

5

Divide & Conquer

11 November 2024

Prof. Dr. Sebastian Wild

Learning Outcomes

Unit 5: *Divide & Conquer*

1. Know the steps of the Divide & Conquer paradigm.
2. Be able to solve simple Divide & Conquer recurrences.
3. Be able to design and analyze new algorithms using the Divide & Conquer paradigm.
4. Know the performance characteristics of selection-by-rank algorithms.

Outline

5 Divide & Conquer

- 5.1 Divide & Conquer Recurrences
- 5.2 Order Statistics
- 5.3 Linear-Time Selection
- 5.4 Fast Multiplication
- 5.5 Majority
- 5.6 Closest Pair of Points in the Plane

Divide and conquer

Divide and conquer *idiom* (Latin: *divide et impera*)

to make a group of people disagree and fight with one another
so that they will not join together against one

(Merriam-Webster Dictionary)

~> in politics & algorithms, many independent, small problems are better than one big one!

Divide-and-conquer algorithms:

1. Break problem into smaller, independent subproblems. (Divide!)
2. Recursively solve all subproblems. (Conquer!)
3. Assemble solution for original problem from solutions for subproblems.

Examples:

- ▶ Mergesort
- ▶ Quicksort
- ▶ Binary search
- ▶ (arguably) Tower of Hanoi

5.1 Divide & Conquer Recurrences

Back-of-the-envelope analysis

- ▶ before working out the details of a D&C idea,
it is often useful to get a quick indication of the resulting performance
 - ▶ don't want to waste time on something that's not competitive in the end anyways!
- ▶ since D&C is naturally recursive, running time often not obvious
instead: given by a recursive equation
- ▶ unfortunately, rigorous analysis often tricky

- ▶ Remember mergesort?

$$C(n) = \begin{cases} 0 & n \leq 1 \\ C(\lfloor n/2 \rfloor) + C(\lceil n/2 \rceil) + 2n & n \geq 2 \end{cases}$$

$$\rightsquigarrow C(n) = 2n \lfloor \lg(n) \rfloor + 2n - 4 \cdot 2^{\lfloor \lg(n) \rfloor} \text{ 🧐}$$
$$= \Theta(n \log n) \text{ 😊}$$

- ▶ the following method works for many typical cases to give the right **order of growth**

The Master Method

- ▶ Assume a stereotypical D&C algorithm
 - ▶ a recursive calls on n/b (for some constant $a \geq 1$)
 - ▶ subproblems of size n/b (for some constant $b > 1$)
 - ▶ with non-recursive “conquer” effort $f(n)$ (for some function $f : \mathbb{R} \rightarrow \mathbb{R}$)
 - ▶ base case effort d (some constant $d > 0$)

\rightsquigarrow running time $T(n)$ satisfies

$$T(n) = \begin{cases} a \cdot T\left(\frac{n}{b}\right) + f(n) & n > 1 \\ d & n \leq 1 \end{cases}$$

Theorem 5.1 (Master Theorem)

With $c := \log_b(a)$, we have for the above recurrence:

- (a) $T(n) = \Theta(n^c)$ if $f(n) = O(n^{c-\varepsilon})$ for constant $\varepsilon > 0$.
- (b) $T(n) = \Theta(n^c \log n)$ if $f(n) = \Theta(n^c)$.
- (c) $T(n) = \Theta(f(n))$ if $f(n) = \Omega(n^{c+\varepsilon})$ for constant $\varepsilon > 0$ **and** f satisfies the regularity condition $\exists n_0, \alpha < 1 \forall n \geq n_0 : a \cdot f\left(\frac{n}{b}\right) \leq \alpha f(n)$.

