Computer Algorithms

Lecture 6: Divide-and-Conquer – Ch 5 – Cont'd

Dr. Manal Helal, Spring 2014.

http://moodle.manalhelal.com

Lecture Learning Objectives

1. Use a Divide & Conquer algorithm design strategy to solve an appropriate problem such as tree traversals , multiplication, closest pair and/or convex-hull.

Divide-and-Conquer

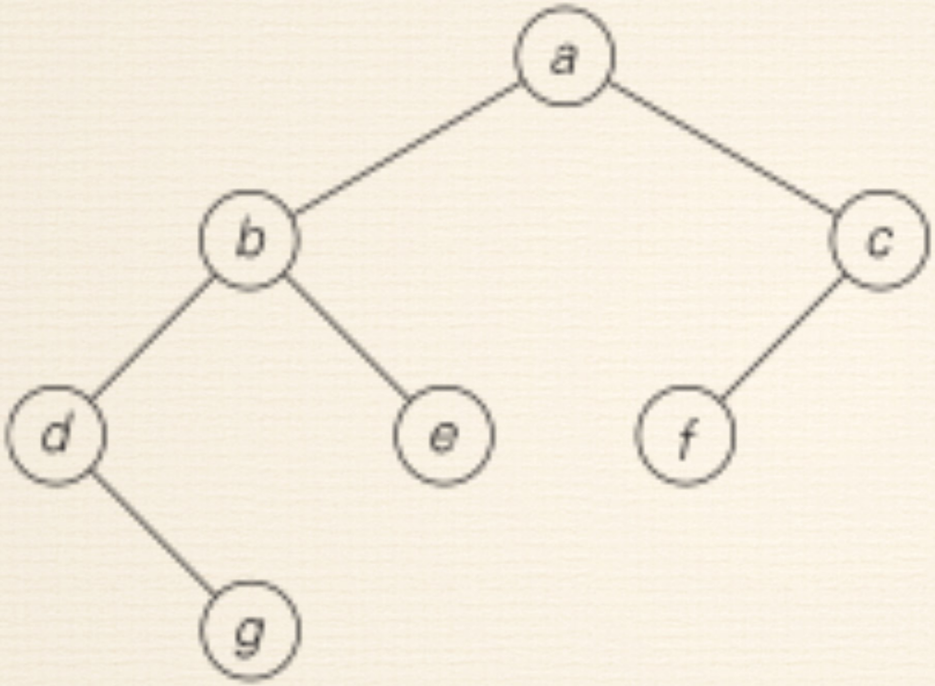
The most-well known algorithm design strategy:

1. Divide instance of problem into two or more smaller instances

2. Solve smaller instances recursively

3. Obtain solution to original (larger) instance by combining these solutions

Binary Tree Traversals



preorder: *a, b, d, g, e, c, f*
inorder: *d, g, b, e, a, f, c*
postorder: *g, d, e, b, f, c, a*

Binary Tree Algorithms

Binary tree is a divide-and-conquer ready structure!

Ex. 1: Classic traversals (preorder, inorder, postorder)

Algorithm *Inorder*(T)

if $T \neq \varnothing$

Inorder(T_{left})

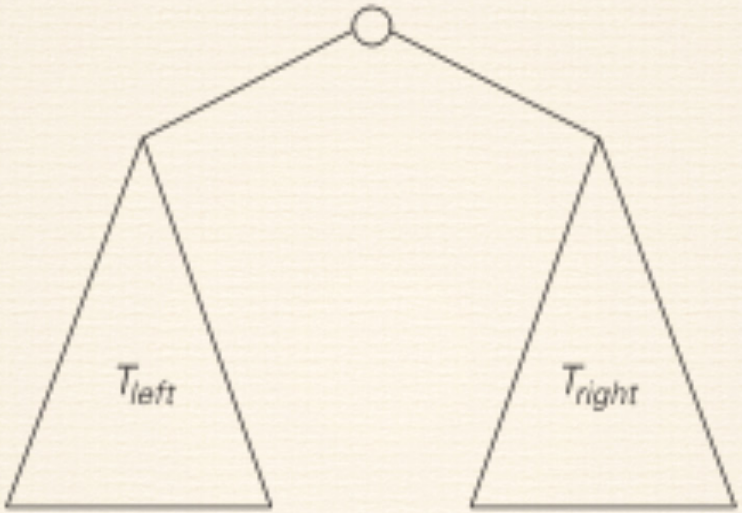
print(root of T)

Inorder(T_{right})

Efficiency: $\Theta(n)$

Binary Tree Algorithms (cont.)

