Overview

- Closest Pair
- Non-dominated points
- Acknowledgements
Closest Pair

- **Input:** \( n \) points \((x_1, y_1), \ldots, (x_n, y_n) \in \mathbb{R}^2\)
- **Output:** indices \( 1 \leq i < j \leq n \) which minimizes the distance
- Unit cost model!
- Simplifying assumption: all \( x \) coordinates are distinct.
- **Exercise:** remove this assumption, but preserve the running time.
Closest Pair

- **Input:** $n$ points $(x_1, y_1), \ldots, (x_n, y_n) \in \mathbb{R}^2$
- **Output:** indices $1 \leq i < j \leq n$ which minimizes the distance
- Exhaustive search: compute all distances and output minimum one - running time $O(n^2)$
Closest Pair

- **Input:** $n$ points $(x_1, y_1), \ldots, (x_n, y_n) \in \mathbb{R}^2$
- **Output:** indices $1 \leq i < j \leq n$ which minimizes the distance
- Exhaustive search: compute all distances and output minimum one - running time $O(n^2)$
- Can we do better?

- Divide and conquer!
  1. Vertical line $\Lambda$ that separates points into 2 halves (left and right of $\Lambda$)
  2. Use median finding algorithm from previous lecture.
  3. Let $L$ and $R$ be the set of points to left and right of $\Lambda$, respectively
  4. Solve closest pair for $L$ and for $R$. Suppose the smallest distance $\delta$ is between points of $L$. 
  5. Can we do better?
Closest Pair

- **Input:** $n$ points $(x_1, y_1), \ldots, (x_n, y_n) \in \mathbb{R}^2$
- **Output:** indices $1 \leq i < j \leq n$ which minimizes the distance
- Exhaustive search: compute all distances and output minimum one - running time $O(n^2)$

- Divide and conquer!
  - Vertical line $\Lambda$ that separates points into 2 halves (left and right of $\Lambda$)
    - Use median finding algorithm from previous lecture.
Closest Pair

- **Input:** \( n \) points \((x_1, y_1), \ldots, (x_n, y_n) \in \mathbb{R}^2\)
- **Output:** indices \( 1 \leq i < j \leq n \) which minimizes the distance

Exhaustive search: compute all distances and output minimum one - running time \( O(n^2) \)

Divide and conquer!

1. Vertical line \( \Lambda \) that separates points into 2 halves (left and right of \( \Lambda \))
   - Use median finding algorithm from previous lecture.
2. Let \( L \) and \( R \) be the set of points to left and right of \( \Lambda \), respectively
Closest Pair

- **Input:** $n$ points $(x_1, y_1), \ldots, (x_n, y_n) \in \mathbb{R}^2$
- **Output:** indices $1 \leq i < j \leq n$ which minimizes the distance

Exhaustive search: compute all distances and output minimum one - running time $O(n^2)$

**Divide and conquer!**

1. Vertical line $\Lambda$ that separates points into 2 halves (left and right of $\Lambda$)
   - Use median finding algorithm from previous lecture.
2. Let $L$ and $R$ be the set of points to left and right of $\Lambda$, respectively
3. Solve closest pair for $L$ and for $R$. Suppose the smallest distance $\delta$ is between points of $L$.

Are we done?
Closest Pair

- **Input:** $n$ points $(x_1, y_1), \ldots, (x_n, y_n) \in \mathbb{R}^2$
- **Output:** indices $1 \leq i < j \leq n$ which minimizes the distance
- Exhaustive search: compute all distances and output minimum one - running time $O(n^2)$

**Divide and conquer!**

1. Vertical line $\Lambda$ that separates points into 2 halves (left and right of $\Lambda$)
   Use median finding algorithm from previous lecture.
2. Let $L$ and $R$ be the set of points to left and right of $\Lambda$, respectively
3. Solve closest pair for $L$ and for $R$. Suppose the smallest distance $\delta$ is between points of $L$.

Are we done?

Nope. Need to check if smallest distance is between points crossing from $L$ to $R$. 

