Divide-and-Conquer

"Divide et impera"
"Veni, vidi, vici"

- Julius Caesar
100BC - 44BC

Divide-and-Conquer

Most widespread application of divide-and-conquer.

- Break up problem into two pieces of equal size.
- Solve two sub-problems independently by recursion.
- Combine two results in overall solution in linear time.

Consequence.

- Brute force / naïve solution: N^2.
- Divide-and-conquer: N log N.

Familiar example.

- Mergesort.

This course.

- Counting inversions, closest pair of points, order statistics, fast matrix multiplication, fast integer multiplication, FFT.

Why Does It Matter?

Run time (nanoseconds)		$1.3 N^3$	$10 N^2$	$47 N \log_2 N$	48 N
Time to solve a problem of size	1000	1.3 seconds	10 msec	0.4 msec	0.048 msec
	10,000	22 minutes	1 second	6 msec	0.48 msec
	100,000	15 days	1.7 minutes	78 msec	4.8 msec
	million	41 years	2.8 hours	0.94 seconds	48 msec
	10 million	41 millennia	1.7 weeks	11 seconds	0.48 seconds
Max size problem solved in one	second	920	10,000	1 million	21 million
	minute	3,600	77,000	49 million	1.3 billion
	hour	14,000	600,000	2.4 trillion	76 trillion
	day	41,000	2.9 million	50 trillion	1,800 trillion
N multiplied by 10, time multiplied by		1,000	100	10+	10

Orders of Magnitude

Seconds	Equivalent
1	1 second
10	10 seconds
10^2	1.7 minutes
10^3	17 minutes
10^4	2.8 hours
10^5	1.1 days
10^6	1.6 weeks
10^7	3.8 months
10^8	3.1 years
10^9	3.1 decades
10^{10}	3.1 centuries
. . .	forever
10^{21}	age of universe

Meters Per Second	Imperial Units	Example
10^{-10}	1.2 in / decade	Continental drift
10^{-8}	1 ft / year	Hair growing
10^{-6}	3.4 in / day	Glacier
10^{-4}	1.2 ft / hour	Gastro-intestinal tract
10^{-2}	2 ft / minute	Ant
1	2.2 mi / hour	Human walk
10^2	220 mi / hour	Propeller airplane
10^4	370 mi / min	Space shuttle
10^6	620 mi / sec	Earth in galactic orbit
10^8	62,000 mi / sec	1/3 speed of light

Powers of 2	2^{10}	thousand
	2^{20}	million
	2^{30}	billion

A Useful Recurrence Relation

T(N) = worst case running time on input of size N.

- Note: O(1) is "standard" abuse of notation.

$$T(N) \le \begin{cases} O(1) & \text{if } N \le n_0 \\ \underbrace{T(\lceil N/2 \rceil)}_{\text{solve left half}} + \underbrace{T(\lfloor N/2 \rfloor)}_{\text{solve right half}} + \underbrace{O(N)}_{\text{combine}} & \text{otherwise} \end{cases}$$

Alternate informal form: $T(N) \le T(N/2) + O(N)$.

- Implicitly assumes N is a power of 2.
- Implicitly assume $T(N) \in O(1)$ for sufficiently small N.

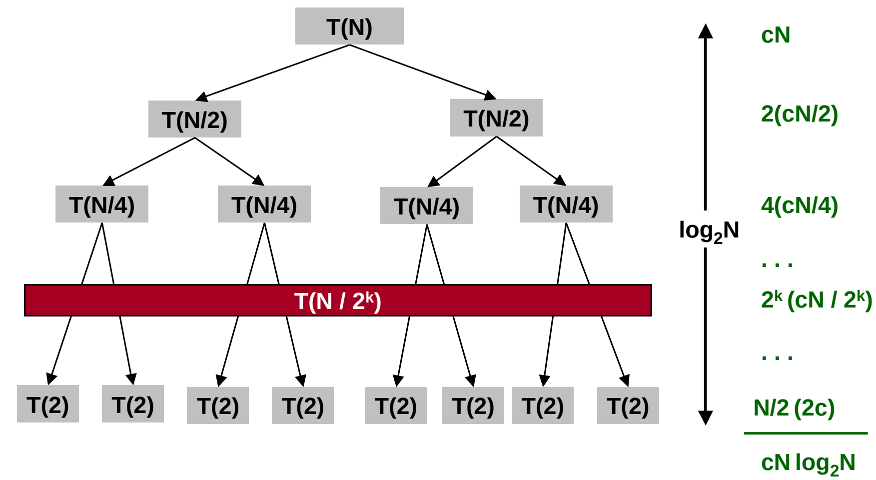
Solution.

- Any function satisfying above recurrence is $\in O(N \log_2 N)$.
- Equivalently, $O(\log_b N)$ for any $b > 1$.

Analysis of Recurrence: Recursion Tree

Assuming N is a power of 2.

$$T(N) = \begin{cases} 0 & \text{if } N = 1 \\ 2T(N/2) + cN & \text{otherwise} \end{cases}$$



Analysis of Recurrence

$$T(N) \le \begin{cases} 0 & \text{if } N = 1 \\ \underbrace{T(\lceil N/2 \rceil)}_{\text{solve left half}} + \underbrace{T(\lfloor N/2 \rfloor)}_{\text{solve right half}} + \underbrace{cN}_{\text{combine}} & \text{otherwise} \end{cases}$$

$$\Rightarrow T(N) \le cN \lceil \log_2 N \rceil$$

Proof by induction on N.

