Esercitazioni di Informatica 3

Divide et Impera

Paolo Costa

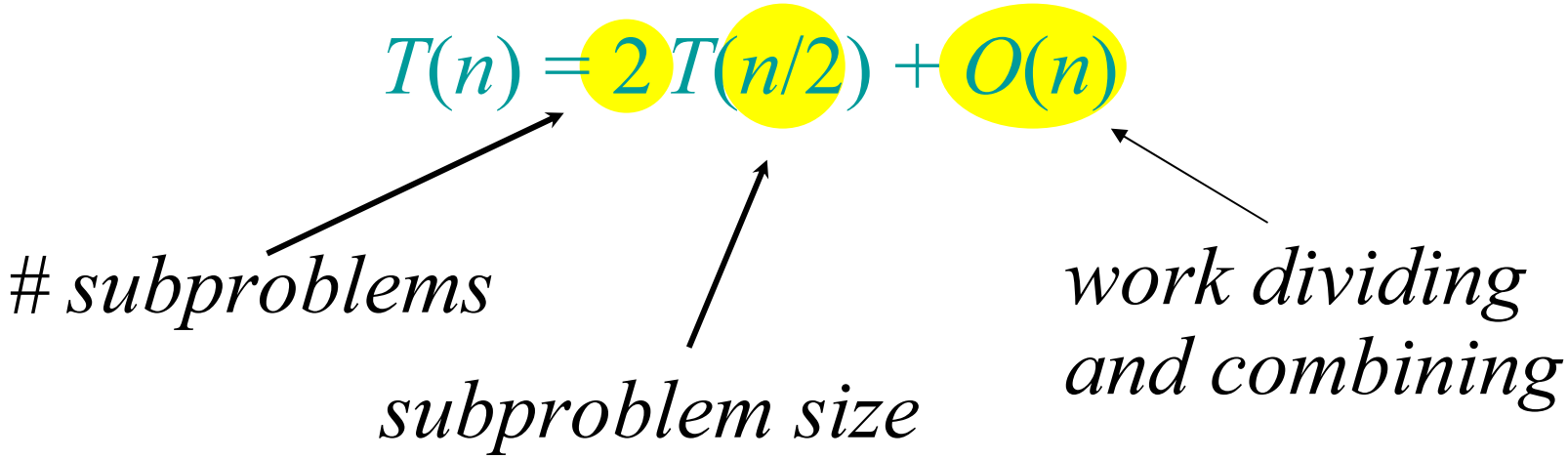
paolo.costa@polimi.it

The divide-and-conquer design paradigm

1. ***Divide*** the problem (instance) into subproblems.
2. ***Conquer*** the subproblems by solving them recursively.
3. ***Combine*** subproblem solutions.

Example: merge sort

1. ***Divide:*** Trivial.
2. ***Conquer:*** Recursively sort 2 subarrays.
3. ***Combine:*** Linear-time merge.



Master theorem (reprise)

we have already seen the following

$\mathsf{T}(n) = a\mathsf{T}(n/b)+cn^k$, for $n > 1$
$\mathsf{T}(1) = \mathsf{d}$,

Solution of the recurrence depends on the ratio $r = b^k/a$

$\mathsf{T}(n) = \Theta(n^{\log_b a})$, if $a > b^k$
$\mathsf{T}(n) = \Theta(n^k \log n)$, if $a = b^k$
$\mathsf{T}(n) = \Theta(n^k)$, if $a < b^k$

Merge sort: $a = 2, b = 2 \Rightarrow n^{\log_b a} = n$
$\Rightarrow$ CASE 2 $(k = 0) \Rightarrow T(n) = \Theta(n \lg n)$.

Binary search

Find an element in a sorted array:

1. ***Divide:*** Check middle element.
2. ***Conquer:*** Recursively search 1 subarray.
3. ***Combine:*** Trivial.

