Divide-and-Conquer

Reading: CLRS Sections 2.3, 4.1, 4.2, 4.3, 28.2, 33.4.

CSE 6331 Algorithms
Steve Lai
Divide and Conquer

- Given an instance $x$ of a problem, the divide-and-conquer method works as follows:

  function DAC(x)
   
   if $x$ is sufficiently small then
     solve it directly
   
   else
     divide $x$ into smaller subinstances $x_1, x_2, \ldots, x_k$;
     $y_i \leftarrow$ DAC($x_i$), for $1 \leq i \leq k$;
     $y \leftarrow$ combine($y_1, y_2, \ldots, y_k$);
     return($y$)
Analysis of Divide-and-Conquer

- Typically, \( x_1, \ldots, x_k \) are of the same size, say \( \lfloor n/b \rfloor \).
- In that case, the time complexity of DAC, \( T(n) \), satisfies a recurrence:

\[
T(n) = \begin{cases} 
  c & \text{if } n \leq n_0 \\
  kT(\lfloor n/b \rfloor) + f(n) & \text{if } n > n_0 
\end{cases}
\]

- Where \( f(n) \) is the running time of dividing \( x \) and combining \( y_i \)'s.
- What is \( c \) ?
- What is \( n_0 \) ?
Mergesort: Sort an array $A[1..n]$

- **procedure** mergesort($A[i..j]$)
  
  // Sort $A[i..j]$/

  if $i = j$ then return  // base case //

  $m \leftarrow \lfloor (i + j)/2 \rfloor$

  mergesort($A[i..m]$)

  mergesort($A[m+1..j]$)

  merge($A[i..m]$, $A[m+1..j]$)

- Initial call: mergesort($A[1..n]$)
Analysis of Mergesort

- Let $T(n)$ denote the running time of mergesorting an array of size $n$.
- $T(n)$ satisfies the recurrence:

$$T(n) = \begin{cases} 
  c & \text{if } n \leq 1 \\
  T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + \Theta(n) & \text{if } n > 1
\end{cases}$$

- Solving the recurrence yields:

$$T(n) = \Theta(n \log n)$$

- We will learn how to solve such recurrences.
Linked-List Version of Mergesort

- function mergesort(i, j)

  // Sort A[i..j]. Initially, link[k] = 0, 1 ≤ k ≤ n.//

  global A[1..n], link[1..n]

  if i = j then return(i)  // base case //

  m ← \lfloor (i + j)/2 \rfloor

  ptr1 ← mergesort(i, m)

  ptr2 ← mergesort(m + 1, j)  \text{ divide and conquer }

  ptr ← merge(ptr1, ptr2)

  return(ptr)
Solving Recurrences

- Suppose a function $T(n)$ satisfies the recurrence

\[
T(n) = \begin{cases} 
  c & \text{if } n \leq 1 \\
  3T(\lfloor n/4 \rfloor) + n & \text{if } n > 1 
\end{cases}
\]

where $c$ is a positive constant.

- Wish to obtain a function $g(n)$ such that $T(n) = \Theta(g(n))$.

- Will solve it using various methods: Iteration Method, Recurrence Tree, Guess and Prove, and Master Method.
Iteration Method

Assume $n$ is a power of 4. Say, $n = 4^m$. Then,

$T(n) = n + 3T(n/4)$

$= n + 3\left[\frac{n}{4} + 3T(n/16)\right]$  

$= n + 3\left(\frac{n}{4}\right) + 9\left[\frac{(n/16)}{4} + 3T(n/64)\right]$  

$= n + (3/4)n + (3/4)^2 n + 3^3 T(n/4^3)$  

$= n \left[1 + \frac{3}{4} + \left(\frac{3}{4}\right)^2 + \cdots + \left(\frac{3}{4}\right)^{m-1}\right] + 3^m T\left(\frac{n}{4^m}\right)$  

$= n \Theta(1) + O(n) = \Theta(n)$

So, $T(n) = \Theta(n \mid n \text{ a power of 4}) \Rightarrow T(n) = \Theta(n)$. (Why?)
Remark

- We have applied Theorem 7 to conclude $T(n) = \Theta(n)$ from $T(n) = \Theta(n \mid n$ a power of 4).
- In order to apply Theorem 7, $T(n)$ needs to be nondecreasing.
- It will be a homework question for you to prove that $T(n)$ is indeed nondecreasing.
Recurrence Tree

solving problems

1 of size $n$

\[
\downarrow \uparrow
\]

