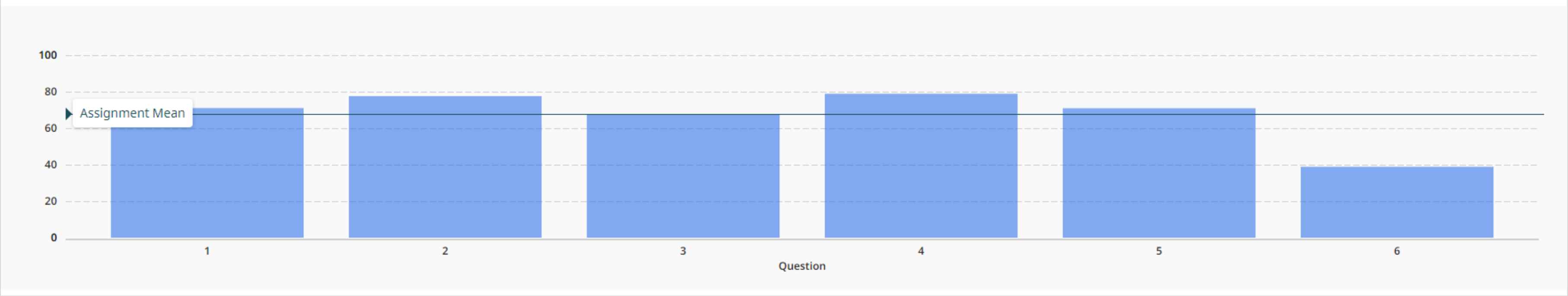
CSE 417
Algorithms and Complexity

Winter 2023
Lecture 17
Divide and Conquer

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;
        }
```

Recursive Matrix Multiplication

Multiply 2 x 2 Matrices:

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

r = ae + bf
s = ag + bh
t = ce + df
u = cg + dh

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?

- How much work in combining the results?

- What is the recurrence?

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

- Recurrence:

Strassen's Algorithm

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

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

$\log_2 7 = 2.8073549221$

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

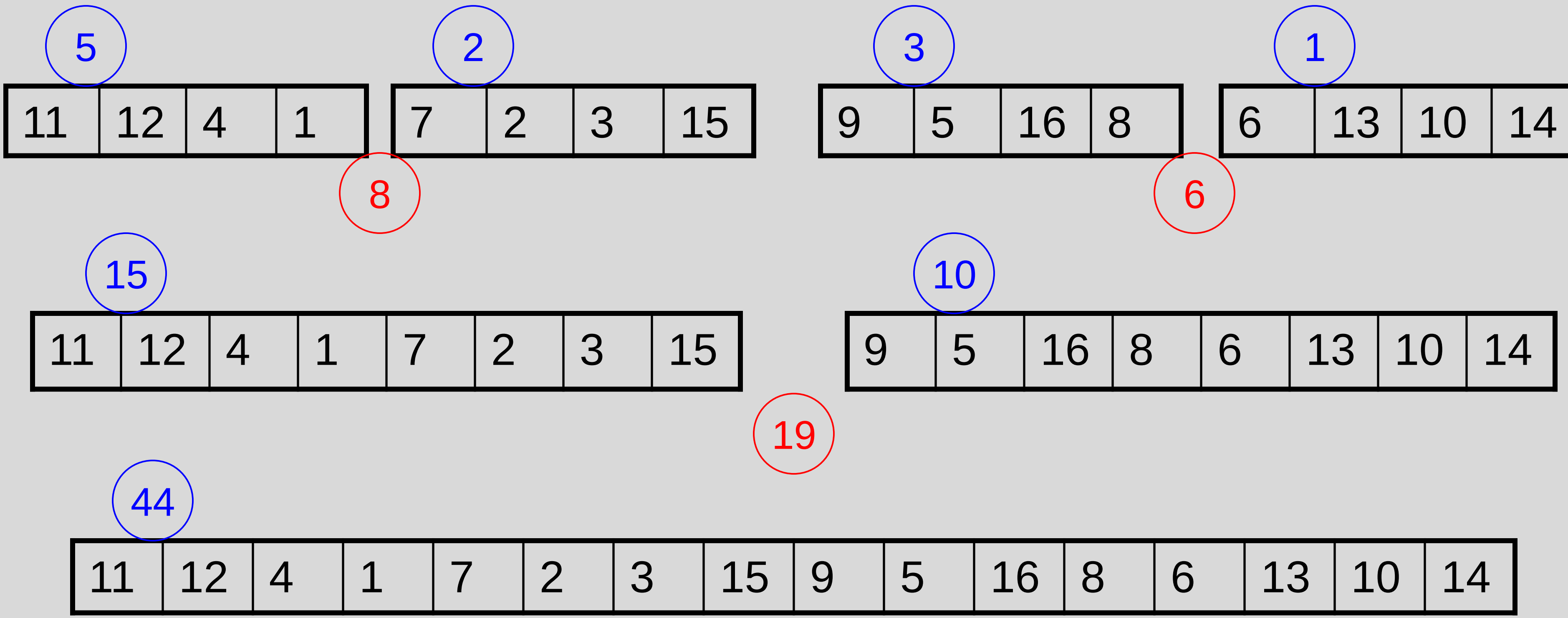
11	12	4	1	7	2	3	15	9	5	16	8	6	13	10	14

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

1	2	3	4	7	11	12	15

5	6	8	9	10	13	14	16

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

1	4	11	12

2	3	7	15

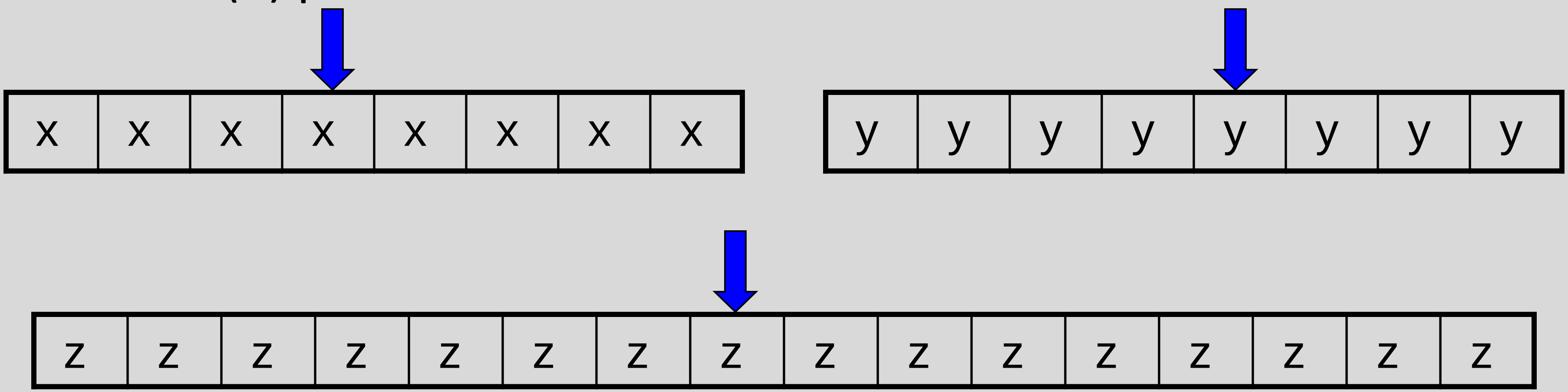
5	8	9	16

6	10	13	14

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 recurrence”
  - $T(n) = 2\ T(n/2) + cn$

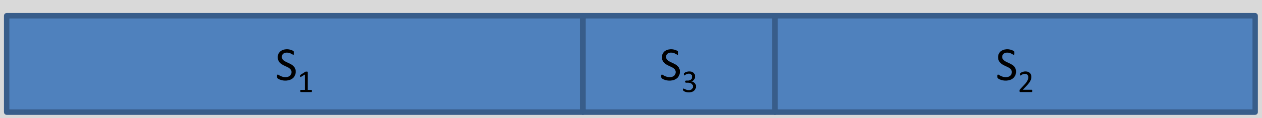
Computing the Median

- 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
    $S_1 = \{y \text{ in } A \mid y < x\}$
    $S_2 = \{y \text{ in } A \mid y > x\}$
    $S_3 = \{y \text{ in } A \mid y = x\}$
    if ($|S_2| >= k$)
        return Select(S_2, k)
    else if ($|S_2| + |S_3| >= k$)
        return x
    else
        return Select(S_1, $k - |S_2| - |S_3|$)
