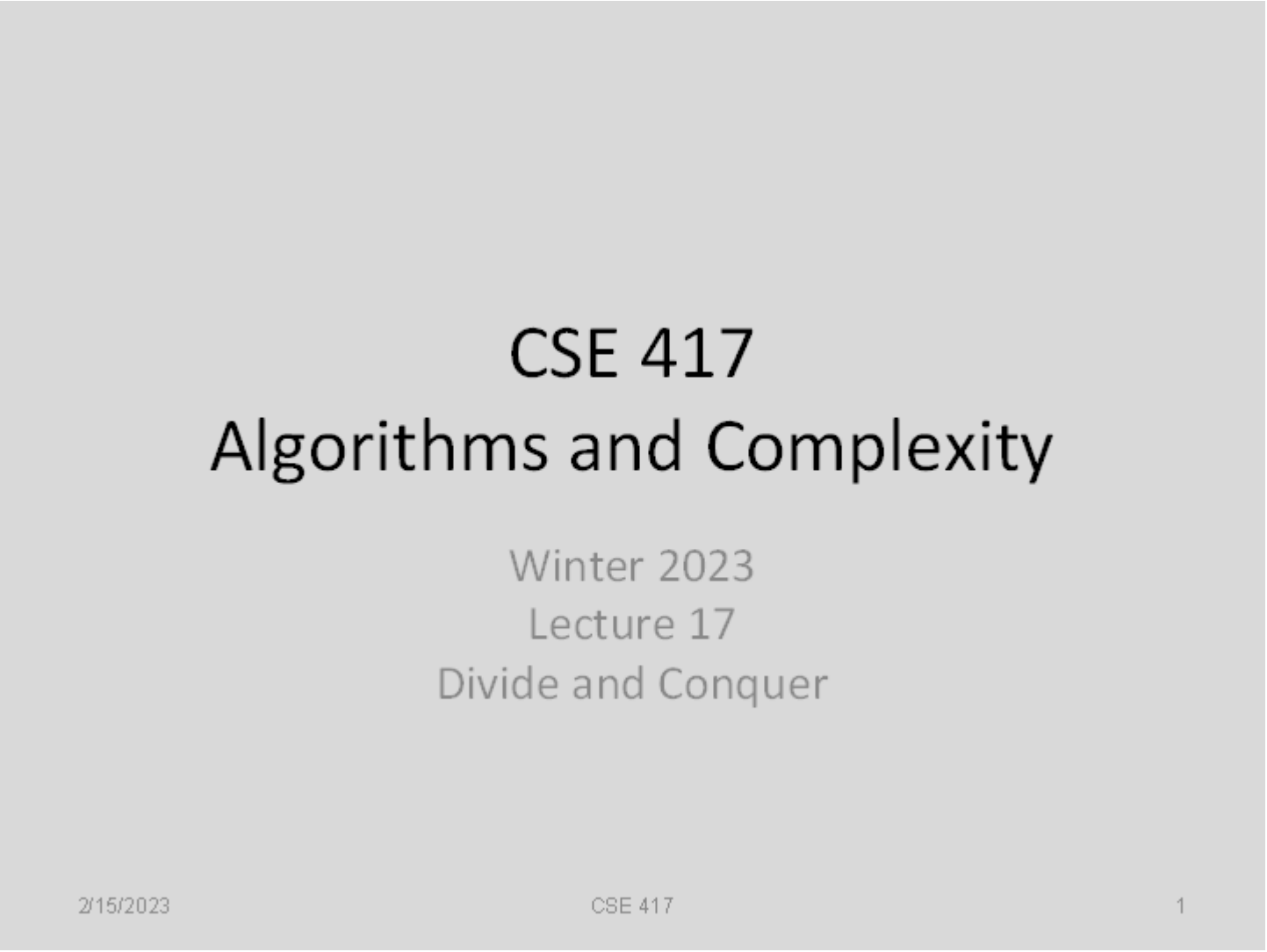
Lecture17


CSE 417
Algorithms and Complexity
Winter 2023
Lecture 17
Divide and Conquer
2/15/2023
CSE 417
1

Announcements

- Midterm stats (out of 60)
  - Mean: 40.46, Median: 42.0, Std Dev: 11.23

- Today: Divide and Conquer
- Friday: Dynamic Programming
- Monday: Presidents' Day

