CS 255, Spring 2014, SJSU

Divide and Conquer

Fernando Lobo

Divide and Conquer

- A technique for solving problems (we'll study other techniques later).
- Consists of 3 steps:
  - Divide a problem in one or more subproblems.
  - Conquer (solve) recursively each subproblem.
  - Combine the results of the subproblems to obtain the solution for the original problem.

Divide and Conquer

Running time is almost always given by:

$$T(n) = D(n) + aT(n/b) + C(n)$$

- $D(n)$ is the time spent in the divide step.
- a is the number of subproblems.
- n/b is the size of each subproblem.
- $C(n)$ is the time spent in the combine step.

Divide and Conquer

- We usually put together $D(n) + C(n)$ and call it $f(n)$, the time spent on non-recursive work.
- Gives $T(n) = aT(n/b) + f(n)$, ready for applying the Master Method.

Example 1: MergeSort

$$\begin{aligned} T(n) &= \Theta(1) + 2T(n/2) + \Theta(n) \\ &= 2T(n/2) + \Theta(n) \\ &= \Theta(n \lg n) \end{aligned}$$

Example 2: Binary Search

- Divide step: look at the element in the mid position of the array.
- Conquer step: solve recursively for one of the subarrays (left or right)
- Combine step: do nothing.
- Recurrence: $T(n) = T(n/2) + \Theta(1)$.
- Using the Master Method, $a = 1$, $b = 2$, $f(n) = c$ (with c constant). $n^{\log_b a} = n^{\log_2 1} = n^0 = 1 = \Theta(f(n))$. This is case 2 of the Master Method. The complexity is $T(n) = \Theta(\lg n)$

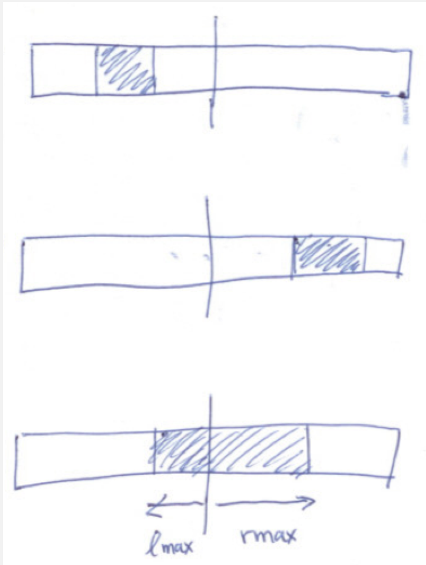
Example 3: Maximum subarray problem

- Given an $A[1..n]$ of real numbers, find the maximum sum that can be obtained by summing over the elements of subarray $A[i..j]$, with $1 \leq i \leq j \leq n$.
- Sample input: $A = [-2, 11, -4, 13, -5, 2]$. Ouput: 20, corresponding to the sum of the elements in $A[2..4]$.
- Brute-force solution: Compute the sum for $A[i..j]$ and keep track of the best so far.
- Running time: $\Theta(n^3)$. With a little optimization can get to $\Theta(n^2)$.
- Can we do better?

Example 3: D&C solution

- Divide the array in the middle.
- One of three cases can occur:
  1. the best subarray is entirely on the left subarray.
  2. the best subarray is entirely on the right subarray.
  3. the best subarray starts in the left and ends in the right subarray.
- Solution is the best of the three cases.
- Cases 1 and 2 can be computed recursively (conquer step). Case 3 can be computed in linear time.