3 of size $n/4$

\[
\downarrow \uparrow
\]

$3^2$ of size $n/4^2$

\[
\downarrow \uparrow
\]

$3^3$ of size $n/4^3$

\[
\downarrow \uparrow
\]

\vdots

$3^{m-1}$ of size $n/4^{m-1}$

\[
\downarrow \uparrow
\]

$3^m$ of size $n/4^m$

\[
\downarrow \uparrow
\]

time needed

$n$

$3 \cdot n/4$

$3^2 \cdot n/4^2$

$3^3 \cdot n/4^3$

\vdots

$3^{m-1} \cdot n/4^{m-1}$

$3^m \cdot \Theta(1)$
Guess and Prove

- Solve \( T(n) = \begin{cases} 
  c & \text{if } n \leq 1 \\
  3T(\lfloor n/4 \rfloor) + \Theta(n) & \text{if } n > 1 
\end{cases} \)

- First, guess \( T(n) = \Theta(n) \), and then try to prove it.
- Sufficient to consider \( n = 4^m, m = 0, 1, 2, \ldots \)
- Need to prove: \( c_1 4^m \leq T(4^m) \leq c_2 4^m \) for some \( c_1, c_2 \) and all \( m \geq m_0 \) for some \( m_0 \). We choose \( m_0 = 0 \) and prove by induction on \( m \).
- IB: When \( m = 0 \), \( c_1 4^0 \leq T(4^0) \leq c_2 4^0 \) if \( c_1 \leq c \leq c_2 \).
- IH: Assume \( c_1 4^{m-1} \leq T(4^{m-1}) \leq c_2 4^{m-1} \) for some \( c_1, c_2 \).
• **IS:** $T(4^m) = 3T(4^{m-1}) + \Theta(4^m)$

  \[
  \leq 3c_2 4^{m-1} + c'_2 4^m \quad \text{for some constant } c'_2
  \]

  \[
  = (3c_2/4 + c'_2) 4^m
  \]

  \[
  \leq c_2 4^m \quad \text{if } c'_2 \leq c_2/4
  \]

\[
T(4^m) = 3T(4^{m-1}) + \Theta(4^m)
\]

  \[
  \geq 3c_1 4^{m-1} + c'_1 4^m \quad \text{for some constant } c'_1
  \]

  \[
  = (3c_1/4 + c'_1) 4^m
  \]

  \[
  \geq c_1 4^m \quad \text{if } c'_1 \geq c_1/4
  \]

• Let $c_1, c_2$ be such that $c_1 \leq c \leq c_2$, $c'_2 \leq c_2/4$, $c_1/4 \leq c'_1$. Then, $c_1 4^m \leq T(4^m) \leq c_2 4^m$ for all $m \geq 0$. 
The Master Theorem

- Definition: $f(n)$ is polynomially smaller than $g(n)$, denoted as $f(n) \ll g(n)$, iff $f(n) = O(g(n)n^{-\varepsilon})$, or $f(n)n^\varepsilon = O(g(n))$, for some $\varepsilon > 0$.

- For example, $1 \ll \sqrt{n} \ll n^{0.99} \ll n \ll n^2$.

- Is $1 \ll \log n$? Or $n \ll n \log n$?

- To answer these, ask yourself whether or not $n^\varepsilon = O(\log n)$.

- For convenience, write $f(n) \approx g(n)$ iff $f(n) = \Theta(g(n))$.

- Note: the notations $\ll$ and $\approx$ are good only for this class.
The Master Theorem

If $T(n)$ satisfies the recurrence $T(n) = aT(n/b) + f(n)$, then $T(n)$ is bounded asymptotically as follows.

1. If $f(n) \ll n^{\log_b a}$, then $T(n) = \Theta(n^{\log_b a})$.
2. If $f(n) \gg n^{\log_b a}$, then $T(n) = \Theta(f(n))$.
3. If $f(n) \approx n^{\log_b a}$, then $T(n) = \Theta(f(n)\log n)$.
4. If $f(n) \approx n^{\log_b a}\log^k n$, then $T(n) = \Theta(f(n)\log n)$.