- Base case: N = 1.
- Define $n_1 = \lfloor n/2 \rfloor$, $n_2 = \lceil n/2 \rceil$.
- Induction step: assume true for 1, 2, . . . , N – 1.

$$\begin{aligned} T(N) &\le T(n_1) + T(n_2) + cn \\ &\le cn_1 \lceil \log_2 n_1 \rceil + cn_2 \lceil \log_2 n_2 \rceil + cn \\ &\le cn_1 \lceil \log_2 n_2 \rceil + cn_2 \lceil \log_2 n_2 \rceil + cn \\ &= cn \lceil \log_2 n_2 \rceil + cn \\ &\le cn(\lceil \log_2 n \rceil - 1) + cn \\ &= cn \lceil \log_2 n \rceil \end{aligned}$$

$$\begin{aligned} n_2 &= \lceil n/2 \rceil \\ &\le \left\lceil 2^{\lceil \log_2 n \rceil} / 2 \right\rceil \\ \Rightarrow \log_2 n_2 &\le \lceil \log_2 n \rceil - 1 \end{aligned}$$

Counting Inversions

Web site tries to match your preferences with others on Internet.

- You rank N songs.
- Web site consults database to find people with similar rankings.

Closeness metric.

- My rank = { 1, 2, . . . , N }.
- Your rank = { $a_1, a_2, \ldots, a_N$ }.
- Number of inversions between two preference lists.
- Songs i and j inverted if i < j, but $a_i > a_j$

Songs

	A	B	C	D	E
Me	1	2	3	4	5
You	1	3	4	2	5

Inversion

Inversions

{3, 2}, {4, 2}

Counting Inversions

Brute-force solution.

- Check all pairs i and j such that i < j.
- $\Theta(N^2)$ comparisons.

Note: there can be a quadratic number of inversions.

- Asymptotically faster algorithm must compute total number without even looking at each inversion individually.

Counting Inversions: Divide-and-Conquer

Divide-and-conquer.

1	5	4	8	10	2	6	9	12	11	3	7

Counting Inversions: Divide-and-Conquer

Divide-and-conquer.

- Divide: separate list into two pieces.

1	5	4	8	10	2	6	9	12	11	3	7	O(1)

1	5	4	8	10	2

6	9	12	11	3	7

Counting Inversions: Divide-and-Conquer

Divide-and-conquer.

- Divide: separate list into two pieces.
- Conquer: recursively count inversions in each half separately.

1	5	4	8	10	2	6	9	12	11	3	7	O(1)

1	5	4	8	10	2

6	9	12	11	3	7

2T(N / 2)

5 blue-blue inversions

8 green-green inversions

Counting Inversions: Divide-and-Conquer

Divide-and-conquer.
- Divide: separate list into two pieces.
- Conquer: recursively count inversions in each half.
- Combine: count inversions where a_i and a_j are in different halves.

1	5	4	8	10	2	6	9	12	11	3	7	O(1)

1	5	4	8	10	2	6	9	12	11	3	7	2T(N / 2)

5 blue-blue inversions 8 green-green inversions

9 blue-green inversions: O(N^2)
{5-3, 4-3, 8-6, 8-3, 8-7, 10-6, 10-9, 10-3, 10-7}

Counting Inversions: Divide-and-Conquer

Divide-and-conquer.
- Divide: separate list into two pieces.
- Conquer: recursively count inversions in each half.
- Combine: count inversions where a_i and a_j are in different halves.
- Return sum of three quantities.

1	5	4	8	10	2	6	9	12	11	3	7	O(1)

1	5	4	8	10	2	6	9	12	11	3	7	2T(N / 2)

5 blue-blue inversions 8 green-green inversions

9 blue-green inversions: O(N^2)
{5-3, 4-3, 8-6, 8-3, 8-7, 10-6, 10-9, 10-3, 10-7}

Total = 5 + 8 + 9 = 22. O(1)

Counting Inversions: Divide-and-Conquer

Divide-and-conquer.
- Divide: separate list into two pieces.
- Conquer: recursively count inversions in each half.
- ➡ Combine: count inversions where a_i and a_j are in different halves.
- Return sum of three quantities.

1	5	4	8	10	2	6	9	12	11	3	7	O(1)

1	5	4	8	10	2	6	9	12	11	3	7	2T(N / 2)

5 blue-blue inversions 8 green-green inversions

9 blue-green inversions:
{5-3, 4-3, 8-6, 8-3, 8-7, 10-6, 10-9, 10-3, 10-7}

Can we do this step in sub-quadratic time?

Total = 5 + 8 + 9 = 22. O(1)

Counting Inversions: Good Combine

Combine: count inversions where a_i and a_j are in different halves.
- Key idea: easy if each half is sorted.
- Sort each half.
- Count inversions.

1	5	4	8	10	2	6	9	12	11	3	7

1	2	4	5	8	10	3	6	7	9	11	12	O(N log N)

9 blue-green inversions: 4 + 2 + 2 + 1 + 0 + 0 O(N)

$$T(N) = T(\lfloor N/2 \rfloor) + T(\lceil N/2 \rceil) + O(N \log N) \Rightarrow T(N) = O(N \log^2 N)$$

Counting Inversions: Better Combine

Combine: count inversions where a_i and a_j are in different halves.
- Assume each half is pre-sorted.
- Count inversions.
- Merge two sorted halves into sorted whole.

