Divide-and-Conquer
CIS 780, Autumn 2004, Lai

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

1 Introduction

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

function $DAQ(x)$
    if $x$ is sufficiently small or simple then
        solve it directly
    else
        divide $x$ into smaller subinstances $x_1, \ldots, x_k$;
        for $i \leftarrow 1$ to $k$ do $y_i \leftarrow DAQ(x_i)$;
        combine the $y_i$’s to obtain a solution $y$ for $x$;
        return($y$)
    endif

It is often that $x_1, \ldots, x_k$ are of the same size, say $\lfloor n/b \rfloor$. In that case, the time complexity of DAQ can be expressed as a recurrence:

$$T(n) = \begin{cases} 
    c, & \text{if } n \leq n_0 \\
    kT(\lfloor n/b \rfloor) + f(n), & \text{if } n > n_0 
\end{cases}$$
2 Merge Sort

Sort an array:

procedure mergesort(A[1..n], i, j)
    // Sort A[i..j] //
    if i ≥ j then return
    m ← (i + j) div 2
    mergesort(A, i, m)
    mergesort(A, m + 1, j)
    merge(A, i, m, j)
end

Sort a linked list

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)
    m ← (i + j) div 2
    ptr1 ← mergesort(i, m)
    ptr2 ← mergesort(m + 1, j)
    ptr ← merge(ptr1, ptr2)
    return(ptr)
end
3 Solving Recurrences

A function $T(n)$ satisfies the following recurrence:

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

where $c$ is a positive constant. Obtain a function $g(n)$ such that $T(n) = O(g(n))$.

3.1 Iteration Method and Recurrence Tree

Assume $n$ to be a power of 4, say, $n = 4^m$.

$$T(n) = n + 3T(n/4)$$
$$= n + 3[n/4 + 3T(n/16)]$$
$$= n + 3(n/4) + 9[(n/16) + 3T(n/64)]$$
$$= n + n(3/4) + n(3/4)^2 + 3^3T(n/4^3)$$
$$= n[1 + 3/4 + (3/4)^2 + (3/4)^3 + \ldots + (3/4)^{m-1}] + 3^mT(n/4^m)$$
$$= n[1 + 3/4 + (3/4)^2 + (3/4)^3 + \ldots + (3/4)^{m-1}] + c3^m$$
$$= n(1 - (3/4)^m)/(1 - 3/4) + c3^m$$
$$\leq 4n + c4^m$$
$$= 4n + cn$$
$$= O(n)$$

So, $T(n) = O(n | n \text{ a power of 4})$. It can be verified that $T(n)$ is asymptotically nondecreasing and $n$ is smooth. Thus, $T(n) = O(n)$. 


3.2 Guess and Prove

Solve

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

- First, we guess that \( T(n) = O(n) \).

- Then we prove the guess is right, by showing \( T(n) \leq bn \) for some constant \( b \).

- It suffices to consider \( n = 4^m, m = 1, 2, \ldots \).

  **Induction Base:** \( T(1) \leq b \) if \( b \) is chosen to be large enough.

  **Induction Hypothesis:** Assume \( T(4^k) \leq b4^k \) for \( k < m \).

  **Induction Step:** Show \( T(n) \leq bn \) for \( n = 4^m \) as follows.

\[
T(n) = n + 3T(n/4) \\
\leq n + 3bn/4 \\
\leq (1 + 3b/4)n \\
\leq bn, \text{ if } b \geq 4
\]

So, letting \( b \geq 4 + T(1) \), we have \( T(n) \leq bn \) for all \( n \). This establishes \( T(n) = O(n) \).
3.3 Master’s Theorem

**Theorem 1** If \( T(n) = aT(n/b) + f(n) \), then \( T(n) \) is bounded asymptotically as follows.

1. If \( f(n) = O(n^{\log_b a n^{-\epsilon}}) \), then \( T(n) = \Theta(n^{\log_b a}) \).
2. If \( f(n) = \Omega(n^{\log_b a n^{\epsilon}}) \), then \( T(n) = \Theta(f(n)) \).
3. If \( f(n) = \Theta(n^{\log_b a}) \), then \( T(n) = \Theta(n^{\log_b a \log n}) \).
4. If \( f(n) = \Theta(n^{\log_b a \log^k n}) \), then \( T(n) = \Theta(n^{\log_b a \log^{k+1} n}) \).

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

In the theorem, \( n/b \) should be interpreted as \( \lceil n/b \rceil \) or \( \lfloor n/b \rfloor \).

Applying the Master theorem to

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

we immediately obtain \( T(n) = \Theta(n) \), as \( n^{\log_b a} = n^{\log_4 3} < n \).

3.4 More Examples

- \( T(n) = 9T(n/3) + n. \)
- \( T(n) = T(2n/3) + 1 \)
- \( T(n) = 3T(n/4) + n \log n. \)
- \( T(n) = 7T(n/2) + \Theta(n^2). \)
- \( T(n) = 2T(n/2) + n \log n. \)
- \( T(n) = T(n/3) + T(2n/3) + n. \)
4 Strassen’s Algorithm for Matrix Multiplication

- **Problem**: Compute $C = AB$, where $A$ and $B$ are $n \times n$ matrices.

