S. KIMMEL

Announcements: quiz Friday, equatio, Plickers

Course Organization

- Algorithmic Paradigms
  - Divide & Conquer ✱
  - Dynamic Programming
  - Greedy
- Applications to graph Problems
- What problems can we not solve efficiently

Recall goal:

Appreciate beauty, creativity, surprising cleverness of algorithm design. Next algorithm hopefully touches on some of these...

example:

Multiplication - basic operation used frequently in many applications

Input: 2 n-digit integers a, b. $a = a_{n-1} \cdots a_1 a_0$, $b = b_{n-1} \cdots b_1 b_0$ $\left(a = \sum_{i=0}^{n-1} a_i 10^i\right)$

Output: Integer $c = a \times b$

S. KIMMEL

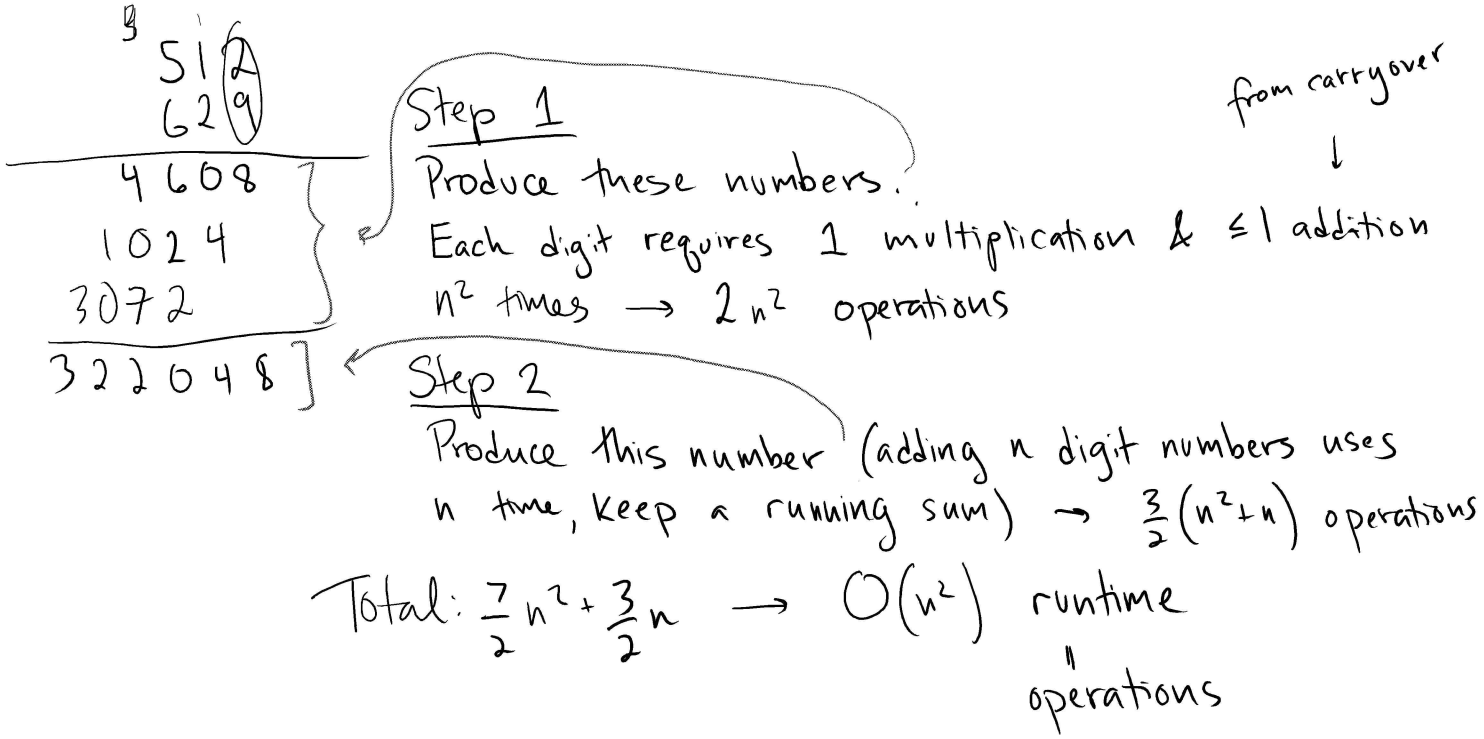
Q: Suppose multiplying/adding 2 one-digit numbers takes $O(1)$ time.

Using "grade school" algorithm, what is runtime of multiplying a and b?

A) $O(n)$ **B) $O(n^2)$** C) $O(\log(n))$ D) Not enough information

Use this example to remind ourselves about runtime and big-O notation

ex:



Teaser: We'll be able to do in $O(n^{1.58})$ using Karatsuba multiplication!

S. KIMMEL

Example brings up 2 guiding principles:

Principles of Algorithm Analysis

1. Use worst-case analysis. e.g. Counted 1 addition operation for carryover for every multiplication, even though might not actually use.
   - upper bounds hold for all instances
   - don't need extra info about problem
   - easier
2. Use asymptotic analysis (big-O notation)

Q. Why do we use big-O notation?

- Only care about big problems (otherwise do by hand or using brute force algorithm) and for big problems, lower order terms don't matter
  e.g. Insertion Sort: $\frac{1}{2}n^2$ Merge Sort: $6n(\log_2 n + 1)$
- Independent of architecture
- Easier
- Predicts how time will scale as problem size increases

S.KIMMEL

Divide & Conquer (& Combine)

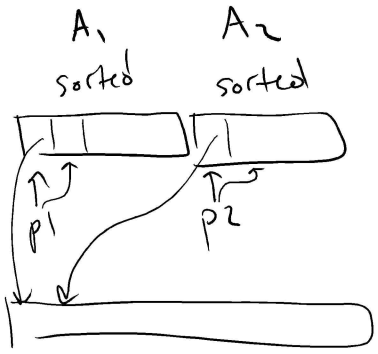
- Split big problem into smaller versions of same problem.
- Solve smaller problems via recursion
- combine solutions to get solution to big problem

Already Seen!

