CSE 331: Introduction to Algorithm Analysis and Design

Divide and Conquer

1 Closest Points

1.1 Problem

Input: A set of n 2d points $P = \{p_1, \dots, p_n\}$ such that $p_i = (x_i, y_i) \in \mathbb{R}^2$. Assume all x and y are distinct.

Output: The pair of points that are closest (minimum distance)

1.2 Definitions

Distance: $d(p_i, p_j) = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}$

1.3 Algorithm

```
1. Copy the input points $P$ into $P_x$ and $P_y$ and sort by $x$ and $y$ respectively
2. return $closestPair(P_x, P_y)$
```

```
$closestPair(P_x, P_y)$:
    1. If $n < 4$ find closest pair by brute force and return them
    2. Divide $P_x$ in half as $Q$ and $R$
    3. Define $x*$ as the median $x$ value in $P$ which divides $Q$ and $R$
    4. Compute $Q_x$, $Q_y$, $R_x$, and $R_y$
    5. $(q_0, q_1) = closestPair(Q_x, Q_y)$
    6. $(r_0, r_1) = closestPair(R_x, R_y)$
    7. $\delta = min(d(q_0, q_1), d(r_0, r_1))$
    8. $S_y =$ All points $(x, y) \in P$ such that $abs(x - x*) < \delta$ sorted by $y$
    9. return $closestInBox(S_y, (q_0, q_1), (r_0, r_1))$
```

```
$closestInBox(S_y, (q_0, q_1), (r_0, r_1))$:
    1. $\delta = min(d(q_0, q_1), d(r_0, r_1))$
```

2. For each $s \in S_y$ check the distance between s and the next 15 points in S_y
3. If any checked distance is less than δ, set δ to that distance and store the compared points
4. return the points resulting in the min δ

1.4 Correctness

Proof. As with the other recursive algorithms we've seen in this class we will prove correctness by induction following the structure of the algorithm.

Base case: For $n < 4$ the algorithm finds the closest pair of points by brute force. Since all possibilities are checked, the correct solution must have been found.

Induction step: We will assume that the algorithm is correct for all $0 \leq n \leq k$ and prove that it is correct for $n = k + 1$. By the inductive hypothesis we can assume that the two recursive calls of $closestPair$ return the correct pair of points. To finish this proof we must show that the way we merge the two sub-solutions is correct. To do this we consider all the points that are within δ in x value from $x*$. If any point has an x value further than δ away from $x*$ then it must be further than δ from any point in the other sub-problem and therefore cannot be closer than δ to any point in either set. Thus, only points in S need to be considered when looking for a pair of points with distance less than δ.

We now consider each point in S in order of y value and consider the next 15 points while each is being considered. Our claim is that we only need to check the next 15 points and the 16th point and beyond cannot be within δ of the point being considered. Let's consider the 2δ by 2δ box in S with $x*$ at the center and the current point at the bottom of the box. Now divide this box into 16 smaller boxes of size $\delta/2$ by $\delta/2$. Note that since each of these boxes are fully contained in either Q or R that at most 1 point can be in each box since the max distance two points in the same box could be is less than δ which would mean they are closer than δ which is a contradiction of the inductive hypothesis. Since the current point is at the bottom of the larger box, any point above this box must be at least 2δ away from this point which is greater than δ. With 16 boxes with at most 16 points (including the current point) that could be within δ we only need to check the next 15 points. If a point with a smaller y value than the current point is within δ then that distance would have been found when the lower point was checking its next 15 points. Since the algorithm is checking every pair that could possibly be closer than δ to each other it must have found any such pair of points

□

1.5 Runtime

1. Copy the input points P into P_x and P_y and sort by x and y respectively $O(n \cdot \log n)$ to sort
2. return $closestPair(P_x, P_y)$ $O(n \cdot \log n)$ by analysis below and solving the recurrence relation $T(n) = cn + 2T(n/2)$ which was shown to be $O(n \cdot \log n)$ in the mergeSort runtime analysis. This is the overall runtime of the algorithm.

$closestPair(P_x, P_y)$:

1. If $n < 4$ find closest pair by brute force and return them $O(1)$ since 4 is a constant
2. Divide P_x in half as Q and R $O(n)$. Potentially $O(1)$ depending on implementation details but this wouldn't help the overall asymptotic runtime
3. Define $x*$ as the median x value in P which divides Q and R $O(1)$
4. Compute Q_x, Q_y, R_x, and R_y $O(n)$. Sorted by x is given since we divide on x. Sorted by y can be done in $O(n)$ by iterating over P_y and copying the points into Q_y and R_y depending on their x values
5. $(q_0, q_1) = closestPair(Q_x, Q_y)$ $T(n/2)$
6. $(r_0, r_1) = closestPair(R_x, R_y)$ $T(n/2)$
7. $\delta = min(d(q_0, q_1), d(r_0, r_1))$ $O(1)$
8. S_y = All points $(x, y) \in P$ such that $abs(x - x*) < \delta$ sorted by y $O(n)$ by using the same method used to compute Q_y and R_y
9. return $closestInBox(S_y, (q_0, q_1), (r_0, r_1))$ $O(n)$ by analysis below

$closestInBox(S_y, (q_0, q_1), (r_0, r_1))$:

1. $\delta = min(d(q_0, q_1), d(r_0, r_1))$ $O(1)$
2. For each $s \in S_y$ check the distance between s and the next 15 points in S_y $O(n)$. Since 15 is a constant we are doing constant work for each of the $O(n)$ points (worst-case all n points are in S)
3. If any checked distance is less than δ, set δ to that distance and store the compared points $O(1)$
4. return the points resulting in the min δ $O(1)$