

COL 351 : ANALYSIS & DESIGN OF ALGORITHMS

LECTURE 5

DIVIDE & CONQUER IV :

COUNTING INVERSIONS AND MATRIX MULTIPLICATION

AUG 04, 2026

|

ROHIT VAISH

COUNTING INVERSIONS

COUNTING INVERSIONS

input: a length n array A of distinct integers

output: the number of **inversions** of A

COUNTING INVERSIONS

input: a length n array A of distinct integers

output: the number of **inversions** of A

\
pairs (i, j) of array indices with
 $i < j$ and $A[i] > A[j]$.

COUNTING INVERSIONS

input: a length n array A of distinct integers

output: the number of **inversions** of A

\
pairs (i, j) of array indices with
 $i < j$ and $A[i] > A[j]$.

e.g.,

1	3	5	2	4	6
---	---	---	---	---	---

inversions =

COUNTING INVERSIONS

input: a length n array A of distinct integers

output: the number of **inversions** of A

\
pairs (i, j) of array indices with
 $i < j$ and $A[i] > A[j]$.

e.g.,

1	3	5	2	4	6
---	---	---	---	---	---

inversions =

$i=2, j=4$

COUNTING INVERSIONS

input: a length n array A of distinct integers

output: the number of **inversions** of A

\
pairs (i, j) of array indices with
 $i < j$ and $A[i] > A[j]$.

e.g.,

1	3	5	2	4	6
---	---	---	---	---	---

inversions =

$i=2, j=4$

$i=3, j=4$

COUNTING INVERSIONS

input: a length n array A of distinct integers

output: the number of **inversions** of A

\
pairs (i, j) of array indices with
 $i < j$ and $A[i] > A[j]$.

e.g.,

1	3	5	2	4	6
---	---	---	---	---	---

inversions =

$$i=2, j=4$$

$$i=3, j=4$$

$$i=3, j=5$$

COUNTING INVERSIONS

input: a length n array A of distinct integers

output: the number of **inversions** of A

\
pairs (i, j) of array indices with
 $i < j$ and $A[i] > A[j]$.

e.g.,

1	3	5	2	4	6
---	---	---	---	---	---

inversions = 3

$i=2, j=4$

$i=3, j=4$

$i=3, j=5$

COUNTING INVERSIONS

input: a length n array A of distinct integers

output: the number of **inversions** of A

\
pairs (i, j) of array indices with
 $i < j$ and $A[i] > A[j]$.

e.g.,

1	3	5	2	4	6
---	---	---	---	---	---

inversions = 3

1 3 5 2 4 6
• • • • • • elements

• • • • • •
1 2 3 4 5 6 indices

COUNTING INVERSIONS

input: a length n array A of distinct integers

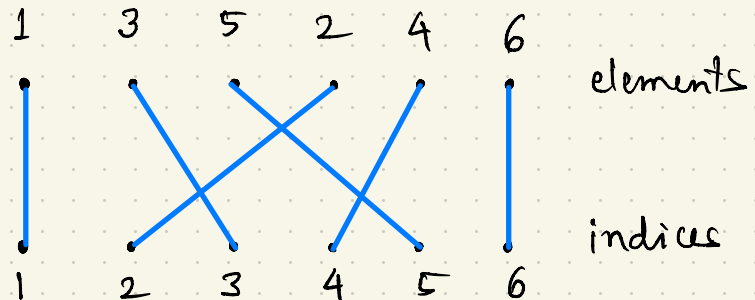
output: the number of **inversions** of A

\n
pairs (i, j) of array indices with
 $i < j$ and $A[i] > A[j]$.

e.g.,

1	3	5	2	4	6
---	---	---	---	---	---

inversions = 3



COUNTING INVERSIONS

input: a length n array A of distinct integers

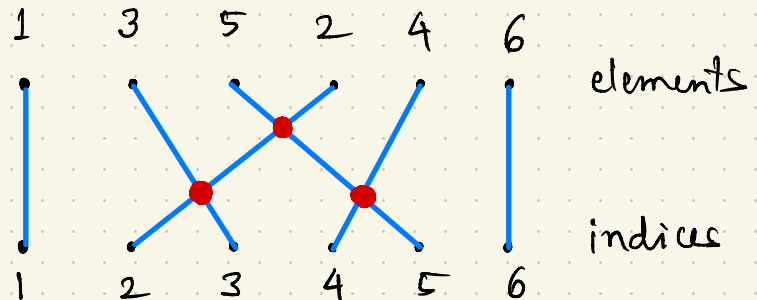
output: the number of **inversions** of A

