# Announcements

- Divide and Conquer and Recurrences
  - Recurrence Techniques
  - Fast Matrix Multiplication
  - Counting Inversions (5.3)
  - Closest Pair (5.4)
  - Integer Multiplication (5.5)
  - Quicksort and Median Finding
- Dynamic Programming
- Midterm, Friday, February 9

---

## Integer Arithmetic

### 9715480283945084383094856701043643845790217965702956767
+ 1242431098234099057329075097179898430928779579277597977
= 209506709303468099431856846868779409766717133476767930

Runtime for standard algorithm to add two n digit numbers:

\[ 209506709303468099431856846868779409766717133476767930 \]
\[ + 1242431098234099057329075097179898430928779579277597977 \]
\[ = 3337508121314574053108625415568785339121622364506355740 \]

---

### 5920175091777634709677679342929097012308956679993010921

Runtime for standard algorithm to multiply two n digit numbers:

\[ 5920175091777634709677679342929097012308956679993010921 \]
\[ \times 209506709303468099431856846868779409766717133476767930 \]
\[ = 121757151934837638839538180392074559844380956679993010921 \]

---

## Recursive Multiplication Algorithm (First attempt)

<table>
<thead>
<tr>
<th>x</th>
<th>y</th>
<th>xy</th>
</tr>
</thead>
<tbody>
<tr>
<td>x1</td>
<td>y1</td>
<td>x1y1</td>
</tr>
<tr>
<td>x0</td>
<td>y0</td>
<td>x0y0</td>
</tr>
</tbody>
</table>

xy = (x1, 2^n/2 + x0) (y1, 2^n/2 + y0)

= x1y1 2^n + (x1y0 + x0y1) 2^n/2 + x0y0

Recurrence:

Run time:

---

## Simple algebra

\[ x = x_1 \cdot 2^{n/2} + x_0 \]
\[ y = y_1 \cdot 2^{n/2} + y_0 \]
\[ xy = x_1y_1 \cdot 2^n + (x_1y_0 + x_0y_1) \cdot 2^{n/2} + x_0y_0 \]
\[ p = (x_1 + x_0)(y_1 + y_0) = x_1y_1 + x_1y_0 + x_0y_1 + x_0y_0 \]

---

## Karatsuba’s Algorithm

Multiply n-digit integers x and y

\[ x = x_1 \cdot 2^{n/2} + x_0 \]
\[ y = y_1 \cdot 2^{n/2} + y_0 \]

Recursively compute

\[ a = x_1y_1 \]
\[ b = x_0y_0 \]
\[ p = (x_1 + x_0)(y_1 + y_0) \]

Return \[ a \cdot 2^n + (p - a - b)2^n/2 + b \]

Recurrence: \[ T(n) = 3T(n/2) + cn \]

---

\[ \log_3 3 = 1.58496250073 \]
Quicksort [Tony Hoare, 1959]

QuickSort(S):
1. Pick an element \( v \) in S. This is the pivot value.
2. Partition \( S \) into two disjoint subsets, \( S_1 \) and \( S_2 \) such that:
   - elements in \( S_1 \) are all \( < v \)
   - elements in \( S_2 \) are all \( > v \)
3. Return concatenation of QuickSort(\( S_1 \)), \( v \), QuickSort(\( S_2 \))

Recursion ends if QuickSort( ) receives an array of length 0 or 1.

Quicksort – worst case

- Pivot selection: choose first element
- Sort \([1,2,3,4,5,6,\ldots N]\)

Quicksort - pragmatics

- Pivot selection rules
  - Median of first, middle, and last
  - Choose random element
- In place implementation
- Algorithm engineering for partitioning
- Recursion cutoff for small problems

Average case analysis for Quicksort

- All inputs equally likely
  - Or random elements used for pivot
  - Or input is randomly shuffled
- \( QS(n) \) = average number of comparisons for Quicksort on input of size \( n \).

Building a recurrence

Pivot chosen at random. The chance of having \( i \) elements less than the pivot is \( 1/n \).

\[
T(n) = (n - 1) + \frac{1}{n} \sum_{i=0}^{n-1} (T(i) + T(n - i - 1)).
\]

Solution: \( T(N) \approx 2N \ln N \)

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?
Problem generalization

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

Randomized Selection

• Choose the element at random
• Analysis can show that the algorithm has expected run time O(n)

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

BFPRT Algorithm

• A very clever choose algorithm . . .

Split into n/5 sets of size 5
M be the set of medians of these sets
Let x be the median of M

BFPRT runtime

|S_1| < 3n/4, |S_2| < 3n/4

Split into n/5 sets of size 5
M be the set of medians of these sets
x be the median of M
Construct S_1 and S_2
Recursive call in S_1 or S_2

Select(A, k)

```plaintext
Select(A, k)

Choose element x from A
S_1 = \{ y \in A | y < x \}
S_2 = \{ y \in A | y > x \}
S_3 = \{ y \in A | y = x \}
if (|S_2| >= k)
    return Select(S_2, k)
else if (|S_1| + |S_3| >= k)
    return x
else
    return Select(S_1, k - |S_2| - |S_3|)
```
<table>
<thead>
<tr>
<th>BFPRT Recurrence</th>
<th>A theoretical aside</th>
</tr>
</thead>
<tbody>
<tr>
<td>( T(n) \leq T(3n/4) + T(n/5) + c \cdot n )</td>
<td>• How many comparisons are needed in the worst case to find the median?</td>
</tr>
<tr>
<td></td>
<td>• BFPRT showed that this is at most ( 18 \cdot n )</td>
</tr>
<tr>
<td></td>
<td>• Best known results in ( 3 \cdot n ) (but it's complicated)</td>
</tr>
<tr>
<td></td>
<td>• The lower bound was shown to be at least ( 2 \cdot n ) by Bent and John</td>
</tr>
<tr>
<td></td>
<td>‒ Improved to ( 2.01 \cdot n ) by Zwick</td>
</tr>
</tbody>
</table>