# Part3-Divide & Conquer Problem

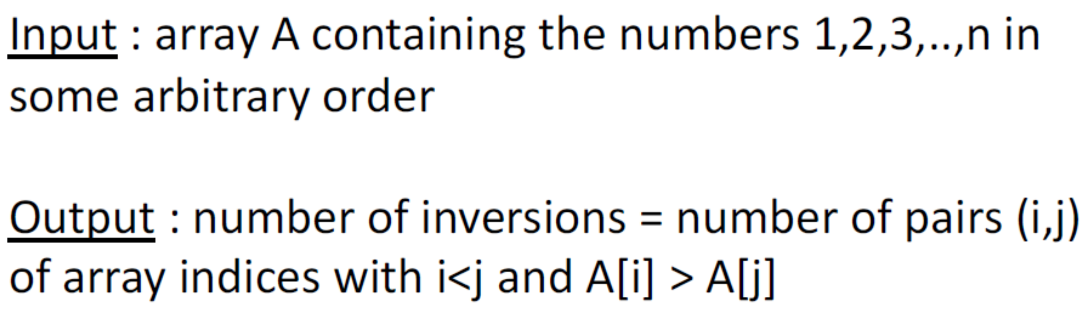
| | |
|---|---|
| Completed on | @2020/03/15 |
| Key videos | https://www.coursera.org/learn/algorithms-divide-conquer/lecture/IUiUk/o-n-log-n-algorithm-for-counting-inversions-ii https://www.coursera.org/learn/algorithms-divide-conquer/lecture/YXmYz/strassens-subcubic-matrix-multiplication-algorithm https://www.coursera.org/learn/algorithms-divide-conquer/lecture/nf0jk/o-n-log-n-algorithm-for-closest-pair-i-advanced-optional https://www.coursera.org/learn/algorithms-divide-conquer/lecture/cER7y/o-n-log-n-algorithm-for-closest-pair-ii-advanced-optional |
| Note | In-depth analysis to three typical divide-and-conquer algorithms with lower complexity, i.e., Counting Inversions, Strassen's Matrix Multiplication Algorithm, and Algorithm for Closest Pair. |

## Note

▼ O(n log n) Algorithm for Counting Inversions I

- 基本问题：计算一个无序数组中逆序对的数量 ⇒ 应用于相似性度量（similarity measure between two ranked lists）

  如：[1, 3, 5, 2, 4, 6]有[3,2], [5,2], [5,4]三对

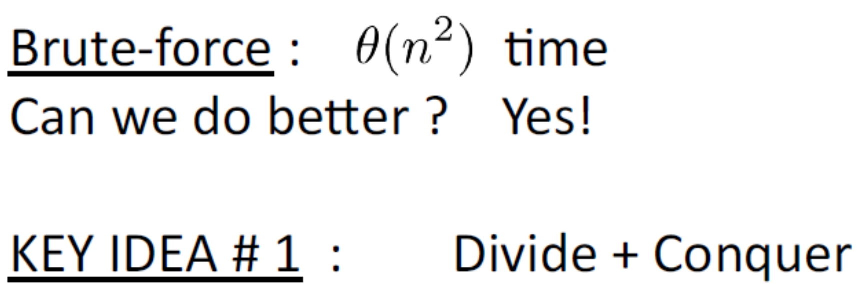


Input : array A containing the numbers 1,2,3,..,n in some arbitrary order

Output : number of inversions = number of pairs (i,j) of array indices with i<j and A[i] > A[j]