\n
pairs (i, j) of array indices with
 $i < j$ and $A[i] > A[j]$.

e.g.,

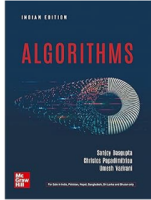
1	3	5	2	4	6
---	---	---	---	---	---

inversions = 3



COUNTING INVERSIONS

Customers who viewed this item also viewed



Algorithms

Sanjoy Dasgupta

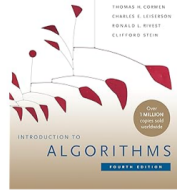
★★★★★ 6

Paperback

₹673.00

Get it by **Wednesday, July 31**

FREE Delivery by Amazon



INTRODUCTION TO
ALGORITHMS, FOURTH
EDITION

Charles E. Leiserson

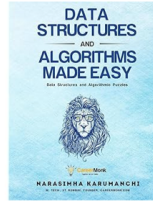
★★★★★ 569

Hardcover

₹3,300.00

FREE Delivery

Only 1 left in stock.



Data Structures And
Algorithms Made Easy:...

> Narasimha Karumanchi

★★★★★ 3,117

Paperback

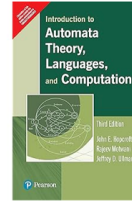
#1 Best Seller in

Algorithms

₹604.00

Get it by **Thursday, August 1**

FREE Delivery by Amazon



Introduction to Automata
Theory, Languages, and
Computation, 3e
Hopcroft

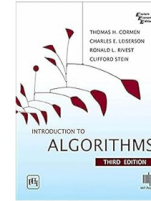
★★★★★ 386

Paperback

₹749.00

Get it by **Wednesday, July 31**

FREE Delivery by Amazon



Introduction to
Algorithms (Eastern
Economy Edition)
> Thomas H. Cormen

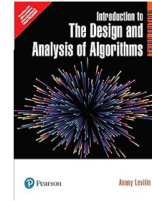
★★★★★ 1,579

Paperback

₹1,650.00

₹160.00 shipping

Only 1 left in stock.



Introduction to
The Design and Analysis of
Algorithms, 3/e
> Anany Levitin

★★★★★ 208

Paperback

₹709.00

Get it by **Wednesday, July 31**

FREE Delivery by Amazon

Collaborative filtering : Similar users get similar recommendations

COUNTING INVERSIONS

Brute force algorithm: check every pair of indices (i, j)

COUNTING INVERSIONS

Brute force algorithm: check every pair of indices (i, j)

— $\theta(n^2)$

COUNTING INVERSIONS

Brute force algorithm: check every pair of indices (i, j)

$\theta(n^2)$

Can we do better?

COUNTING INVERSIONS

Brute force algorithm: check every pair of indices (i, j)

$\theta(n^2)$

Can we do better?

Yes! $O(n \log n)$ algorithm via divide-and-conquer.

