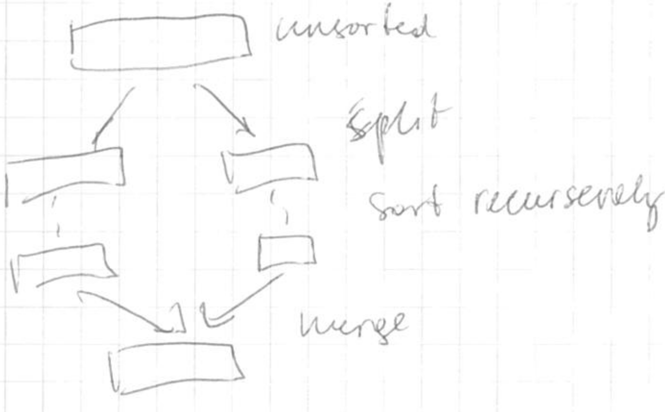
# Algorithms: Design and Analysis P 1

## Week 1

### Merge Sort

most transparent example of Divide & Conquer

Input: Unsorted   output: Sorted



Merge:

C = output [len = n]

A = 1st arr , B = 2nd array // sorted

i = j = 1

for k = 1 to n

    if A(i) < B(j)

        C(k) = A(i)

        i++

    else [B(j) < A(i)]

        C(k) = B(j)

        j++

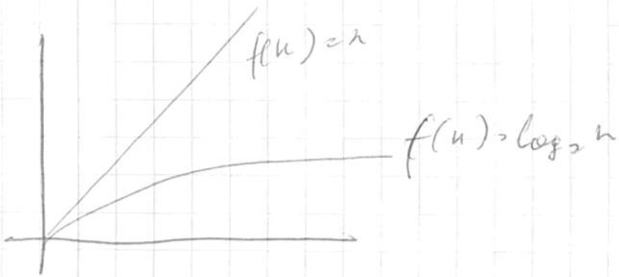
end

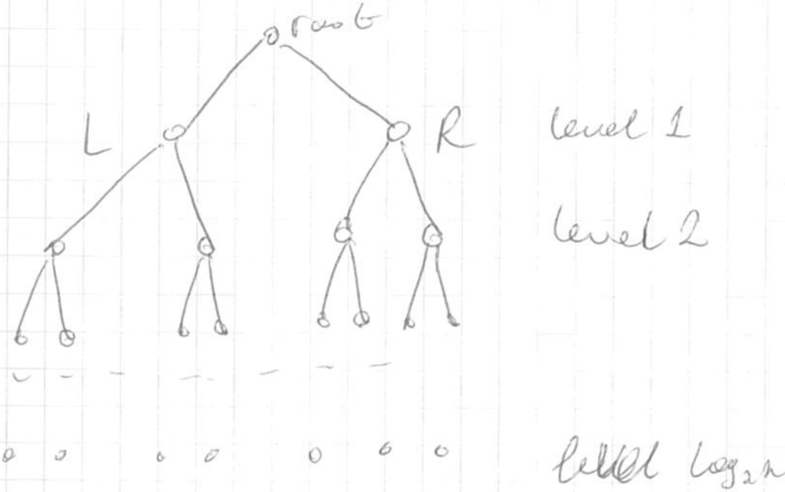
running time

$\leq 4m + 2$

$\leq 6m$

MSort requires $\leq 6n \log_2 n + 6n$ permuts
to sort n numbers




Total # of operations on level j

$$\le \underbrace{2^j}_{\text{\# of level-}j\text{ subproblems}} \cdot 6\Big(\underbrace{\frac{n}{2^j}}_{\text{subproblem size at level } j}\Big) = 6n \text{ (independent of the level!)}$$

number of levels: $\log_2 n + 1$

$$\Downarrow$$

$$\le \underline{6n(\log_2 n + 1)}$$

# Guided principles

1. „Worst-case analysis"

over running time

for general-purpose

~~2.~~ Average-case analysis

benchmarks ← you have to have domain knowledge

2 don't ~~play~~ pay attention to constants

- easier
-

3. Asymtotic analysis

for large input sets sizes of n

$$\underbrace{6n\log_2 n + 6n}_{\text{Merge Sort}} \quad \text{"better than"} \quad \underbrace{\frac{1}{2}n^2}_{\text{Insertion}}$$

