W, 09/04/19

Fall'19 CSCE 629

Analysis of Algorithms

Fang Song
Texas A&M U

Lecture 3

- Divide-&-Conquer
- Fast multiplication
- Matrix multiplication

Recap: sorting

- Merge sort
  - Divide array into two halves.
  - Recursively sort each half.
  - Merge two halves to make sorted whole.
  - Runtime: $T(n) = 2T(n/2) + O(n) = O(n\log n)$ [will show]
- Quick sort
  - Divide array into two "nice" halves: $L \leq pivot \leq \ R$
  - Recursively sort each half.
  - Merge (trivial).
  - Runtime: $O(n\log n)$ average-case [will come back] and $O(n^2)$ worst-case [HW]
- Question: can we improve it, i.e., below $O(n\log n)$? [will come back]

Divide-&-Conquer

You can see a pattern …

1. Divide
   - Divide the given instance of the problem into several independent smaller instances of the same problem.
2. Delegate
   - Solve smaller instances recursively, i.e., delegate each smaller instance to the Recursion Fairy.
3. Combine
   - Combine solutions of smaller instance into the final solution for the given instance.

Multiplication

- Input: n-bit integers a, b (in binary)
- Output: $c = ab$

$13 = (1101)_2, 14 = (1110)_2$
$13 \times 14 = (1101)_2 \times (1110)_2$
$= (10110110)_2 = 182$

subscript 2 means binary rep.

- Recall: the grade-school algorithm
  - Compute n intermediate products
  - Do n additions
  - Running time: $\Theta(n^2)$

- Can we do better?

```
         1101
×        1110
-------------
         0000
        11010
       110100
+     1101000
-------------
     10110110
```

Multiplication by divide-&-conquer

- Attempt #1
  - Write $a = a_1 2^{n/2} + a_0, b = b_1 2^{n/2} + b_0$
  - Observe $ab = a_1 b_1 2^n + (a_1 b_0 + a_0 b_1) 2^{n/2} + a_0 b_0$
  - OK! Multiply $n/2$-bit integers recursively

- Running time
  - Exercise. Work out the recurrence relation

$$T(n) = 4T(n/2) + \Theta(n)$$

  - Alas! This is still $\Theta(n^2)$

$$(1101)_2 = 2^2(11)_2 + (01)_2$$
$$(1110)_2 = 2^2(11)_2 + (10)_2$$
$$(1101)_2 \times (1110)_2$$
$$= 2^4(11)_2 \times (11)_2$$
$$+2^2(11)_2 \times (10)_2$$
$$+2^2(01)_2 \times (11)_2$$
$$+(01)_2 \times (10)_2$$
$= \cdots$ [Verify on your own]

Karatsuba's idea

- From 4 to 3

$$ab = a_1 b_1 2^n + (a_1 b_0 + a_0 b_1) 2^{n/2} + a_0 b_0$$

$$= x 2^n + (z - x - y) 2^{n/2} + y$$

$$\underbrace{(a_1 + a_0)(b_1 + b_0)}_{z} = \underbrace{a_1 b_1}_{x} + \underbrace{a_0 b_0}_{y} + (a_1 b_0 + a_0 b_1)$$

- Running time $T(n) = 3T(n/2) + \Theta(n) = O(n^{1.59})$
  - Significant improvement over n^2 when n is big

Karatsuba's fast multiplication algorithm

- Input: n-bit integers a, b (in binary)
- Output: $c = ab$

FastMultiply(a, b, n): // Assume n is a power of 2 for simplicity

```
if n = 1
    Return x · y
else
    a_1 ← a/2^{n/2}, b_1 ← b/2^{n/2}, a_0 ← a mod 2^{n/2}, b_0 ← b mod 2^{n/2}
    x ← FastMultiply (a_1, b_1, n/2),
    y ← FastMultiply (a_0, b_0, n/2),
    z ← FastMultiply (a_1 + a_0, b_1 + b_0, n/2),
    Return x2^n + (z − x − y)2^{n/2} + y
```

… faster multiplication

Anatolii Karatsuba 1960 Arnold Schönhage, Volker Strassen 1971 David Harvey, Joris van der Hoeven 2019





