Linear Time Sorting

Time to clear the air…

- Quicksort
  - Best case is ~ merge sort's worst-case
  - In expectation: 39% *more* comparisons
- Mergesort
  - Most implementations do *not* sort in place
    - Need *n* or *n/2* memory
    - Lots of copying!
  - In place requires O($n \log^2 n$)
    - *(Katajainen, Pasanen & Teuhola 1996)*

…so what gives?

- In practice,
  - There are other operations occurring than comparisons
  - Quicksort has lower costing high-frequency operations
  - More "engineering" tricks to speed
    - Choose sample median (usually 3)
    - Swap to insertion for small (<10) arrays

The Take-away

- Home: 10^8; Super: 10^12

Insertion Sort (N^2)

computer	thousand	million	billion
home	instant	2.8 hours	317 years
super	instant	1 second	1.6 weeks

Mergesort (N log N)

thousand	million	billion
instant	1 sec	18 min
instant	instant	instant

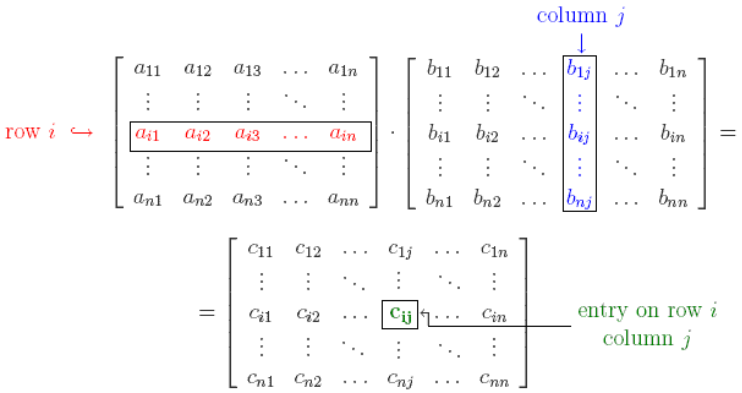
Source: cs.princeton.edu

Quicksort (N log N)

thousand	million	billion
instant	0.3 sec	6 min
instant	instant	instant

- Lesson 1: Good algorithms are better than super computers
- Lesson 2: Great algorithms are better than good ones
- Lesson 3:Coefficients actually do matter in practice: implement with hardware in mind!

Matrix Multiplication



Source: math.cornell.edu

Matrix Multiplication

Input: Two $n \times n$ (square) matrices, $A = (a_{ij})$ and $B = (b_{ij})$.

Output: $n \times n$ matrix $C = (c_{ij})$, where $C = A \cdot B$, i.e.,

$$c_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj}$$

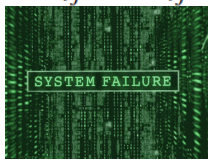
for $i, j = 1, 2, \ldots, n$.

The Obvious Method

SQUARE-MAT-MULT(A, B, n)

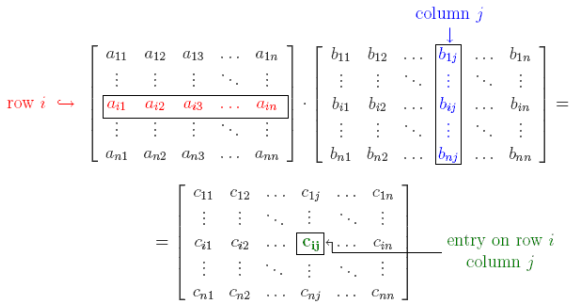
let C be a new $n \times n$ matrix
for $i = 1$ **to** n
  for $j = 1$ **to** n
    $c_{ij} = 0$
    for $k = 1$ **to** n
      $c_{ij} = c_{ij} + a_{ik} \cdot b_{kj}$
return C

$\Rightarrow \Theta(n^3)$



Can we do better?

