Announcements

Announcements

- HW1 is due tonight
  - Note on P3: you can treat the instruments as identical $\underline{a}\ \underline{\quad}\ \underline{b} = \underline{b}\ \underline{\quad}\ \underline{a}$
  - If you already solved under the assumption that they are not identical, that's ok- just make a note of your assumption
- HW2 will be posted today

Announcements

- The first set of oral exams will be scheduled the week of March 15
- Later this week I will send an e-mail with information about signing up for a 30-minute examination slot
- Procedure:
  - On March 7 (Sunday), I will post 6 questions
  - You need to submit written solutions by March 14. – don't need to detailed
  - During your examination period, the examiner (me or a TA) will pick 2 of the problem
  - You will describe your solution to that problem, and answer follow-up questions from the examiner. Your solution should match the written solution you submitted (so people going later in the week don't have an advantage)
  - The examiner will then ask you to find and explain the solution to a modified version of the problem
  - Your grade will be based solely on your performance in the oral exam

$\boxed{1.5}$

Ok if hand-written

Divide-and-Conquer

Overview

A divide-and-conquer algorithm has three main steps:

equivalent to larger problem

1. Break the problem into smaller subproblems
2. Recursively solve the subproblems
3. Combine the subproblem solutions to solve the original problem

In-Class Exercise: Depth of a Binary Tree

binary

- Given a tree *T*, design a divide-and-conquer algorithm to find it's depth (the maximum distance from the root to a node).

~~GetHe~~

GetDepth (T):
  if T has no nodes:
    return 0
  else:
    return max (GetDepth (left subtree),
              GetDepth (right subtree)) + 1

$O(n)$ where n = # nodes

In-Class Exercise

Matrix Multiplication

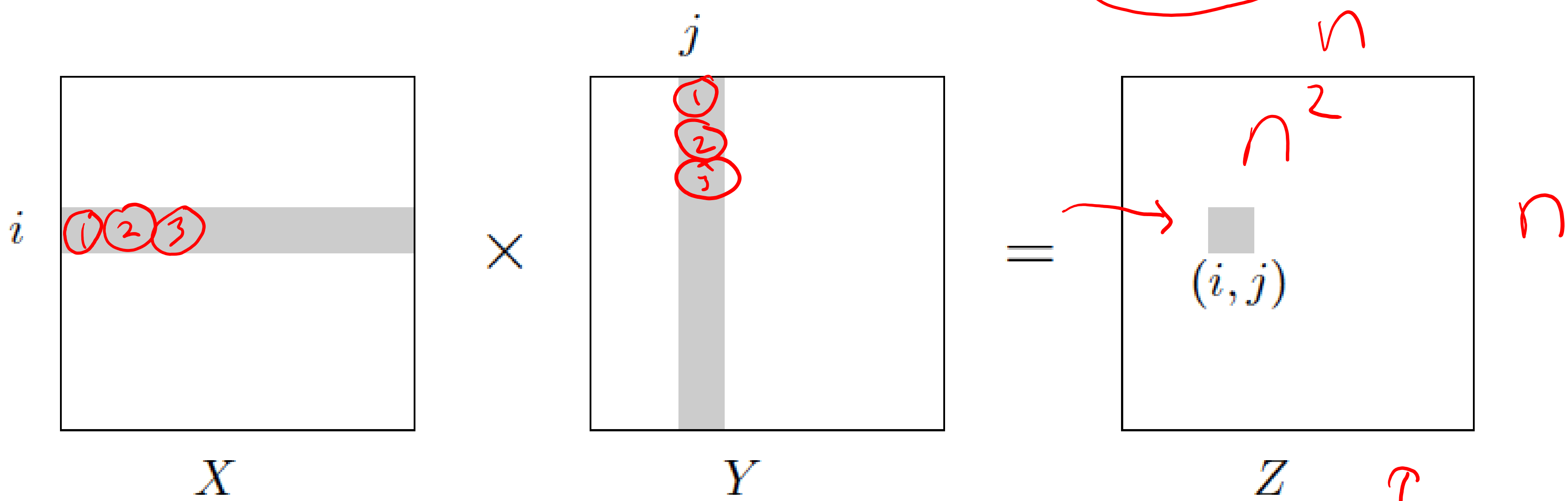
n rows, n cols

The standard algorithm:

matrix addition = $O(n^2)$

$O(n)$

$$Z_{ij} = \sum_{k=1}^{n} X_{ik} Y_{kj}.$$



What is the running time? $O(n^3)$: n^2 elements in Z, each takes $O(n^2)$ time

Matrix Multiplication

MMult(X,Y)

A more efficient method was discovered by Volker Strassen in 1969!

$n/2 \times n/2$

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

MMult(A,E)

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

$T(n)$ = running time of MMult on two $n \times n$ matrices

What is the running time of this divide-and-conquer strategy?

$T(n) = 8T(\frac{n}{2}) + O(n^2)$

$a = 8 \quad b = 2 \quad d = 2$

$\log_b a = 3 > d : O(n^3)$

Matrix Multiplication

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

But it turns out that there is a way to decompose the problem into 7 subproblems, not 8!

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

$$\begin{aligned} P_1 &= A(F - H) \\ P_2 &= (A + B)H \\ P_3 &= (C + D)E \\ P_4 &= D(G - E) \end{aligned} \qquad \begin{aligned} P_5 &= (A + D)(E + H) \\ P_6 &= (B - D)(G + H) \\ P_7 &= (A - C)(E + F) \end{aligned}$$

Prev:
8 mult
4 add

Strassen:
7 mult
18 add/sub

What is the running time?

$T(n) = 7T(\frac{n}{2}) + O(n^2)$

$O(n^{\log_2 7} = n^{2.8})$

In-Class Exercise

10

- MaxSubarraySum problem
  - Given an array *S* with real numbers (some may be negative), find indices *i*, *j* with *i* < *j* such that the sum of values *S*[*i*], …, *S*[*j*] is maximized
  - Hint: how can you combine results from subarrays?

$[1, -2, \underbrace{5, -1, 6}]$

$O(n^3) \left\langle \begin{array}{l} O(n^2) \text{ pairs } (i,j) \\ O(n) \text{ time for each pair} \end{array} \right.$

S = [L | R]

slow method
consider all pairs (i, j): i < j
take sum of vals in between
return best i, j

In-Class Exercise

```
MSS(S):
  L = MSS(left half)
  R = MSS(right half)
  M = best subarray
        that crosses mid
  return best of L, R, M

# Finding the best
  that spans midpoint

# first, find best piece on right
sum = 0; best = 0
for i = mid → length(S):
    sum += S[i]
    if sum > best:
        best = sum
        best_j = i
```

$S = [\ [\ \ ]\ ;\ [\ \ ]\ ]$

$T(n) = 2T(\frac{n}{2}) + O(n)$

$a=2, b=2, d=1$

$O(n \log n)$

best

$O(n)$

$O(n)$

best

In-Class Exercise: Fast Sorting

$O(n + k \log k) < O(n \log y)$

- Suppose you are given an unsorted array of *N* integers, and you know that the smallest value is *x* and the largest is *y*.

x x+1 y

C = [ ]

0 y-x

- Assume that *y – x* is very small compared to *N*
- Can you sort this in better than O(*N* log *N*) time?

k = y-x

S = [ ]

O(k) 2. Iterate through C, remove 0 values

O(k log k) 3. Sort what is left, Use counts to create sorted S.

1. Create a "counter" to count how often each value appears. O(n) time to iterate through S.