Divide and Conquer

- Algorithm paradigm
  - Break problems into subproblems until easy to solve
  - Work is split between "divide", "combine", and "base'' components
- Standard examples
  - MergeSort and QuickSort
- Analysis tool: Recurrences

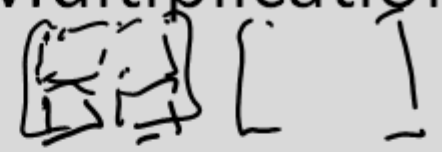
Matrix Multiplication

- N X N Matrix, A B = C

```
for (int i = 0; i < n; i++)
      for (int j = 0; j < n; j++) {
            int t = 0;
            for (int k = 0; k < n; k++)
                  t = t + A[i][k] * B[k][j];
            C[i][j] = t;
      }
```

$O(n^3)$

Recursive Matrix Multiplication



Multiply 2 x 2 Matrices:

$$\begin{vmatrix} r & s \\ t & u \end{vmatrix} = \begin{vmatrix} a & b \\ c & d \end{vmatrix} \begin{vmatrix} e & g \\ f & h \end{vmatrix}$$

$$\left.\begin{aligned} r &= ae + bf \\ s &= ag + bh \\ t &= ce + df \\ u &= cg + dh \end{aligned}\right\} \quad 8 \times \quad 4 +$$

A N x N matrix can be viewed as a 2 x 2 matrix with entries that are (N/2) x (N/2) matrices.

The recursive matrix multiplication algorithm recursively multiplies the (N/2) x (N/2) matrices and combines them using the equations for multiplying 2 x 2 matrices

Recursive Matrix Multiplication

- How many recursive calls are made at each level? 8 calls on $\frac{n}{2} \times \frac{n}{2}$
- How much work in combining the results? $O(n^2)$
- What is the recurrence? $T(n) = 8T(\frac{n}{2}) + n^2$

What is the run time for the recursive Matrix Multiplication Algorithm?

- Recurrence: $T(n) = 8\,T(\frac{n}{2}) + n^2$

n^2

$8\left(\frac{n}{2}\right)^2 = 2n^2$

$8^{\log n} = 2^{3\log n} = n^3$

8 mult
4 add

Strassen's Algorithm

7 mult, 18 add

Multiply 2 x 2 Matrices:

$$\begin{vmatrix} r & s \\ t & u \end{vmatrix} = \begin{vmatrix} a & b \\ c & d \end{vmatrix} \begin{vmatrix} e & g \\ f & h \end{vmatrix}$$

$r = p_1 + p_2 - p_4 + p_6$

$s = p_4 + p_5$

$t = p_6 + p_7$

$u = p_2 - p_3 + p_5 - p_7$

Where:

$p_1 = (b - d)(f + h)$

$p_2 = (a + d)(e + h)$

$p_3 = (a - c)(e + g)$

$p_4 = (a + b)h$

$p_5 = a(g - h)$

$p_6 = d(f - e)$

$p_7 = (c + d)e$

From Aho, Hopcroft, Ullman 1974

Recurrence for Strassen's Algorithms

$n \geq 64$

- $T(n) = 7\,T(n/2) + cn^2$
- What is the runtime?

$$T(n) = 7^{\log n} = 2^{\log n \cdot \log 7}$$
$$= n^{\log 7}$$
$$\approx n^{2.81}$$
$$n^{2.04\ldots}$$

$\log_2 7 = 2.8073549221$

Strassen's Algorithms

- Treat n x n matrices as 2 x 2 matrices of n/2 x n/2 submatrices
- Use Strassen's trick to multiply 2 x 2 matrices with 7 multiplies
- Base case standard multiplication for single entries
- Recurrence: $T(n) = 7\,T(n/2) + cn^2$
- Solution is $O(7^{\log n}) = O(n^{\log 7})$ which is about $O(n^{2.807})$

Inversion Problem

- Let $a_1, \ldots a_n$ be a permutation of 1 . . n
- (a_i, a_j) is an inversion if i < j and $a_i > a_j$