- Is $\Theta(n^3)$ the best we can do?
  - Must compute n^2 entries
  - Each entry is the sum of n terms



Divide-and-conquer?

$$\begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{pmatrix} = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} \cdot \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix}$$

- Approach
  - Multiply two *n*/2 matrices
  - Sum their product
- Costs
  - 8 multiplications
  - Combining takes *Θ(n)*
- Recurrence: $T(n) = 8T(n/2) + \Theta(n^2) \Rightarrow T(n) = \Theta(n^3)$

$$\begin{aligned} C_{11} &= A_{11} \cdot B_{11} + A_{12} \cdot B_{21} , \\ C_{12} &= A_{11} \cdot B_{12} + A_{12} \cdot B_{22} , \\ C_{21} &= A_{21} \cdot B_{11} + A_{22} \cdot B_{21} , \\ C_{22} &= A_{21} \cdot B_{12} + A_{22} \cdot B_{22} . \end{aligned}$$

But wait! Call now, and get….

Volker Strassen

Volker Strassen is a German mathematician, a professor emeritus in the department of mathematics and statistics at the University of Konstanz. Wikipedia

Born: April 29, 1936 (age 76), Düsseldorf

Awards: Knuth Prize

Strassen Method

- Step 1: divide

$$\begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{pmatrix} = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} \cdot \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix}$$

Strassen Method

- Step 2: add /subtract

$$\begin{aligned} S_1 &= B_{12} - B_{22} , \\ S_2 &= A_{11} + A_{12} , \\ S_3 &= A_{21} + A_{22} , \\ S_4 &= B_{21} - B_{11} , \\ S_5 &= A_{11} + A_{22} , \\ S_6 &= B_{11} + B_{22} , \\ S_7 &= A_{12} - A_{22} , \\ S_8 &= B_{21} + B_{22} , \\ S_9 &= A_{11} - A_{21} , \\ S_{10} &= B_{11} + B_{12} . \end{aligned}$$

- Step 3: multiply

$$\begin{aligned} P_1 &= A_{11} \cdot S_1 \\ P_2 &= S_2 \cdot B_{22} \\ P_3 &= S_3 \cdot B_{11} \\ P_4 &= A_{22} \cdot S_4 \\ P_5 &= S_5 \cdot S_6 \\ P_6 &= S_7 \cdot S_8 \\ P_7 &= S_9 \cdot S_{10} \end{aligned}$$

$\Theta(n/2)$.

Strassen Method

- Step 4: combine

$$\begin{aligned} C_{11} &= P_5 + P_4 - P_2 + P_6 \,, \\ C_{12} &= P_1 + P_2 \,, \\ C_{21} &= P_3 + P_4 \,, \\ C_{22} &= P_5 + P_1 - P_3 - P_7 \,. \end{aligned}$$

- E.g., $C_{12} = P_1 + P_2$

$$\begin{aligned} P_1 &= A_{11} \cdot S_1 = A_{11} \cdot B_{12} - A_{11} \cdot B_{22} \\ P_2 &= S_2 \cdot B_{22} = A_{11} \cdot B_{22} + A_{12} \cdot B_{22} \end{aligned}$$

$$C_{12} = A_{11} \cdot B_{12} + A_{12} \cdot B_{22}$$

What's the difference?

Consider the Mudd-O-Matic computer, capable of performing 10^9 operations second…

	n = 10^5
n^3	11 days
$n^{2.81}$	31 hours

Improving on Strassen!

- Coppersmith-Winograd: $O(n^{2.3737})$

Don Coppersmith

What's the difference?

Consider the Mudd-O-Matic computer, capable of performing 10^9 operations second…

	n = 10^5
n^3	11 days
$n^{2.81}$	31 hours
$n^{2.37}$	12 minutes

Putnam Fellows (top 5 on Putnam)

Name	Team	Years			
Don Coppersmith	MIT	1968	1969	1970	1971
Arthur Rubin	Purdue, Caltech	1970	1971	1972	1973
Bjorn Poonen	Harvard	1985	1986	1987	1988
Ravi D. Vakil	Toronto	1988	1989	1990	1991
Gabriel D. Carroll	UC Berkeley, Harvard	2000	2001	2002	2003
Reid W. Barton	MIT	2001	2002	2003	2004
Daniel Kane	MIT	2003	2004	2005	2006
Brian R. Lawrence	Caltech	2007	2008	2010	2011
Edward L. Kaplan	Carnegie Tech	1939	1940	1941	
David J. Grabiner	Princeton	1986	1987	1988	