COUNTING INVERSIONS

Call an inversion (i, j) where $i < j$

left inversion if $i, j \leq n/2$

right inversion if $i, j > n/2$

split inversion if $i \leq \frac{n}{2} < j$

COUNTING INVERSIONS

Call an inversion (i, j) where $i < j$

left inversion if $i, j \leq n/2$

right inversion if $i, j > n/2$

split inversion if $i \leq \frac{n}{2} < j$

1	3	5	2	4	6
---	---	---	---	---	---

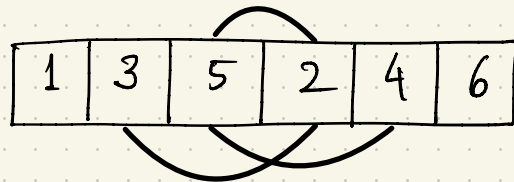
COUNTING INVERSIONS

Call an inversion (i, j) where $i < j$

left inversion if $i, j \leq n/2$

right inversion if $i, j > n/2$

split inversion if $i \leq \frac{n}{2} < j$



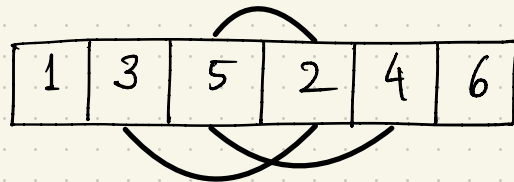
COUNTING INVERSIONS

Call an inversion (i, j) where $i < j$

left inversion if $i, j \leq n/2$

right inversion if $i, j > n/2$

split inversion if $i \leq \frac{n}{2} < j$



all split inversions

COUNTING INVERSIONS

Call an inversion (i, j) where $i < j$

left inversion if $i, j \leq n/2$ 

right inversion if $i, j > n/2$ 

compute these recursively

split inversion if $i \leq \frac{n}{2} < j$  compute these in "combine" step

HIGH-LEVEL ALGORITHM

input: an array A of n distinct integers

output: the number of inversions of A

HIGH-LEVEL ALGORITHM

input: an array A of n distinct integers

output: the number of inversions of A

if $n \leq 1$ **return** 0

HIGH-LEVEL ALGORITHM

input: an array A of n distinct integers

output: the number of inversions of A

```
if n ≤ 1 return 0
```

else $l :=$ recursively count inversions on left half of A

$$h := \text{ " " " right " " }$$
$$s := \text{Count split inversions of } A$$

return l + r + s

HIGH-LEVEL ALGORITHM

input: an array A of n distinct integers

output: the number of inversions of A

```
if n ≤ 1      return 0
```

else $l :=$ recursively count inversions on left half of A

$$h := \text{ " " " " right " " }$$
$$s := \text{Count split inversions of } A$$

return l + r + s

$$A = \left[\frac{n}{2} + 1 \mid \frac{n}{2} + 2 \mid \dots \mid n \mid 1 \mid 2 \mid \dots \mid \frac{n}{2} \right]$$

$$\# \text{ split inversions} = ?$$

$$A = \left[\begin{array}{|c|c|c|c|c|c|c|c|} \hline \frac{n}{2}+1 & \frac{n}{2}+2 & \dots & n & 1 & 2 & \dots & \frac{n}{2} \\ \hline \end{array} \right]$$

$$\# \text{ split inversions} = n^2/4$$

$$A = \left[\frac{n}{2} + 1 \mid \frac{n}{2} + 2 \mid \dots \mid n \mid 1 \mid 2 \mid \dots \mid \frac{n}{2} \right]$$

$$\# \text{ split inversions} = n^2/4$$

Count *quadratic* no. of inversions in *linear*, i.e., $O(n)$, time?

—
 suffices for $O(n \log n)$ time overall



Piggyback on Merge Sort



Piggyback on Merge Sort

Suppose A has **no** split inversion.



Piggyback on Merge Sort

Suppose A has **no** split inversion.

Then, every element in
left half of A $<$ every element in
right half of A



Piggyback on Merge Sort

Suppose A has **no** split inversion.

Then, every element in
left half of A $<$ every element in
right half of A

What does **merge** subroutine do for such array?



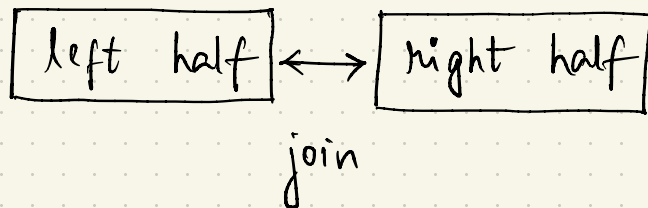
Piggyback on Merge Sort

Suppose A has **no** split inversion.

Then, every element in
left half of A $<$ every element in
right half of A

What does **merge subroutine** do for such array?

Concatenation!



What does merge subroutine do when there **are** split inversions?

What does merge subroutine do when there **are** split inversions?

Consider merging

1	3	5
---	---	---

and

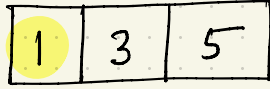
2	4	6
---	---	---

Output

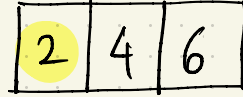
--	--	--	--	--	--

What does merge subroutine do when there **are** split inversions?

