Problem 1. In the lecture, we presented an algorithm for solving the closest pair problem in $O(n \log n)$ expected time. However, our algorithm requires knowing the precise value of r, which is the distance between the closest pair found from recursion. Computing r precisely would require the "square root" operation, which is not an atomic operation of the real-RAM model. In this problem, you will see how this issue can be circumvented.

(a) In the lecture's algorithm, we imposed a grid where each cell has side length $r/\sqrt{2}$. Suppose that we instead impose a grid whose side length is $c \cdot r/\sqrt{2}$ for some positive constant $c < 1$. Explain how the algorithm can be modified to still find the closest pair correctly in $O(n \log n)$ expected time.

(b) Let p and q be two points whose Euclidean distance is $dist(p, q)$. Given the coordinates of p and q, explain how to obtain in $O(1)$ time a value r' satisfying $dist(p, q)/\sqrt{2} \leq r' \leq dist(p, q)$.

Solution.

(a) Let $c = \frac{1}{\sqrt{2}}$. Note that in this setup, each cell c contains at most 1 point (i.e. $|c(P)| \leq 2$), and still there are at most $4n$ non-empty cells. We now prove that by packing lemma, there would be at most $O(1)$ r-neighbouring cells. For a cell c, select the point that maximizes the minimum distance to the boundary of the cell, this is actually the center of the square (we have to pick a center for the ball in packing lemma). Then, we can pick this point to be the center of the circle, and let the radius of this circle $r_c = r + \sqrt{2}r/2$, we can first walk at most $\sqrt{2}r/2$ far away from the center to reach the boundary of the cell c, and then every point no further from the cell c can be reached. The packing lemma gives at most $(1 + \left\lceil \frac{2(r+\sqrt{2}r/2)}{r/2} \right\rceil)^2 = (1 + \lceil 4(1 + \sqrt{2}/2) \rceil)^2 = (1 + \lceil 4(1 + \sqrt{2}/2) \rceil)^2 = 64$ cells to be covered. It is fine if no square root is involved, in this case, let $r_c = 2r$, and packing lemma gives $(1 + 8)^2 = 81$.
For implementation of the algorithm, we can apply a single BFS from the source cell c_1, whenever a cell c' found by the algorithm satisfies the requirement of r_c, add the neighbouring cells in. We have argued that only $O(1)$ cells will be considered. Since there are only $O(n)$ c_1s, there will be only in total $|c_1(P) \times c_2(P)| = O(n)$ cells to be considered. The merge part of the algorithm is $O(n)$ expected as required.

(b) Let $r' = \max\{\Delta_x, \Delta_y\}$, where $\Delta_x = |p_x - q_x|, \Delta_y = |p_y - q_y|$.
Note that taking the absolute of a value x would require one comparison and based on the result of comparison we need to multiply that value by -1 if necessary, this takes at most two atomic operations. We will need one comparison for calculating the maximum.
Now, without loss of generality, suppose $\Delta_x \leq \Delta_y$, then $r' = \Delta_y$ and

$$dist(p, q) = \sqrt{(\Delta_x)^2 + (\Delta_y)^2} \geq \sqrt{\Delta_y^2} = \Delta_y = r'.$$

Suppose that $r' < dist(p, q)/\sqrt{2}$, then

$$dist(p, q) \leq \sqrt{r'^2 + r'^2} < \sqrt{(dist(p, q)/\sqrt{2})^2 + (dist(p, q)/\sqrt{2})^2} = dist(p, q),$$

which is a contradiction.

Problem 2. Design an algorithm that solves the closest pair problem in $\mathbb{R}^d$ in $O(n \log n)$ expected time.

Solution. Recursively (by finding the median m among the x-coordinates of the n points, evenly divide these points by adding a hyperplane $\alpha : x = m$), call the procedure to solve the left and right subproblem, let the results returned be p, q and p', q', and let $r = \min\{dist(p,q), dist(p',q')\}$. We instead impose a grid with side length $r/\sqrt{d}$.

procedure ClosestPair(P)
    $m \leftarrow$ the median of the x-coordinates among these n points
    $\alpha \leftarrow$ the hyperplane $x = m$
    $P_1 \leftarrow$ points on the left of α
    $P_2 \leftarrow$ points on the right of α
    $(p', q') \leftarrow$ClosestPair(P_1)
    $(p'', q'') \leftarrow$ClosestPair(P_2)
    for every non-empty cell c_1 on the left of α **do**
        for every r-neighbor cell c_2 of c_1 of the right of α **do**
            calculate the distance of each pair of points $(p_1, p_2) \in c_1(P) \times c_2(P)$, let (p, q) be the pair with smallest distance among the pairs investigated and also $(p', q'), (p'', q'')$
        end for
    end for
    return (p, q)
end procedure

Again we verify that:

(a) Every cell contains at most $O(1)$ points.
Proof. If a point is not at the corner, the r-radius ball spanned from the point could immediately cover the cell. For the case of a point lying at one of the corners of a hypercube, the only point not covered is on the other endpoint of the diagonal.

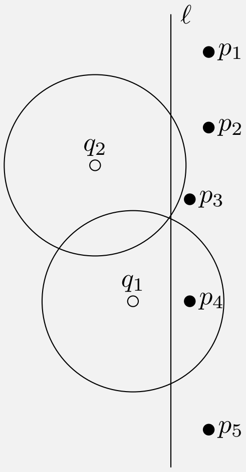
(b) Every cell has at most $O(1)$ r-neighbouring cells, this can be verified by the packing lemma.
Proof. Locate the ball at the center of the cell, extend the surface of the cell by distance of d. The ball can cover the resulting hypercube with radius of

$$\left(1 + \left\lceil \frac{2(r + r/\sqrt{d})}{r/\sqrt{d}} \right\rceil\right)^d = \lceil 2\sqrt{d} + 3 \rceil^d = O(1).$$

(c) There are at most $2^d \cdot n = O(n)$ non-empty cells (for hashing), since a point can locate in the middle of two neighboring coordinates in at most d dimensions.