- The straightforward method requires $\Theta(n^3)$ time, using the formula $c_{ij} = \sum_{1 \leq k \leq n} a_{ik}b_{kj}$.

- Toward the end of 1960s, Strassen showed how to multiply matrices in $O(n^{\log 7}) = O(n^{2.81})$ time. (For $n = 100$, $n^{2.81} \approx 416869$.)

- 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, assume $n$ 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

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

• Multiplication of two \( 2 \times 2 \) matrices can be done using only 7 scalar multiplications as follows. Let

\[
\begin{align*}
M_1 &= (A_{21} + A_{22} - A_{11})(B_{22} - B_{12} + B_{11}) \\
M_2 &= A_{11}B_{11} \\
M_3 &= A_{12}B_{21} \\
M_4 &= (A_{11} - A_{21})(B_{22} - B_{12}) \\
M_5 &= (A_{21} + A_{22})(B_{12} - B_{11}) \\
M_6 &= (A_{12} - A_{21} + A_{11} - A_{22})B_{22} \\
M_7 &= A_{22}(B_{11} + B_{22} - B_{12} - B_{21})
\end{align*}
\]

Then

\[
C' = \begin{pmatrix} M_2 + M_3 & M_1 + M_2 + M_5 + M_6 \\ M_1 + M_2 + M_4 - M_7 & M_1 + M_2 + M_4 + M_5 \end{pmatrix}.
\]

• Time complexity:

\[ T(n) = 7T(n/2) + \Theta(n^2). \]

Solving the recurrence yields \( T(n) = \Theta(n^{\log_2 7}) = O(n^{2.81}). \)
5 The Closest Pair Problem

5.1 Problem Statement

Given a set of \( n \) points \( A = \{(x_i, y_i) : 1 \leq i \leq n\} \) in the plane, find two points in \( A \) with smallest distance.

5.2 Basic Idea

Straightforward method: \( O(n^2) \)

Divide and conquer: \( O(n \log n) \)

Basic idea:

1. Break up \( A \) into \( A = B \cup C' \).
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 \((p, q)\) such that \( \text{dist}(p, q) = \min\{\text{dist}(p_k, q_k) : 1 \leq k \leq 3\} \) is smallest.
5.3 Main Program

Assume that the coordinates of the \( n \) points are stored in \( X[1..n] \) and \( Y[1..n] \), with \((X[i], Y[i])\) the \( i \)th point. For simplicity, let \( A[i] = (X[i], Y[i]) \).

Assume that the points have been sorted such that \( X[1] \leq X[2] \leq \cdots \leq X[n] \).

**First Attempt**

**procedure** Closest-Pair\((A[i..j], (p, q))\)

// This procedure returns a closest pair \((p, q)\) in \( A[i..j] \).

**global** \( X[0..n+1], Y[0..n+1] \)

**case**

\( j - i = 0: \) \((p, q) \leftarrow (0, n + 1);\)

\( j - i = 1: \) \((p, q) \leftarrow (i, j);\)

\( j - i > 1: \) \( m \leftarrow (i + j) \text{ div } 2 \)

Closest-Pair\((A[i..m], (p_1, q_1))\)

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

\( \text{ptr1} \leftarrow \text{Mergesort}(A[i..m])\)

\( \text{ptr2} \leftarrow \text{Mergesort}(A[m + 1..j])\)

\( \text{ptr} \leftarrow \text{Merge} (\text{ptr1}, \text{ptr2})\)

\( \delta \leftarrow \min\{\text{dist}(p_1, q_1), \text{dist}(p_2, q_2)\}\)

Closest-Pair-Between-Two-Sets\((A[i..j], \text{ptr}, \delta, (p_3, q_3))\)

\((p, q) \leftarrow \text{select the pair } (p_k, q_k), 1 \leq k \leq 3 \text{ with shortest dis}\)

**endcase**
procedure Closest-Pair(A[i..j], (p, q), ptr)

// This procedure performs two functions.
First, it finds a closest pair (p, q) in A[i..j].
Second, it sorts A[i..j] on y-coordinate, and yields a linked list ptr.//
global X[0..n+1], Y[0..n+1], Link[1..n]
// Initially, X[0] ← Y[0] ← −∞, X[n + 1] ← Y[n + 1] ← ∞,
and Link[k] ← 0, 1 ≤ k ≤ n.//
case
    j − i = 0: (p, q) ← (0, n + 1); ptr ← i
    j − i = 1: (p, q) ← (i, j)
        if Y[i] ≤ Y[j] then {ptr ← i; Link[i] ← j}
        else {ptr ← j; Link[j] ← i}
    j − i > 1: m ← (i + j) div 2
        Closest-Pair(A[i..m], (p_1, q_1), ptr_1)
        Closest-Pair(A[m + 1..j], (p_2, q_2), ptr_2)
        Merge(ptr_1, ptr_2, ptr)
        δ ← min{dist(p_1, q_1), dist(p_2, q_2)}
        Closest-Pair-Between-Two-Sets(A[i..j], ptr, δ, (p_3, q_3))
        (p, q) ← select the pair (p_k, q_k), 1 ≤ k ≤ 3 with shortest dist
endcase
5.4 Subroutine

To find a closest pair between $A[i..m]$ and $A[m + 1..j]$ with distance $< \delta$, where $m = (i + j) \text{ div } 2$.