3	7	10	14	18	19

2	11	16	17	23	25

13 blue-green inversions: 6 + 3 + 2 + 2 + 0 + 0 O(N)

2	3	7	10	11	14	16	17	18	19	23	25

O(N)

$$T(N) = T(\lfloor N/2 \rfloor) + T(\lceil N/2 \rceil) + O(N) \Rightarrow T(N) = O(N \log N)$$

Closest Pair

Given N points in the plane, find a pair that is closest together.
- For concreteness, we assume Euclidean distances.
- Foundation of then-fledgling field of computational geometry.
- Graphics, computer vision, geographic information systems, molecular modeling, air traffic control.

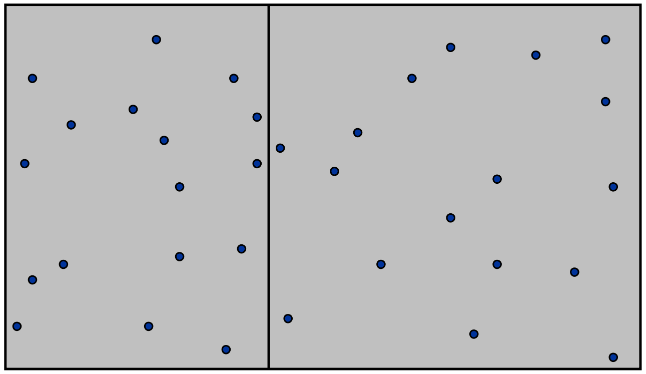
Brute force solution.
- Check all pairs of points p and q.
- $\Theta(N^2)$ comparisons.

One dimensional version (points on a line).
- O(N log N) easy.

Assumption to make presentation cleaner.
- No two points have same x coordinate.

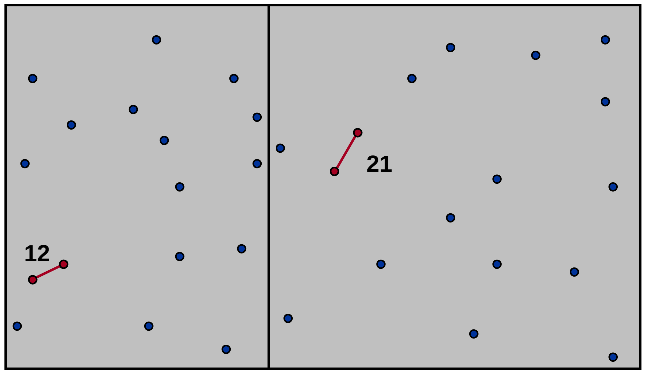
Closest Pair

Algorithm.
- Divide: draw vertical line so that roughly N / 2 points on each side.



Closest Pair

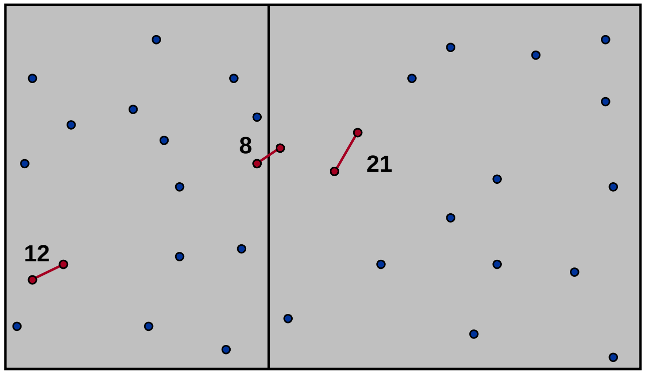
Algorithm.
- Divide: draw vertical line so that roughly N / 2 points on each side.
- Conquer: find closest pair in each side recursively.



Closest Pair

Algorithm.

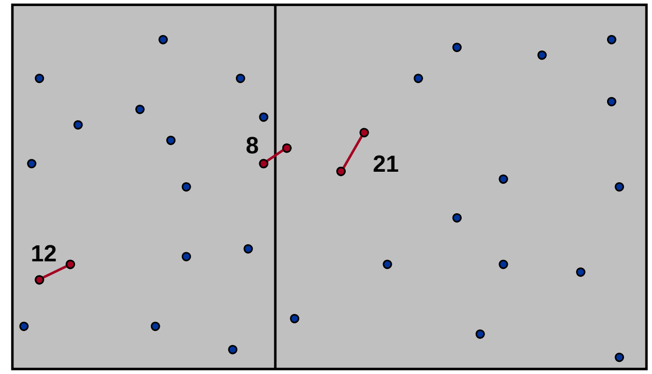
- Divide: draw vertical line so that roughly N / 2 points on each side.
- Conquer: find closest pair in each side recursively.
- Combine: find closest pair with one point in each side.



Closest Pair

Algorithm.