↓

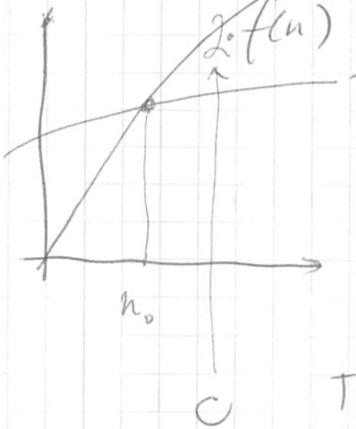
for large n!

Fast Algorithm -

those worst-case running time grows slowly for input size

## Big Oh

$T(n) = O(f(n))$




eventually $T(n)$ is bound above $f(n)$

if only there exists constants $\underbrace{c, n_0 > 0}_{\text{independent of } n}$ such that

$$T(n) \le c \cdot f(n)$$
for all $n \geq n_0$

(1) if $T(n) = a_k n^k + \ldots + a_1 n + a_0$

(polynom)

$$\Downarrow$$

$$T(n) = O(n^k)$$

Proof: $T(n) \le |a_k| n^k + \ldots + |a_1| n + |a_0|$

$\le |a_k| n^k + \ldots + a_1 n^k + |a_0| n^k$

$\le c \cdot n^k$

QED

② for every $k \geq 1$, $n^k$ is not $O(n^{k-1})$

Proof. by contradiction

Suppose $n^k = O(n^{k-1})$

then

$n^k \leq c\,n^{k-1} \quad \forall n \geq n_0$

$\Downarrow$

$n \leq c$ not true.

[contradiction]

Big Omega and Theta

Omega $\quad T(n) = \Omega(f(n))$

$T(n) \geq c\,f(n) \quad \forall n \geq 0, \forall c.$

Theta

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

if $T(n) = O(f(n))$ and

$T(n) = \Omega(f(n))$

(sandwich between $O$ and $\Omega$)

Let $T(n) = \frac{1}{2}n^2 + 3n$

$T(n) = \Omega(n)$

$T(n) = \Theta(n^2)$

$T(n) = O(n^3)$

<u>Little-Oh</u>

$T(n) = o(f(n))$

$T(n) \leq c \cdot f(n) \quad \forall n \geq n_0$

for all $c > 0$

# Divide & Conquer

## Counting Inversions

① Divide
into smaller
subproblems

② Conquer
using recursion

Clean up

③ Combine

Input: Array A
containing 1..n
in some arbitrary order

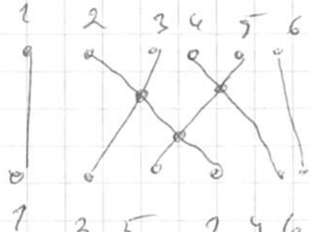
Output: number of
inversions

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

1 3 5 2 4 6

Inversions

(3,2) (5,2) (5,4




number of crosses –
number of inversions

motivation

how closed two ranked lists?

(2 friends with movies)

eg. "collaborative filtering"

The largest possible number of inversions:

$$\binom{n}{2} = \cancel{C^n_2}\ C^2_n = \frac{n(n-1)}{2}$$

for 6: $C^2_6 = \dfrac{6(6-1)}{2} = 3 \cdot 5 = 15$

Options

① Brute Force $\Theta(n^2)$ time

Can we do better?