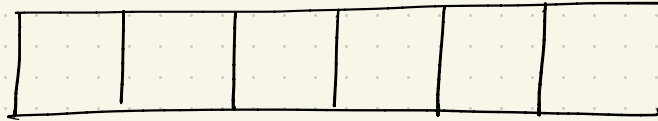
Consider merging



and

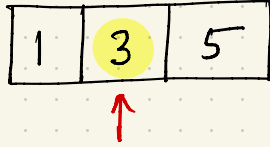


Output

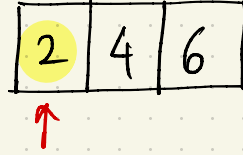


What does merge subroutine do when there **are** split inversions?

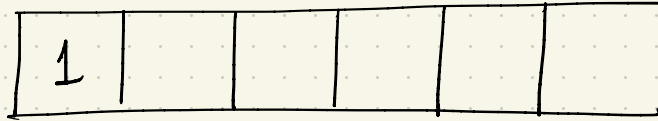
Consider merging



and



Output



What does merge subroutine do when there **are** split inversions?

Consider merging

1	3	5
---	---	---

 and

2	4	6
---	---	---

.

Output

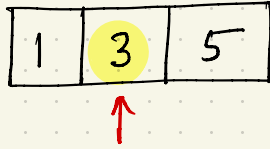
1	2				
---	---	--	--	--	--



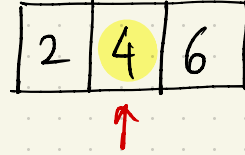
When "2" gets copied in output, the split inversions $(3, 2)$ and $(5, 2)$ are **exposed**.

What does merge subroutine do when there **are** split inversions?

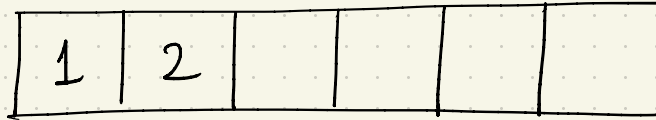
Consider merging



and

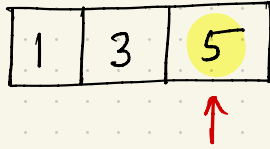


Output

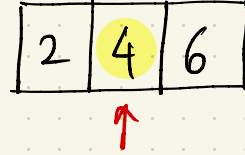


What does merge subroutine do when there **are** split inversions?

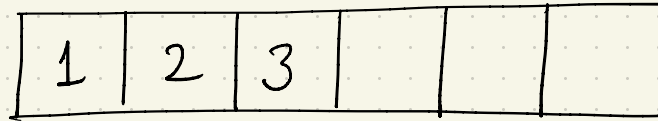
Consider merging



and



Output



What does merge subroutine do when there **are** split inversions?

Consider merging

1	3	5
---	---	---

 and

2	4	6
---	---	---

Output

1	2	3	4		
---	---	---	---	--	--



When "4" gets copied in output, the split inversion $(5, 4)$ is **exposed**.

What does merge subroutine do when there **are** split inversions?

Consider merging

1	3	5
---	---	---

and

2	4	6
---	---	---

↑

Output

1	2	3	4	5	
---	---	---	---	---	--

What does merge subroutine do when there **are** split inversions?

Consider merging

1	3	5
---	---	---

 and

2	4	6
---	---	---

.

Output

1	2	3	4	5	6
---	---	---	---	---	---

Claim: Let A be an array, and let L and R be sorted versions of first and second halves of A .

The elements $x \in L$ and $y \in R$ form a **split inversion** if and only if

$\text{merge}(L, R)$ **copies** y to the output **before** x .

Claim: Let A be an array, and let L and R be sorted versions of first and second halves of A .

The elements $x \in L$ and $y \in R$ form a **split inversion** if and only if

$\text{merge}(L, R)$ **copies** y to the output **before** x .

Proof: Since $x \in L$ and $y \in R$

(x, y) split inversion $\iff y < x \iff y$ copied before x .



ALGORITHM FOR COUNTING INVERSIONS

input: an array A of n distinct integers

output: the number of inversions of A

if $n \leq 1$ **return** 0

else $l :=$ recursively count inversions on left half of A

$r :=$ " " " right " "

$s :=$ Count split inversions of A

return $l + r + s$

ALGORITHM FOR COUNTING INVERSIONS

input: an array A of n distinct integers

output: the number of inversions of A

if $n \leq 1$ **return** 0

else l := recursively count inversions on left half of A

r := " " " right " "

s := Count split inversions of A

return $l + r + s$

ALGORITHM FOR COUNTING INVERSIONS

input: an array A of n distinct integers

output: the number of inversions of A

if $n \leq 1$ **return** 0

else $(L, l) :=$ recursively ^{sort and} count inversions on left half of A

$r :=$ " " " right "