- 解法：暴力 or 分治

  暴力算法，双层循环（double loop）

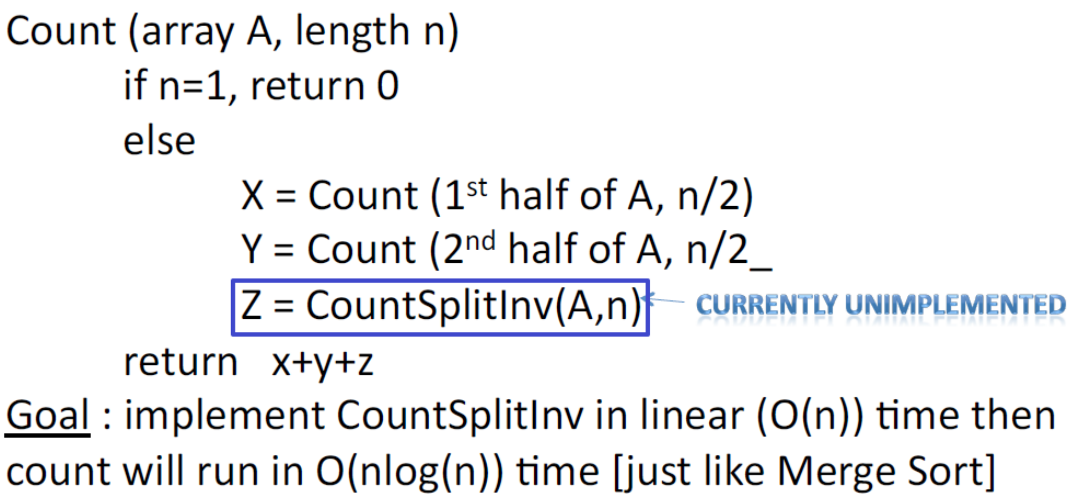


Brute-force : $\theta(n^2)$ time

Can we do better ? Yes!

KEY IDEA # 1 : Divide + Conquer

- 分治算法，按照之前MergeSort的方法，对于N个数，分治算法的复杂度为O(logn)，因此只需保证其中的算法复杂度为O(n)即可低于暴力算法！

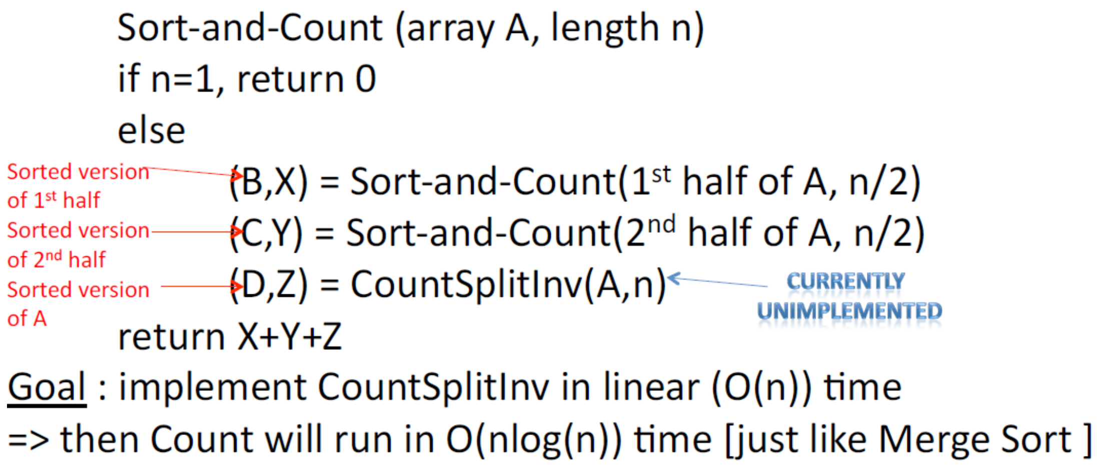


```
Count (array A, length n)
    if n=1, return 0
    else
        X = Count (1st half of A, n/2)
        Y = Count (2nd half of A, n/2_
        Z = CountSplitInv(A,n)  <- CURRENTLY UNIMPLEMENTED
    return  x+y+z
```

Goal : implement CountSplitInv in linear (O(n)) time then count will run in O(nlog(n)) time [just like Merge Sort]