$L_0$: vertical line passing through the point $A[m]$.

$L_1$ and $L_2$: as shown.

We observe that:

1. We need to consider only points between $L_1$ and $L_2$.

2. For a point $r$ in $A[i..m]$, we need to consider only the points in $A[m + 1..j]$ that are inside the square of $\delta \times \delta$. There are at most three such points.

3. Similarly, for a point $r$ in $A[m + 1..j]$, we need to consider at most three points in $A[i..m]$.
procedure Closest-Pair-Between-Two-Sets(A[i..j], ptr, δ, (p₃, q₃))

// Find the closest pair between A[i..m] and A[m + 1..j] with dist. < δ.
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]
(p₃, q₃) ← (0, n + 1)
b₁ ← b₂ ← b₃ ← 0
c₁ ← c₂ ← c₃ ← n + 1
m ← (i + j) div 2
k ← ptr

while k ≠ 0 do
    if |X[k] − X[m]| < δ then
        case
            k ≤ m: compute d ← min{dist(k, cᵢ) : 1 ≤ i ≤ 3};
            if d < δ then update δ and (p₃, q₃);
            b₃ ← b₂; b₂ ← b₁; b₁ ← k
            k > m: compute d ← min{dist(k, bᵢ) : 1 ≤ i ≤ 3};
            if d < δ then update δ and (p₃, q₃);
            c₃ ← c₂; c₂ ← c₁; c₁ ← k
        endcase
    endif
    k ← Link[k]
endwhile
6 Convex Hull

6.1 Problem Statement

- Given a set $A$ of $n$ points in the plane, we want to find the convex hull of $A$.
- The convex hull of $A$ is the smallest convex polygon that contains all the points in $A$.

6.2 Sketch of Algorithm

- Let $A = \{p_1, p_2, \ldots, p_n\}$. Denote the convex hull of $A$ by $CH(A)$.
- An observation: the segment $p_i p_j$ is an edge of $CH(A)$ if all points of $A$ are on the same side of $p_i p_j$.
- Straightforward method: $\Omega(n^2)$
- Divide and conquer: $O(n \log n)$
- Basic ideas:
  1. Let $A$ be sorted by the $x$-coordinates of the points.
  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.
6.3 Combine $CH(B)$ and $CH(C)$ to get $CH(A)$

1. We need to find the “upper bridge” and the “lower bridge” that connect the two convex hulls.

2. Find the upper bridge as follows:
   
   (a) $v :=$ the rightmost point in $CH(B)$;  
       $w :=$ the leftmost point in $CH(C)$.
   
   (b) Loop
       
       if clockwise_neighbor($v$) lies above the line $vw$ then
       
       $v :=$ clockwise_neighbor($v$)
       
       else if clockwise_neighbor($w$) lies above the line $vw$ then
       
       $w :=$ clockwise_neighbor($w$)
       
       else
       
       exit from the loop
       
       End of loop
   
   (c) $vw$ is the upper bridge.

3. Find the lower bridge similarly.
7 Supplementary Problems

1. Why do we need to sort the given points by \( x \) at the beginning of the closest-pair algorithm?

2. Consider the closest-pair algorithm. Suppose we do not sort \( A[i..j] \) by \( y \)-coordinate in Closest-Pair\((A[i..j], (p, q), ptr)\), but instead we sort the \( n \) points \( A[1..n] \) by \( y \) into a linked list in the beginning of the algorithm, immediately after they are sorted by \( x \).

   (a) Does the modified algorithm work correctly?
   (b) What is its time complexity?

3. Let \( X[1..n] \) and \( Y[1..n] \) be two sets of integers, each sorted in nondecreasing order. Write a divide-and-conquer algorithm that finds the \( n \)th smallest of the \( 2n \) combined elements. Your algorithm must run in \( O(\log n) \) time. You may assume that all the \( 2n \) elements are distinct.

4. The time complexity of an algorithm satisfies the following recurrence equation:

\[
T(n) = \begin{cases} 
2T(\lfloor \sqrt{n} \rfloor) + 1 & \text{if } n > 2 \\
1 & \text{otherwise}
\end{cases}
\]

Solve the equation and express \( T(n) \) in terms of the \( \Theta \) notation. (You may ignore the “\( \lceil \cdot \rceil \)” in your analysis.)