$s :=$ Count split inversions of A

return $l + r + s$

ALGORITHM FOR COUNTING INVERSIONS

input: an array A of n distinct integers

output: the number of inversions of A

if $n \leq 1$ **return** 0

else $(L, l) :=$ recursively ^{sort and} count inversions on left half of A

$r :=$ " " " right "

$s :=$ Count split inversions of A

return $l + r + s$

ALGORITHM FOR COUNTING INVERSIONS

input: an array A of n distinct integers

output: the number of inversions of A

if $n \leq 1$ **return** 0

else $(L, l) :=$ recursively ^{sort and} count inversions on left half of A

$(R, r) :=$ " ^{sort and} " right " "

$s :=$ Count split inversions of A

return $l + r + s$

ALGORITHM FOR COUNTING INVERSIONS

input: an array A of n distinct integers

output: the number of inversions of A

if $n \leq 1$ return 0

else $(L, l) :=$ recursively ^{sort and} count inversions on left half of A

$(R, r) :=$ " ^{sort and} " " right " "

$s :=$ Count split inversions of A

return $l + r + s$

ALGORITHM FOR COUNTING INVERSIONS

input: an array A of n distinct integers

output: the number of inversions of A

if $n \leq 1$ **return** 0

else $(L, l) :=$ recursively ^{sort and} count inversions on left half of A

$(R, r) :=$ " ^{sort and} " right " "

$(B, s) :=$ ^{merge and} count split inversions of A (using L and R)

return $l + r + s$

ALGORITHM FOR COUNTING INVERSIONS

input: an array A of n distinct integers

output: the number of inversions of A

if $n \leq 1$ return 0

else $(L, l) :=$ recursively ^{sort and} count inversions on left half of A

$(R, r) :=$ " ^{sort and} " right " "

$(B, s) :=$ ^{merge and} Count split inversions of A (using L and R)

return $l + r + s$

How to implement this in linear time?

RECALL MERGE SUBROUTINE

```
initialize  $i := 1$   
initialize  $j := 1$   
for  $k := 1$  to  $n$   
    if  $L[i] < R[j]$  then  
         $B[k] := L[i]$   
        increment  $i$  by 1  
    else  
         $B[k] := R[j]$   
        increment  $j$  by 1  
return  $B$ 
```

MERGE - AND - COUNT - SPLIT - INVERSIONS

```
initialize  $i := 1$   
initialize  $j := 1$   
for  $k := 1$  to  $n$   
    if  $L[i] < R[j]$  then  
         $B[k] := L[i]$   
        increment  $i$  by 1  
    else  
         $B[k] := R[j]$   
        increment  $j$  by 1  
  
return  $B$ 
```

MERGE-AND-COUNT-SPLIT-INVERSIONS

initialize $s := 0$

initialize $i := 1$

initialize $j := 1$

for $k := 1$ to n

if $L[i] < R[j]$ then

$B[k] := L[i]$

increment i by 1

else

$B[k] := R[j]$

increment j by 1

$s := s + \underbrace{\left(\frac{n}{2} - i + 1\right)}_{\# \text{ left in } L}$

return (B, s)

Theorem : For every input array A of length $n \geq 1$,
our algorithm correctly computes the number
of inversions of A and runs in $O(n \log n)$ time.

Theorem : For every input array A of length $n \geq 1$,
our algorithm correctly computes the number
of inversions of A and runs in $O(n \log n)$ time.

Proof : Exercise

TAKEAWAY

Sometimes *strengthening* the induction hypothesis can help!
($\text{sort} \rightarrow \text{sort} + \text{count}$)

MATRIX MULTIPLICATION

MATRIX MULTIPLICATION

input: Two $n \times n$ matrices X and Y of integers

output: the product $Z = X \cdot Y$ where
$$Z_{ij} = \sum_{k=1}^n x_{ik} y_{kj}$$

MATRIX MULTIPLICATION

input: Two $n \times n$ matrices X and Y of integers

output: the product $Z = X \cdot Y$ where $Z_{ij} = \sum_{k=1}^n x_{ik} y_{kj}$

The diagram illustrates the calculation of a single element Z_{ij} in the product matrix Z . It shows the dot product of a row from matrix X and a column from matrix Y .