In case 2, it is required that $af(n/b) \leq cf(n)$ for some $c < 1$, which is satisfied by most $f(n)$ that we shall encounter.

In the theorem, $n/b$ should be interpreted as $\lfloor n/b \rfloor$ or $\lceil n/b \rceil$. 
Examples: solve these recurrences

- \( T(n) = 3T(n/4) + n. \)
- \( T(n) = 9T(n/3) + n. \)
- \( T(n) = T(2n/3) + 1. \)
- \( T(n) = 3T(n/4) + n \log n. \)
- \( T(n) = 7T(n/2) + \Theta(n^2). \)
- \( T(n) = 2T(n/2) + n \log n. \)
- \( T(n) = T(n/3) + T(2n/3) + n. \)
\[ T(n) = aT(n / b) + f(n) \]

solving problems

1 of size \( n \)
\[ \downarrow \uparrow \]
\( a \) of size \( n / b \)
\[ \downarrow \uparrow \]
\( a^2 \) of size \( n / b^2 \)
\[ \downarrow \uparrow \]
\[ \vdots \]
\( a^{\log_b n - 1} \) of size \( n / b^{\log_b n - 1} \)
\[ \downarrow \uparrow \]
\( a^{\log_b n} \) of size \( n / b^{\log_b n} \)

\[ a^{\log_b n - 1} \cdot f(n / b^{\log_b n - 1}) \]
\[ a^{\log_b n} \cdot \Theta(1) \]

time needed

\[ f(n) \]
\[ a \cdot f(n / b) \]
\[ a^2 \cdot f(n / b^2) \]
\[ \vdots \]
\[ T(n) = aT(n/b) + f(n) \]

1 of size \( n \)

\[ \downarrow \uparrow \]

\( a \) of size \( n/b \)

\[ \downarrow \uparrow \]

\( a^2 \) of size \( n/b^2 \)

\[ \downarrow \uparrow \]

\[ \vdots \]

\( a^{\log_b n} \) of size \( n/b^{\log_b n} \)

\[ \downarrow \uparrow \]

\( a^{\log_b n} \) of size \( n/b^{\log_b n} \)

\[ \downarrow \uparrow \]

\( a^{\log_b n} \) of size \( n/b^{\log_b n} \)

\[ \vdots \]

\[ a^{\log_b n} \cdot f(n/b^{\log_b n-1}) \]

\[ a^{\log_b n} \cdot \Theta(1) \]

\[ T(n) = \sum_{i=0}^{\log_b n-1} a^i f\left(\frac{n}{b^i}\right) + n^{\log_b a} \]

(Note: \( \log_b n = \frac{\log_a n}{\log_a b} = \log_a n \cdot \log_b a \))
Suppose $f(n) = \Theta\left(n^{\log_b a}\right)$.

Then $f\left(\frac{n}{b^i}\right) = \Theta\left(\left(\frac{n}{b^i}\right)^{\log_b a}\right) = \Theta\left(\frac{n^{\log_b a}}{b^{i\log_b a}}\right) = \Theta\left(\frac{n^{\log_b a}}{a^i}\right)$,

and thus $a^i f\left(\frac{n}{b^i}\right) = \Theta\left(n^{\log_b a}\right)$. Then, we have

$$T(n) = \sum_{i=0}^{\log_b n-1} a^i f\left(\frac{n}{b^i}\right) + n^{\log_b a} \quad \text{(from the previous slide)}$$

$$= \Theta\left(\sum_{i=0}^{\log_b n-1} n^{\log_b a} + n^{\log_b a}\right) = \Theta\left(n^{\log_b a} \log n\right) = \Theta\left(f(n) \log n\right)$$
When recurrences involve roots

- Solve \( T(n) = \begin{cases} 2T\left(\sqrt{n}\right) + \log n & \text{if } n > 2 \\ c & \text{otherwise} \end{cases} \)

- Suffices to consider only powers of 2. Let \( n = 2^m \).

- Define a new function \( S(m) = T(2^m) = T(n) \).

- The above recurrence translates to

\[
S(m) = \begin{cases} 2S\left(\frac{m}{2}\right) + m & \text{if } m > 1 \\ c & \text{otherwise} \end{cases}
\]

- By Master Theorem, \( S(m) = \Theta(m \log m) \).