3 cases



Example 3: pseudocode

```
MAX-SUBARRAY(A, left, right)
1  if left == right
2      return A[left]
3  else mid = (left + right)/2
4      maxLeft = MAX-SUBARRAY(A, left, mid)
5      maxRight = MAX-SUBARRAY(A, mid + 1, right)
6      lmax = sum = A[mid]
7      for i = mid − 1 downto left
8          sum = sum + A[i]
9          lmax = max(lmax, sum)
10     rmax = sum = A[mid + 1]
11     for i = mid + 2 to right
12         sum = sum + A[i]
13         rmax = max(rmax, sum)
14     maxLeftRight = lmax + rmax
15     return max3(maxLeft, maxRight, maxLeftRight)
```


Example 3: running time with D&C

- Lines 6–14 compute Case 3 in $\Theta(n)$ time.
- Running time:

$$\begin{aligned} T(n) &= 2T(n/2) + \Theta(n) \\ &= \Theta(n \lg n) \end{aligned}$$

- We were able to reduce from $\Theta(n^2)$ to $\Theta(n \lg n)$

Example 4: Integer multiplication

- Obvious algorithms: primary school method.
- Example:

```
     23
   * 12
 ------
     46
   + 23
 ------
    276
```

Example 4: Integer multiplication

- Works with any base. Let $x = 1100$ and $y = 1101$, both in base 2. $xy = ?$

```
     1100
   * 1101
 --------
     1100
    0000
   1100
+ 1100
---------
 10011100
```

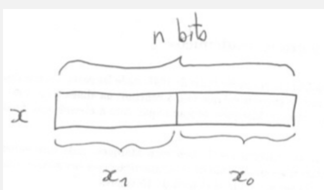

Example 4: Integer multiplication

- Let's assume both numbers have n digits ($n = 4$ in the previous example.)
- Running time of algorithm as a function of n: $\Theta(n^2)$.
- Can we do better?
- Yes, with divide and conquer.

Example 4: Divide and Conquer (1st attempt)

Idea:

- Divide x into two parts, with x_1 digits in the left part (the most significant ones) and x_0 digits in the right part (the least significant ones). Do the same thing for y.



- $x = x_1 2^{n/2} + x_0$. Likewise, $y = y_1 2^{n/2} + y_0$.

$$
\begin{aligned}
xy &= (x_1 2^{n/2} + x_0)(y_1 2^{n/2} + y_0) \\
&= x_1 y_1 2^n + (x_1 y_0 + x_0 y_1) 2^{n/2} + x_0 y_0
\end{aligned}
$$

Example 4: Divide and Conquer (1st attempt)

- The computation of 2^n and $2^{n/2}$ do not require multiplications. Can be implemented with bit shifts.
- Multiplication of 2 n-bit numbers can be done with 4 multiplications of smaller sized ($n/2$) numbers + a constant number of additions of n-bit numbers.
- Recurrence: $T(n) = 4T(n/2) + \Theta(n)$.
- Using the Master Method: $a = 4$, $b = 2$.
  $n^{\log_b a} = n^{\log_2 4} = n^2$
- Case 1: $T(n) = \Theta(n^2)$

Example 4: Divide and Conquer (1st attempt)

- Summary: D&C algorithm is not better than the naive algorithm. It's more complicated and has the same time complexity!

Example 4: Divide and Conquer (2nd attempt)

- Can we make it with 3 recursive calls?
- If yes we would obtain the following recurrence:

$$T(n) = 3T(n/2) + \Theta(n) = \Theta(n^{\log_2 3}) \approx \Theta(n^{1.59})$$

- It's indeed possible with a little trick.

Example 4: Divide and Conquer (2nd attempt)

- Want to obtain:
  $xy = x_1y_12^n + (x_1y_0 + x_0y_1)2^{n/2} + x_0y_0$
- Trick:
  $(x_1 + x_0)(y_1 + y_0) = x_1y_1 + x_1y_0 + x_0y_1 + x_0y_0$
- The multiplication has the 4 multiplications that we want.
- If we compute x_1y_1 e x_0y_0 recursively, we can obtain xy with 3 recursive calls:
  - $(x_1 + x_0)(y_1 + y_0)$ // mult1
  - x_1y_1 // mult2
  - x_0y_0 // mult3