▼ O(n log n) Algorithm for Counting Inversions II

- 进一步优化，受到MergeSort启发，进行排序并计数



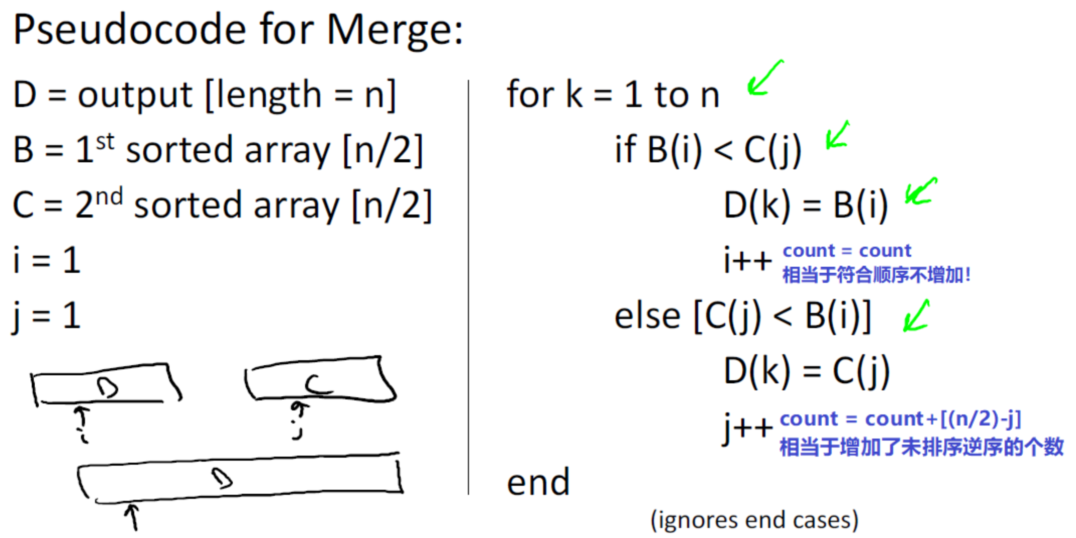
```
Sort-and-Count (array A, length n)
if n=1, return 0
else
    (B,X) = Sort-and-Count(1st half of A, n/2)
    (C,Y) = Sort-and-Count(2nd half of A, n/2)
    (D,Z) = CountSplitInv(A,n)  <- CURRENTLY UNIMPLEMENTED
return X+Y+Z
```

Sorted version of 1st half
Sorted version of 2nd half
Sorted version of A

Goal : implement CountSplitInv in linear (O(n)) time

=> then Count will run in O(nlog(n)) time [just like Merge Sort ]

- 算法实现：

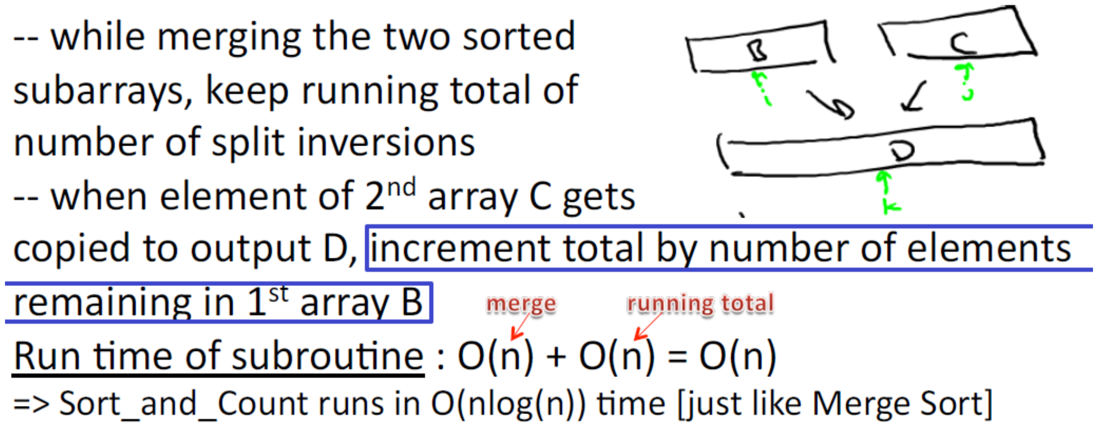


Pseudocode for Merge:

```
D = output [length = n]
B = 1st sorted array [n/2]
C = 2nd sorted array [n/2]
i = 1
j = 1

for k = 1 to n
    if B(i) < C(j)
        D(k) = B(i)
        i++    count = count  相当于符合顺序不增加!
    else [C(j) < B(i)]
        D(k) = C(j)
        j++    count = count+[(n/2)-j]  相当于增加了未排序逆序的个数
end
```

(ignores end cases)



-- while merging the two sorted subarrays, keep running total of number of split inversions

-- when element of 2nd array C gets copied to output D, increment total by number of elements remaining in 1st array B

<u>Run time of subroutine</u> : O(n) + O(n) = O(n)