Matrix X is represented by a horizontal green line, with the label "row i " to its left. Below it is the letter X .

Matrix Y is represented by a vertical green line, with the label "col j " above it. Below it is the letter Y .

The result is matrix Z , represented by a square bracket containing a green dot. An arrow points from the label Z_{ij} to this dot. Below it is the letter Z .

The equation is written as: $\left[\text{row } i \right] \cdot \left[\text{col } j \right] = \left[\text{dot} \right] = Z_{ij}$

MATRIX MULTIPLICATION

input: Two $n \times n$ matrices X and Y of integers

output: the product $Z = X \cdot Y$ where $Z_{ij} = \sum_{k=1}^n x_{ik} y_{kj}$

The diagram illustrates the calculation of a single element Z_{ij} in the product matrix Z . It shows the dot product of a row from matrix X and a column from matrix Y .

Matrix X is represented by a horizontal green line, with the label "row i " to its left. Below it is the letter X .

Matrix Y is represented by a vertical green line, with the label "col j " above it. Below it is the letter Y .

The result is matrix Z , represented by a square bracket with a green dot inside. An arrow points from the label Z_{ij} to this dot. Below it is the letter Z .

The equation is written as: $\left[\text{row } i \right] \cdot \left[\text{col } j \right] = \left[\text{dot} \right]$

input size: $O(n^2)$

MATRIX MULTIPLICATION

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

$X \qquad \qquad Y \qquad \qquad Z$

MATRIX MULTIPLICATION

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} \cdot \begin{bmatrix} e & f \\ g & h \end{bmatrix} = \begin{bmatrix} ae + bg & \\ & \end{bmatrix}$$

$X \qquad Y \qquad Z$

MATRIX MULTIPLICATION

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} \cdot \begin{bmatrix} e & f \\ g & h \end{bmatrix} = \begin{bmatrix} ae + bg & af + bh \end{bmatrix}$$

$X \qquad Y \qquad Z$

MATRIX MULTIPLICATION

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

$X \qquad Y \qquad Z$

MATRIX MULTIPLICATION

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

$X \qquad Y \qquad Z$

MATRIX MULTIPLICATION

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

Running time for $n \times n$ matrices : ?

MATRIX MULTIPLICATION

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

Running time for $n \times n$ matrices : $O(n^3)$

assuming multiplication/addition of two numbers takes $O(1)$ time

MATRIX MULTIPLICATION

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

$X \qquad \qquad Y \qquad \qquad Z$

Running time for $n \times n$ matrices : $O(n^3)$

Can we do better?

APPLYING DIVIDE & CONQUER

APPLYING DIVIDE & CONQUER

$$X = \left[\begin{array}{c|c} L & R \end{array} \right]_{n \times n}$$

non-square matrices



APPLYING DIVIDE & CONQUER

$$X = \left[\begin{array}{c|c} A & B \\ \hline C & D \end{array} \right]_{n \times n}$$

A, B, C, D are $\frac{n}{2} \times \frac{n}{2}$ matrices

(assuming n is even)

APPLYING DIVIDE & CONQUER

$$X = \begin{bmatrix} A & B \\ C & D \end{bmatrix}_{n \times n}$$

$$Y = \begin{bmatrix} E & F \\ G & H \end{bmatrix}_{n \times n}$$

APPLYING DIVIDE & CONQUER

$$X = \begin{bmatrix} A & B \\ C & D \end{bmatrix}_{n \times n}$$

$$Y = \begin{bmatrix} E & F \\ G & H \end{bmatrix}_{n \times n}$$

$$\text{Then, } X \cdot Y = \begin{bmatrix} A \cdot E + B \cdot G & A \cdot F + B \cdot H \\ C \cdot E + D \cdot G & C \cdot F + D \cdot H \end{bmatrix}$$

verify!

RECURSIVE ALGORITHM

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

RECURSIVE ALGORITHM

Step 1: Recursively compute the eight matrix products

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

RECURSIVE ALGORITHM

Step 1: Recursively compute the eight matrix products

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

Step 2: Do the necessary additions ($\Theta(n^2)$ time)

RECURSIVE ALGORITHM

Step 1: Recursively compute the eight matrix products

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

Step 2: Do the necessary additions ($\Theta(n^2)$ time)

Fact: Above algorithm has running time $\Theta(n^3)$ (by Master theorem)
— coming soon!