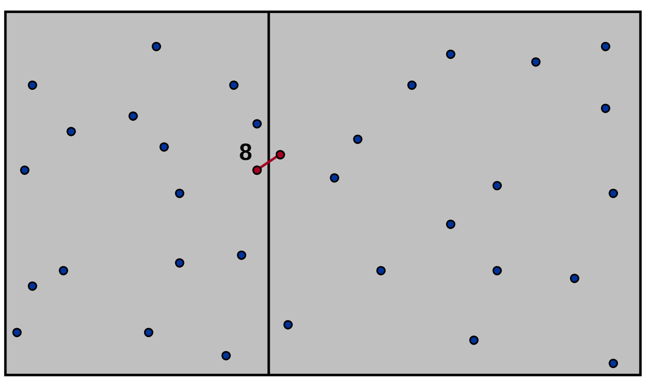
- Divide: draw vertical line so that roughly N / 2 points on each side.
- Conquer: find closest pair in each side recursively.
- Combine: find closest pair with one point in each side.
- Return best of 3 solutions.



Closest Pair

Algorithm.

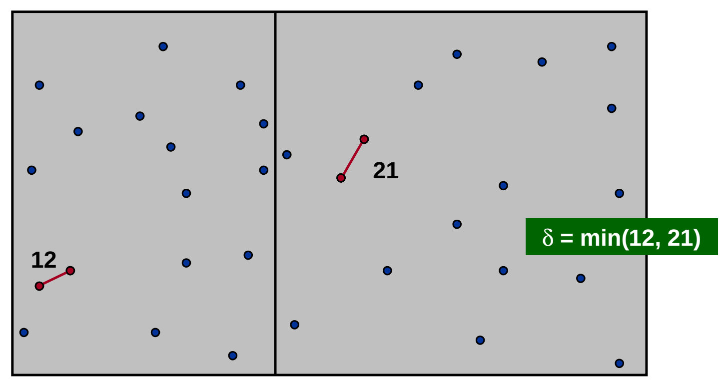
- Divide: draw vertical line so that roughly N / 2 points on each side.
- Conquer: find closest pair in each side recursively.
- Combine: find closest pair with one point in each side.
- Return best of 3 solutions.



Closest Pair

Key step: find closest pair with one point in each side.

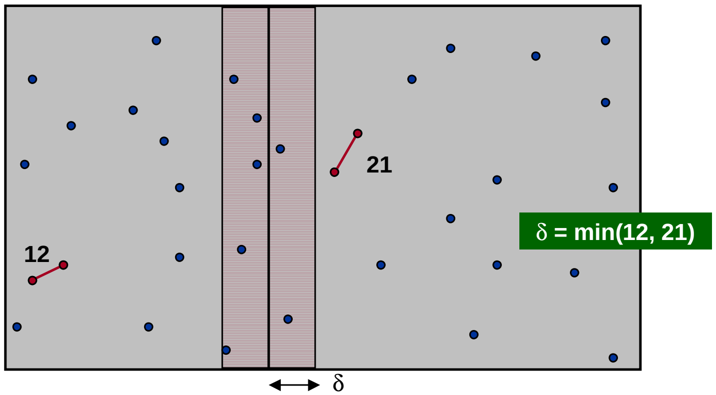
- Extra information: closest pair entirely in one side had distance δ.



Closest Pair

Key step: find closest pair with one point in each side.

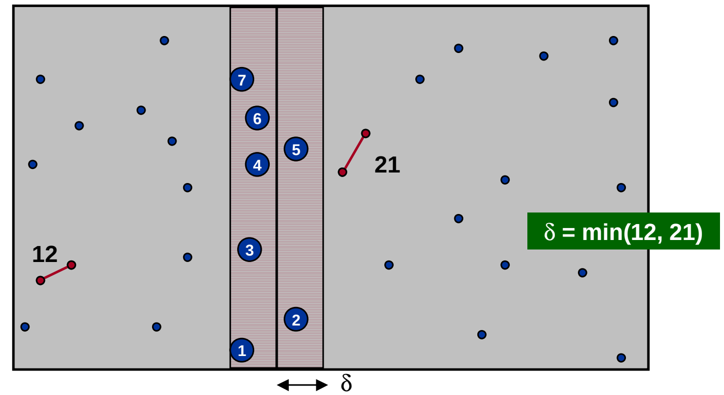
- Extra information: closest pair entirely in one side had distance δ.
- Observation: only need to consider points S within δ of line.



Closest Pair

Key step: find closest pair with one point in each side.

- Extra information: closest pair entirely in one side had distance δ.
- Observation: only need to consider points S within δ of line.
- Sort points in strip S by their y coordinate.
  - suffices to compute distances for pairs within constant number of positions of each other in sorted list!

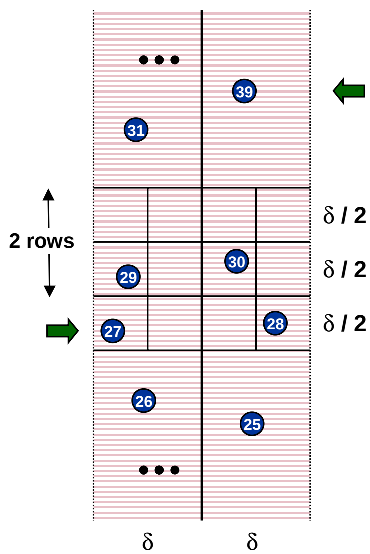


Closest Pair

S = list of points in the strip sorted by their y coordinate.

Crucial fact: if p and q are in S, and if d(p, q) < δ , then they are within 11 positions of each other in S.