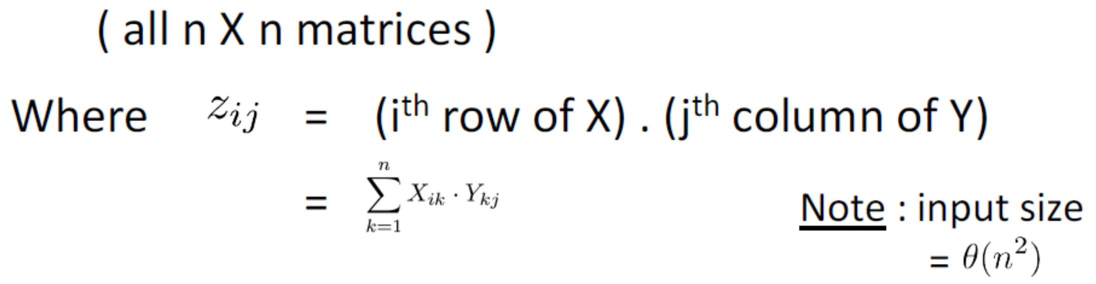
merge running total

=> Sort_and_Count runs in O(nlog(n)) time [just like Merge Sort]

- Strassen's Subcubic Matrix Multiplication Algorithm
  - 一般乘法：输入两个NxN的矩阵，输出结果

    考虑最简单的2×2矩阵，输出2×2个元素，每个元素有2次乘法得到结果，最终算法复杂度：

$$T(n)=\mathcal{O}(n^{\log_2 8})=\mathcal{O}(n^3)$$



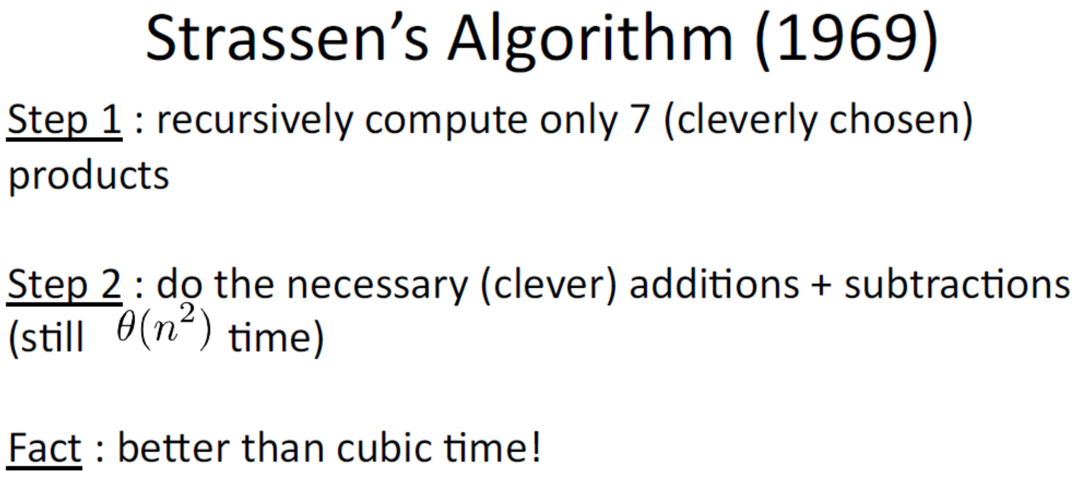
( all n X n matrices )

Where $z_{ij}$ = (ith row of X) . (jth column of Y)

$= \sum_{k=1}^{n} X_{ik} \cdot Y_{kj}$

Note : input size $= \theta(n^2)$

- 分治思想：Strassen's Algorithm相比于一般乘法算法，中间仅需要7次乘法运算！



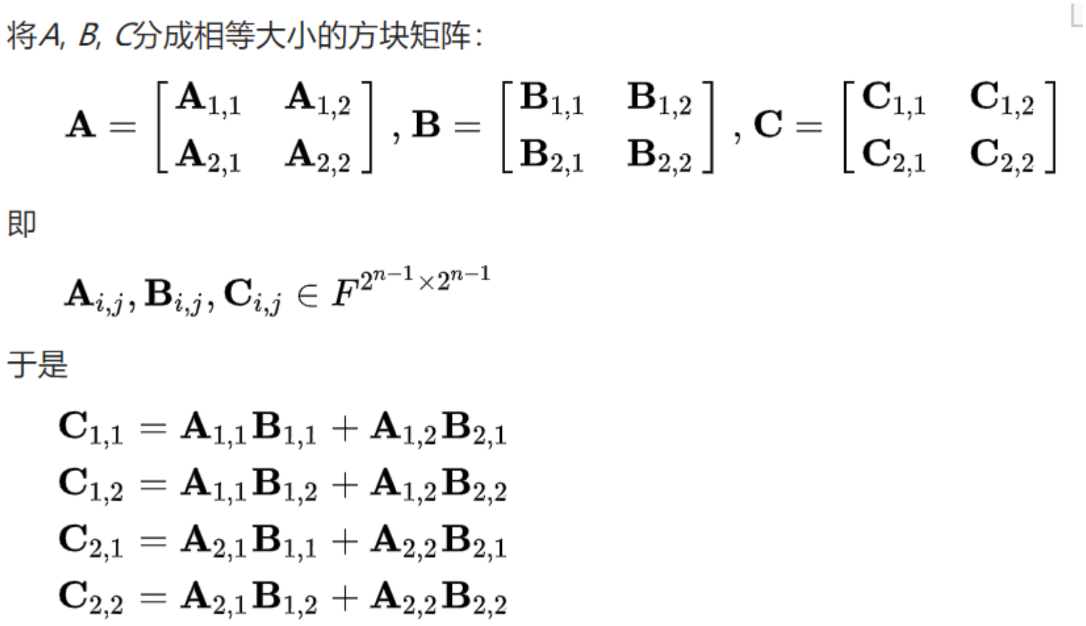
## Strassen’s Algorithm (1969)

