WEEK 2 L1

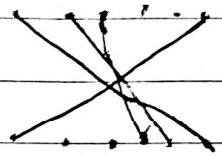
1. Divide into smaller sub-problems
2. Conquer via recursive calls
3. Combine solutions of subproblems into one for the original problem

EXAMPLE 1

Input: Array A containing numbers 1, 2, 3, ..., in some arbitrary order

Output: number of inversions = number of pairs (a, b) such that a precedes b and $a > b$

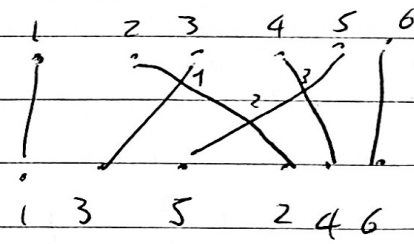
((..i..j..), A[i] > A[j])



EXAMPLE 2

Example 1 ~~1 3 5 2 4 6~~ (1, 3, 5, 2, 4, 6)

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



Motivation: numerical similarity measure between two Lists eg:

Online shopping, people that bought your stuff also bought:

Largest number of inversions in a size n set is:

$$\binom{n}{2} = \frac{n(n-1)}{2}$$

10

Approaches:

Brute force double loop, $O(n^2)$

Can we do better? Yes, with divide and conquer.

Call an inversion (i, j):

x ① - left inversion if $i, j \leq n/2$ → can compute recursively

y ② - right if $i, j > n/2$

z ③ - split if $i \leq n/2$, $j > \frac{n}{2}$ How? Clear up.

For [1, 3, 5, 2, 4, 6] ① and ② both return 0!

So all of our inversions are split and come from ③

if z is linear, $O(n \log(n))$

L2

To do that, we'll piggyback on Merge sort.

For every recursive call, we don't only count the inversions but also order the array.

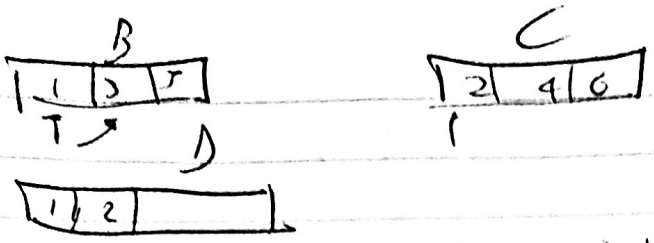
We can do that because it doesn't change the number of split inversions

z merges in time n, ~~every time the ... the ...~~

[1|3|5] [2|4|6]

i j

every time you compare i and j, if i < j you move i to the right. if i > j you count 3 and move j to the right



when copying 2 from the right array, you are exposing 2 different splits! (3,2) and (5,2).

Later when 8 gets copied it only exposes 1.

Q General Asin:

The split mergings involving an element y of the 2nd (or bell) array C are precisely the numbers left in the first array (sorted) A when y is copied to the output d

This is because they are ordered.

Proof: let ~~x be~~ $x \in B$,

① if x is copied to D before y, $x < y$, no mergings

② if y is copied to D before x, then $y < x$, ~~not all remain~~ since $x \le x_{n} \le x_{n-1} \le \dots \le x_{n}$ additional splits are exposed

Run time is $O(n) + O(n) = O(n)$

L3

We'll be talking about $n \times n$ matrices, but applies to non-square ones as well

$$\boxed{x} \cdot \boxed{y} = \boxed{z}$$

$$z_{ij} = \begin{pmatrix} \text{ith row} \\ \text{of } x \end{pmatrix} \cdot \begin{pmatrix} \text{jth column} \\ \text{of } y \end{pmatrix} = \sum_{k=1} x_{ik} \cdot y_{kj}$$

12

Note: We are led to believe that the best algorithm for this will run $n \geq \Omega(n^2)$.

~~BUT~~

@ $\Theta(n^3)$,

n elements by n elements to compute, each requiring n operations.

DIV n CONQ:

$$x = \left(\begin{array}{c|c} A & B \\ \hline C & D \end{array}\right) \quad y = \left(\begin{array}{c|c} E & F \\ \hline G & H \end{array}\right)$$

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

① 8 products

② 2 additions ($\Theta(n^2)$ time)

Fact: ② Run time is still $\Theta(n^3)$

DIFFERENT APPROACH? ~~NEEDED~~

④ Step 1: 7 products ~~is the goal~~ with Strassen

Step 2: find all ~~Still~~ $\Theta(n^2)$ time

more adds and subs but

The seven Products are:

$P_1 = A(F-H)$

$P_2 = (A+B)H$

$P_3 = (C+D)E$

$P_4 = D(G-E)$

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

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

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

$$X \cdot Y = \left(\begin{array}{c|c} AE+BG & AF+BH \\ \hline CE+DG & CF+DH \end{array} \right) = \left(\begin{array}{c|c} P_5+P_4-P_2+P_6 & P_1+P_2 \\ \hline P_3+P_4 & P_1+P_5-P_3-P_7 \end{array} \right)$$

$$AE + AH + DE + DH + DG - DE - AH - BH + BG + DH - BG - DH$$

$$= AE + BG$$

$$AF - AH + AE + AH + DE + DH - CE - DE - AE + AF + CE + CF$$

L6

L6

The master method is a general tool to determine the running time of D&C Algorithms.

