{n(n+1)}{2} + T(1) - 1$. This is clearly $O(n^2)$.

(d) It is clear that $T(n) \geq \sqrt{n}$ and that $T(n) \leq O(n)$. The latter can easily be seen by induction. Now, suppose we guess that the answer is $T(n) = Cn^\alpha$, for some $\alpha \in [1/2, 1]$. Then, for an inductive proof to go through, we will need

$$C(1/2^\alpha + 1/3^\alpha) + n^{1/2-\alpha} \leq C.$$

What is the smallest α for which we can find a C such that the above holds? We can do a quick search. Turns out $\alpha = 0.79$ works.

(e) $T(n) = T(n^{1/2}) + 4 = T(n^{1/4}) + 8 = \cdots = T(n^{1/2^r}) + 4r$, for any r. If we set r such that $1/2^r = 1/\log n$, then the $T(n^{1/2^r})$ term becomes $T(1)$. Thus we have $T(n) = O(\log \log n)$.

(f) In the first case, the ith level in the tree has 3^i terms that are $T(n/2^i)$, and 3^{i-1} terms that are $(n^2/4^{i-1})$. Thus the 'work' done at the ith level is $n^2(3/4)^{(i-1)}$. Overall the tree has $k = \log_2 n$ levels. At the last level, the $T(1)$ terms add up to $O(3^k) = O(n^{\log_2 3})$. Thus the overall bound we get is $T(n) = O(n^2)$ (as $\sum_i (3/4)^i = O(1)$, and $\log_2 3 < 2$).

In the second case, the dominant terms are the $T(1)$ terms in the last level. This gives $T(n) = O(n^{\log_2 3})$.

In the final case, the work at each level works out to precisely $n^{\log_2 3}$ (irrespective of the level). Thus the overall bound we get is $T(n) = O(n^{\log_2 3} \log n)$.

2: Sorting nearby numbers

Algorithm: We maintain an array B of size M, in which the jth element is the number of times the number $\min_i A[i] + j$ appears in the array A. This array is initialized to zero, and is then populated in time $O(n)$, by performing a sweep through A.

Next, we can do a sweep through $B[]$, and using the counts, output the sorted version of A.

Running time: The two sweeps are both linear in the size of the corresponding arrays. This gives the bound $O(n + M)$.

3: Selecting in a Union

First, note that the answer can only lie in $A[0, \dots, k-1]$ or $B[0, \dots, k-1]$ (i.e., the remaining elements in the two arrays are irrelevant). Thus we will suppose that $N = k$.

Next, note that if we can find an r s.t. $A[r-1]$ satisfies $B[k-r-1] \le A[r-1] \le B[k-r]$, then $A[r-1]$ is the kth smallest element in the union. With this in mind, let us do a "binary search" in A as follows: start with $r = k/2$, and check if the above happens. If not, either $A[r-1] < B[k-r-1]$ or $A[r-1] > B[k-r-1]$. In the former case, the kth smallest element must be to the right of $A[r]$, thus we set $r = 3k/4$ (doing floor/ceiling as in regular binary search). Meanwhile, in the latter case, the kth smallest element must be to the left of $A[r]$, and we then proceed, setting $r = k/4$.

This process continues, and we either find the kth smallest element, or we find two consecutive indices $r-1, r$ with the guarantee that the answer lies between them. In this case, the kth smallest element in the union is $B[k-r-1]$.

4: Closest pair of restaurants in Manhattan

(a) Have points at $(1,0), (4,0), (5,0), (8,0)$. Then $x = 4.5$ is a vertical line that partitions the points into two sets of size $n/2$. In this case, $d = 3$, while the right answer is 1.

(b) Proof of hint: tile the $d \times d$ square S with 6 squares, each of size $(d/2) \times (d/3)$. Now, if we had 7 points in S, there have to be 2 points in one of the tiles (pigeon hole principle), and thus these two points have a Manhattan distance at most $d/2 + d/3 < d$.

Once we have this, we can reason about the algorithm as follows. First, the min pairwise distance can occur either between two points on the same side of x, or between points on different sides. In the former case, d will be the answer. In the latter case, let us denote by a and b the points that achieve this min distance. It is clear that the x-coordinates of a, b lie in $[x-d, x+d]$ (else the distance between a, b is $> d$). Thus we can throw away all the other points outside the strip, as step 3 does. Now consider sorting the points by y coordinates. In this ordering, we may suppose without loss of generality, that a appears before b. If there are ≤ 12 points between a and b in the ordering, then $\|a - b\|$ will be considered as a candidate distance, we will find it. Now, can there be ≥ 13 points in the ordering 'between' a and b? Suppose there are. First, note that the difference in y-coordinates of a, b is at most d (else the distance is $> d$). See fig below.

At least one of R_1 and R_2 contain ≥ 7 points. Thus there are points in either R_1 or R_2 that are $< d$ apart. This contradicts the fact that d is the shortest distance in the recursive calls in step (2).

(c) (Skipping description – pretty straightforward). We get the $O(n \log n)$ because of the sorting in step 4 of the algorithm. The rest of step 4 takes $O(n)$ time.

[Recurrence easy to solve]

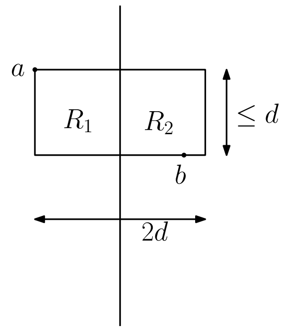

Figure 1: Figure showing possible relative positions of a, b

5: Linear Time Median

(You can look up any standard reference, e.g., the Dasgupta textbook.)