RECURSIVE ALGORITHM

Step 1: Recursively compute the eight matrix products

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

Step 2: Do the necessary additions ($\Theta(n^2)$ time)

Fact: Above algorithm has running time $\Theta(n^3)$ (by Master theorem)
— coming soon!

Is there a way to save a recursive call?

STRASSEN'S ALGORITHM

STRASSEN'S ALGORITHM

Step 1: Recursively compute only seven matrix products

STRASSEN'S ALGORITHM

Step 1: Recursively compute only seven matrix products

Step 2: Do the necessary additions and subtractions (still $\theta(n^2)$ time)

STRASSEN'S ALGORITHM

Step 1: Recursively compute only seven matrix products

Step 2: Do the necessary additions and subtractions (still $\Theta(n^2)$ time)

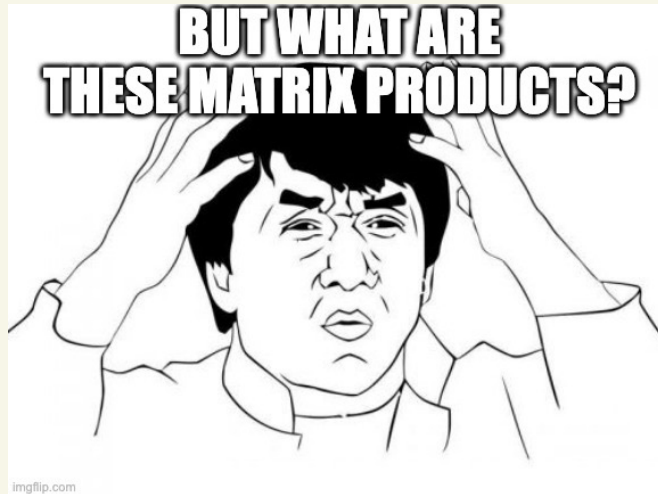
Spoiler alert: Better than $O(n^3)$ time (by Master theorem)

STRASSEN'S ALGORITHM

Step 1: Recursively compute only seven matrix products

Step 2: Do the necessary additions and subtractions (still $\Theta(n^2)$ time)

Spoiler alert: Better than $O(n^3)$ time (by Master theorem)



STRASSEN'S ALGORITHM

$$X = \left[\begin{array}{c|c} A & B \\ \hline C & D \end{array} \right]_{n \times n} \quad Y = \left[\begin{array}{c|c} E & F \\ \hline G & H \end{array} \right]_{n \times n}$$

$$X \cdot Y = \left[\begin{array}{cc} A \cdot E + B \cdot G & A \cdot F + B \cdot H \\ C \cdot E + D \cdot G & C \cdot F + D \cdot H \end{array} \right]$$

STRASSEN'S ALGORITHM

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

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

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

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

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

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

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

$$X = \left[\begin{array}{c|c} A & B \\ \hline C & D \end{array} \right]_{n \times n} \quad Y = \left[\begin{array}{c|c} E & F \\ \hline G & H \end{array} \right]_{n \times n}$$

$$X \cdot Y = \left[\begin{array}{cc} A \cdot E + B \cdot G & A \cdot F + B \cdot H \\ C \cdot E + D \cdot G & C \cdot F + D \cdot H \end{array} \right]$$

STRASSEN'S ALGORITHM

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

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

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

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

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

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

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

$$X = \left[\begin{array}{c|c} A & B \\ \hline C & D \end{array} \right]_{n \times n} \quad Y = \left[\begin{array}{c|c} E & F \\ \hline G & H \end{array} \right]_{n \times n}$$

$$X \cdot Y = \left[\begin{array}{cc} A \cdot E + B \cdot G & A \cdot F + B \cdot H \\ C \cdot E + D \cdot G & C \cdot F + D \cdot H \end{array} \right]$$

Claim: $X \cdot Y = \left[\begin{array}{cc} P_5 + P_4 - P_2 + P_6 & P_1 + P_2 \\ P_3 + P_4 & P_1 + P_5 - P_3 - P_7 \end{array} \right]$

