Divide & Conquer Algorithmic Approach

Recall: Mergesort is the exemplar

Recall: Under certain circumstances, the Master Theorem helps us analyze these algorithms.

Divide & Conquer Paradigm

1. Divide the input into smaller subproblems
2. Conquer the subproblems, using recursion
3. Combine the solutions of subproblems into solution for original problem.

Problem: Counting Inversions

Input: array A of n distinct integers.

Output: the number of inversions of A

3 1 5 4 2

out of order

out of order

Inversions are:

1,3 2,3 2,5 2,4 4,5 # inversions = 5

Why are we interested?

- recommendation algorithms

Brute Force algorithm to count inversions:

```
BF_CountInversions(array A)
/* Output = number of pairs of elements
/* that are out of order in A */

    numInv = 0
    for i = 1 to n-1 do
        for j = i+1 to n do
            if A[i] > A[j]
                numInv ++

    return numInv.
```

7 ... 3
i   j

$$\sum_{i=1}^{n} i = \frac{n(n+1)}{2} \qquad \sum_{i=1}^{n-1} i = \frac{(n-1)n}{2}$$

Running time? $O\left(\sum_{i=1}^{n-1} i\right) = O(n^2)$

Counting Inversions using Divide & Conquer

$\underbrace{7 \quad 3 \quad 1 \quad 6}_{\text{recursion}: 4} \;\Big|\; \underbrace{5 \quad 4 \quad 2 \quad 8}_{\text{recursion}: 2+1=3}$

- Divide A into left-half and right half
- count inversions in left half
- count inversions in right half
- count inversions $\underbrace{\text{between the two halves}}_{\text{"split" inversions}}$

return sum of these 3.

7 3 | 1 6

left right

Suppose sorted

7 6
3 1

compare

left right

element i	1	3	6	7
# split inversions with i		0	1	0

Merge_and_Count_Split_Inversions (C, D)

/* C and D are sorted lists of length $\frac{n}{2}$

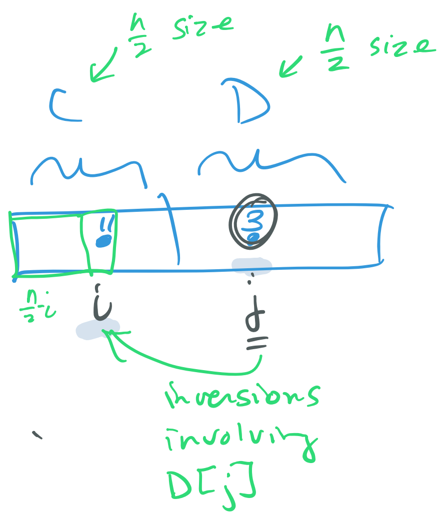
/* Sum, $\forall\, d \in D$, # elements $c \in C$ where

/* $d < c$, and output B, contents of

/* C and D in sorted order

/* Simplifying assumption: $|C| = |D|$.

$i = 1 \quad j = 1 \quad splitInv = 0$



for $K = 1$ to n do

if $C[i] < D[j]$

$B[K] = C[i]$

$i = i + 1$

else

$B[K] = D[j]; \quad j = j + 1$

$splitInv \mathrel{+}= \frac{n}{2} - i + 1$

return (B, splitInv)

Running Time: Mergesort: $T(n) = 2T(n/2) + n$

$n \in \Theta(n')$ ∴

by case 2 of MT, Mergesort_with Inversion counting runs in $\Theta(n \log n)$

MergeSort_and_Count_Inversions (A)