Example: Find 9

3	5	7	8	9	12	15

Binary search

Find an element in a sorted array:

1. ***Divide:*** Check middle element.
2. ***Conquer:*** Recursively search 1 subarray.
3. ***Combine:*** Trivial.

Example: Find 9

Binary search

Find an element in a sorted array:

1. ***Divide:*** Check middle element.
2. ***Conquer:*** Recursively search 1 subarray.
3. ***Combine:*** Trivial.

Example: Find 9

3 5 7 8 9 12 15

Binary search

Find an element in a sorted array:

1. ***Divide:*** Check middle element.
2. ***Conquer:*** Recursively search 1 subarray.
3. ***Combine:*** Trivial.

Example: Find 9

Binary search

Find an element in a sorted array:

1. ***Divide:*** Check middle element.
2. ***Conquer:*** Recursively search 1 subarray.
3. ***Combine:*** Trivial.

Example: Find 9

3 5 7 8 9 12 15

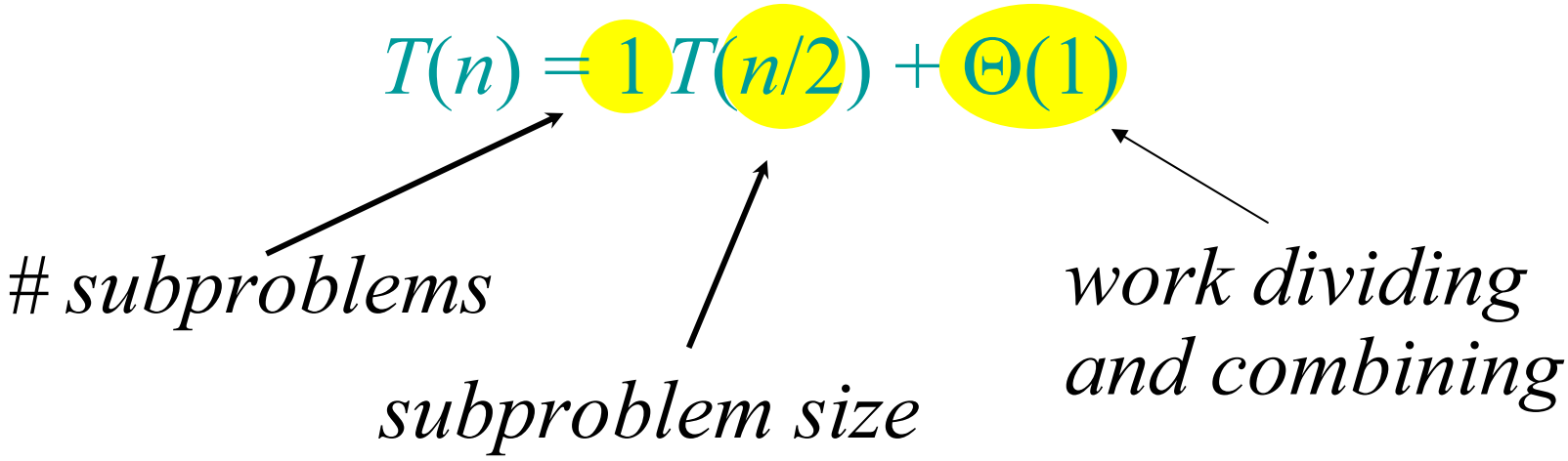
Binary search

Find an element in a sorted array:

1. ***Divide:*** Check middle element.
2. ***Conquer:*** Recursively search 1 subarray.
3. ***Combine:*** Trivial.

Example: Find 9

Recurrence for binary search



$$n^{\log_b a} = n^{\log_2 1} = n^0 = 1 \Rightarrow \text{CASE 2 } (k = 0)$$

$$\Rightarrow \quad T(n) = \Theta(\lg n) \ .$$

Powering a number

Problem: Compute a^n, where $n \in \mathbb{N}$.

Naive algorithm: $\Theta(n)$.