A Breakthrough On Matrix Product

NOVEMBER 29, 2011

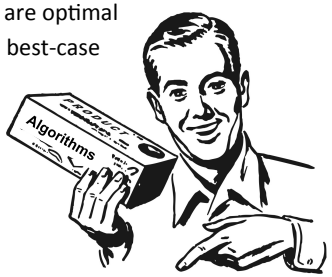
by rjlipton *tags:* breakthrough, galactic, matrix exponent, matrix product, Strassen

Beating Coppersmith-Winograd

Virginia Vassilevska Williams is a theoretical computer scientist who has worked on a number of problems, including a very neat question about **cheating** in the setup of tournaments, and a bunch of novel papers exemplified by **this one** on graph and matrix problems with runtime exponents of 3 that have long been begging to be improved.

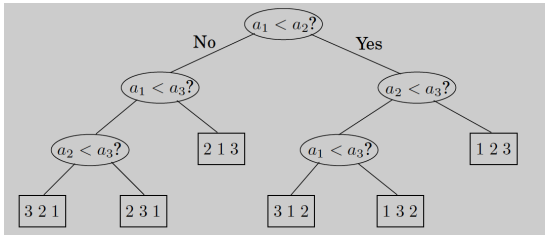
Lower-bounding sort

- Bounding problems
  - Establish lower bounds for solving general problem
  - Why do we care?
    - We want to show that our algorithms are optimal
    - E.g., algorithms worst-case = problem best-case
- How long does it take to sort?
  - Insertion, Merge, Quick
    -
  - Can we do better?
    - to examine input



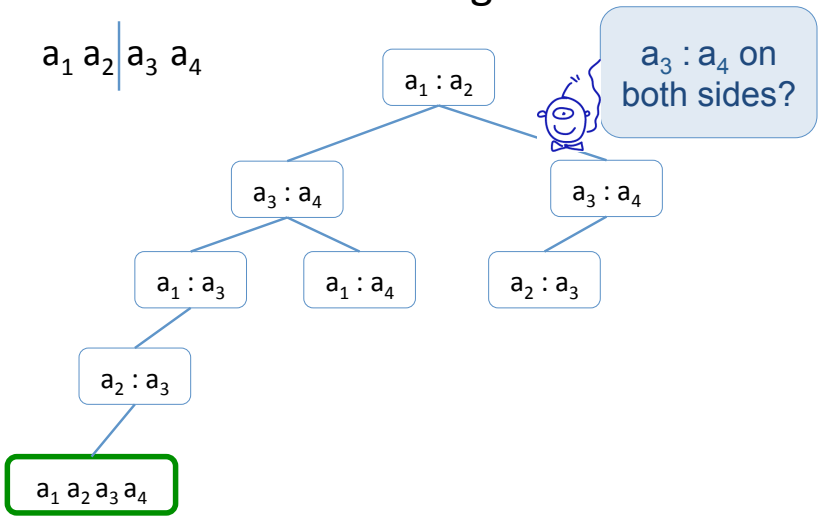
Comparison-based search

- Decision tree:



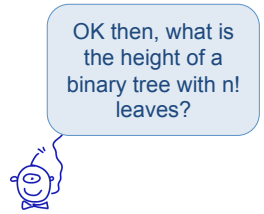
- Comparisons form a binary tree

What does the comparison tree look like for Mergesort?