  $xy = x_1y_12^n + (mult1 - x_1y_1 - x_0y_0)2^{n/2} + x_0y_0$

Example 4: Pseudocode

RECURSIVE-MULTIPLY(x, y)
  Divide x in the mid position and obtain x_1 and x_0
  Dividir y in the mid position and obtain y_1 and y_0
  mult1 = RECURSIVE-MULTIPLY$(x_1 + x_0, y_1 + y_0)$
  mult2 = RECURSIVE-MULTIPLY(x_1, y_1)
  mult3 = RECURSIVE-MULTIPLY(x_0, y_0)
  return $mult2 * 2^n + (mult1 - mult2 - mult3) * 2^{n/2} + mult3$

Example 5: Matrix multiplication

$$\begin{bmatrix} A_{1,1} & A_{1,2} & \cdots & A_{1,n} \\ A_{2,1} & A_{2,2} & \cdots & A_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ A_{n,1} & A_{n,2} & \cdots & A_{n,n} \end{bmatrix} \cdot \begin{bmatrix} B_{1,1} & B_{1,2} & \cdots & B_{1,n} \\ B_{2,1} & B_{2,2} & \cdots & B_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ B_{n,1} & B_{n,2} & \cdots & B_{n,n} \end{bmatrix}$$

- Input: $A_{i,j}$, $B_{i,j}$ with $i, j = 1, 2, \ldots, n$.
- Output: $C_{i,j} = A \cdot B = \sum_{k=1}^{n} A_{i,k} \cdot B_{k,j}$
- $C_{i,j}$ is given by the product of line i by column j.

Example 5: Matrix multiplication

Standard algorithm: 3 loops that iterate through $1 \,.\,.\, n$.

```
MATRIX-MULTIPLY(A, B, C)
    for i = 1 to n
        for j = 1 to n
            C[i, j] = 0
            for k = 1 to n
                C[i, j] = C[i, j] + A[i, k] * B[k, j]
```

Complexity: $\Theta(n^3)$.

Example 5: Divide and Conquer (1st attempt)

- Idea: divide a $n \times n$ matrix into 4 $\frac{n}{2} \times \frac{n}{2}$ matrices.

$$\underbrace{\begin{bmatrix} r & s \\ t & u \end{bmatrix}}_{C} = \underbrace{\begin{bmatrix} a & b \\ c & d \end{bmatrix}}_{A} \cdot \underbrace{\begin{bmatrix} e & f \\ g & h \end{bmatrix}}_{B}$$

$$r = ae + bg$$
$$s = af + bh$$
$$t = ce + dg$$
$$u = cf + dh$$

- Gives 8 multiplications of $\frac{n}{2} \times \frac{n}{2}$ matrices and 4 additions of $\frac{n}{2} \times \frac{n}{2}$ matrices.

Example 5: Divide and Conquer (1st attempt)

- Running time: $T(n) = 8T(n/2) + \Theta(n^2)$.
- Using the Master Method: $a = 8$, $b = 2$. $n^{\log_b a} = n^{\log_2 8} = n^3$
- Case 1: $T(n) = \Theta(n^3)$
- Summary: A complicated algorithm that is not better than the standard one.

Example 5: Divide and Conquer (2nd attempt, Strassen's algorithm)

Idea: go from 8 to 7 multiplications!

- Gives recurrence: $T(n) = 7T(n/2) + \Theta(n^2)$.
- Using the Master Method: $T(n) = \Theta(n^{\log_2 7}) \approx \Theta(n^{2.81})$

Examplo 5: Strassen's algorithm

$$
\begin{aligned}
P_1 &= a \cdot (f - h)\\
P_2 &= (a + b) \cdot h\\
P_3 &= (c + d) \cdot e\\
P_4 &= d \cdot (g - e)\\
P_5 &= (a + d) \cdot (e + h)\\
P_6 &= (b - d) \cdot (g + h)\\
P_7 &= (a - c) \cdot (e + f)
\end{aligned}
$$

$\boxed{\text{7 mults, 18 additions}}$

$$
\begin{aligned}
r &= P_5 + P_4 - P_2 + P_6\\
s &= P_1 + P_2\\
t &= P_3 + P_4\\
u &= P_5 + P_1 - P_3 - P_7
\end{aligned}
$$

- You can verify it!