Step 1 : recursively compute only 7 (cleverly chosen) products

Step 2 : do the necessary (clever) additions + subtractions (still $\theta(n^2)$ time)

Fact : better than cubic time!

- 算法解析（来源wiki）



将*A*, *B*, *C*分成相等大小的方块矩阵：

$$\mathbf{A} = \begin{bmatrix} \mathbf{A}_{1,1} & \mathbf{A}_{1,2} \\ \mathbf{A}_{2,1} & \mathbf{A}_{2,2} \end{bmatrix}, \mathbf{B} = \begin{bmatrix} \mathbf{B}_{1,1} & \mathbf{B}_{1,2} \\ \mathbf{B}_{2,1} & \mathbf{B}_{2,2} \end{bmatrix}, \mathbf{C} = \begin{bmatrix} \mathbf{C}_{1,1} & \mathbf{C}_{1,2} \\ \mathbf{C}_{2,1} & \mathbf{C}_{2,2} \end{bmatrix}$$

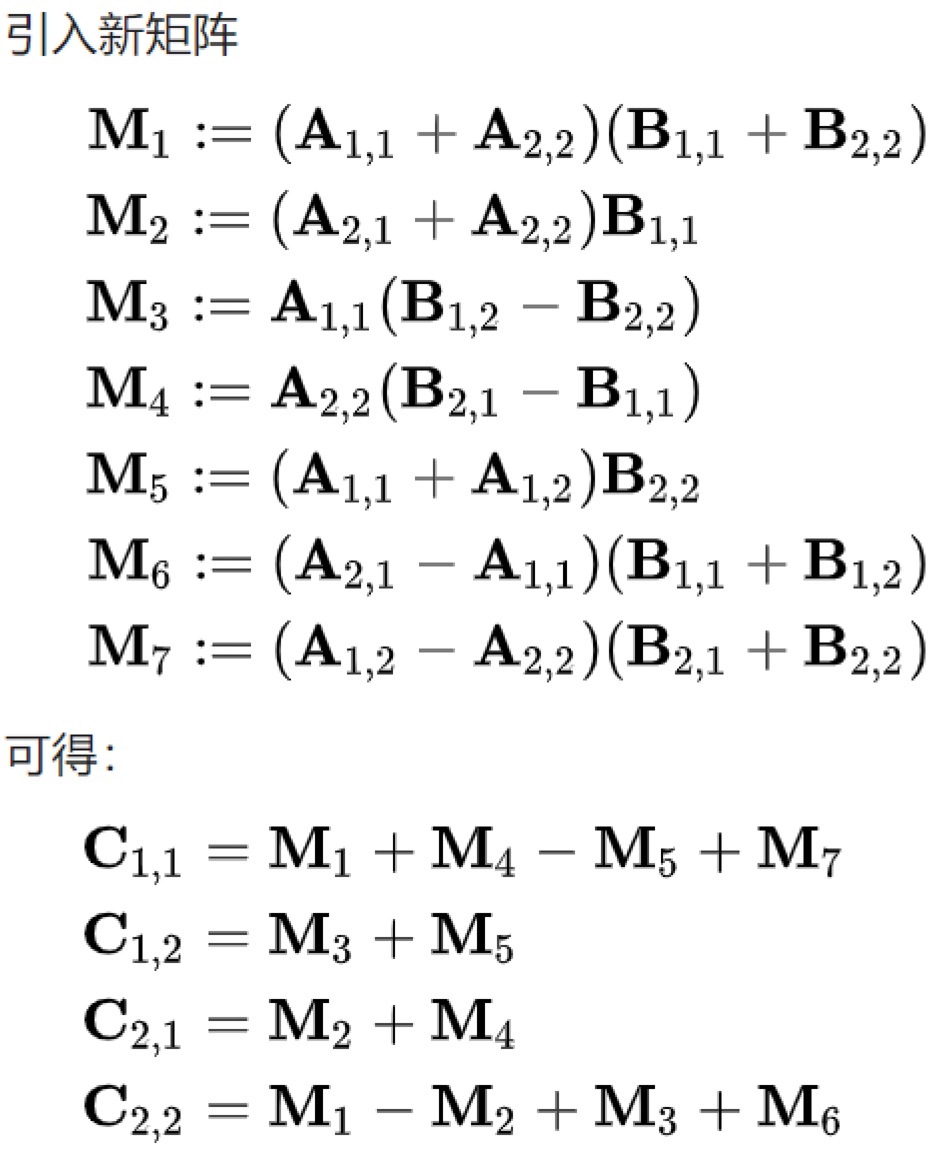
即

$$\mathbf{A}_{i,j}, \mathbf{B}_{i,j}, \mathbf{C}_{i,j} \in F^{2^{n-1} \times 2^{n-1}}$$

于是

$$\begin{aligned} \mathbf{C}_{1,1} &= \mathbf{A}_{1,1}\mathbf{B}_{1,1} + \mathbf{A}_{1,2}\mathbf{B}_{2,1} \\ \mathbf{C}_{1,2} &= \mathbf{A}_{1,1}\mathbf{B}_{1,2} + \mathbf{A}_{1,2}\mathbf{B}_{2,2} \\ \mathbf{C}_{2,1} &= \mathbf{A}_{2,1}\mathbf{B}_{1,1} + \mathbf{A}_{2,2}\mathbf{B}_{2,1} \\ \mathbf{C}_{2,2} &= \mathbf{A}_{2,1}\mathbf{B}_{1,2} + \mathbf{A}_{2,2}\mathbf{B}_{2,2} \end{aligned}$$

引入新矩阵标记M，是由A、B矩阵元素加减的乘积组合



引入新矩阵

$$
\begin{aligned}
\mathbf{M}_1 &:= (\mathbf{A}_{1,1} + \mathbf{A}_{2,2})(\mathbf{B}_{1,1} + \mathbf{B}_{2,2}) \\
\mathbf{M}_2 &:= (\mathbf{A}_{2,1} + \mathbf{A}_{2,2})\mathbf{B}_{1,1} \\
\mathbf{M}_3 &:= \mathbf{A}_{1,1}(\mathbf{B}_{1,2} - \mathbf{B}_{2,2}) \\
\mathbf{M}_4 &:= \mathbf{A}_{2,2}(\mathbf{B}_{2,1} - \mathbf{B}_{1,1}) \\
\mathbf{M}_5 &:= (\mathbf{A}_{1,1} + \mathbf{A}_{1,2})\mathbf{B}_{2,2} \\
\mathbf{M}_6 &:= (\mathbf{A}_{2,1} - \mathbf{A}_{1,1})(\mathbf{B}_{1,1} + \mathbf{B}_{1,2}) \\
\mathbf{M}_7 &:= (\mathbf{A}_{1,2} - \mathbf{A}_{2,2})(\mathbf{B}_{2,1} + \mathbf{B}_{2,2})
\end{aligned}
$$