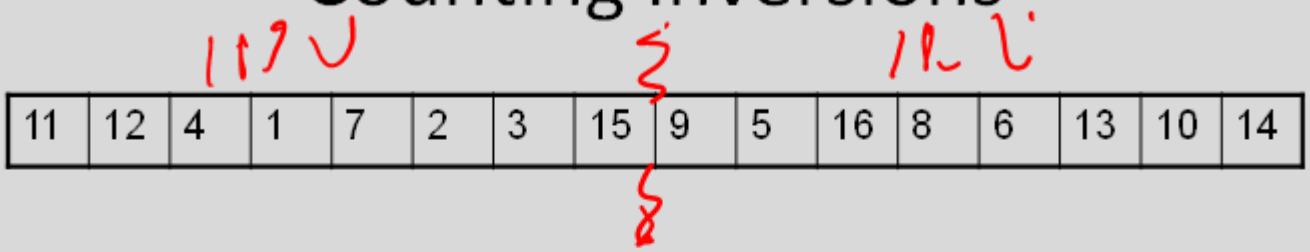
4, 6, 1, 7, 3, 2, 5

- Problem: given a permutation, count the number of inversions
- This can be done easily in $O(n^2)$ time
  - Can we do better?

Application

- Counting inversions can be use to measure how close ranked preferences are
  - People rank 20 movies, based on their rankings you cluster people who like that same type of movie