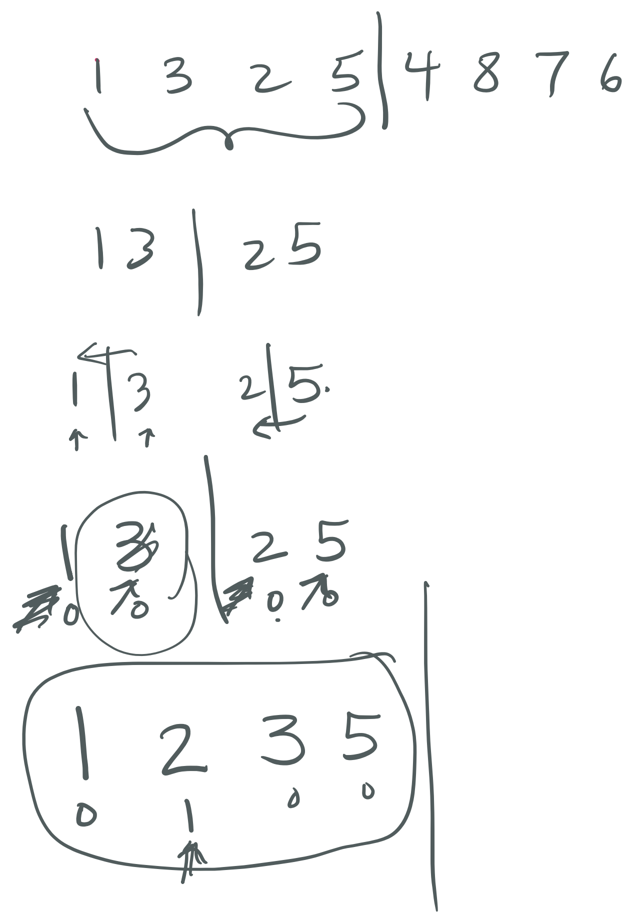
/* A is an array of n integers

/* simplifying assumption: n is even */

⋮

for you to do

⋮



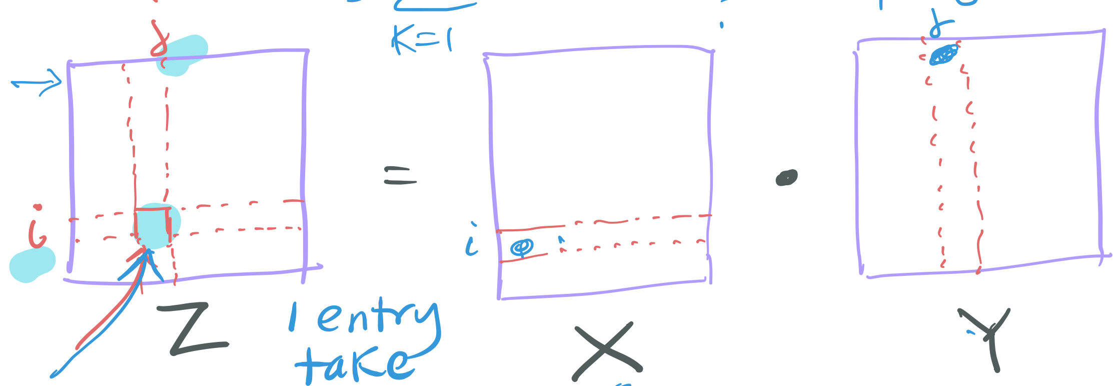
Running Time?

$$\begin{bmatrix} 1 & 2 \\ 3 & 1 \end{bmatrix} \cdot \begin{bmatrix} 2 & 1 \\ 3 & 1 \end{bmatrix} = \begin{bmatrix} 2+6 & 3 \\ 9 & 4 \end{bmatrix}$$

Strassen's Matrix Multiplication

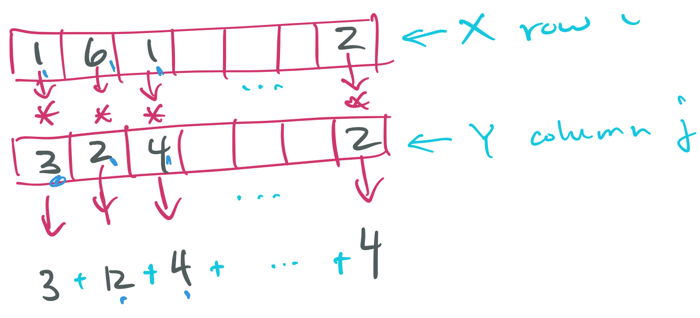
X, Y are n×n matrices (n^2 entries each)

$$Z[i][j] = \sum_{k=1}^{n} X[i][k] \cdot Y[k][j]$$



1 entry take $\Theta(n) = X_i \cdot Y_j$

"dot product" :



Illustrating with 2×2 matrices

$$\begin{pmatrix} a & b \\ c & d \end{pmatrix} \cdot \begin{pmatrix} e & f \\ g & h \end{pmatrix} = \begin{pmatrix} ae+bg & af+bh \\ ce+dg & cf+dh \end{pmatrix}$$