}

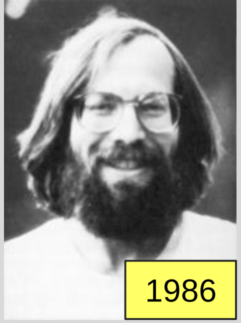


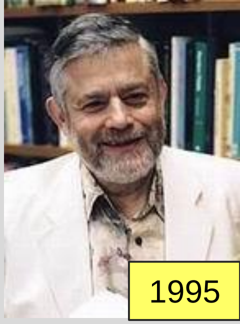
Deterministic Selection

- What is the run time of select if we can guarantee that choose finds an x such that $|S_1| < 3n/4$ and $|S_2| < 3n/4$ in O(n) time

- What is the run time of select if we can guarantee that choose finds an x such that $|S_1| < 3n/4$ and $|S_2| < 3n/4$ in O(n) time

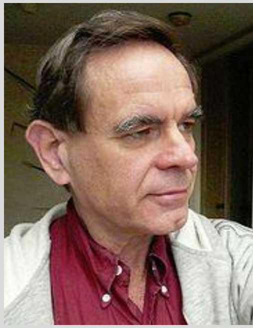
BFPRT Algorithm





- 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

```
    9715480283945084383094856701043643845790217965702956767
+   1242431098234099057329075097179898430928779579277597977
```

Runtime for standard algorithm to add two n digit numbers:

```
    2095067093034680994318596846868779409766717133476767930
X   5920175091777634709677679342929097012308956679993010921
```

Runtime for standard algorithm to multiply two n digit numbers:

Recursive Multiplication Algorithm (First attempt)

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

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

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

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

Recurrence:

Run time:

Simple algebra

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

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

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

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

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

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

$\log_2 3 = 1.58496250073\ldots$