Verification for r

$$
\begin{aligned}
r &= P_5 + P_4 - P_2 + P_6\\
&= (a + d) \cdot (e + h)\\
&\quad + d \cdot (g - e) - (a + b) \cdot h\\
&\quad + (b - d) \cdot (g + h)\\
&= ae + ah + de + dh + dg - de - ah - bh + bg + bh - dg - dh\\
&= ae + bg
\end{aligned}
$$

Example 6: Closest pair of points

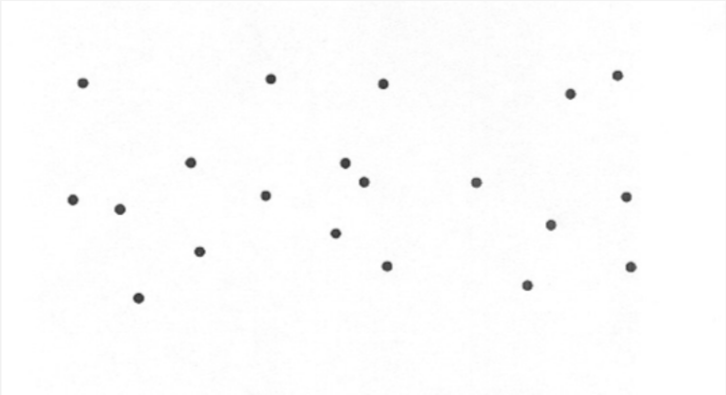
- Problem: Given a set of points in the XY plane, find the closest pair of points.
- Input: $P = \{p_1, p_2, \ldots, p_n\}$ with $p_i = (x_i, y_i)$.
- Output: A pair of points p_i and p_j with minimal distance.

- Easy to find a $\Theta(n^2)$ algorithm, just compute the distance between every pair of points and keep track of the pair with minimal distance.
- We shall see a $\Theta(n \lg n)$ algorithm using D & C.

Aplications

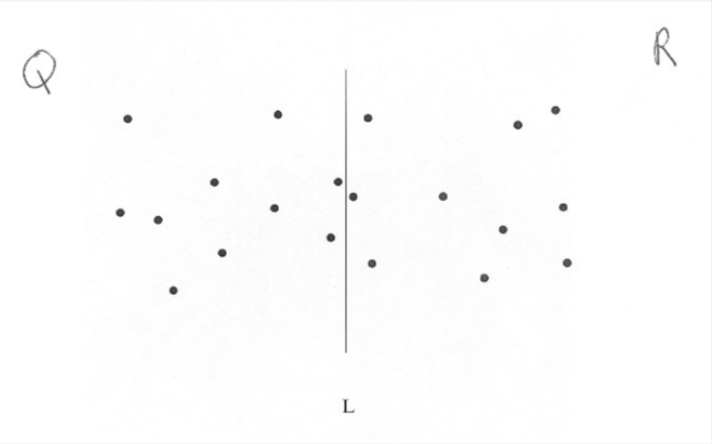
- Computer graphics.
- Geographical information systems.
- Air traffic control.
- etc.

Any idea?