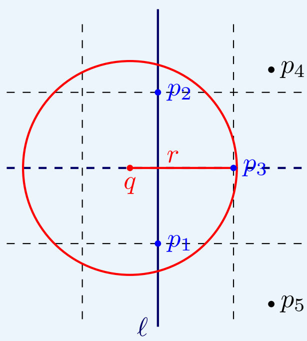
This does not alter the expected time complexity in the merge step.

Problem 3. Let ℓ be a vertical line, and P be a set of n points on the right of ℓ. Define r as the distance of the closest pair of P. It is known that every point in P has distance at most r from ℓ.
We are now given a point q on the left of ℓ. Denote by $D_q(r)$ the disc that centers at q and has radius r. Define an r-bounded nearest neighbor (NN) of q as a point $p \in P \cap D_q(r)$ having the smallest distance to q.



For example, in the above figure, $P = \{p_1, p_2, \ldots, p_5\}$, and r is the distance of p_2 and p_3. The (only) r-bounded NN of q_1 is p_4, whereas q_2 has no r-bounded NNs. The two circles illustrate $D_{q_1}(r)$ and $D_{q_2}(r)$.
Consider the following approach for finding an r-bounded NN of q. First, sort $P \cup \{q\}$ by y-coordinate. Then, inspect the 20 points positioned before and after q in the sorted list, respectively; namely, 40 points are inspected in total. Prove that all r-bounded NNs (if they exist) of q must be among those 40 points.
Hint: Impose a grid, and 40 is rather conservative.

Solution. Impose a grid with side length $r/\sqrt{2}$. Align the grid lines with ℓ and y-coordinate of q, respectively. Then, the circle expanded from q with radius r will intersect with at most $\left(1 + \left\lceil \frac{2r}{r/\sqrt{2}} \right\rceil\right)^2 = 16$ cells (by packing lemma). Now every cell contains at most 2 points. There are in total at most $2 \times 16 = 32$ points required to be investigated.



Note: actually the circle can only intersect with at most 8 cells on the right of ℓ because the center cannot cross the line (so the factor $\frac{1}{2}$ is due to symmetry). We can also expect that the factor 2 cannot always be achieved.

Problem 4. Let P be a set of points in $\mathbb{R}^2$. Give an algorithm to find the closest pair of P in $O(n \log n)$ time deterministically.
Hint: Use the finding in Problem 3.

Solution. Note that we will need points to be sorted in y-coordinate, then only a constant number of points are required to be investigated.

We can first sort all points in x-coordinate in $O(n \log n)$ time. Use divide and conquer again, the sub-procedure will be able to "merge-sort" all the points in y-coordinate, and thus merging will cost only $O(n)$ time in the procedure.

procedure ClosestPair($P[1 \ldots n], l, r$)
    $m \leftarrow (l + r)/2$
    $(p', q') \leftarrow$ClosestPair(P,l, m)
    $(p'', q'') \leftarrow$ClosestPair(P,$m + 1, r$)
    $r \leftarrow \min\{dist(p', q'), dist(p'', q'')\}$
    add all points in $P[m + 1 \ldots r]$ satisfying $x_p - x_m \leq r$ to a list L
    for $i \leftarrow 1$ to m **do**
        if $dist(i, \ell) \leq r$ **then**
            Maintain y_{pos} satisfying $y_i \geq y_{pos}$ in L
            for $j \leftarrow \max\{0, pos - 32\}$ to $\min\{pos + 32, |L|\}$ **do**
                calculate the distance of each pair of points $(p_i, L[j])$, let (p, q) be the pair with smallest distance among the pairs investigated and also $(p', q'), (p'', q'')$
            end for
        end if
    end for
    merge $P[l \ldots r]$ by their y-coordinates, in ascending order
    return (p, q)
end procedure

This implementation is efficient in a sense that: there is no extra memory space needed to pass to the subprocesses for temporary storage of array elements: the algorithm discards some property used and create new ordering property for ancestors, sorting of y-coordinates are made in-situ; maintenance of y_{pos} is simple since a pointer should only shift right throughout the loop, making the total time shifted $O(n)$; merging requires only $O(n)$, since the subproblem has made the two subarrays y-monotone. This algorithm should give a deterministically $O(n \log n)$ running time.

Problem 5. Let P be a set of points in $\mathbb{R}^d$. Give an algorithm to find the closest pair of P in $O(n \log n)$ time deterministically.
Hint: How do you generalize your algorithm for Problem 4?

Solution. We adapt the algorithm in Problem 4. Note the following changes:

We split the points by a hyperplane instead of a line. We only add points with distance less than r into L. The constant 32 has to be changed due to dimensionality as described by the following.

Impose a grid with side length $r/\sqrt{d}$. Align the grid lines with the hyperplane $\alpha : x = x$-coordinate of $P[mid]$ and y-coordinate of q, respectively. Then, the circle expanded from q with radius r will intersect with at most $\left(1 + \left\lceil \frac{2r}{r/\sqrt{d}} \right\rceil\right)^d = (1 + \lceil 2\sqrt{d} \rceil)^d = O(1)$ cells (by packing lemma). Now every cell remains to contain at most 2 points. There are in total at most $O(1)$ points required to be investigated.