Ex. 2: Computing the height of a binary tree



$h(T) = \max\{h(T_L), h(T_R)\} + 1$ if $T \neq \varnothing$ and $h(\varnothing) = -1$

Efficiency: $\Theta(n)$

Multiplication of Large Integers

Consider the problem of multiplying two (large) n-digit integers represented by arrays of their digits such as:

A = 12345678901357986429
B = 87654321284820912836
The grade-school algorithm:

$$
\begin{array}{r}
a_1 \; a_2 \ldots \; a_n \\
b_1 \; b_2 \ldots \; b_n \\
\hline
(d_{10})\, d_{11} d_{12} \ldots d_{1n} \\
(d_{20})\, d_{21} d_{22} \ldots d_{2n} \\
\ldots \ldots \ldots \ldots \ldots \ldots \ldots \\
(d_{n0})\, d_{n1} d_{n2} \ldots d_{nn} \\
\hline
\end{array}
$$

```
    1980 = a
x   2315 = b
----------
    9900
   1980
  5940
+ 3960
----------
4573700 = a x b
```

Efficiency: n^2 one-digit multiplications

First Divide-and-Conquer Algorithm

A small example: $A * B$ where $A = 23 = 2.10^1 + 3.10^0$ and $B = 14 = 1.10^1 + 4.10^0$.

$23 * 14 = (2.10^1 + 3.10^0) * (1.10^1 + 4.10^0)$

$= (2 * 1)10^2 + (2 * 4 + 3 * 1)10^1 + (3 * 4)10^0$.

A bigger example: $A * B$ where $A = 2135$ and $B = 4014$
$A = (21 \cdot 10^2 + 35)$, $B = (40 \cdot 10^2 + 14)$
So, $A * B = (21 \cdot 10^2 + 35) * (40 \cdot 10^2 + 14)$
$= 21 * 40 \cdot 10^4 + (21 * 14 + 35 * 40) \cdot 10^2 + 35 * 14$

In general, if $A = A_1A_2$ and $B = B_1B_2$ (where A and B are *n*-digit, A_1, A_2, B_1, B_2 are $n/2$-digit numbers),
$A * B = A_1 * B_1 \cdot 10^n + (A_1 * B_2 + A_2 * B_1) \cdot 10^{n/2} + A_2 * B_2$

Recurrence for the number of one-digit multiplications $M(n)$:
$$M(n) = 4M(n/2), \quad M(1) = 1$$
Solution: $M(n) = n^2$

First Divide-and-Conquer Pseudo-code

Algorithm *Divide-Mult(a,b):*

if *a or b has one digit,* ***then:***

  *return a * b.*

else:

  Let n be the number of digits in max{a, b}.

  Let a_L *and* a_R *be left and right halves of a.*

  Let b_L *and* b_R *be left and right halves of b.*

  Let x_1 *hold Divide-Mult(*a_L, b_L*).*

  Let x_2 *hold Divide-Mult(*a_L, b_R*).*

  Let x_3 *hold Divide-Mult(*a_R, b_L*).*

  Let x_4 *hold Divide-Mult(*a_R, b_R*).*

  return $x_1 * 10^n + (x_2 + x_3) * 10^{n/2} + x_4$.

end of if

```
aL = 19 | 80 = aR
bL = 23 | 15 = bR

          aL          aR
x         bL          bR
--------------------------------
          aL bR       aR bR
+ aL bL   aR bL
--------------------------------
  aL bL   aL bR + aR bL   aR bR
```

Second Divide-and-Conquer Algorithm

$A * B = A_1 * B_1 \cdot 10^n + (A_1 * B_2 + A_2 * B_1) \cdot 10^{n/2} + A_2 * B_2$

The idea is to decrease the number of multiplications from 4 to 3:

$(A_1 + A_2) * (B_1 + B_2) = A_1 * B_1 + (A_1 * B_2 + A_2 * B_1) + A_2 * B_2,$

I.e., $(A_1 * B_2 + A_2 * B_1) = (A_1 + A_2) * (B_1 + B_2) - A_1 * B_1 - A_2 * B_2,$ which requires only 3 multiplications at the expense of (4-1) extra add/sub.