STRASSEN'S ALGORITHM

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

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

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

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

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

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

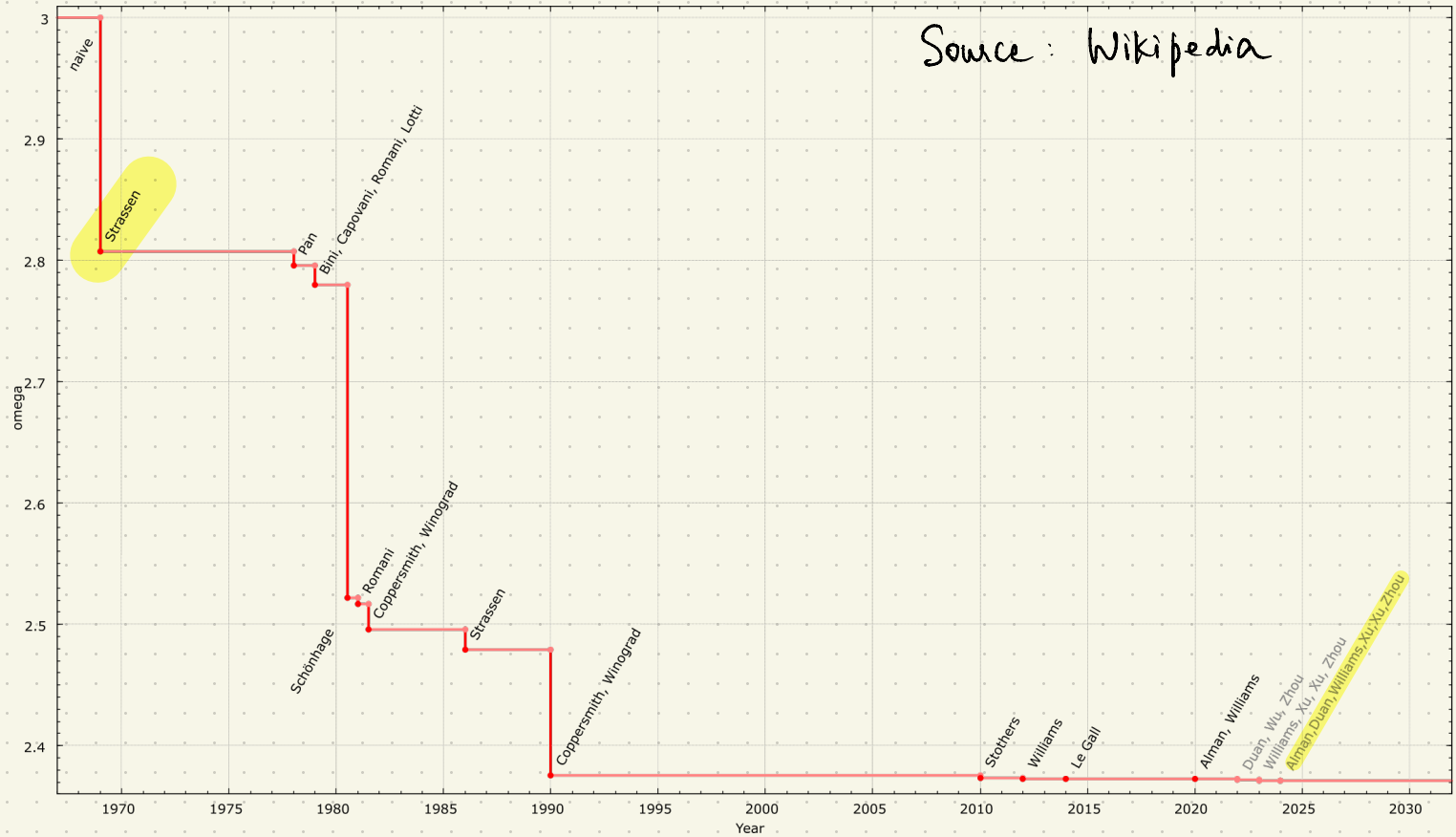
$$P_7 = (A - C) \cdot (E + F)$$

$$X = \begin{bmatrix} A & B \\ C & D \end{bmatrix}_{n \times n} \quad Y = \begin{bmatrix} E & F \\ G & H \end{bmatrix}_{n \times n}$$

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

Claim: $X \cdot Y = \begin{bmatrix} P_5 + P_4 - P_2 + P_6 & P_1 + P_2 \\ P_3 + P_4 & P_1 + P_5 - P_3 - P_7 \end{bmatrix}$ Strassen's algorithm saves a recursive call!

MATRIX MULTIPLICATION



MATRIX MULTIPLICATION