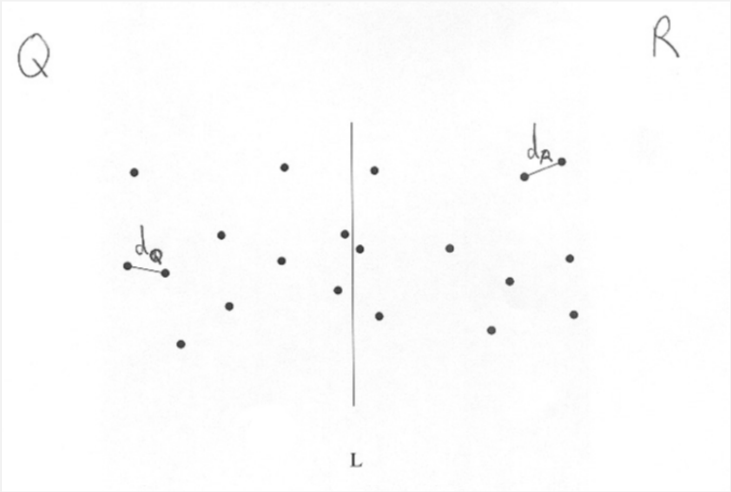
Divide step

- Divide the set P into two subsets, Q e R, each with $n/2$ points.
- The division is made using the median x coordinate.



Conquer step

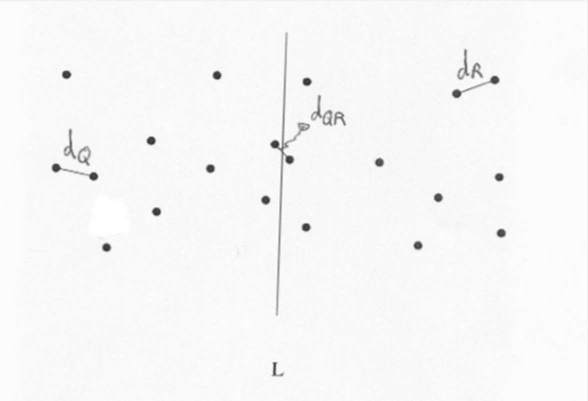
- Solve recursively for Q and R.



- If the number of points is sufficiently small (say, $n <= 3$) compute the solution in a brute force way.

Combine step: the tough part

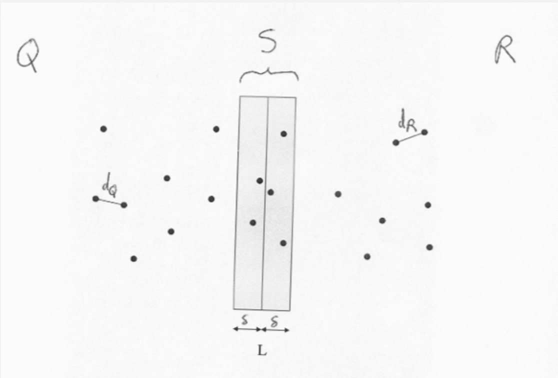
- Need to do it in $\Theta(n)$ in order to obtain an overall complexity of $\Theta(n \lg n)$.



- The minimum distance will be the minimum of three things: d_Q, d_R, d_{QR} (min distance between a point in Q and a point in R).

Combine step

- Let $\delta = min(d_Q, d_R)$



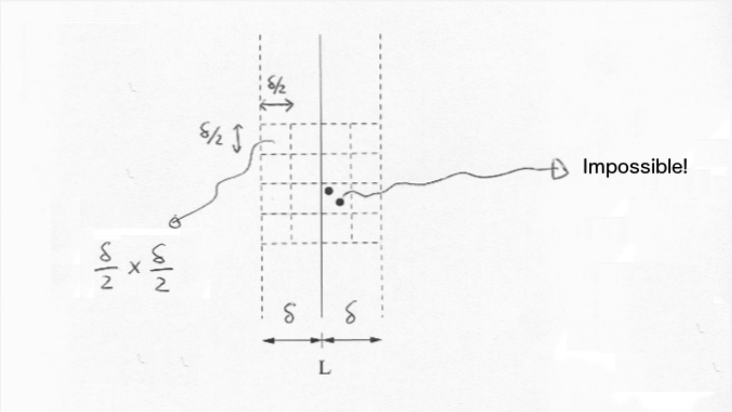
- To find out if there is a pair of points p_i and p_j, such that $p_i \in Q$, $p_j \in R$ and $dist(p_i, p_j) < \delta$, we only need to verify those points which are at most δ distance away from L.