Closest Pair

- **Input:** $n$ points $(x_1, y_1), \ldots, (x_n, y_n) \in \mathbb{R}^2$
- **Output:** indices $1 \leq i < j \leq n$ which minimizes the distance
- Exhaustive search: compute all distances and output minimum one - running time $O(n^2)$

- Divide and conquer!
  1. Vertical line $\Lambda$ that separates points into 2 halves (left and right of $\Lambda$)
     Use median finding algorithm from previous lecture.
  2. Let $L$ and $R$ be the set of points to left and right of $\Lambda$, respectively
  3. Solve closest pair for $L$ and for $R$. Suppose the smallest distance $\delta$ is between points of $L$.
     
     Are we done?
     
     Nope. Need to check if smallest distance is between points crossing from $L$ to $R$.
     
     Checking crossing pairs seems as hard as the original problem!
Closest Pair

- **Input:** $n$ points $(x_1, y_1), \ldots, (x_n, y_n) \in \mathbb{R}^2$
- **Output:** indices $1 \leq i < j \leq n$ which minimizes the distance
- Exhaustive search: compute all distances and output minimum one - running time $O(n^2)$

- Divide and conquer!
  1. Vertical line $\Lambda$ that separates points into 2 halves (left and right of $\Lambda$)
     Use median finding algorithm from previous lecture.
  2. Let $L$ and $R$ be the set of points to left and right of $\Lambda$, respectively
  3. Solve closest pair for $L$ and for $R$. Suppose the smallest distance $\delta$ is between points of $L$.

  **Observation:** only need to check if $\exists$ crossing pair with distance $< \delta$
Closest Pair

- **Input:** \( n \) points \((x_1, y_1), \ldots, (x_n, y_n) \in \mathbb{R}^2\)
- **Output:** indices \(1 \leq i < j \leq n\) which minimizes the distance
- Exhaustive search: compute all distances and output minimum one - running time \(O(n^2)\)

- **Divide and conquer:**
  1. Vertical line \(\Lambda\) that separates points into 2 halves (left and right of \(\Lambda\))
     - Use median finding algorithm from previous lecture.
  2. Let \(L\) and \(R\) be the set of points to left and right of \(\Lambda\), respectively
  3. Solve closest pair for \(L\) and for \(R\). Suppose the smallest distance \(\delta\) is between points of \(L\).

    **Observation:** only need to check if \(\exists\) crossing pair with distance \(< \delta\)
    
    Could just pay attention to points with \(x\)-coordinate within \(\delta\) to line \(\Lambda\)... but still all points can be there...
Closest pair - boxing up

- Make $\delta/2 \times \delta/2$ boxes!

- Each square box has $\leq 1$ point from our set

- Maximum distance inside square is $\delta/\sqrt{2}$

- Each point only needs to compute distances with points within two horizontal layers

- All other distances are $> \delta$

- Hence, each point needs only check its distance with $\leq 11$ other points!

- Now we only need to check $O(n)$ pairs
Closest pair - boxing up

- Make $\delta/2 \times \delta/2$ boxes!
- Each square box has $\leq 1$ point from our set

  Maximum distance inside square is $\delta/\sqrt{2}$

$$\sqrt{(\frac{\delta}{2})^2 + (\frac{\delta}{2})^2} = \frac{\delta}{\sqrt{2}}$$
Closest pair - boxing up

- Make \( \frac{\delta}{2} \times \frac{\delta}{2} \) boxes!
- Each square box has \( \leq 1 \) point from our set
  - Maximum distance inside square is \( \frac{\delta}{\sqrt{2}} \)
- Each point only needs to compute distances with points within two horizontal layers
  - All other distances are \( > \delta \)
Closest pair - boxing up

- Make $\delta/2 \times \delta/2$ boxes!
- Each square box has $\leq 1$ point from our set
  - Maximum distance inside square is $\delta/\sqrt{2}$
- Each point only needs to compute distances with points within two horizontal layers
  - All other distances are $> \delta$
- Hence, each point needs only check its distance with $\leq 11$ other points!