Divide-and-conquer algorithm:

$$a^n = \begin{cases} a^{n/2} \cdot a^{n/2} & \text{if } n \text{ is even;} \\ a^{(n-1)/2} \cdot a^{(n-1)/2} \cdot a & \text{if } n \text{ is odd.} \end{cases}$$

$$T(n) = T(n/2) + \Theta(1) \implies T(n) = \Theta(\lg n) \ .$$

Matrix multiplication

Input: $A = [a_{ij}], B = [b_{ij}].$
Output: $C = [c_{ij}] = A \cdot B.$

$\Big\}$ $i, j = 1, 2, \ldots, n.$

$$\begin{bmatrix} c_{11} & c_{12} & \cdots & c_{1n} \\ c_{21} & c_{22} & \cdots & c_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ c_{n1} & c_{n2} & \cdots & c_{nn} \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix} \cdot \begin{bmatrix} b_{11} & b_{12} & \cdots & b_{1n} \\ b_{21} & b_{22} & \cdots & b_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ b_{n1} & b_{n2} & \cdots & b_{nn} \end{bmatrix}$$

$$c_{ij} = \sum_{k=1}^{n} a_{ik} \cdot b_{kj}$$

Standard algorithm

for $i \leftarrow 1$ **to** n
  do for $j \leftarrow 1$ **to** n
    do $c_{ij} \leftarrow 0$
      for $k \leftarrow 1$ **to** n
        do $c_{ij} \leftarrow c_{ij} + a_{ik} \cdot b_{kj}$

Running time = $\Theta(n^3)$

Divide-and-conquer algorithm

IDEA:

$n \times n$ matrix = 2×2 matrix of $(n/2) \times (n/2)$ submatrices:

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

$$C = A \cdot B$$

$$\left.\begin{aligned} r &= ae + bg \\ s &= af + bh \\ t &= ce + dg \\ u &= cf + dh \end{aligned}\right\} \begin{array}{l} 8 \text{ mults of } (n/2) \times (n/2) \text{ submatrices} \\ 4 \text{ adds of } (n/2) \times (n/2) \text{ submatrices} \end{array}$$

Analysis of D&C algorithm

$$T(n) = 8\,T(n/2) + \Theta(n^2)$$

 # submatrices

submatrix size

work adding submatrices

$$n^{\log_b a} = n^{\log_2 8} = n^3 \;\Rightarrow\; \text{CASE 1} \;\Rightarrow\; T(n) = \Theta(n^3).$$

No better than the ordinary algorithm.

Strassen's idea

- Multiply 2×2 matrices with only 7 recursive mults.

$$P_1 = a \cdot (f - h)$$
$$P_2 = (a + b) \cdot h$$
$$P_3 = (c + d) \cdot e$$
$$P_4 = d \cdot (g - e)$$
$$P_5 = (a + d) \cdot (e + h)$$
$$P_6 = (b - d) \cdot (g + h)$$
$$P_7 = (a - c) \cdot (e + f)$$

$$r = P_5 + P_4 - P_2 + P_6$$
$$s = P_1 + P_2$$
$$t = P_3 + P_4$$
$$u = P_5 + P_1 - P_3 - P_7$$

7 mults, 18 adds/subs.
Note: No reliance on commutativity of mult!

Strassen's idea

- Multiply 2×2 matrices with only 7 recursive mults.

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

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

Strassen's algorithm

1. ***Divide:*** Partition A and B into $(n/2)\times(n/2)$ submatrices. Form terms to be multiplied using $+$ and $-$.
2. ***Conquer:*** Perform 7 multiplications of $(n/2)\times(n/2)$ submatrices recursively.
3. ***Combine:*** Form C using $+$ and $-$ on $(n/2)\times(n/2)$ submatrices.

$$T(n) = 7\,T(n/2) + \Theta(n^2)$$

Analysis of Strassen

$$T(n) = 7\,T(n/2) + \Theta(n^2)$$

$n^{\log_b a} = n^{\log_2 7} \approx n^{2.81} \Rightarrow$ CASE 1 $\Rightarrow T(n) = \Theta(n^{\lg 7})$.