A Recursive Algorithm

Express $T(n)$ in terms of running time of recursive calls.

Base case: $T(1) \leq$ a constant

For all $n > 1$: $T(n) \leq 4\,T\left(\frac{n}{2}\right) + O(n)$

(Work done by rec. calls)

A Better Recursive Algorithm

Algorithm #2 (Gauss): recursively compute ac, bd, $(a+b)(c+d)$

Base Case: $T(1) \leq k$

For all $n > 1$ $T(n) \leq 3\,T\left(\frac{n}{2}\right) + O(n)$

L7

General Master Method

Assumption: all of subproblems have same size

Recurrence format:

① Base Case: $T(n) \leq$ a constant for all sufficiently small n

② for all larger n:

$$T(n) \leq a\,T\left(\frac{n}{b}\right) + O\left(n^d\right)$$

a = number of recursive calls

b = factor by which input problem shrinks

d = exponent in the running time outside of recursive calls.

("combine step" (and split) of the original problem if it takes time)

(independent of n)

$T(n)$ is upper bounded by 1 of 3 cases.

$$T(n) = \begin{cases} O(n^d \log(n)) & \text{if } a = b^d \quad \text{1st case} \\ O(n^d) & \text{if } a < b^d \quad \text{2nd case} \\ O(n^{\log_b(a)}) & \text{if } a > b^d \quad \text{3rd case} \end{cases}$$

L7

1) Example 1: Merge Sort

$a = 2$ (2 calls)

$b = 2$ (size of subinput = size input / 2)

$d = 1$ (Linear time)

$a = b^d \longrightarrow O(n^d \log(n)) = O(n \log(n))$

2) For binary search of sorted array:

$(a, b, d) = 1, 2, 0$ ← one comparison, not dependent on n

$a = 1$ call, half the size $\quad b^d = 2^0 = 1 = a$

$O(n^d \log(n)) \xrightarrow{d=0} O(\log(n))$

3) Multiplication of integers:

Without Gauss's trick:

$a = 4$

$b = 2$

$c = 1$

$b^c < a$, $T(n) = O(n^{\log_b(a)}) = O(n^{\log_2 4}) = O(n^2)$

With:

$a = 3$

$b = 2$

$c = 1$

$O(n^{\log_b(a)}) = O(n^{\log_2 3}) = O(n^{1.5})$

5) Strassen's Matrix Multiplication

$a = 7$
$b = 2$
$d = 2$

$b^d < a$ since $4 < 7$

Case 3

$T(n) = O(n^{\log_2 7}) = O(n^{2.81})$ instead of $O(n^3)$

6) Fictitious Recurrence

$T(n) \geq 2T\left(\frac{n}{2}\right) + O(n^2)$

$a = 2$
$b = 2$
$d = 2$

$b^d = 4 > 2 \Rightarrow$

Case 2

$T(n) = O(n^d) = O(n^2)$

Proof of the master method

Proof of the master method

$$T(n) \leq a\,T\left(\frac{n}{b}\right) + O(n^d)$$

Preamble:

Assume recurrence is

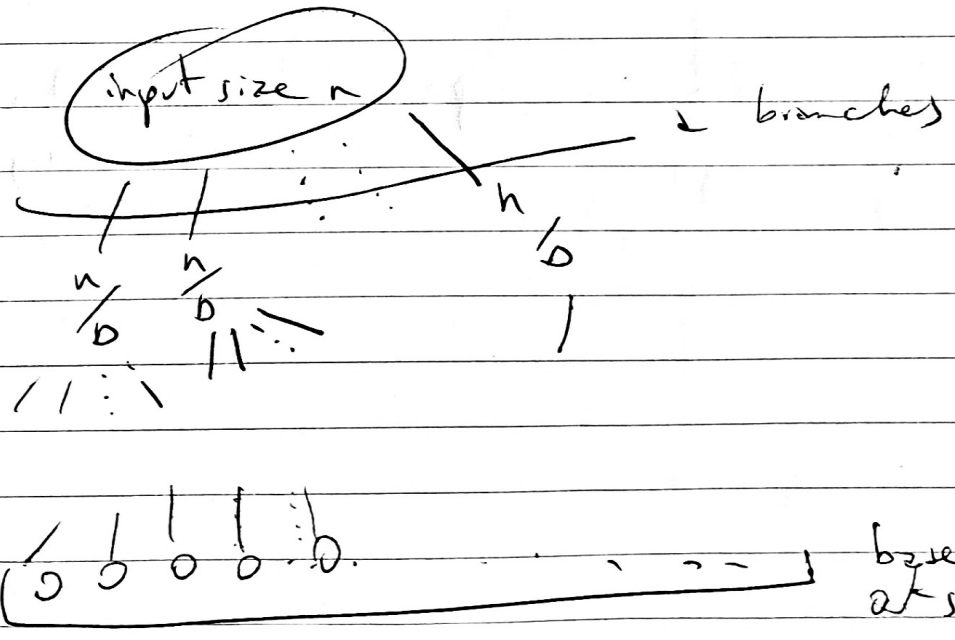
i) $T(1) \leq c$

ii) $T(n) \leq a\,T\left(\frac{n}{b}\right) + cn^d$ (for some constant c which we previously ignored)

and n is a power of b (general case more tedious)