- So, \( T(n) = \Theta(\log n \log \log n) \)
Strassen's Algorithm for Matrix Multiplication

- Problem: Compute $C = AB$, given $n \times n$ matrices $A$ and $B$.
- The straightforward method requires $\Theta(n^3)$ time, using the formula $c_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj}$.
- Toward the end of the 1960s, Strassen showed how to multiply matrices in $O(n^{\log_2 7}) = O(n^{2.81})$ time.
- For $n = 100$, $n^{2.81} \approx 416,869$, and $n^3 = 1,000,000$.
- The time complexity was reduced to $O(n^{2.521813})$ in 1979, to $O(n^{2.521801})$ in 1980, and to $O(n^{2.376})$ in 1986.
- In the following discussion, $n$ is assumed to be a power of 2.
• Write

\[
A = \begin{pmatrix}
A_{11} & A_{12} \\
A_{21} & A_{22}
\end{pmatrix}, \quad
B = \begin{pmatrix}
B_{11} & B_{12} \\
B_{21} & B_{22}
\end{pmatrix}, \quad
C = \begin{pmatrix}
C_{11} & C_{12} \\
C_{21} & C_{22}
\end{pmatrix}
\]

where each \( A_{ij}, B_{ij}, C_{ij} \) is a \( n / 2 \times n / 2 \) matrix.

• Then

\[
\begin{pmatrix}
C_{11} & C_{12} \\
C_{21} & C_{22}
\end{pmatrix} = \begin{pmatrix}
A_{11} & A_{12} \\
A_{21} & A_{22}
\end{pmatrix} \begin{pmatrix}
B_{11} & B_{12} \\
B_{21} & B_{22}
\end{pmatrix}.
\]

• If we compute \( C_{ij} = A_{i1}B_{1j} + A_{i2}B_{2j} \), the running time \( T(n) \) will satisfy the recurrence \( T(n) = 8T(n / 2) + \Theta(n^2) \).

• \( T(n) \) will be \( \Theta(n^3) \), not better than the straightforward one.

• Good for parallel processing. What's the running time using \( \Theta(n^3) \) processors?
Strassen showed

\[
\begin{pmatrix}
C_{11} & C_{12} \\
C_{21} & C_{22}
\end{pmatrix} = \begin{pmatrix}
M_2 + M_3 & M_1 + M_2 + M_5 + M_6 \\
M_1 + M_2 + M_4 - M_7 & M_1 + M_2 + M_4 + M_5
\end{pmatrix}
\]

where

\[
M_1 = (A_{21} + A_{22} - A_{11}) \times (B_{22} - B_{12} + B_{11})
\]

\[
M_2 = A_{11} \times B_{11}
\]

\[
M_3 = A_{12} \times B_{21}
\]

\[
M_4 = (A_{11} - A_{21}) \times (B_{22} - B_{12})
\]

\[
M_5 = (A_{21} + A_{22}) \times (B_{12} - B_{11})
\]

\[
M_6 = (A_{12} - A_{21} + A_{11} - A_{22}) \times B_{22}
\]

\[
M_7 = A_{22} \times (B_{11} + B_{22} - B_{12} - B_{21})
\]

\[
T(n) = 7T(n / 2) + \Theta(n^2) \implies T(n) = \Theta(n^{\log_2 7})
\]
The Closest Pair Problem

- Problem Statement: Given a set of \( n \) points in the plane, \( A = \{(x_i, y_i) : 1 \leq i \leq n\} \), find two points in \( A \) whose distance is smallest among all pairs.

- Straightforward method: \( \Theta(n^2) \).

- Divide and conquer: \( O(n \log n) \).
The Divide-and-Conquer Approach

1. Partition $A$ into two sets: $A = B \cup C$.
2. Find a closest pair $(p_1, q_1)$ in $B$.
3. Find a closest pair $(p_2, q_2)$ in $C$.
4. Let $\delta = \min\{\text{dist}(p_1, q_1), \text{dist}(p_2, q_2)\}$.
5. Find a closest pair $(p_3, q_3)$ between $B$ and $C$ with distance less than $\delta$, if such a pair exists.
6. Return the pair of the three which is closest.

- **Question**: What would be the running time?
- **Desired**: $T(n) = 2T(n/2) + O(n) \Rightarrow T(n) = O(n \log n)$. 
• Now let's see how to implement each step.