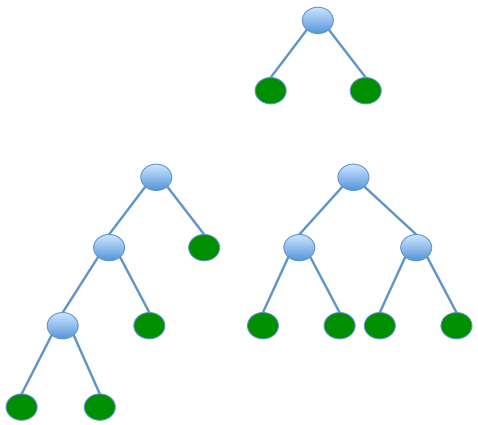
Two Key Observations

- Every comparison-based sorting algorithm for n elements has a binary comparison tree with at least n! leaves
- The *depth* of that tree (number of comparisons on the longest path) is a lower bound on the worst-case running time.



Binary trees and their heights…

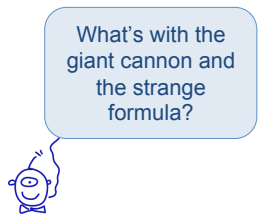
- A tree with 2 leaves has height at least 1
- A tree with 4 leaves has height at least…
- A tree with k leaves has height at least…



Putting it all together…

- Any comparison based sorting algorithm on n items has a corresponding binary tree with n! leaves.
- The height of that tree is at least _____ which is a lower bound on the worst-case running time.
- Finishing up…

$$n! \sim \sqrt{2\pi n}\left(\frac{n}{e}\right)^n$$



Bounding comparison-based search

- Observations
  - Number of permutation (leaves) for *n* elements:
  - Binary tree of depth *d* has at most: leaves
  -
  - Why?
  - (Stirling's approximation)
- Any decision tree sorting *n* elements has depth
  - Depth *d* ≥

Counting sort

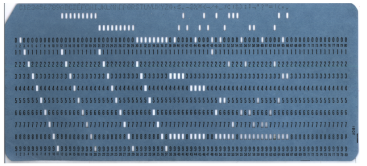
- **Assumption**: numbers to be sorted are integers in {0,1,…,k}

```
COUNTING-SORT(A, B, n, k)
  let C[0..k] be a new array
  for i = 0 to k
      C[i] = 0
  for j = 1 to n
      C[A[j]] = C[A[j]] + 1
  for i = 1 to k
      C[i] = C[i] + C[i − 1]
  for j = n downto 1
      B[C[A[j]]] = A[j]
      C[A[j]] = C[A[j]] − 1
```

Counting Sort Properties

- **Stable**: keys with same values appear in same order as input
- Analysis:
  - $\Theta(n+k) \rightarrow \Theta(n)$ if $k = O(n)$
- What values of *k* are practical?
  - 64/32 bit values?
  - 16 bit? Probably not
  - 8 bit? Maybe, depending on *n*
  - 4-bit? Probably, unless *n* is really small
- Used in Radix sort

Radix sort



- How IBM made its money
- Punch card readers for census tabulation
- Early versions of the sorters involved human operators!

Earlier sorters [edit source | edit beta]

Original census sorter, 1890, manual.[2]

Sorting cards became an issue during the 1900 agricultural census, so Herman Hollerith developed the 1901 Hollerith Automatic Horizontal Sorter,[3] a sorter with horizonal pockets.[4]

In 1908, he designed the more compact Hollerith 070 Vertical Sorting Machine[5] that sorted 250 cards per minute.[2][4]

The IBM Type 71 Vertical Sorter came out in 1928. It had 12 pockets that could hold 80 cards. It could sort 150 cards per minute.[6]

The IBM Type 75, Model 1, 19??, 400 cards per minute[2]

The IBM Type 75, Model 2, 19??, 250 cards per minute[2]