Idea: generalize analysis used with merge sort (a recursion tree)

The Rec Tree:



Total work at level j (ignoring work in recursive calls)

is $\leq a^j \cdot c\left(\left\lceil \frac{n}{b^j} \right\rceil\right)^d$ (work per level j subproblem)

# of level j subproblems (size of each level j subproblem)

$$= cn^d \cdot \left(\frac{a}{b^d}\right)^j$$

Sum over all levels:

$$\text{total work} \le c n^d \cdot \sum_{j=0}^{\log_b n} \left[\frac{a}{b^d}\right]^j$$

TBC

Upper bound on work at level j: $\lfloor L \rfloor 8'$

$$c n^d \cdot \left(\frac{a}{b^d}\right)^j$$

Interpretation $\frac{a}{b^d}$

Tug of war between forces of good (b, d) and of evil (

a = rate at which subproblems proliferate as you go deeper in the recursive tree (RSP)

b^d = rate of work shrinkage (n^d time for a problem of size n, size shrinks by b, work $\propto n^d$, $n \propto \frac{1}{b}$, work $\propto \left(\frac{1}{b}\right)^d$, work shrinkage is $\propto b^d$

Intuition for 3 Cases

① RSP = RWS same amount of work at each level $O(n^d \log(n))$ same work at root d & l ...

② RSP < RWS less work each level (most work at root) $O(n^d)$ because n^d dominates over later work done in sub problems

③ RSP > RWS, the amount of work increases at each sublevel, leaves of tree dominate.
We might expect $O(\#\text{ leaves})$, but $\propto$ ~~# leaves~~ ~~input~~ ... on 8 calls, or ... (levels ... root to bottom) (???)

Completion of the proof of the master method. L10

Total work $\leq \boxed{cn^d} \cdot \boxed{\sum_{j=0}^{\log_b(n)} \left(\frac{a}{b^d}\right)^j}$ (*)

1) If $a = b^d$, then $= 1$, $= \log_b(n)$, (*) $= cn^d \log_b(n) = O(n^d \log n)$

Remember: $1 + r + r^2 + \dots + r^K = \frac{r^{K+1} - 1}{r - 1}$

For sums, if $r < 1$ is constant, $\sum r^K \leq \left(\frac{1}{1-r}\right)$ = a constant.
if $r > 1$ is constant, $\leq r^K \cdot \left(1 + \frac{1}{r-1}\right)$ — {independent of K, levels}

2) $a < b^d$, so $r < 1$, (*) $\leq cn^d \cdot \sum_{k=1}^{\text{whatever}} (r)^k \leq cn^d \cdot \text{constant}$
$r < 1$
$= O(n^d)$

3) (*) $\leq cn^d \cdot \sum_{j=0}^{\log_b(n)} (r)^j \leq cn^d \cdot \left(\frac{a}{b^d}\right)^{\log_b(n)} \cdot \text{constant} =$
$r > 1$

$c_2 = c \cdot \text{constant}$

$\leq c_2 n^d \cdot (a)^{\log_b(n)} \cdot b^{-d \log_b n} = cn^d \cdot (a)^{\log_b(n)} \cdot n^{-d}$

$= O\left(a^{\log_b(n)}\right) = O\left(n^{\log_b(a)}\right)$

$a^{\log_b(n)} = b^{\log_b(a) \log_b(n)} = n^{\log_b(a)}$

Problem Scribblings for WEEK 2.

$a = 7$
$b = 3$
$d = 2$
$b^d = 9 > 7 = a$, case 2, $O(n^2)$

22

$a = 9$
$b = 3$
$d = 2$

$a = b^d = 9$ so case 1 $O(n^d \log(n)) = (n^2 \log(n))$

$a = 5$
$b = 3$
$d = 1$

~~$a < b^d$, case 2, $O(n^d)$~~

$5 = a > b^d = 3$, case 3, $O(n^{\log_b(a)}) = O(n^{\log_3(5)})$

$a = 1$
$b = 2$
$c = 0$

$n^d \log(n) = \log(n)$

$T(n) < T(\frac{n}{2})$

for $n = \log m$ $m = \log n$ $n = 2^{\log n}$ $T(2^{\log n}) < T(2^{(\log n)/2})$

$T(m)$

$a = 1$
$b = 2$
$d = 0$