Now we only need to check $O(n)$ pairs$^1$

---

$^1$Before boxing needed to check $\Omega(n^2)$ pairs
Algorithm

1. Find vertical line Λ
2. Recursively solve $L$, $R$ subproblems
3. Linear scan to remove points $> \delta$ far (horizontally) from Λ
4. Sort points by $y$-coordinate, store them in array $A$
5. For each point in $A$, compute distances to next 11 points in $A$
6. Return minimum distance found.

Correctness: by arguments in previous slides.

Running time:
- We can first sort $y$-coordinates prior to recursing, and this sorted array can still be used in recursion. Thus, running time (with sorted input):
  \[ T(n) = 2T(n/2) + O(n) \]
- Adding the time to sort doesn't change total runtime.

\[ T(n) = O(n \log n) \]
Algorithm

1. Find vertical line $\Lambda$
2. Recursively solve $L, R$ subproblems
3. Linear scan to remove points $> \delta$ far (horizontally) from $\Lambda$
4. Sort points by $y$-coordinate, store them in array $A$
5. For each point in $A$, compute distances to next 11 points in $A$
6. Return minimum distance found.

- **Correctness:** by arguments in previous slides.
Algorithm

1. Find vertical line Λ
2. Recursively solve $L, R$ subproblems
3. Linear scan to remove points $> \delta$ far (horizontally) from Λ
4. Sort points by $y$-coordinate, store them in array $A$
5. For each point in $A$, compute distances to next 11 points in $A$
6. Return minimum distance found.

- **Correctness:** by arguments in previous slides.
- **Running time:** (naive)

\[
T(n) = 2T(n/2) + O(n \log n) \Rightarrow T(n) = O(n \log^2 n)
\]
Algorithm

1. Find vertical line $\Lambda$
2. Recursively solve $L$, $R$ subproblems
3. Linear scan to remove points $> \delta$ far (horizontally) from $\Lambda$
4. Sort points by $y$-coordinate, store them in array $A$
5. For each point in $A$, compute distances to next 11 points in $A$
6. Return minimum distance found.

- **Correctness:** by arguments in previous slides.
- **Running time:** (sorting in beginning)
  We can first sort $y$-coordinates prior to recursing, and this sorted array can still be used in recursion. Thus, running time (with sorted input):

$$T(n) = 2T(n/2) + O(n) \Rightarrow T(n) = O(n \log n)$$

adding the time to sort doesn’t change total runtime.
Closest Pair

Non-dominated points

Acknowledgements
Non-dominated points

- Given two points \((x_1, y_1)\) and \((x_2, y_2)\)

\((x_1, y_1)\) dominates \((x_2, y_2)\) if \(x_1 > x_2\) and \(y_1 > y_2\).
Non-dominated points

- **Input:** set of $n$ points $S := \{(x_1, y_1), \ldots, (x_n, y_n)\}$
- **Output:** all *non-dominated* points of $S$
- **Model:** unit-cost model
- **Assumptions:** (for simplicity) distinct $x$ values
Non-dominated points

- **Input:** set of $n$ points $S := \{(x_1, y_1), \ldots, (x_n, y_n)\}$
- **Output:** all *non-dominated* points of $S$
- Naive algorithm:
  
  For each point $(x_i, y_i)$ check against all other points, if it is dominated or not.

  **Running time:** $O(n^2)$
Non-dominated points

- **Input:** set of $n$ points $S := \{(x_1, y_1), \ldots, (x_n, y_n)\}$
- **Output:** all *non-dominated* points of $S$
- Naive algorithm:
  
  For each point $(x_i, y_i)$ check against all other points, if it is dominated or not.

  **Running time:** $O(n^2)$

- Can we do better?

  - Divide and conquer!
    
    1. Sort points according to $x$-coordinate
    2. Recursively solve two subproblems $n/2$ points to the left of middle (denoted $S_L$), $n/2$ points to the right of middle (denoted $S_R$)
    3. How do we combine?
      