• Trivial: steps 2, 3, 4, 6.
Step 1: Partition $A$ into two sets: $A = B \cup C$.

- A natural choice is to draw a vertical line to divide the points into two groups.
- So, sort $A[1..n]$ by $x$-coordinate. (Do this only once.)
- Then, we can easily partition any set $A[i..j]$ by

$$A[i..j] = A[i..m] \cup A[m+1..j]$$

where $m = \lfloor (i + j)/2 \rfloor$. 
Step 5: Find a closest pair between $B$ and $C$ with distance less than $\delta$, if exists.

- We will write a procedure

\[
\text{Closest-Pair-Between-Two-Sets}(A[i..j], \text{ptr}, \delta, (p3, q3))
\]

which finds a closest pair between $A[i..m]$ and $A[m+1..j]$ with distance less than $\delta$, if exists.

- The running time of this procedure must be no more than $O(|A[i..j]|)$ in order for the final algorithm to be $O(n \log n)$. 
Data Structures

- Let the coordinates of the $n$ points be stored in $X[1..n]$ and $Y[1..n]$.
- For simplicity, let $A[i] = (X[i], Y[i])$.
- For convenience, introduce two dummy points: $A[0] = (-\infty, -\infty)$ and $A[n+1] = (\infty, \infty)$
- We will use these two points to indicate "no pair" or "no pair closer than $\delta$".
- Introduce an array $\text{Link}[1..n]$, initialized to all 0's.
Main Program

- Global variable: $A[0..n + 1]$
- Sort $A[1..n]$ such that $X[1] \leq X[2] \leq \cdots \leq X[n]$. That is, sort the given $n$ points by $x$-coordinate.
- Call Procedure Closest-Pair with appropriate parameters.
Procedure Closest-Pair\((A[i..j], (p, q))\)  //Version 1//

\{\*returns a closest pair \((p, q)\) in \(A[i..j]\)*\}

- If \(j - i = 0\): \((p, q) \leftarrow (0, n + 1)\);
- If \(j - i = 1\): \((p, q) \leftarrow (i, j)\);
- If \(j - i > 1\): \(m \leftarrow \lfloor (i + j)/2 \rfloor\)

  Closest-Pair\((A[i..m], (p_1, q_1))\)
  Closest-Pair\((A[m + 1..j], (p_2, q_2))\)

\(\text{ptr} \leftarrow \text{mergesort} A[i..j] \text{ by y-coordinate into a linked list}\)
\(\delta \leftarrow \min\{\text{dist}(p_1, q_1), \text{dist}(p_2, q_2)\}\)

Closest-Pair-Between-Two-Sets\((A[i..j], \text{ptr}, \delta, (p_3, q_3))\)
\((p, q) \leftarrow \text{closest of the three } (p_1, q_1), (p_2, q_2), (p_3, q_3)\)
**Time Complexity** of version 1

- Initial call: \( \text{Closest-Pair}(A[1..n], (p, q)) \).

- Assume \( \text{Closest-Pair-Between-Two-Sets} \) needs \( \Theta(n) \) time.

- Let \( T(n) \) denote the worst-case running time of \( \text{Closest-Pair}(A[1..n], (p, q)) \).

- Then, \( T(n) = 2T(n/2) + \Theta(n \log n) \).

- So, \( T(n) = \Theta(n \log^2 n) \).

- Not as good as desired.
How to reduce the time complexity to $O(n \log n)$?

- Suppose we use Mergesort to sort $A[i..j]$:
  
  $ptr \leftarrow$ Sort $A[i..j]$ by $y$-coordinate into a linked list

- Rewrite the procedure as version 2.

- We only have to sort the base cases and perform "merge."

- Here we take a free ride on Closest-Pair for dividing.