② D&C

For inversion: $C(i,j)$

$$\begin{cases} \text{Left inversion} & \text{if } i, j \le n/2 \\ \text{Right inversion} & \text{if } i, j > n/2 \\ \text{Split inversion} & \text{if } i \le \frac{n}{2} < j \end{cases}$$

L and R can be separated

for S a separate subroutine is needed

Count ( array A, length n )

if n = 1 return 0

else

X = Count ( $1^{st}$ half of A, $\frac{n}{2}$ )

Y = Count ( $2^{nd}$ half, $\frac{n}{2}$ )

Z = Count Split ( A, n )

return x + y + z

Count Split should be linear to get running time $O(n \log n)$

Idea: have recursive calls both count inversions and sort

(Merge subroutine naturally uncovers split inversions)

Rename Count ⇒ Sort-and-Count

Sort-and-Count (array A, len n)

if $n=1$ return 0

← sorted version of 1st half

$(B, X) = $ S-a-C (1st half, $\frac{n}{2}$)

$(C, Y) = $ S-a-C (2nd half, $\frac{n}{2}$)

$(D, Z) = $ Count Split Inv (A, n)

sorted version of ~~2nd~~ A ~~half~~

sorted version of 2nd half

Example

all inversions are split

Consider merging

L

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

and

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

R

When 2 copied to output, discovers the split inversions (3, 2) and (5, 2)

When 4 copied to output, discover (5, 4)

(When in R less than in L – inversion!)

Claim: the split inversions involving an element y of the 2nd array C are precisely the numbers left in the 1st array B when y is copied to the output D.

Proof: Let $x$ be an element of the 1st array B

1. if $x$ copied to D before $y$, then $x < y$

   $\Rightarrow$ no inversions

2. if $y$ copied to D before $x$, then

   $y < x$,

   $\Rightarrow$ $x$ and $y$ are a (split) inversion

Q.E.D.

Merge-and-Count Split Inv

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

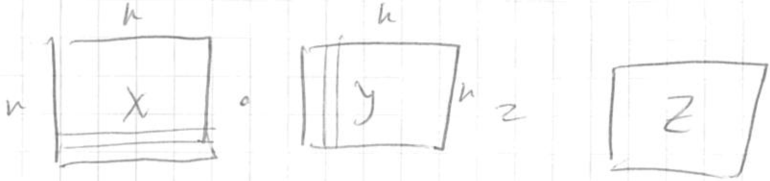
- when element of 2nd array C is copied to output D, increment total ~~number~~ by number of elements remaining in 1st array B

$$\overset{\text{merge}}{O(n)} + O(n) = O(n)$$

$$\Downarrow$$

Sort_And-Count runs $O(n \log n)$

# Matrix Multiplication




$Z_{ij} = (i^{th} \text{ row} \cdot X)(j^{th} \text{ col of } Y) =$

$$= \sum_{k=1}^{n} x_{ik} \cdot y_{kj}$$

$\begin{pmatrix} \text{input size} \\ \Theta(n^2) \end{pmatrix}$

D&C

1. Divide
2. Conquer
3. Combine

$$X = \left(\begin{array}{c|c} A & B \\ \hline C & D \end{array}\right)$$

$$Y = \left(\begin{array}{c|c} E & F \\ \hline G & H \end{array}\right)$$

$A - H$ – $n/2 \times n/2$ matrices

$$X \cdot Y = \left(\begin{array}{c|c} AE + BG & AF + BH \\ \hline CE + DG & CF + DH \end{array}\right)$$

Step 1: recursively compute 8 products

Step 2: do additions ( $\Theta(n^2)$ time )

Strassen's Algorithm

Step 1 recursively compute 7 products

Step 2 do (clever) additions ($O(n^2)$ time)

Better than cubic
(see Master Method!)

7 products:

not important

$P_1 = A(F-H)$ $P_2 = (A+B)H$

$P_3 = (C+D)E$ $P_4 = D(G-E)$

$P_5 = (A+D)(E+H)$ $P_6 = (B-D)(G+H)$

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

...

$$X \cdot Y = \begin{pmatrix} AE+BG & AF+BH \\ CE+DG & CF+DH \end{pmatrix} =$$

$$\begin{pmatrix} P_5+P_4-P_2+P_6 & P_1+P_2 \\ P_3+P_4 & P_1+P_5-P_3-P_7 \end{pmatrix}$$