      (astute) Observation: no point in $S_L$ dominates a point in $S_R$
      
      Need to eliminate points from $S_L$ which are dominated by a point in $S_R$

      These must be the points with $y$-coordinate larger than the largest height of $S_R$!
Non-dominated points

- **Input:** set of $n$ points $S := \{(x_1, y_1), \ldots, (x_n, y_n)\}$
- **Output:** all *non-dominated* points of $S$
- Naive algorithm:
  
  For each point $(x_i, y_i)$ check against all other points, if it is dominated or not.

  **Running time:** $O(n^2)$

- Can we do better?
- Divide and conquer!
  1. Sort points according to $x$-coordinate
  2. Recursively solve two subproblems $n/2$ points to the left of middle (denoted $S_L$), $n/2$ points to the right of middle (denoted $S_R$)
  3. How do we combine?
    - (astute) Observation: no point in $S_L$ dominates a point in $S_R$
    - Need to eliminate points from $S_L$ which are dominated by a point in $S_R$
    - These must be the points with $y$-coordinate larger than the largest height of $S_R$!
Combining solutions to subproblems

Let $ND_L = [P_1, \ldots, P_a]$ and $ND_R = [Q_1, \ldots, Q_b]$ be non-dominated points of $S_L, S_R$, respectively, sorted by $x$-coordinate.
Combining solutions to subproblems

- Let $ND_L = [P_1, \ldots, P_a]$ and $ND_R = [Q_1, \ldots, Q_b]$ be non-dominated points of $S_L, S_R$, respectively, sorted by $x$-coordinate.
- Must be the case that $y(Q_1) > y(Q_j)$ for all $j > 1$!
Combining solutions to subproblems

- Let $ND_L = [P_1, \ldots, P_a]$ and $ND_R = [Q_1, \ldots, Q_b]$ be non-dominated points of $S_L, S_R$, respectively, sorted by $x$-coordinate.
- Must be the case that $y(Q_1) > y(Q_j)$ for all $j > 1$!
- Thus, only need to compare $y(P_i)$ with $y(Q_1)$!
- $O(n)$ time to combine!
Combining solutions to subproblems

- Let $ND_L = [P_1, \ldots, P_a]$ and $ND_R = [Q_1, \ldots, Q_b]$ be non-dominated points of $S_L, S_R$, respectively, *sorted by* $x$-coordinate.
- Must be the case that $y(Q_1) > y(Q_j)$ for all $j > 1$!
- Thus, only need to compare $y(P_i)$ with $y(Q_1)$!
- $O(n)$ time to combine!

Algorithm

1. Sort points by $x$-coordinate
2. Recursively solve two subproblems $n/2$ points to the left of middle (denoted $S_L$), $n/2$ points to the right of middle (denoted $S_R$)
3. Combine points as above (linear scan)
4. Output non-dominated points
Combining solutions to subproblems

- Let $ND_L = [P_1, \ldots, P_a]$ and $ND_R = [Q_1, \ldots, Q_b]$ be non-dominated points of $S_L, S_R$, respectively, *sorted by* $x$-coordinate.
- Must be the case that $y(Q_1) > y(Q_j)$ for all $j > 1$!
- Thus, only need to compare $y(P_i)$ with $y(Q_1)$!
- $O(n)$ time to combine!

**Algorithm**

1. Sort points by $x$-coordinate
2. Recursively solve two subproblems $n/2$ points to the left of middle (denoted $S_L$), $n/2$ points to the right of middle (denoted $S_R$)
3. Combine points as above (linear scan)
4. Output non-dominated points

**Running time:**

1. sorting $O(n \log n)$
2. Recursion (for sorted input):
   
   $$T(n) = 2T(n/2) + O(n) \Rightarrow T(n) = O(n \log n)$$
3. **Total runtime:** $O(n \log n)$
Acknowledgement

- Based on Prof. Lau's lecture 4
  https://cs.uwaterloo.ca/~lapchi/cs341/notes/L04.pdf
- Based on Prof. Brown’s lecture (see course webpage)
References I

Kleinberg, John and Tardos, Eva (2006)  
Algorithm Design.  
Addison Wesley