- Let's call S to such a set.

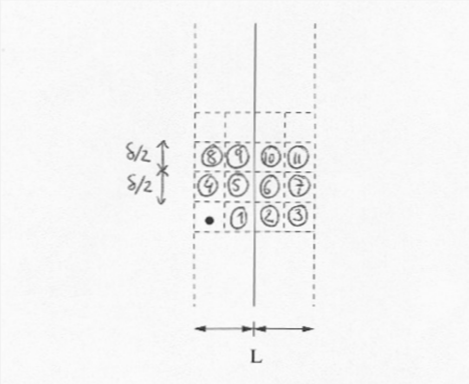
- Note that S can contain all of the initial n points.

- So we might still need to check the distances between every point in Q with every point in R, which would give a $\Theta(n^2)$ algorithm.

- There can't be more than one point per box. Otherwise they would be on the same side (both in Q or both in R) and their distance would be less than $\delta \implies$ a contradiction.



- Let s_1 and s_2 be elements of S and $dist(s_1, s_2) < \delta$. Then, s_1 and s_2 are at most 11 positions apart from each other in the sequence S_y (S sorted on y).



- Why? Because there can only be one point per box.
- 12 or more positions apart $\implies dist(s_1, s_2) > \delta$.

- No need to sort S on each recursive call.
- If we did that we would get the recurrence $T(n) = 2T(n/2) + \Theta(n \lg n)$.
- Cannot apply Master Method but it can be shown that $T(n) = \Theta(n \lg n \lg n)$.
- To obtain a recurrence $T(n) = 2T(n/2) + \Theta(n)$, we need to do the combine step in $\Theta(n)$.

- The idea is to pre-sort array P on the x coordinate (obtaining array P_x) and on the y coordinate (obtaining array P_y).
- Each recursive call receives two arrays of points (sorted on x and sorted on y).

CLOSEST-PAIR(P, n)

$P_x = P$ sorted on x // $\Theta(n \lg n)$
$P_y = P$ sorted on y // $\Theta(n \lg n)$
$(p1, p2) =$ CLOSEST-PAIR-REC(P_x, P_y, n)

$\textsc{Closest-Pair-Rec}(P_x, P_y, n)$

if $n \leq 3$

Compute the distance between every pair of points and return the pair with minimal distance.

else

// divide P into Q and R

Build Q_x, Q_y, R_x, R_y // $\Theta(n)$

$(q1, q2) = \textsc{Closest-Pair-Rec}(Q_x, Q_y, nQ)$

$(r1, r2) = \textsc{Closest-Pair-Rec}(R_x, R_y, nR)$

$dQ = dist(q1, q2)$

$dR = dist(r1, r2)$

$\delta = min(dQ, dR)$

$x^* = x$ coordinate of the last point in array Q_x

// line $L : x == x^*$

// $S =$ points in P that are at most

// δ distance from L.

Build S_y from P_y // $\Theta(n)$

(cont...)

For each $s \in S_y$, compute the distance of s to each of next 11 points in S_y. Let $(s1, s2)$ be the pair with minimal distance.

if $dist(s1, s2) < \delta$

return $(s1, s2)$

elseif $dQ < dR$

return $(q1, q2)$

else return $(r1, r2)$

Further details

- How to obtain Q_x, Q_y, R_x, R_y in $\Theta(n)$?
- The divide step was made through the median x coordinate. What if there's multiple points with the same x?
- Work for you to think about.

Additional notes

- This algorithm was invented in the 70s by Shamos and Hoey.
- The 11 point limit can be reduced. Your textbook refers 7 points only. And even that can be reduced.