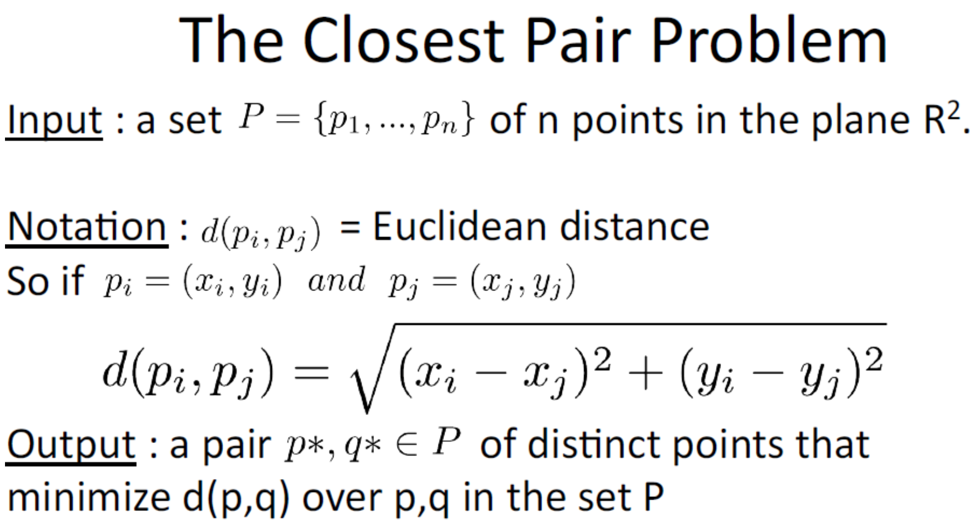
可得：

$$
\begin{aligned}
\mathbf{C}_{1,1} &= \mathbf{M}_1 + \mathbf{M}_4 - \mathbf{M}_5 + \mathbf{M}_7 \\
\mathbf{C}_{1,2} &= \mathbf{M}_3 + \mathbf{M}_5 \\
\mathbf{C}_{2,1} &= \mathbf{M}_2 + \mathbf{M}_4 \\
\mathbf{C}_{2,2} &= \mathbf{M}_1 - \mathbf{M}_2 + \mathbf{M}_3 + \mathbf{M}_6
\end{aligned}
$$

最终，输出矩阵的四个元素可以由七个矩阵M加减组合得到!

▼ O(n log n) Algorithm for Closest Pair I [Advanced - Optional]

- 问题描述与解法分析



## The Closest Pair Problem

Input : a set $P = \{p_1, ..., p_n\}$ of n points in the plane R$^2$.

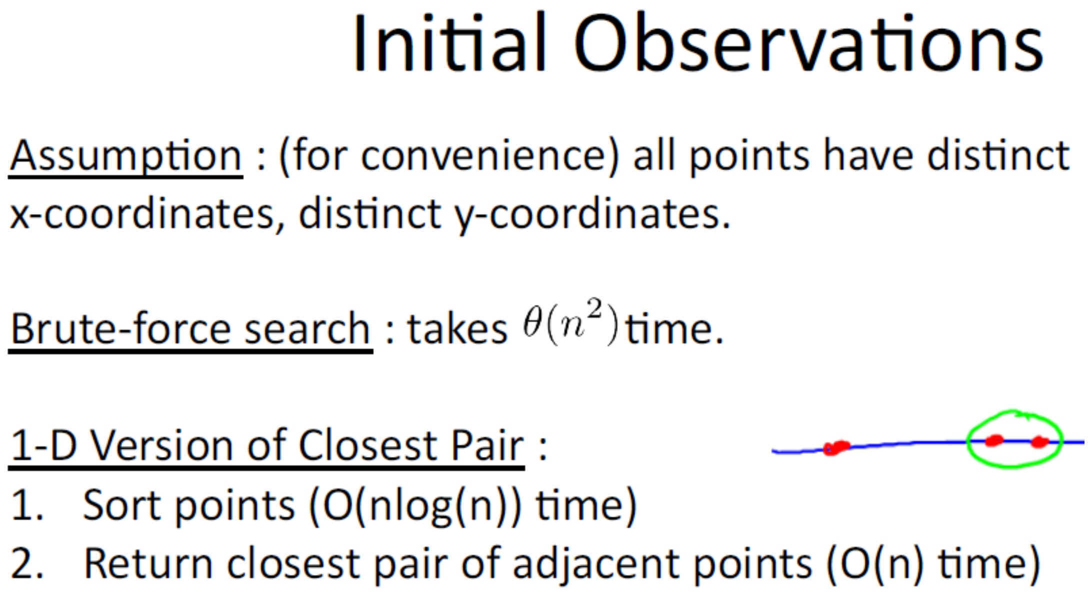
Notation : $d(p_i, p_j)$ = Euclidean distance

So if $p_i = (x_i, y_i)$ *and* $p_j = (x_j, y_j)$

$$d(p_i, p_j) = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}$$

Output : a pair $p*, q* \in P$ of distinct points that minimize d(p,q) over p,q in the set P

- 解法分析



## Initial Observations

Assumption : (for convenience) all points have distinct x-coordinates, distinct y-coordinates.

Brute-force search : takes $\theta(n^2)$time.