The number 2.81 may not seem much smaller than 3, but because the difference is in the exponent, the impact on running time is significant. In fact, Strassen's algorithm beats the ordinary algorithm on today's machines for $n \geq 30$ or so.

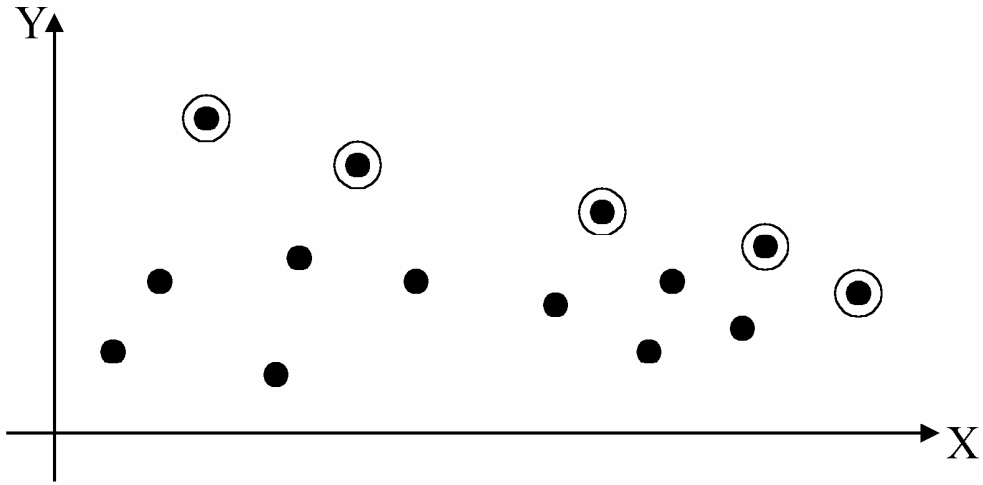
Best to date (of theoretical interest only): $\Theta(n^{2.376\cdots})$.

2-D maxima finding problem

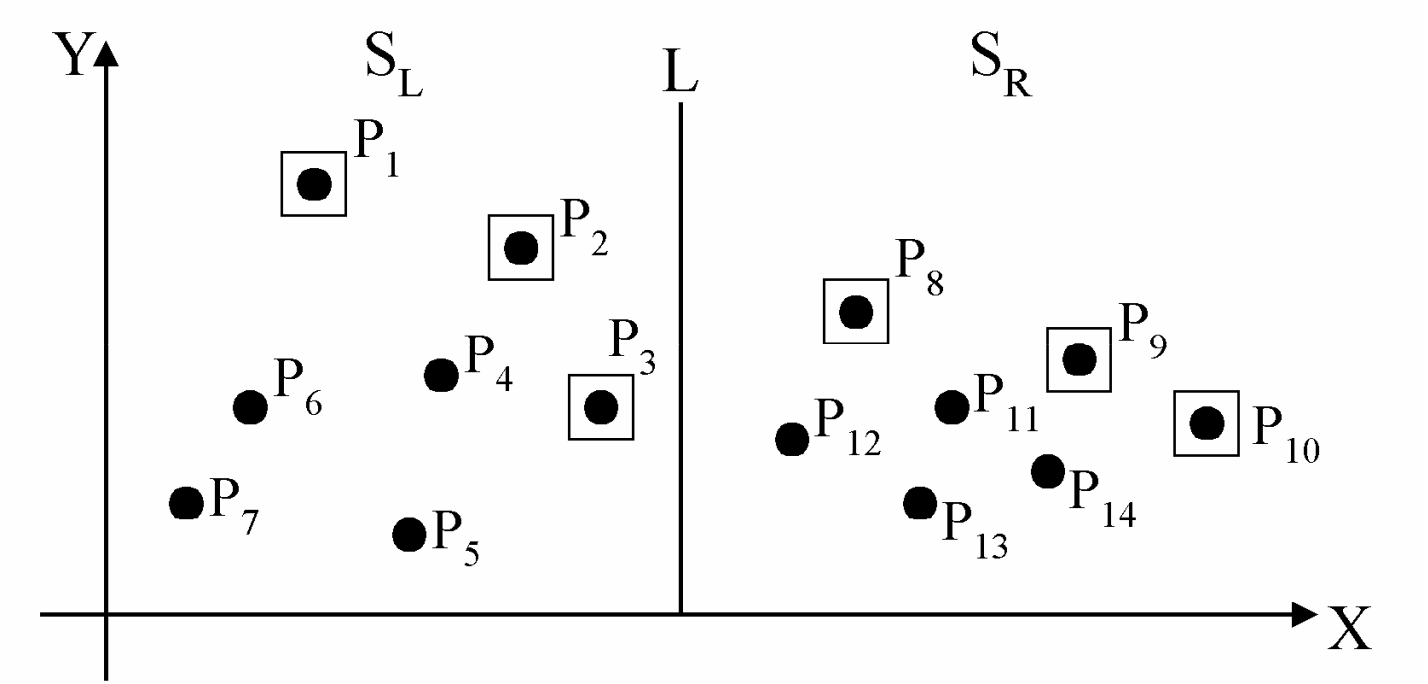
- **Def** : A point (x_1, y_1) dominates (x_2, y_2) if $x_1 > x_2$ and $y_1 > y_2$. A point is called a maxima if no other point dominates it
- Straightforward method : Compare every pair of points.