Counting Inversions

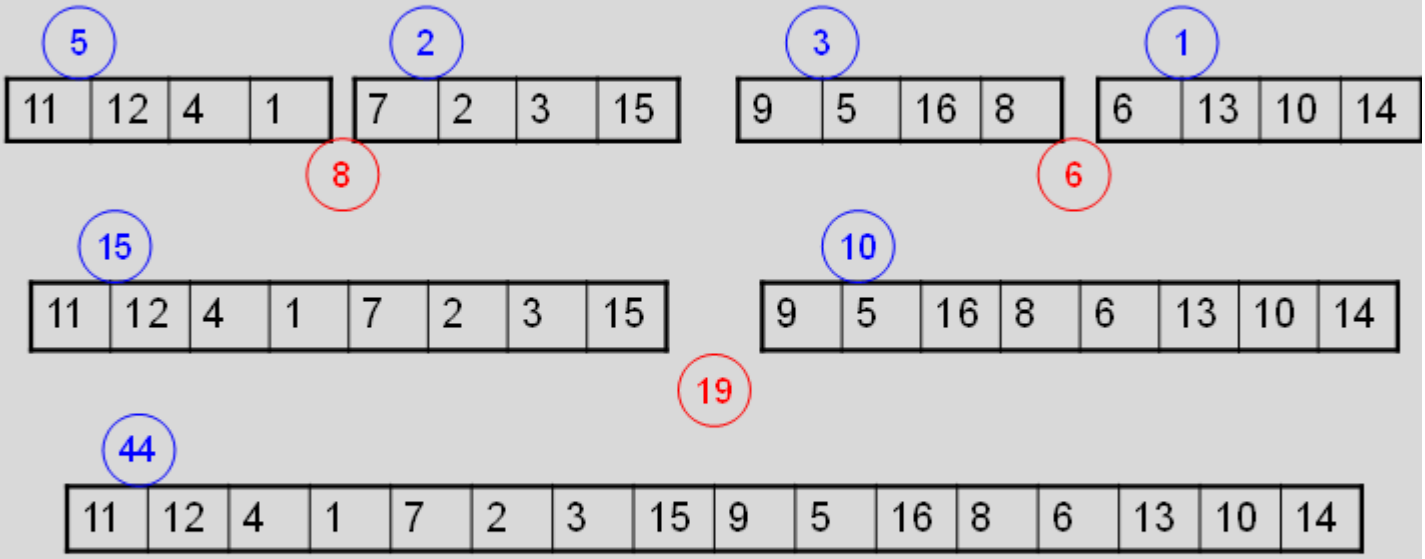


Count inversions on lower half

Count inversions on upper half

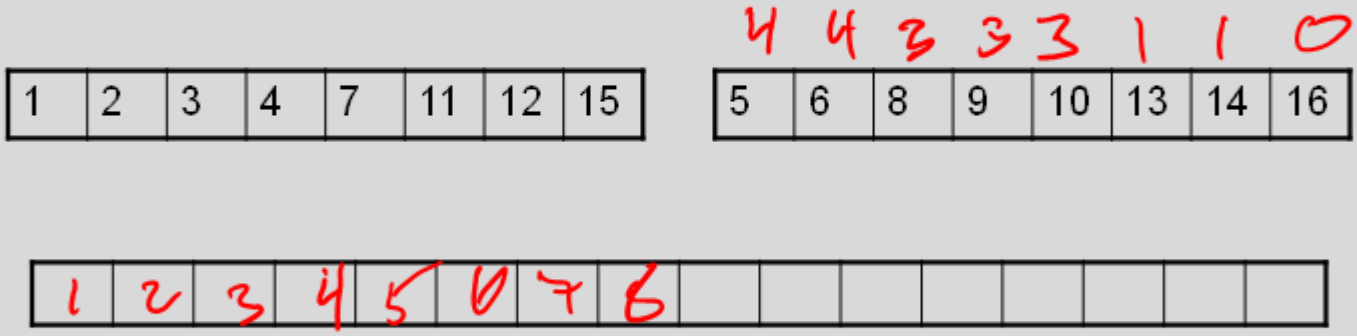
Count the inversions between the halves

Count the Inversions



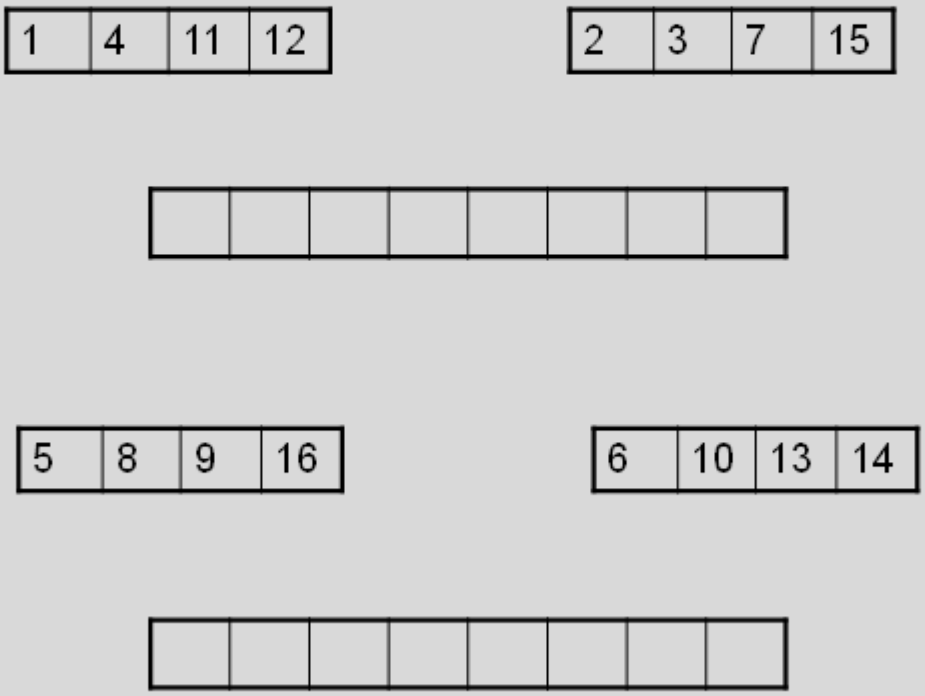
Problem – how do we count inversions between sub problems in O(n) time?

- Solution – Count inversions while merging



Standard merge algorithm – add to inversion count when an element is moved from the upper array to the solution

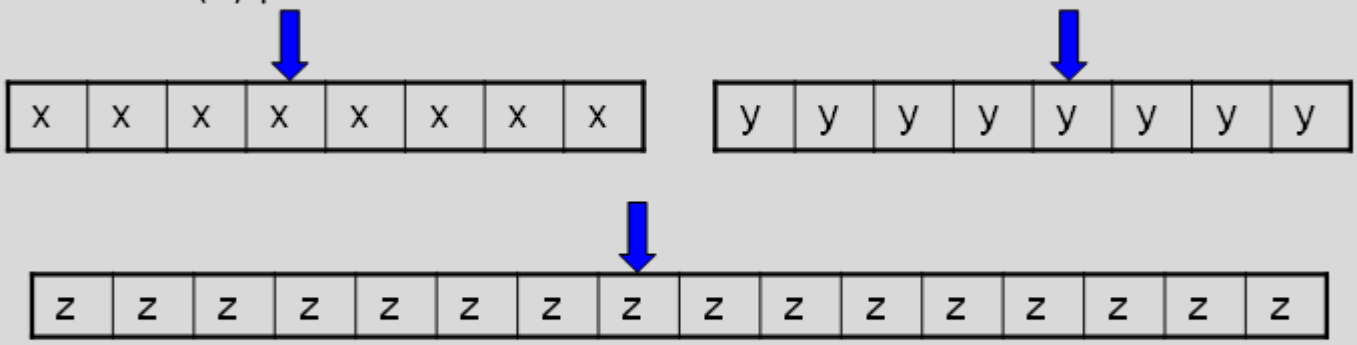
Use the merge algorithm to count inversions