Recurrence for the number of multiplica

$$M(n) = 3M(n/2), \quad M(1) =$$

Solution: $M(n) = 3^{\log 2^n} = n^{\log 2^3} \approx n^{1.585}$

```
    x1 = aL bL
    x2 = aR bR
    x3 = (aL + aR) (bL + bR)

                    aL              aR
  x                 bL              bR
------------------------------------
aL bL       aL bR + aR bL       aR bR
  x1        x3 - x1 - x2          x2
```

Second Divide-and-Conquer Pseudo-code

```
Algorithm Karatsuba(a,b):
if a or b has one digit, then:
    return a * b.
else:
    Let n be the number of digits in max{a, b}.
    Let a_L and a_R be left and right halves of a.
    Let b_L and b_R be left and right halves of b.
    Let x_1 hold Karatsuba(a_L, b_L).
    Let x_2 hold Karatsuba(a_L + a_R, b_L + b_R).
    Let x_3 hold Karatsuba(a_R, b_R).
    return x_1*10^n + (x_2 - x_1 - x_3)*10^(n/2) + x_3.
end of if
```

Exercise

2135 * 4014

Strassen's Matrix Multiplication

Strassen observed [1969] that the product of two matrices can be computed as follows

$$\begin{bmatrix} c_{00} & c_{01} \\ c_{10} & c_{11} \end{bmatrix} = \begin{bmatrix} a_{00} & a_{01} \\ a_{10} & a_{11} \end{bmatrix} * \begin{bmatrix} b_{00} & b_{01} \\ b_{10} & b_{11} \end{bmatrix}$$

$$= \begin{bmatrix} m_1 + m_4 - m_5 + m_7 & m_3 + m_5 \\ m_2 + m_4 & m_1 + m_3 - m_2 + m_6 \end{bmatrix}$$

Formulas for Strassen's Algorithm

$$m_1 = (a_{00} + a_{11}) * (b_{00} + b_{11}),$$
$$m_2 = (a_{10} + a_{11}) * b_{00},$$
$$m_3 = a_{00} * (b_{01} - b_{11}),$$
$$m_4 = a_{11} * (b_{10} - b_{00}),$$
$$m_5 = (a_{00} + a_{01}) * b_{11},$$
$$m_6 = (a_{10} - a_{00}) * (b_{00} + b_{01}),$$
$$m_7 = (a_{01} - a_{11}) * (b_{10} + b_{11}).$$

Analysis of Strassen's Algorithm

If n is not a power of 2, matrices can be padded with zeroes.

Number of multiplications:

$$M(n) = 7M(n/2), \quad M(1) = 1$$

Solution: Since $n = 2^k$,

$M(2^k) = 7M(2^{k-1}) = 7[7M(2^{k-2})] = 7^2M(2^{k-2}) = \ldots$

$= 7^iM(2^{k-i}) \ldots = 7^kM(2^{k-k}) = 7^k$.

Since $k = \log_2 n$,

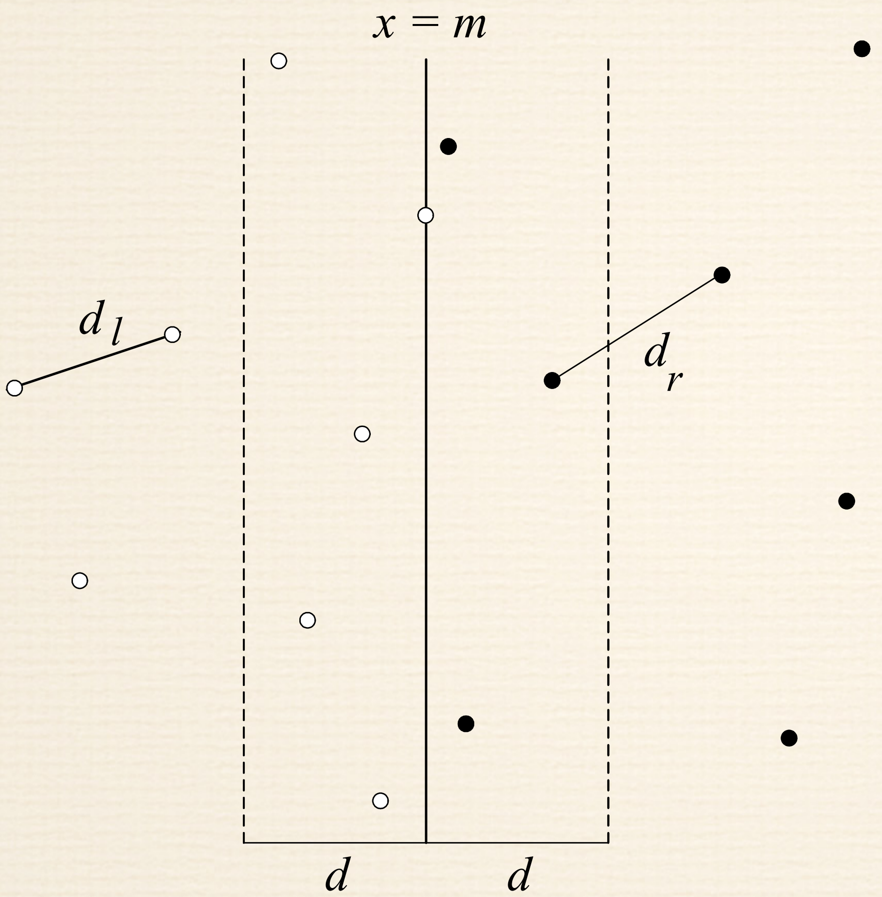
$M(n) = 7^{\log_2 n} = n^{\log_2 7} \approx n^{2.807}$, vs. n^3 of brute-force algorithm.