1-D Version of Closest Pair :

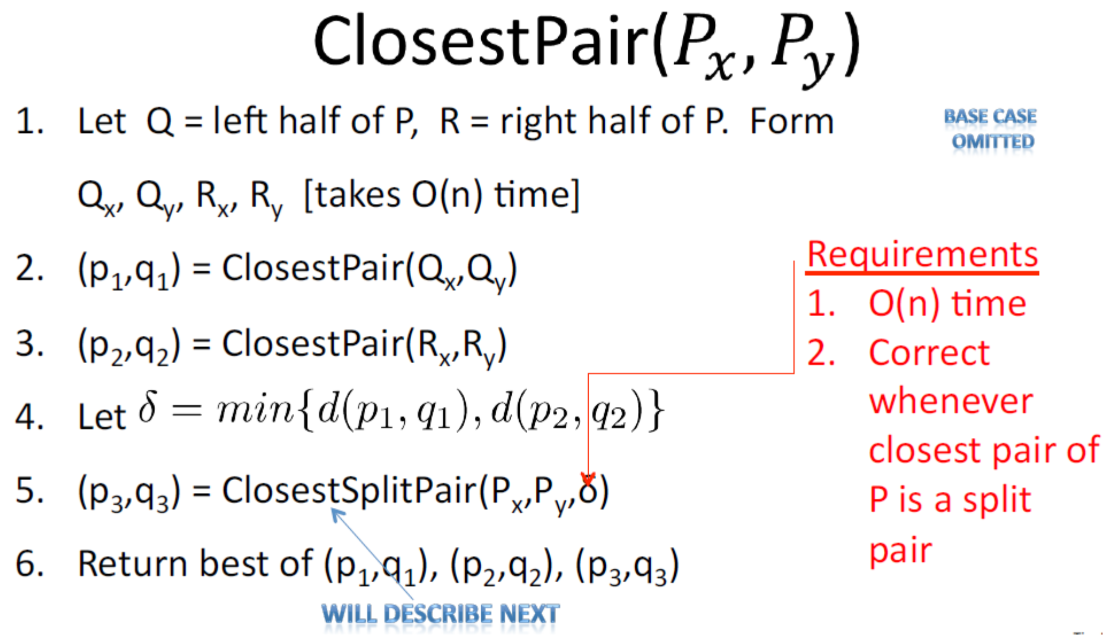
1. Sort points (O(nlog(n)) time)
2. Return closest pair of adjacent points (O(n) time)

暴力解法：双层循环嵌套O(n2)

一维解法：先排序O(nlogn)，然后计算相邻元素距离O(n)

- 二维算法步骤与分析

分别对x与y排序，但是仍然尚不足以解决问题！⇒ 需要额外的合并函数符合O(n)复杂度！



ClosestPair$(P_x, P_y)$

BASE CASE OMITTED

```
1. Let Q = left half of P, R = right half of P. Form
   Q_x, Q_y, R_x, R_y [takes O(n) time]
2. (p_1,q_1) = ClosestPair(Q_x,Q_y)
3. (p_2,q_2) = ClosestPair(R_x,R_y)
4. Let δ = min{d(p_1,q_1), d(p_2,q_2)}
5. (p_3,q_3) = ClosestSplitPair(P_x,P_y,δ)
6. Return best of (p_1,q_1), (p_2,q_2), (p_3,q_3)
```

WILL DESCRIBE NEXT

Requirements

1. O(n) time
2. Correct whenever closest pair of P is a split pair

中间增加了步骤4，提前计算子问题中的最小值

▼ O(n log n) Algorithm for Closest Pair II [Advanced - Optional]

暂未理解！

## References

- Strassen's Subcubic Matrix Multiplication Algorithm

施特拉森演算法
https://zh.wikipedia.org/wiki/%E6%96%BD%E7%89%B9%E6%8B%89%E6%A3%AE%E6%BC%94%E7%AE%97%E6%B3%95

Strassen algorithm
https://en.wikipedia.org/wiki/Strassen_algorithm

- 乘法算法

Matrix multiplication algorithm
https://en.wikipedia.org/wiki/Matrix_multiplication_algorithm

- 分治法与暴力法求最近的二维点对

二维空间最近点对问题 python_Python_荒谬小孩-CSDN博客
https://blog.csdn.net/wangkai0011/article/details/80518314

空间最小距离点对--python_Python_woshilsh的博客-CSDN博客
https://blog.csdn.net/woshilsh/article/details/89956482

python动态演示分治法解决最近对问题
https://www.cnblogs.com/whitehawk/p/10853875.html