Indicate the number of inversions for each element detected when merging

Inversions

- Counting inversions between two sorted lists
  - O(1) per element to count inversions



- Algorithm summary
  - Satisfies the "Standard recurrance"
  - $T(n) = 2\,T(n/2) + cn$ $= O(n \log n)$

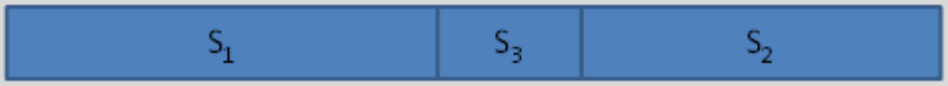
Computing the Median

- Given n numbers, find the number of rank n/2
- One approach is sorting
  - Sort the elements, and choose the middle one
  - Can you do better?

- *Selection*, given n numbers and an integer k, find the k-th largest

Select(A, k)

```
Select(A, k){
        Choose element x from A
        S1 = {y in A | y < x}
        S2 = {y in A | y > x}
        S3 = {y in A | y = x}
        if (|S2| >= k)
                return Select(S2, k)
        else if (|S2| + |S3| >= k)
                return x
        else
                return Select(S1, k - |S2| - |S3|)
}
```



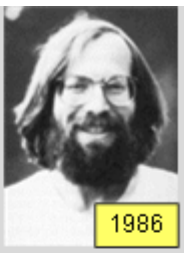
Deterministic Selection

- Random pivot gives an expected O(n) run time. The question of a deterministic algorithm was more challenging.
- What is the run time of select if we can guarantee that *ChoosePivot* finds an x such that $|S_1| < 3n/4$ and $|S_2| < 3n/4$ in O(n) time?

$$T(n) = T(\tfrac{3}{4}n) + n$$

O(n)

BFPRT Algorithm



- A very clever choose algorithm . . .
- Deterministic algorithm that guarantees that $|S_1| < 3n/4$ and $|S_2| < 3n/4$
- Actual recurrence is:

$$T(n) \leq T(3n/4) + T(n/5) + c\,n$$

Integer Arithmetic

```
  971548028394508438309485670104364384579021796570295 6767
+ 124243109823409905732907509717989843092877957927759 7977
```

O(n)

Runtime for standard algorithm to add two n digit numbers:

O(n²)

```
  209506709303468099431859684686779409766717133476767930
X 592017509177763470967767934292909701230895667999301092 0
```

Runtime for standard algorithm to multiply two n digit numbers:

Recursive Multiplication Algorithm (First attempt)

$x = x_1 2^{n/2} + x_0$

$y = y_1 2^{n/2} + y_0$

$xy = (x_1 2^{n/2} + x_0)(y_1 2^{n/2} + y_0)$

$= x_1 y_1 2^n + (x_1 y_0 + x_0 y_1) 2^{n/2} + x_0 y_0$

Recurrence: $T(n) = 4T(\frac{n}{2}) + n$

Run time: $O(n^2)$

2/15/2023 CSE 417 23

Simple algebra

$$x = x_1 2^{n/2} + x_0$$

$$y = y_1 2^{n/2} + y_0$$

$$xy = x_1y_1 2^n + (x_1y_0 + x_0y_1) 2^{n/2} + x_0y_0$$

(1) (2)

x_1y_1 , x_0y_0

(3)

$$p = (x_1 + x_0)(y_1 + y_0) = x_1y_1 + x_1y_0 + x_0y_1 + x_0y_0$$

(4) $- x_1y_1 - x_0y_0$

Karatsuba's Algorithm

Multiply n-digit integers x and y

Let $x = x_1 2^{n/2} + x_0$ and $y = y_1 2^{n/2} + y_0$

Recursively compute

$a = x_1 y_1$

$b = x_0 y_0$

$p = (x_1 + x_0)(y_1 + y_0)$

Return $a2^n + (p - a - b)2^{n/2} + b$

$3^{\log n} = n^{\log 3}$

$= n^{1.58}$

Recurrence: $T(n) = 3T(n/2) + cn$

$\log_2 3 = 1.58496250073\ldots$