Hollerith census machine with sorter

- **Key idea:** sort *least* significant digits first

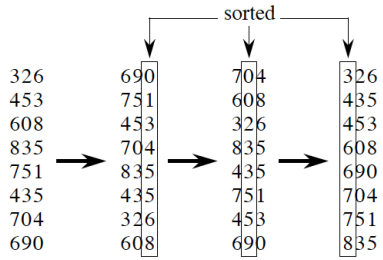
Radix sort

RADIX-SORT(A, d)

for $i = 1$ **to** d

use a stable sort to sort array A on digit i

Example



Radix sort

RADIX-SORT(A, d)

for $i = 1$ **to** d

use a stable sort to sort array A on digit i

- Correctness
  - Proven inductively on *i*
  - Assume digits 1,2,…,*i*-1 are sorted
    - If digits in position *i* are different, remaining digits are irrelevant
    - If digits in position *i* are same, stable sort leaves numbers in sorted order
- Analysis
  - $\Theta(n+k)$ per pass, *d* passes
  - $\Theta(d(n+k)) \Rightarrow \Theta(dn)$ if $k = O(n)$

The breakdown

- *n* words
- *b* bits/word
- Break into *r*-bit digits. Have $d = \lceil b/r \rceil$
- Using counting sort, $k = 2^r - 1$
- Time = $\Theta\left(\frac{b}{r}(n+2^r)\right)$

- Example: 32-bit words, 8-bit digits.

$$b = 32, r = 8, d = \lceil 32/8 \rceil = 4, k = 2^8 - 1 = 255$$

How to choose *r*?

$$\Theta\left(\frac{b}{r}(n+2^r)\right)$$

- Balance *b/r* and $n+2^r$.
- Choosing $r \approx \log n \Rightarrow \Theta\left(\frac{b}{\log n}(n+n)\right) = \Theta(bn/\log n)$

$r < \log n \Rightarrow$

$r > \log n \Rightarrow$

Comparison

- 1 million (2^{20}) 32-bit integers
  - Radix sort: ceil(32/20) = 2 passes
  - Merge sort/quicksort: log 2^{20} = 20 passes
- But I thought we showed sorting to be $\Theta(n \log n)$?
  - Counting sort allows us to gain information about keys without directly comparing them
  - Use keys as array indices

Bucket sort

- Assumes input is generated by a random process that distributed elements uniformly over [0,1)
- Idea:
  - Divide [0,1) into *n* equal-sized *buckets*
  - Distributed the *n* input values into the buckets
  - Sort each bucket
  - Process buckets in order, gathering elements in each one

Bucket Sort

Input: $A[1..n]$, where $0 \le A[i] < 1$ for all i.
Auxiliary array: $B[0..n-1]$ of linked lists, each list initially empty.

```
BUCKET-SORT(A, n)
  let B[0..n − 1] be a new array
  for i = 1 to n − 1
      make B[i] an empty list
  for i = 1 to n
      insert A[i] into list B[⌊n · A[i]⌋]
  for i = 0 to n − 1
      sort list B[i] with insertion sort
  concatenate lists B[0], B[1], . . . , B[n − 1] together in order
  return the concatenated lists
```

Correctness of Bucket Sort

- Consider $A[i], A[j]$; WLOG, let $A[i] \le A[j]$
- Then, $\lfloor n \cdot A[i] \rfloor \le \lfloor n \cdot A[j] \rfloor$ and $A[i]$ is placed into a bucket with the same or lower index as $A[j]$
  - If same, insertion sort fixes order
  - If earlier bucket, concatenation fixes order

```
BUCKET-SORT(A, n)
  let B[0..n − 1] be a new array
  for i = 1 to n − 1
      make B[i] an empty list
  for i = 1 to n
      insert A[i] into list B[⌊n · A[i]⌋]
  for i = 0 to n − 1
      sort list B[i] with insertion sort
  concatenate lists B[0], B[1], . . . , B[n − 1] together in order
  return the concatenated lists
```

Bucket Sort Analysis

- Relies on no bucket getting too many values
- All lines of algorithm besides insertion sort take $\Theta(n)$
- If each bucket gets a constant number (average is 1 per bucket) of elements, then sorting all buckets takes $\mathrm{O}(n)$
  - Requires careful analysis

Expected-case analysis

- *Probabilistic analysis* uses probability to analyze an algorithm whose running time depends on the distribution of inputs
- Different than using a *randomized algorithm*, where randomization imposes a distribution
- If not drawn from a uniform distribution, all bets are off
  - Still correct, performance analysis just doesn't stand