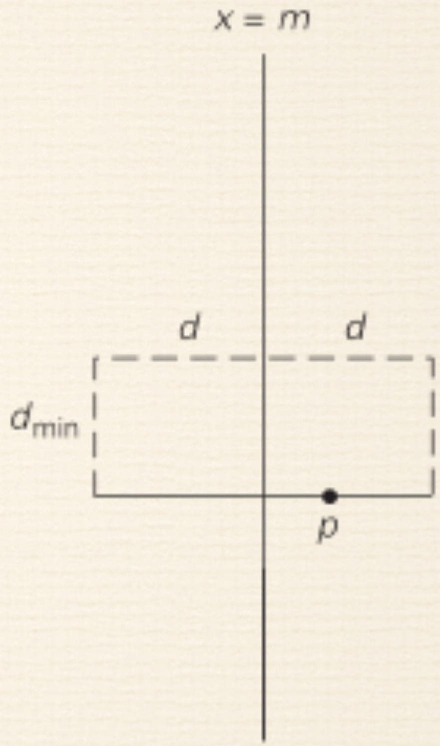
Algorithms with better asymptotic efficiency are known but they are even more complex.

Closest-Pair Problem by Divide-and-Conquer

1. Divide the set into two equal sized parts by the line *l*, and recursively compute the minimal distance in each part.
2. Let *d* be the minimal of the two minimal distances**:** **O(*1*)**
3. Eliminate points that lie farther than *d* apart from *l***:** **O(*n*)**
4. Sort the remaining points according to their *y*-coordinates**:** **O(*n log n*)**
5. Scan the remaining points in the *y* order and compute the distances of each point to its five neighbours(why?)**:** **O(*n*)**
6. If any of these distances is less than *d* then update *d***:** **O(*1*)**

Closest Pair by Divide-and-Conquer (cont.)





```
ALGORITHM  EfficientClosestPair(P, Q)
    //Solves the closest-pair problem by divide-and-conquer
    //Input: An array P of n ≥ 2 points in the Cartesian plane sorted in
    //       nondecreasing order of their x coordinates and an array Q of the
    //       same points sorted in nondecreasing order of the y coordinates
    //Output: Euclidean distance between the closest pair of points
    if n ≤ 3
        return the minimal distance found by the brute-force algorithm
    else
        copy the first ⌈n/2⌉ points of P to array P_l
        copy the same ⌈n/2⌉ points from Q to array Q_l
        copy the remaining ⌊n/2⌋ points of P to array P_r
        copy the same ⌊n/2⌋ points from Q to array Q_r
        d_l ← EfficientClosestPair(P_l, Q_l)
        d_r ← EfficientClosestPair(P_r, Q_r)
        d ← min{d_l, d_r}
        m ← P[⌈n/2⌉ − 1].x
        copy all the points of Q for which |x − m| < d into array S[0..num − 1]
        dminsq ← d^2
        for i ← 0 to num − 2 do
            k ← i + 1
            while k ≤ num − 1 and (S[k].y − S[i].y)^2 < dminsq
                dminsq ← min((S[k].x − S[i].x)^2 + (S[k].y − S[i].y)^2, dminsq)
                k ← k + 1
    return sqrt(dminsq)
```