- No two points lie in same box.
- Two points at least 2 rows apart have distance ≥ 2δ / 2.



Closest Pair

δ = ClosestPair (p_1, p_2 , . . . , p_N)

O(N log N)	Compute separation line x = x_{med} such that half the points have x coordinate less than x_{med}, and half are greater.
2T(N / 2)	δ_1 = ClosestPair(left half) δ_2 = ClosestPair(right half) δ = min (δ_1 , δ_2)
O(N)	Delete all points further than δ from separation line.
O(N log N)	Sort remaining points in strip by y coordinate.
O(N)	Scan in y order, and compute distance between each point and next 11 neighbors.
O(N)	If any of these distances is less than δ , update δ .

$$T(N) = T(\lfloor N/2 \rfloor) + T(\lceil N/2 \rceil) + O(N \log N) \quad \Rightarrow \quad T(N) = O(N \log^2 N)$$

Closest Pair

Can we achieve O(N log N)?

- Yes. Don't sort points in strip from scratch each time.
- Each recursive call should return two lists: all points sorted by y coordinate, and all points sorted by x coordinate.
- Sorting is accomplished by merging two already sorted lists.

$$T(N) = T(\lfloor N/2 \rfloor) + T(\lceil N/2 \rceil) + O(N) \Rightarrow T(N) = O(N \log N)$$

Integer Arithmetic

Given two N-digit integers a and b, compute a + b.

- O(N) bit operations.

Multiplication: given two N-digit integers a and b, compute ab.

- Brute force solution: $\Theta(N^2)$ bit operations.

Application.

- Cryptography.

1	1	1	1	1	1	0	1	
	1	1	0	1	0	1	0	1
+	0	1	1	1	1	1	0	1
1	0	1	0	1	0	0	1	0

									1	1	0	1	0	1	0	1
								*	0	1	1	1	1	1	0	1
								1	1	0	1	0	1	0	1	0
							0	0	0	0	0	0	0	0	0	
						1	1	0	1	0	1	0	1	0		
					1	1	0	1	0	1	0	1	0			
				1	1	0	1	0	1	0	1	0				
			1	1	0	1	0	1	0	1	0					
		1	1	0	1	0	1	0	1	0						
	0	0	0	0	0	0	0	0	0							
0	1	1	0	1	0	0	0	0	0	0	0	0	0	0	1	0

Divide-and-Conquer Multiplication: First Attempt

To multiply two N-digit integers:

- Multiply four N/2-digit integers.
- Add two N/2-digit integers, and shift to obtain result.

$$\begin{aligned} 123{,}456 \times 987{,}654 &= (10^3 w + x) \times (10^3 y + z) \\ &= 10^6 (wy) + 10^3 (wz + xy) + 10^0 (xz) \\ &= 10^6 (121{,}401) + 10^3 (80{,}442 + 450{,}072) + 10^0 (298{,}224) \\ &= 121{,}401{,}299{,}224 \end{aligned}$$

$$\begin{aligned} w &= 123 \\ x &= 456 \\ y &= 987 \\ z &= 654 \end{aligned}$$

$$ab = (10^{N/2} w + x)(10^{N/2} y + z)$$

$$T(N) = \underbrace{4T(N/2)}_{\text{recursive calls}} + \underbrace{\Theta(N)}_{\text{add, shift}} \Rightarrow T(N) = \Theta(N^2)$$

N is a power of 2 ⇨

Karatsuba Multiplication

To multiply two N-digit integers:

- Add two N/2 digit integers.
- Multiply three N/2-digit integers.
- Subtract two N/2-digit integers, and shift to obtain result.

$$\begin{aligned} 123{,}456 \times 987{,}654 &= (10^3 w + x) \times (10^3 y + z) \\ &= 10^6 (wy) + 10^3 (wz + xy) + 10^0 (xz) \\ &= 10^6 (p) + 10^3 (r - p - q) + 10^0 (q) \\ &= 10^6 (121{,}401) + 10^3 (950{,}139 - 121{,}401 - 298{,}224) + 10^0 (298{,}224) \\ & \quad 121{,}401{,}299{,}224 \end{aligned}$$

$$\begin{aligned} w &= 123 \\ x &= 456 \\ y &= 987 \\ z &= 654 \end{aligned}$$

$$\begin{aligned} p &= wy \\ q &= xz \\ r &= (w + x)(y + z) \end{aligned}$$

$$(wz + xy) = r - p - q$$

Karatsuba Multiplication: Analysis

To multiply two N-digit integers:

- Add two N/2 digit integers.
- Multiply three N/2-digit integers.
- Subtract two N/2-digit integers, and shift to obtain result.

Karatsuba-Ofman (1962).

- $O(N^{1.585})$ bit operations.

$$p = wy,\quad q = xz,\quad r = (w+x)(y+z)$$

$$(wz + xy) = r - p - q$$

$$ab = (10^{N/2} w + x)(10^{N/2} y + z)$$

$$T(N) \le \underbrace{T(\lfloor N/2 \rfloor) + T(\lceil N/2 \rceil) + T(1 + \lceil N/2 \rceil)}_{\text{recursive calls}} + \underbrace{\Theta(N)}_{\text{add, subtract, shift}}$$

$$\Rightarrow T(N) = O(N^{\log_2 3})$$

Matrix Multiplication

Given two N x N matrices A and B, compute C = AB.