Problem: Matrix Multiplication

Input: 2 $n \times n$ matrices X and Y

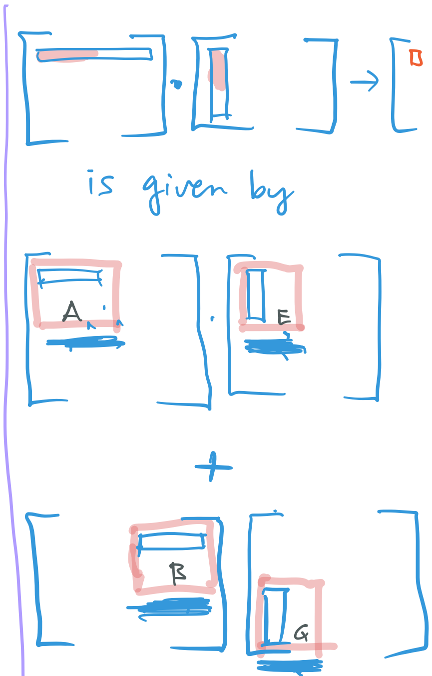
Output: the matrix product $X \cdot Y$

```
StraightforwardMatrixMult(X, Y)
/* input: nxn matrices X and Y
/* output: nxn matrix Z = X·Y */
for i = 1 to n do
    for j = 1 to n do
        Z[i][j] = 0
        for k = 1 to n do
            Z[i][j] += X[i][k]
                       * Y[k][j]
return Z.
```

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

Strassen's Cool idea....

$$\begin{bmatrix} A & B \\ C & D \end{bmatrix} \cdot \begin{bmatrix} E & F \\ G & H \end{bmatrix} = \begin{bmatrix} A\cdot E + B\cdot G & A\cdot F + B\cdot H \\ C\cdot E + D\cdot G & C\cdot F + D\cdot H \end{bmatrix}$$



First, do it recursively:

RecMatMult

/* input: $n \times n$ matrices X and Y

/* output: $Z = X \circ Y$

/* assume: n is a power of 2

if $n = 1$

return the 1×1 matrix $X[1][1] * Y[1][1]$

else

A, B, C, D = submatrices of X, as above

E, F, G, H = submatrices of Y, as above

recursively compute the eight matrix products as above

return the result

$n^2 \in O(n^{3-\varepsilon})$

$$T(n) = 8\,T\left(\tfrac{n}{2}\right) + n^2$$

Master Thm: $n^2 \in O(\underbrace{n^{\log_2 8}}_{n^3})$ $\therefore T(n) \in \Theta(n^3)$

So why is it interesting?

$$\begin{bmatrix} A & B \\ C & D \end{bmatrix} \cdot \begin{bmatrix} E & F \\ G & H \end{bmatrix} = \begin{bmatrix} AE+BG & AF+BH \\ CE+DG & CF+DH \end{bmatrix}$$

Strassen's crazy idea:

$\exists$ 8 $\frac{n}{2} \times \frac{n}{2}$ - sized matrix products to compute,

but they are not independent . . .

can we do fewer matrix mults and get all 8

Submatrices?

$P_1 = A \cdot (F - H)$

$P_2 = (A + B) \cdot H$

$P_3 = (C + D) \cdot E$

$P_4 = D \cdot (G - E)$

$P_5 = (A + D) \cdot (E + H)$

$P_6 = (B - D) \cdot (G + H)$

$P_7 = (A - C) \cdot (E + F)$

Compute the above. Can we now extract

$AE + BG$, $AF + BH$, $CE + DG$, $CF + DH$?

$$\boxed{AE + BG} = P_5 + P_4 - P_2 + P_6$$

$$= (A+D)\cdot(E+H) + D\cdot(G-E) - (A+B)\cdot H + (B-D)\cdot(G+H)$$

$$= AE + \cancel{AH} + \cancel{DE} + \cancel{DH} + \cancel{DG} - \cancel{DE} - \cancel{AH} - \cancel{BH} + BG + \cancel{BH} - \cancel{DG} - \cancel{DH}$$

$$AF + BH = P_1 + P_2$$

$$CE + DG = P_3 + P_4$$

$$CF + DH = P_1 + P_5 - P_3 - P_7$$

Running time of Strassen's Matrix Mult Algorithm . . . .

complexity? $\Theta(n^2)$.