- That is, we combine Mergesort with Closest-Pair.
Procedure Closest-Pair($A[i..j]$, $(p, q)$)  //Version 2//

- If $j - i = 0$: $(p, q) \leftarrow (0, n + 1)$;
- If $j - i = 1$: $(p, q) \leftarrow (i, j)$;
- If $j - i > 1$: $m \leftarrow \lfloor (i + j)/2 \rfloor$
  
  Closest-Pair($A[i..m]$, $(p_1, q_1)$)

Closest-Pair($A[m+1..j]$, $(p_2, q_2)$)

$p_{1r} \leftarrow \text{Mergesort}(A[i..m])$

$p_{2r} \leftarrow \text{Mergesort}(A[m+1..j])$

$p_{\text{new}} \leftarrow \text{Merge}(p_{1r}, p_{2r})$

(the rest is the same as in version 1)
Procedure Closest-Pair\( (A[i..j], (p, q), ptr) \) //final version//

\{*mergesort \( A[i..j] \) by \( y \) and find a closest pair \( (p, q) \) in \( A[i..j] \)*\}

- if \( j - i = 0 \): \( (p, q) \leftarrow (0, n+1); \) \( ptr \leftarrow i \)
- if \( j - i = 1 \): \( (p, q) \leftarrow (i, j); \)
  - if \( Y[i] \leq Y[j] \) then \( \{ ptr \leftarrow i; \) \( Link[i] \leftarrow j \}\)
  - else \( \{ ptr \leftarrow j; \) \( Link[j] \leftarrow i \}\)
- if \( j - i > 1 \): \( m \leftarrow \left\lfloor (i + j)/2 \right\rfloor \)
  
  Closest-Pair\( (A[i..m], (p_1, q_1), ptr1) \)
  
  Closest-Pair\( (A[m+1..j], (p_2, q_2), ptr2) \)
  
  \( ptr \leftarrow \text{Merge}(ptr1, ptr2) \)

(the rest is the same as in version 1)
Time Complexity of the final version

- Initial call: Closest-Pair\( (A[1..n], (p, q), pqr) \).
- Assume Closest-Pair-Between-Two-Sets needs \( \Theta(n) \) time.
- Let \( T(n) \) denote the worst-case running time of Closest-Pair\( (A[1..n], (p, q), pqr) \).
- Then, \( T(n) = 2T(n / 2) + \Theta(n) \).
- So, \( T(n) = \Theta(n \log n) \).
- Now, it remains to write the procedure Closest-Pair-Between-Two-Sets\( (A[i..j], ptr, \delta, (p_3, q_3)) \)
Closest-Pair-Between-Two-Sets

- **Input:** \((A[i..j], \text{ptr}, \delta)\)
- **Output:** a closest pair \((p, q)\) between \(B = A[i..m]\) and \(C = A[m+1..j]\) with distance < \(\delta\), where \(m = \lfloor (i + j) / 2 \rfloor\). If there is no such a pair, return the dummy pair \((0, n + 1)\).
- **Time complexity desired:** \(O(|A[i..j]|)\).
- For each point \(b \in B\), we will compute \(\text{dist}(b, c)\) for \(O(1)\) points \(c \in C\). Similarly for each point \(c \in C\).
- Recall that \(A[i..j]\) has been sorted by \(y\). We will follow the sorted linked list and look at each point.
Closest-Pair-Between-Two-Sets

- \( L_0 \): vertical line passing through the point \( A[m] \).
- \( L_1 \) and \( L_2 \): vertical lines to the left and right of \( L_0 \) by \( \delta \).
- We observe that:
  - We only need to consider those points between \( L_1 \) and \( L_2 \).
  - For each point \( k \) in \( A[i..m] \), we only need to consider the points in \( A[m+1..j] \) that are inside the square of \( \delta \times \delta \).
  - There are at most three such points.
  - And they are among the most recently visited three points of \( A[m+1..j] \) lying between \( L_0 \) and \( L_2 \).
  - Similar argument for each point \( k \) in \( A[m+1..j] \).
For $k$, only need to consider the points in the square less the right and bottom edges.

Red points are apart by at least $\delta$.

Blue points are apart by at least $\delta$. 
Closest-Pair-Between-Two-Sets($A[i..j], \text{ptr}, \delta, (p_3, q_3)$)

// Find the closest pair between $A[i..m]$ and $A[m + 1..j]$ with $\text{dist} < \delta$. If there exists no such a pair, then return the dummy pair $(0, n + 1)$. //

global $X[0..n+1], Y[0..n+1], Link[1..n]$

$(p_3, q_3) \leftarrow (0, n + 1)$

$b_1, b_2, b_3 \leftarrow 0$ //most recently visited 3 points btwn $L_0, L_1$ //

$c_1, c_2, c_3 \leftarrow n + 1$ //such points between $L_0, L_2$ //

$m \leftarrow \lfloor (i + j)/2 \rfloor$

$k \leftarrow \text{ptr}$
while $k \neq 0$ do //follow the linked list until end//