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

- Brute force: $\Theta(N^3)$ time.

$$\begin{pmatrix} 26 & 62 & 98 \\ 80 & 224 & 368 \\ 134 & 386 & 638 \end{pmatrix} = \begin{pmatrix} 0 & 2 & 4 \\ 6 & 8 & 10 \\ 12 & 14 & 16 \end{pmatrix} \times \begin{pmatrix} 1 & 7 & 13 \\ 3 & 9 & 15 \\ 5 & 11 & 17 \end{pmatrix}$$

$$\begin{pmatrix} c_{11} & c_{12} & c_{13} & \cdots & c_{1N} \\ c_{21} & c_{22} & c_{23} & \cdots & c_{2N} \\ c_{31} & c_{32} & c_{33} & \cdots & c_{3N} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ c_{N1} & c_{N2} & c_{N3} & \cdots & c_{NN} \end{pmatrix} = \begin{pmatrix} a_{11} & a_{12} & a_{13} & \cdots & a_{1N} \\ a_{21} & a_{22} & a_{23} & \cdots & a_{2N} \\ a_{31} & a_{32} & a_{33} & \cdots & a_{3N} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ a_{N1} & a_{N2} & a_{N3} & \cdots & a_{NN} \end{pmatrix} \times \begin{pmatrix} b_{11} & b_{12} & b_{13} & \cdots & b_{1N} \\ b_{21} & b_{22} & b_{23} & \cdots & b_{2N} \\ b_{31} & b_{32} & b_{33} & \cdots & b_{3N} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ b_{N1} & b_{N2} & b_{N3} & \cdots & b_{NN} \end{pmatrix}$$

Hard to imagine naïve algorithm can be improved upon.

Matrix Multiplication: Warmup

Warmup: divide-and-conquer.

- Divide: partition A and B into N/2 x N/2 blocks.
- Conquer: multiply 8 N/2 x N/2 recursively.
- Combine: add appropriate products using 4 matrix additions.

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

$$\begin{aligned} C_{11} &= (A_{11} \times B_{11}) + (A_{12} \times B_{21}) \\ C_{12} &= (A_{11} \times B_{12}) + (A_{12} \times B_{22}) \\ C_{21} &= (A_{21} \times B_{11}) + (A_{22} \times B_{21}) \\ C_{22} &= (A_{21} \times B_{12}) + (A_{22} \times B_{22}) \end{aligned}$$

$$T(N) = \underbrace{8T(N/2)}_{\text{recursive calls}} + \underbrace{\Theta(N^2)}_{\text{add, form submatrices}} \Rightarrow T(N) = \Theta(N^3)$$

Matrix Multiplication: Idea

Idea: multiply 2 x 2 matrices with only 7 scalar multiplications.

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

$$\begin{aligned} P_1 &= a \times (g - h) \\ P_2 &= (a + b) \times h \\ P_3 &= (c + d) \times e \\ P_4 &= d \times (f - e) \\ P_5 &= (a + d) \times (e + h) \\ P_6 &= (b - d) \times (f + h) \\ P_7 &= (a - c) \times (e + g) \end{aligned}$$

$$\begin{aligned} r &= P_5 + P_4 - P_2 + P_6 \\ s &= P_1 + P_2 \\ t &= P_3 + P_4 \\ u &= P_5 + P_1 - P_3 - P_7 \end{aligned}$$

- 7 multiplications.
- 18 = 10 + 8 additions and subtractions.

Note: did not rely on commutativity of scalar multiplication.

Matrix Multiplication: Strassen

Generalize to matrices.

- Divide: partition A and B into N/2 x N/2 blocks.
- Compute: 14 N/2 x N/2 matrices via 10 matrix add/subtract.
- Conquer: multiply 7 N/2 x N/2 recursively.
- Combine: 7 products into 4 terms using 8 matrix add/subtract.

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

Analysis.

- Assume N is a power of 2.
- T(N) = # arithmetic operations.

$$T(N) = \underbrace{7T(N/2)}_{\text{recursive calls}} + \underbrace{\Theta(N^2)}_{\text{add, subtract}} \Rightarrow T(N) = \Theta(N^{\log_2 7}) = O(N^{2.81})$$

Beyond Strassen

Can you multiply two 2 x 2 matrices with only 7 scalar multiplications?

- Yes! Strassen (1969). $\Theta(N^{\log_2 7}) = O(N^{2.81})$

Can you multiply two 2 x 2 matrix with only 6 scalar multiplications?

- Impossible (Hopcroft and Kerr, 1971). $\Theta(N^{\log_2 6}) = O(N^{2.59})$

Two 3 x 3 matrices with only 21 scalar multiplications?

- Also impossible. $\Theta(N^{\log_3 21}) = O(N^{2.77})$

Two 70 x 70 matrices with only 143,640 scalar multiplications?

- Yes! (Pan, 1980). $\Theta(N^{\log_{70} 143640}) = O(N^{2.80})$

Decimal wars.

- December, 1979: $O(N^{2.521813})$.
- January, 1980: $O(N^{2.521801})$.

Coppersmith-Winograd (1987): $O(N^{2.376})$.

Strassen in Practice?

Practical considerations.

- Stop recursion around N = 100.
- Numerical stability.
- Harder to parallelize.
- Caching effects.