Time complexity:

$O(n^2)$



Divide-and-conquer for maxima finding



The maximal points of S_L and S_R

The algorithm:

- Input: A set of n planar points.
- Output: The maximal points of S.

Step 1: If S contains only one point, return it as the maxima. Otherwise, find a line L perpendicular to the X-axis which separates the set of points into two subsets S_L and S_R , each of which consisting of n/2 points.

Step 2: Recursively find the maximal points of S_L and S_R .

Step 3: Find the largest y-value of S_R. Project the maximal points of S_L onto L. Discard each of the maximal points of S_L if its y-value is less than the largest y-value of S_R .

- Time complexity: T(n)

  Step 1: O(n)

  Step 2: 2T(n/2)

  Step 3: O(n)

$$T(n)=\begin{cases} 2T(n/2)+O(n)+O(n) & , n>1 \\ 1 & , n=1 \end{cases}$$

Assume $n = 2^k$

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

The Counterfit Coin

You are given a pot of n>=3 coins, n-1 of which are authentic gold coins and 1 of which is a counterfit coin. The counterfit coin might be either heavier or lighter than an authentic coin. Your only means of measuring weights is to use a balance scale with two bowls.

The algorithm

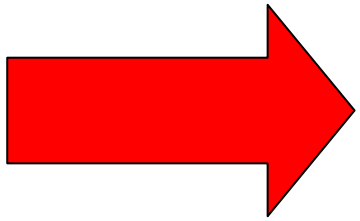
- Divide the pile roughly into thirds, calling the thirds A, B and C.
- If n is not a multiple of three, then keep the extra 1 or 2 coin aside. Now weigh A against B.
- If they are equal, throw them out and recurse with C and the sparse coins.
- If they are unequal, throw out C and the sparse coins and recurse.
- When we get down to 2 coins, just compare each of them against a known good coin (i.e. any coin we have thrown out), revealing the counterfit coin.

Complexity

- At each round we throw away $\frac{1}{3}n$
- The recurrency equation will be:

$$T(n) = T(\frac{2}{3}n) + 1 \quad \text{con} \qquad T(2) = 1$$

a = 1 b = 3/2 k = 0



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

Twelve coins

Find an algorithm to discover counterfit coin with only 3 weighings (estabilishing whether it is heavier or lighter)

Solution at:

http://ssc.unict.it/test/monetesl.htm