1. if $|X[k] - X[m]| < \delta$ then // consider only btwn $L_1, L_2$ //
   if $k \leq m$ then //point $k$ is to the left of $L_0$ //
      compute $d \leftarrow \min\{\text{dist}(k, c_i) : 1 \leq i \leq 3\}$;
      if $d < \delta$ then update $\delta$ and $(p_3, q_3)$;
      $b_3 \leftarrow b_2$; $b_2 \leftarrow b_1$; $b_1 \leftarrow k$;
   else //point $k$ is to the right of $L_0$ //
      compute $d \leftarrow \min\{\text{dist}(k, b_i) : 1 \leq i \leq 3\}$;
      if $d < \delta$ then update $\delta$ and $(p_3, q_3)$;
      $c_3 \leftarrow c_2$; $c_2 \leftarrow c_1$; $c_1 \leftarrow k$;

2. $k \leftarrow \text{Link}[k]$
Convex Hull

- Problem Statement: Given a set of $n$ points in the plane, say, $A = \{p_1, p_2, p_3, \ldots, p_n\}$, we want to find the convex hull of $A$.
- The convex hull of $A$, denoted by $CH(A)$, is the smallest convex polygon that encloses all points of $A$.
- Observation: segment $p_ip_j$ is an edge of $CH(A)$ if all other points of $A$ are on the same side of $p_ip_j$ (or on $p_ip_j$).
- Straightforward method: $\Omega(n^2)$.
- Divide and conquer: $O(n \log n)$. 
Divide-and-Conquer for Convex Hull

0. Assume all $x$-coordinates are different, and no three points are colinear. (Will be removed later.)
1. Let $A$ be sorted by $x$-coordinate.
2. If $|A| \leq 3$, solve the problem directly. Otherwise, apply divide-and-conquer as follows.
3. Break up $A$ into $A = B \cup C$.
4. Find the convex hull of $B$.
5. Find the convex hull of $C$.
6. Combine the two convex hulls by finding the upper and lower bridges to connect the two convex hulls.
Upper and Lower Bridges

• The upper bridge between \( CH(B) \) and \( CH(C) \) is the edge \( vw \), where \( v \in CH(B) \) and \( w \in CH(C) \), such that
  • all other vertices in \( CH(B) \) and \( CH(C) \) are below \( vw \), or
  • the two neighbors of \( v \) in \( CH(B) \) and the two neighbors of \( w \) in \( CH(C) \) are below \( vw \), or
  • the counterclockwise-neighbor of \( v \) in \( CH(B) \) and the clockwise-neighbor of \( w \) in \( CH(C) \) are below \( vw \), if \( v \) and \( w \) are chosen as in the next slide.

• Lower bridge: similar.
Finding the upper bridge

- $v \leftarrow$ the rightmost point in $CH(B)$;
- $w \leftarrow$ the leftmost point in $CH(C)$.
- Loop
  - if counterclockwise-neighbor($v$) lies above line $\overrightarrow{vw}$ then
    - $v \leftarrow$ counterclockwise-neighbor($v$)
  - else if clockwise-neighbor($w$) lies above $\overrightarrow{vw}$ then
    - $w \leftarrow$ clockwise neighbor($w$)
  - else exit from the loop
- $vw$ is the upper bridge.
Data Structure and Time Complexity

• What data structure will you use to represent a convex hull?

• Using your data structure, how much time will it take to find the upper and lower bridges?

• What is the overall running time of the algorithm?

• We assumed:
  (1) no two points in $A$ share the same $x$-coordinate
  (2) no three points in $A$ are colinear

• Now let's remove these assumptions.
Orientation of three points

- Three points: \( p_1(x_1, y_1), p_2(x_2, y_2), p_3(x_3, y_3) \).
- \((p_1, p_2, p_3)\) in that order is counterclockwise if
  \[
  \begin{vmatrix}
  x_1 & y_1 & 1 \\
  x_2 & y_2 & 1 \\
  x_3 & y_3 & 1
  \end{vmatrix} > 0
  \]
- Clockwise if the determinant is negative.
- Colinear if the determinant is zero.