... which uses many Matrix additions

and 7 matrix mults $\left(\frac{n}{2} \times \frac{n}{2}\right)$

as a function of n, for two $n \times n$ matrices:

$$T(n) = 7\,T\left(\frac{n}{2}\right) + \underline{n^2}$$

$$n^2 \in \quad \left(n^{\log_2 7}\right)$$

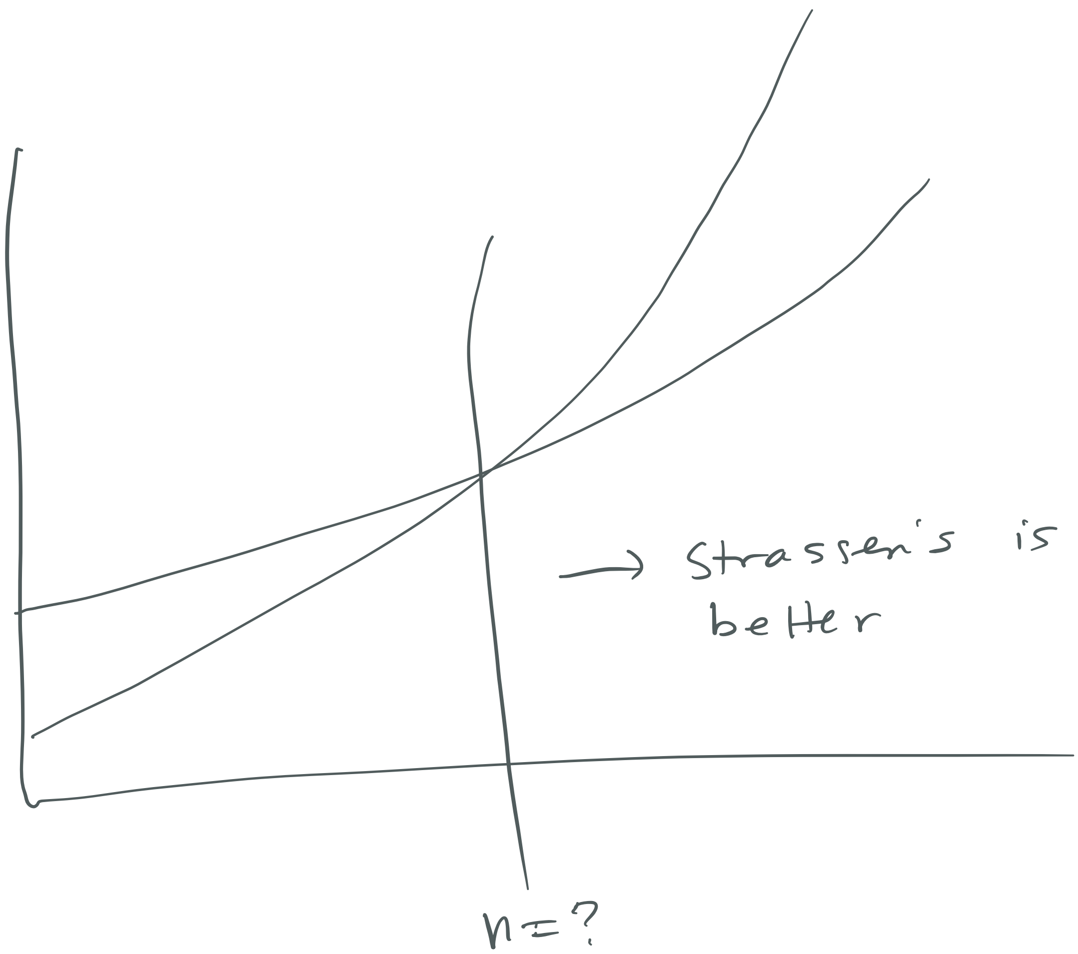
ie $\quad n^2 \in O\left(n^{2.8074-\varepsilon}\right)$

Hence by case $\underline{1}$ of Master Theorem,

Strassen's Matrix mult algorithm runs in

$$\Theta\left(n^{2.8074}\right)$$

$\blacksquare$


Strassen's is better
n=?