Master Theorem – Intuition & Proof Idea

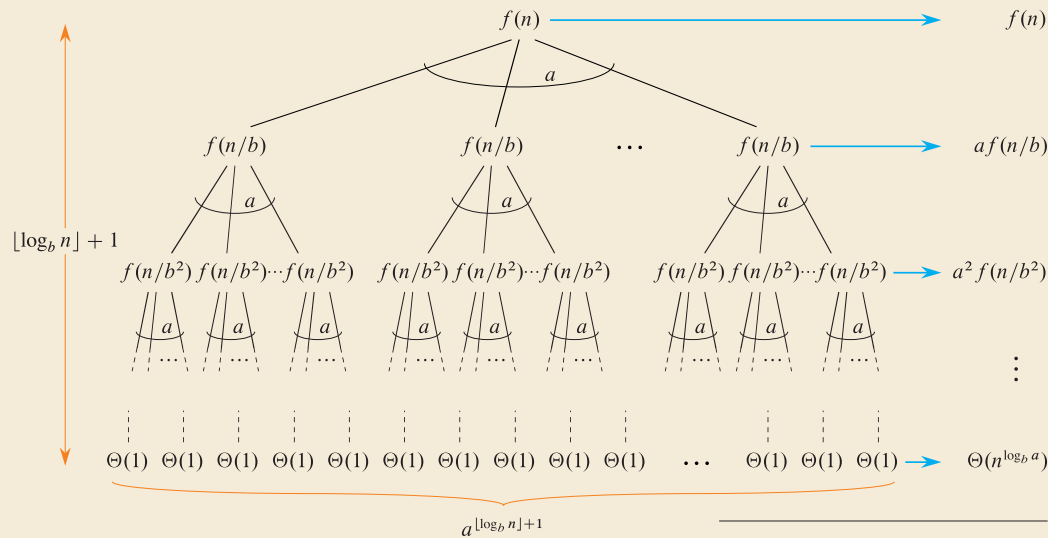


Figure 4.3 of Cormen et al. *Introduction to Algorithms* 4th ed.

Total: $\Theta(n^{\log_b a}) + \sum_{j=0}^{\lceil \log_b n \rceil} a^j f(n/b^j)$

When it's fine to ignore floors and ceilings

The *polynomial-growth condition*

- ▶ $f : \mathbb{R}_{>0} \rightarrow \mathbb{R}$ satisfies the *polynomial-growth condition* if

$$\exists n_0 \forall C \geq 1 \exists D > 1 \quad \forall n \geq n_0 \forall c \in [1, C] : \frac{1}{D}f(n) \leq f(cn) \leq Df(n)$$

- ▶ intuitively: increasing n by up to a factor C (and anywhere in between!) changes the function value by at most a factor $D = D(C)$
(for sufficiently large n)
- ▶ examples: $f(n) = \Theta(n^\alpha \log^\beta(n) \log \log^\gamma(n))$ for constants α, β, γ
 $\rightsquigarrow f$ satisfies the polynomial-growth condition

zero allowed

Lemma 5.2 (Polynomial-growth master method)

If the toll function $f(n)$ satisfies the polynomial-growth condition, then the Θ -class of the solution of a D&C recurrence remains the same when ignoring floors and ceilings on subproblem sizes.

A Rigorous and Stronger Meta Theorem

Theorem 5.3 (Roura's Discrete Master Theorem)

Let $T(n)$ be recursively defined as

$$T(n) = \begin{cases} b_n & 0 \leq n < n_0, \\ f(n) + \sum_{d=1}^D a_d \cdot T\left(\frac{n}{b_d} + r_{n,d}\right) & n \geq n_0, \end{cases}$$


where $D \in \mathbb{N}$, $a_d > 0$, $b_d > 1$, for $d = 1, \dots, D$ are constants, functions $r_{n,d}$ satisfy $|r_{n,d}| = O(1)$ as $n \rightarrow \infty$, and function $f(n)$ satisfies $f(n) \sim B \cdot n^\alpha (\ln n)^\gamma$ for constants $B > 0$, α , γ .

Set $H = 1 - \sum_{d=1}^D a_d (1/b_d)^\alpha$; then we have:

- (a) If $H < 0$, then $T(n) = O(n^{\tilde{\alpha}})$, for $\tilde{\alpha}$ the unique value of α that would make $H = 0$.
- (b) If $H = 0$ and $\gamma > -1$, then $T(n) \sim f(n) \ln(n) / \tilde{H}$ with constant $\tilde{H} = (\gamma + 1) \sum_{d=1}^D a_d b_d^{-\alpha} \ln(b_d)$.
- (c) If $H = 0$ and $\gamma = -1$, then $T(n) \sim f(n) \ln(n) \ln(\ln(n)) / \hat{H}$ with constant $\hat{H} = \sum_{d=1}^D a_d b_d^{-\alpha} \ln(b_d)$.
- (d) If $H = 0$ and $\gamma < -1$, then $T(n) = O(n^\alpha)$.
- (e) If $H > 0$, then $T(n) \sim f(n) / H$.

5.2 Order Statistics

Selection by Rank

- ▶ Standard data summary of numerical data: (Data scientists, listen up!)
 - ▶ mean, standard deviation
 - ▶ min/max (range)
 - ▶ histograms
 - ▶ median, quartiles, other quantiles (a.k.a. order statistics)
- } easy to compute in $\Theta(n)$ time
-  computable in $\Theta(n)$ time?

General form of problem: Selection by Rank

- ▶ **Given:** array $A[0..n)$ of numbers and number $k \in [0..n)$.
- ▶ **Goal:** find element that would be in position k if A was sorted (k th smallest element).
- ▶ $k = \lfloor n/2 \rfloor \rightsquigarrow$ median; $k = \lfloor n/4 \rfloor \rightsquigarrow$ lower quartile
 $k = 0 \rightsquigarrow$ minimum; $k = n - \ell \rightsquigarrow \ell$ th largest

but 0-based &
counting dups

Quickselect

- ▶ Key observation: Finding the element of rank k seems hard.
But computing the rank of a given element is easy!