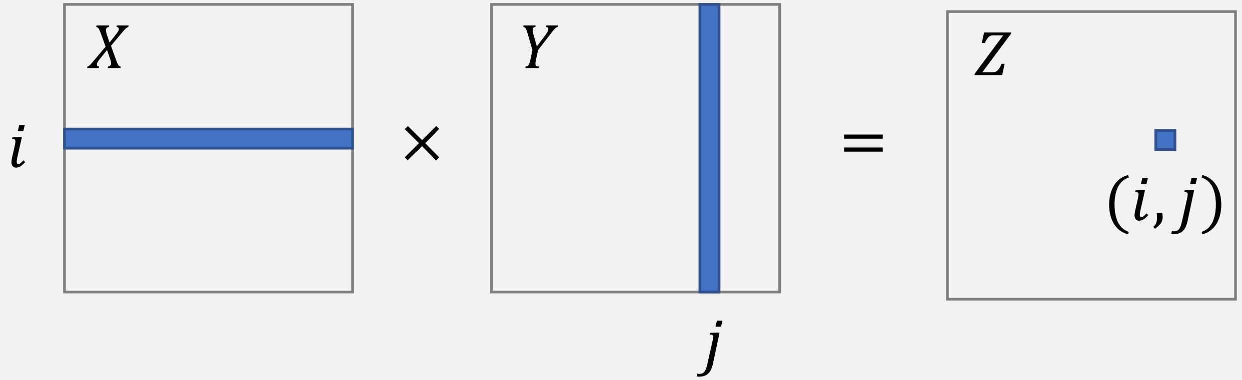
$O(n^{1.585})$ $O(n^{1+1/(\log k)})$ $O(n \log n \log\log n)$ $O(n \log n\, e^{\log^* n})$ $O(n \log n)$


Andrei Toom, Stephen Cook 1966

Martin Fürer 2007

Matrix multiplication

- Input: $X = [x_{ij}], Y = [y_{ij}]$. $i, j = 1, \ldots, n$
- Output: $Z = [z_{ij}] = XY$.



$$z_{ij} = \sum_k x_{ik} y_{kj}$$

- Standard algorithm
  - Running time: $\Theta(n^3)$

MatrixMult(X, Y, Z, n):
 for $i \leftarrow 1$ to n
  for $j \leftarrow 1$ to n
   $z_{ij} \leftarrow 0$
   for $k \leftarrow 1$ to n $z_{ij} = \sum_k x_{ik} y_{kj}$

MatrixMult by divide-&-conquer

- Idea: block-wise multiplication
  - $n \times n$ matrix = 2×2 matrix of $(n/2)\times(n/2)$ submatrices

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

- Running time
  - 8 recursive mults of $(n/2)\times(n/2)$ submatrices
  - 4 adds of $(n/2)\times(n/2)$ submatrices

$$T(n) = 8T(n/2) + O(n^2) = O(n^3)$$

submatrices submatrix size adding submatrices

Strassen's algorithm

- From 8 to 7 …

$$Z = XY = \begin{bmatrix} AE + BG & AF + BH \\ CE + DG & CF + DH \end{bmatrix}$$

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

$P_1 = A(F - H)$ $P_2 = (A + B)H$
$P_3 = (C + D)E$ $P_4 = D(G - E)$
$P_5 = (A + D)(E + H)$
$P_6 = (B - D)(G + H)$
$P_7 = (A - C)(E + F)$

- Running time
  - 7 recursive mults of $(n/2)\times(n/2)$ submatrices
  - 18 adds/subs of $(n/2)\times(n/2)$ submatrices
  - Significant improvement in "big"-data setting

$$T(n) = 7T(n/2) + O(n^2) \approx O(n^{2.81})$$

… faster matrix multiplication

Volker Strassen **1969**

$O(n^{2.81})$

Virginia Vassilevska Williams **2011**

$O(n^{2.3729})$

$O(n^{2.376})$

Don Coppersmith, Shmuel Winograd **1990**

$O(n^{2.3728639})$

François Le Gall 2014