Order Statistics

Given N linearly ordered elements, find i^{th} smallest element.

- Minimum if i = 1.
- Maximum if i = N.
- Median:
  - i = (N+1) / 2 if N is odd
  - i = N/2 or i = N/2 + 1
- Easy to do with O(N) comparisons if i or N – i is a constant.
- Easy to do in general with $O(N \log_2 N)$ comparisons by sorting.

Can we do in worst-case O(N) comparisons?

- Yes. (Blum, Floyd, Pratt, Rivest, Tarjan, 1973)
- Cool and simple idea. Ahead of its time.

Assumption to make presentation cleaner.

- All items have distinct values.

Fast Select

Similar to quicksort, but throw away useless "half" at each iteration.

- Select i^{th} smallest element from $a_1, a_2, \ldots, a_N$.

FastSelect (i^{th}, N, $a_1, a_2, \ldots, a_N$)

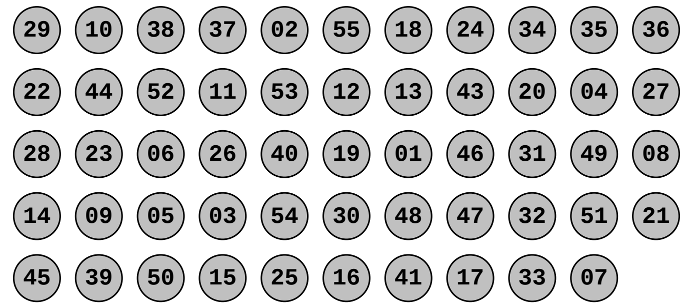
```
x ← FastPartition(N, a_1, a_2, ..., a_N)
k ← rank(x)

if (i == k)
   return x

else if (i < k)
   b[] ← all items of a[] less than x
   return FastSelect(i^th, k-1, b_1, b_2, ..., b_(k-1))

else if (i > k)
   c[] ← all items of a[] greater than x
   return FastSelect((i-k)^th, N-k, c_1, c_2, ..., c_(N-k))
```

x = partition element is kth smallest

Fast Partition

`FastPartition()`.

- Divide N elements into ⌊N/5⌋ groups of 5 elements each, plus extra.

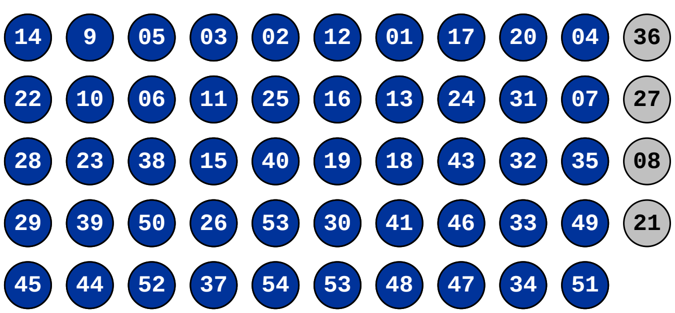


N = 54

Fast Partition

`FastPartition()`.

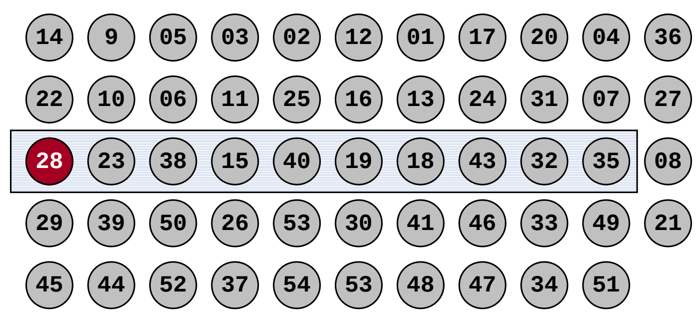
- Divide N elements into ⌊N/5⌋ groups of 5 elements each, plus extra.
- Brute force sort each of the 5-element groups.



Fast Partition

`FastPartition()`.

- Divide N elements into ⌊N/5⌋ groups of 5 elements each, plus extra.
- Brute force sort each of the 5-element groups.
- Find x = "median of medians" using `FastSelect()` recursively.



Fast Selection and Fast Partition

`FastPartition()`.

- Divide N elements into ⌊N/5⌋ groups of 5 elements each, plus extra.
- Brute force sort each of the 5-element groups.
- Find x = "median of medians" using `FastSelect()` recursively.

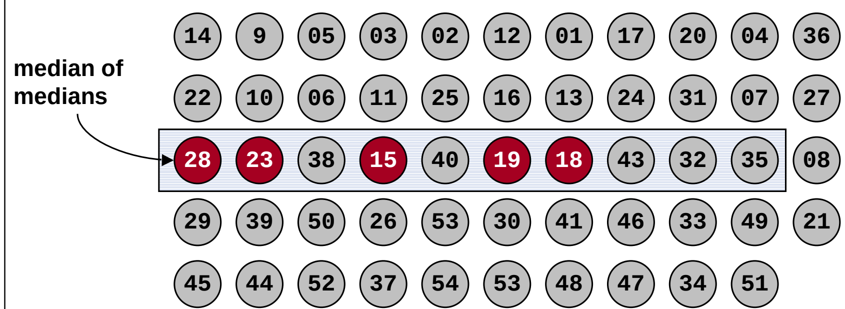
`FastSelect()`.