Merge Sort

Input: Array A of integers of size n
Output: Sorted array

```
MergeSort(A)
    if length(A)==1 : return A        } base case
    A1 = MergeSort( A[1 : n/2] )
    A2 = MergeSort( A[n/2+1 : n] )    } DIVIDE & CONQUER
    p1 = p2 = 1
    for i = 1 to n                    } COMBINE
        if A1[p1] < A2[p2]
            A[i] = A1[p1]
            p1++
        else
            A[i] = A2[p2]
            p2++
    return A
```



S. KIMMEL

Q: Create a Divide & Conquer Algorithm for Multiplication
(a, b n digit numbers, n is a power of 2)

Base Case: If 1 digit numbers, can do in $O(1)$ time

Divide: $a = \overbrace{(a_{n-1} \cdots a_{n/2})}^{a'} \cdot 10^{n/2} + \overbrace{(a_{\frac{n}{2}-1} \cdots a_0)}^{a''}$

$b = \overbrace{(b_{n-1} \cdots b_{\frac{n}{2}})}^{b'} \cdot 10^{n/2} + \overbrace{(b_{n/2-1} \cdots b_0)}^{b''}$

* a', a'', b', b'' each $\frac{n}{2}$ digits.

e.g. $5284 = 5 \times 10^3 + 2 \times 10^2 + 8 \times 10 + 4$

$= 52 \times 10^2 + 84$

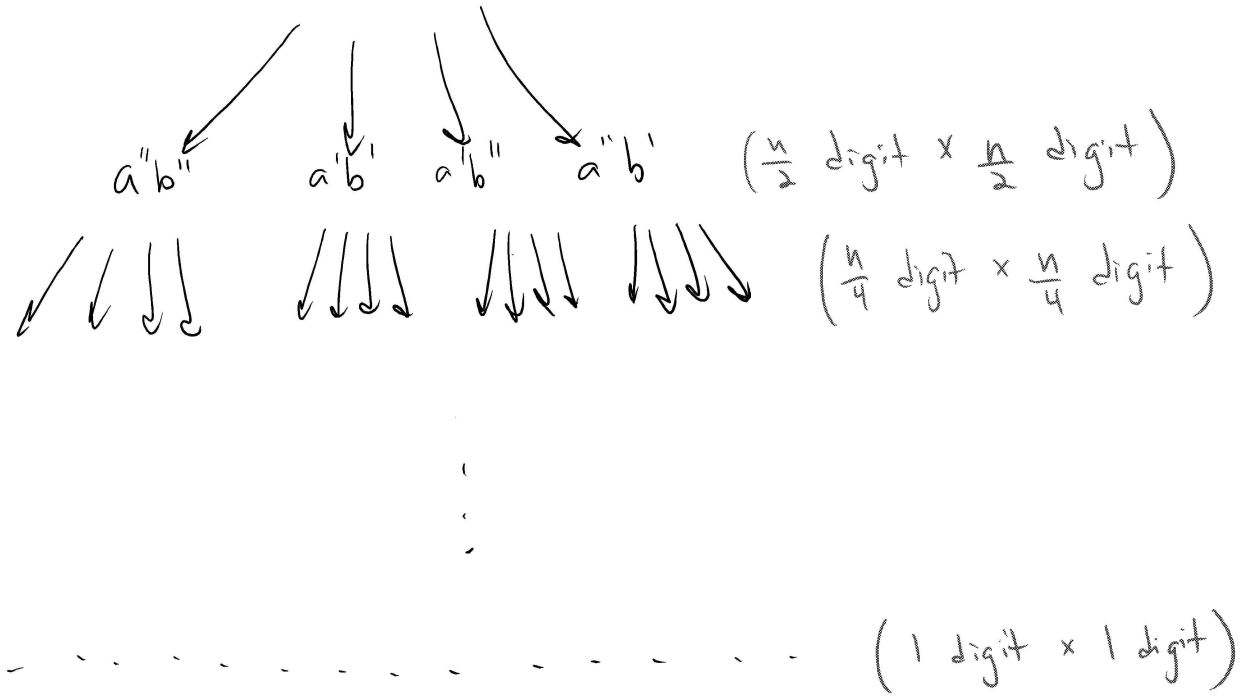
$a \cdot b = (a' \cdot 10^{n/2} + a'') \cdot (b' \cdot 10^{n/2} + b'')$

$= a'b' \cdot 10^n + (a''b' + a'b'') 10^{n/2} + a''b''$

Conquer: Solve smaller problems: $a'b'$, $a''b'$, $a'b''$, $a''b$

Combine: Add $a'b' \cdot 10^n + (a''b' + a'b'') 10^{n/2} + a''b''$

S. KIMMEL



Q: How many 1 digit × 1 digit multiplication problems will this algorithm require? (n digit × n digit)

A) $\log_2 n$ B) n **C) n^2** D) 2^n

Each level increases by factor of 4:

$$\underbrace{4 \cdot 4 \cdot 4 \cdot 4 \cdots 4}_{\log_2 n} = 4^{\log_2 n} = (2^2)^{\log_2 n} = \left(2^{\log_2 n}\right)^2 = n^2$$

This is bad! There are at least n^2 basic operations, so time complexity is $\Omega(n^2)$. No better than regular!