Efficiency of the Closest-Pair Algorithm

Running time of the algorithm is described by

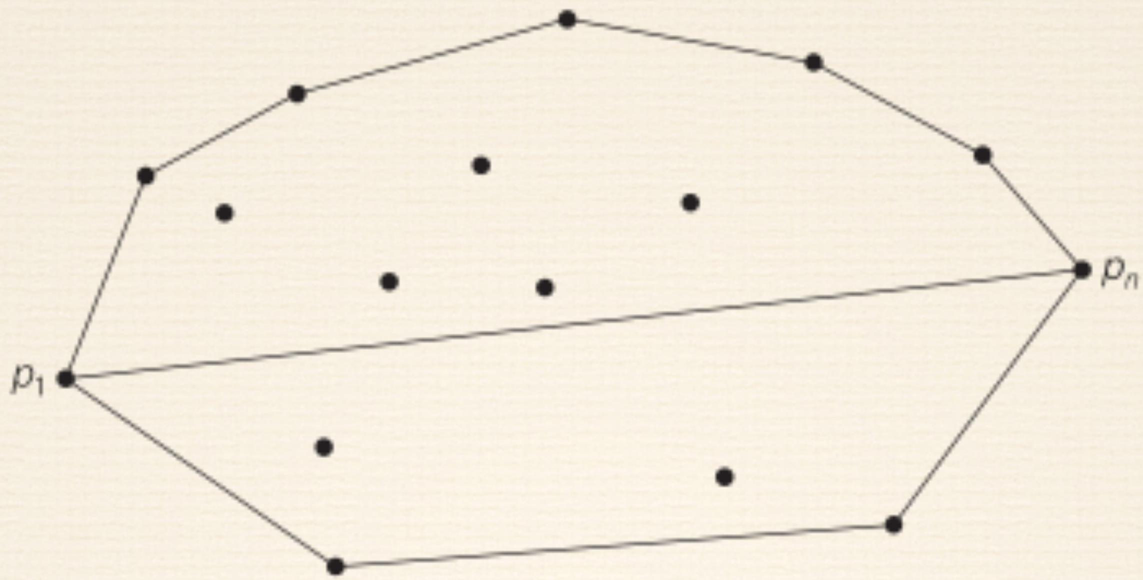
$$T(n) = 2T(n/2) + M(n), \text{ where } M(n) \in O(n)$$

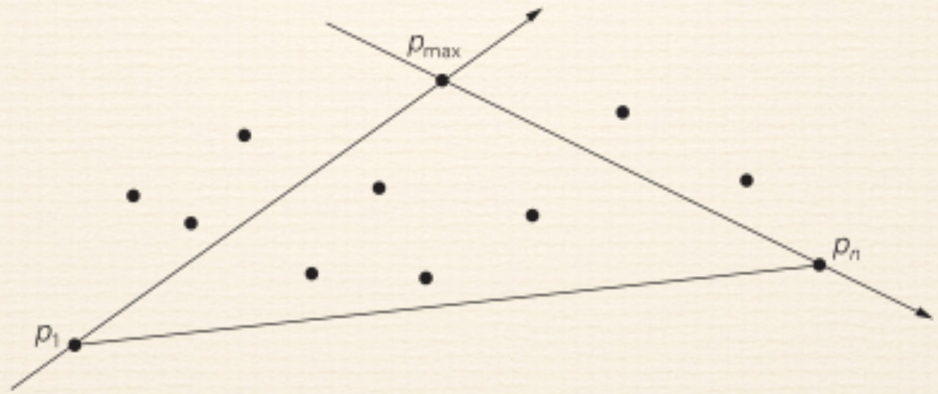
By the Master Theorem (with $a = 2$, $b = 2$, $d = 1$)

$$T(n) \in O(n \log n)$$

Quickhull Algorithm

- *Convex hull*: smallest convex set that includes given points
- Assume points are sorted by *x*-coordinate values
- Identify *extreme points* P_1 and P_2 (leftmost and rightmost)
- Compute *upper hull* recursively:
  - find point P_{max} that is farthest away from line P_1P_2
  - compute the upper hull of the points to the left of line P_1P_{max}
  - compute the upper hull of the points to the left of line $P_{max}P_2$
- Compute *lower hull* in a similar manner


p_n
p_1


p_{max}
p_n
p_1

Efficiency of Quickhull Algorithm

Finding point farthest away from line P_1P_2 can be done in linear time

Time efficiency:

worst case: $\Theta(n^2)$ (as quicksort)

average case: $\Theta(n)$ (under reasonable assumptions about distribution of points given)

If points are not initially sorted by x-coordinate value, this can be accomplished in $O(n \log n)$ time

Several $O(n \log n)$ algorithms for convex hull are known