- Call `FastPartition()`. Let x be partition element used, and let k be its rank.
- Call `FastSelect()` recursively to find i^{th} smallest element.
  - return x if i = k
  - return i^{th} smallest on left side if i < k
  - return $(i-k)^{th}$ smallest on right side if i > k

Fast Selection Analysis

Crux of proof: at least 25% of elements thrown away at each step.

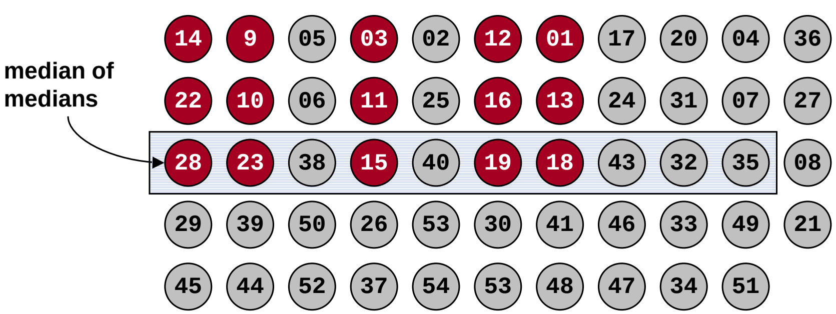
- At least 1/2 of 5 element medians ≤ x
  - at least ⌊⌊N / 5⌋ / 2⌋ = ⌊N / 10⌋ medians ≤ x



Fast Selection Analysis

Crux of proof: at least 25% of elements thrown away at each step.

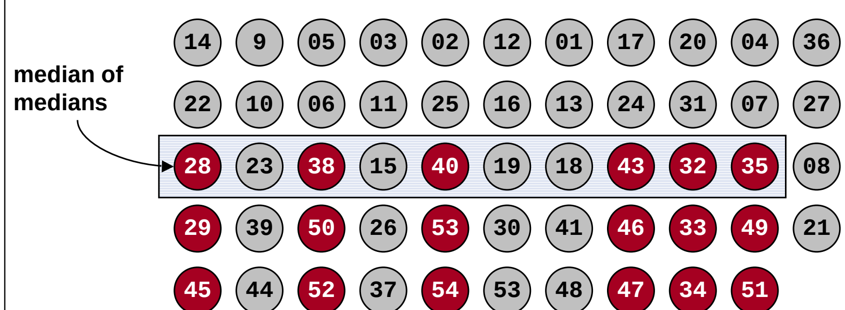
- At least 1/2 of 5 element medians ≤ x
  - at least ⌊⌊N / 5⌋ / 2⌋ = ⌊N / 10⌋ medians ≤ x
- At least 3⌊N / 10⌋ elements ≤ x.



Fast Selection Analysis

Crux of proof: at least 25% of elements thrown away at each step.

- At least 1/2 of 5 element medians ≤ x
  - at least ⌊⌊N / 5⌋ / 2⌋ = ⌊N / 10⌋ medians ≤ x
- At least 3⌊N / 10⌋ elements ≤ x.
- At least 3⌊N / 10⌋ elements ≥ x.



Fast Selection Analysis

Crux of proof: at least 25% of elements thrown away at each step.

- At least 1/2 of 5 element medians $\leq$ x
  - at least $\lfloor \lfloor N/5 \rfloor / 2 \rfloor = \lfloor N/10 \rfloor$ medians $\leq$ x
- At least $3 \lfloor N/10 \rfloor$ elements $\leq$ x.
- At least $3 \lfloor N/10 \rfloor$ elements $\geq$ x.

  $\Rightarrow$ `FastSelect()` called recursively with at most $N - 3 \lfloor N/10 \rfloor$ elements in last step

$$T(N) \leq \underbrace{T(\lfloor N/5 \rfloor)}_{\text{median of medians}} + \underbrace{T(N - 3\lfloor N/10 \rfloor)}_{\text{recursive select}} + \underbrace{O(N)}_{\text{insertion sort}}$$

$$\Rightarrow \quad T(N) = O(N).$$

Fast Selection Analysis

Analysis of recurrence.

$$T(N) \leq \begin{cases} c & \text{if } N < 50 \\ T(\lfloor N/5 \rfloor) + T(N - 3\lfloor N/10 \rfloor) + cN & \text{otherwise} \end{cases}$$

Claim: $T(N) \leq 20cN$.

- Base case: N < 50.
- Inductive step: assume true for 1, 2, . . . , N-1.

$$\begin{aligned} T(N) &\leq T(\lfloor N/5 \rfloor) + T(N - 3\lfloor N/10 \rfloor) + cN \\ &\leq 20c\lfloor N/5 \rfloor + 20c(N - 3\lfloor N/10 \rfloor) + cN \\ &\leq 20c(N/5) + 20c(N) - 20c(N/4) + cN \\ &= 20cN \end{aligned}$$

For $n \geq 50$,
$3\lfloor N/10 \rfloor \geq N/4$.

Linear Time Median Finding Postmortem

Practical considerations.

- Constant (currently) too large to be useful.
- Practical variant: choose random partition element.
  - O(N) expected running time ala quicksort.
- Open problem: guaranteed O(N) with better constant.

Quicksort.

- Worst case O(N log N) if always partition on median.
- Justifies practical variants: median-of-3, median-of-5.