↪ Pick any element $A[b]$ and find its rank j .

↖ count smaller elements

- ▶ $j = k$? ↪ Lucky Duck! Return chosen element and stop
- ▶ $j < k$? ↪ ... not done yet. But: The $j + 1$ elements smaller than $\leq A[b]$ can be excluded!
- ▶ $j > k$? ↪ similarly exclude the $n - j$ elements $\geq A[b]$

▶ partition function from Quicksort:

- ▶ returns the rank of pivot
- ▶ separates elements into smaller/larger

↪ can use same building blocks

```
1 procedure quickselect( $A[l..r]$ ,  $k$ )
2   if  $r - l \leq 1$  then return  $A[l]$ 
3    $b := \text{choosePivot}(A[l..r])$ 
4    $j := \text{partition}(A[l..r], b)$ 
5   if  $j == k$ 
6     return  $A[j]$ 
7   else if  $j < k$ 
8     quickselect( $A[j + 1..n]$ ,  $k - j - 1$ )
9   else //  $j > k$ 
10    quickselect( $A[0..j]$ ,  $k$ )
```

Quickselect – Iterative Code

Recursion can be replaced by loop (*tail-recursion elimination*)

```
1  procedure quickselect( $A[l..r]$ ,  $k$ )
2      if  $r - l \leq 1$  then return  $A[l]$ 
3       $b := \text{choosePivot}(A[l..r])$ 
4       $j := \text{partition}(A[l..r], b)$ 
5      if  $j == k$ 
6          return  $A[j]$ 
7      else if  $j < k$ 
9          quickselect( $A[j + 1..r]$ ,  $k - j - 1$ )
8      else  $j > k$ 
10         quickselect( $A[l..j]$ ,  $k$ )
```

```
1  procedure quickselectIterative( $A[0..n]$ ,  $k$ )
2       $l := 0$ ;  $r := n$ 
3      while  $r - l > 1$ 
4           $b := \text{choosePivot}(A[l..r])$ 
5           $j := \text{partition}(A[l..r], b)$ 
6          if  $j \geq k$  then  $r := j - 1$ 
7          if  $j \leq k$  then  $l := j + 1$ 
8      return  $A[k]$ 
```

- implementations should usually prefer iterative version
- analysis more intuitive with recursive version

Quickselect – Analysis

```
1 procedure quickselect( $A[l..r]$ ,  $k$ )
2   if  $r - \ell \leq 1$  then return  $A[l]$ 
3    $b := \text{choosePivot}(A[l..r])$ 
4    $j := \text{partition}(A[l..r], b)$ 
5   if  $j == k$ 
6     return  $A[j]$ 
7   else if  $j < k$ 
8     quickselect( $A[j + 1..n]$ ,  $k - j - 1$ )
9   else //  $j > k$ 
10    quickselect( $A[0..j]$ ,  $k$ )
```

► cost = #cmps

► costs depend on n and k

► **worst case:** $k = 0$, but always $j = n - 2$

\rightsquigarrow each recursive call makes n one smaller at cost $\Theta(n)$

$\rightsquigarrow T(n, k) = \Theta(n^2)$ worst case cost

average case:

► let $T(n, k)$ expected cost when we choose a pivot uniformly from $A[0..n]$

\rightsquigarrow formulate recurrence for $T(n, k)$ similar to BST/Quicksort recurrence

$$T(n, k) = n + \frac{1}{n} \sum_{r=0}^{n-1} [r = k] \cdot 0 + [k < r] \cdot T(r, k) + [k > r] \cdot T(n - r - 1, k - r - 1)$$

Quickselect – Average Case Analysis

$$\blacktriangleright T(n, k) = n + \frac{1}{n} \sum_{r=0}^{n-1} [r = k] \cdot 0 + [k < r] \cdot T(r, k) + [k > r] \cdot T(n - r - 1, k - r - 1)$$

$$\blacktriangleright \text{Set } \hat{T}(n) = \max_{k \in [0..n)} T(n, k)$$

$$\rightsquigarrow \hat{T}(n) \leq n + \frac{1}{n} \sum_{r=0}^{n-1} \max\{\hat{T}(r), \hat{T}(n - r - 1)\}$$

\blacktriangleright analyze hypothetical, worse algorithm:

if $r \notin [\frac{1}{4}n, \frac{3}{4}n)$, discard pivot and repeat with new one!

$$\rightsquigarrow \hat{T}(n) \leq \tilde{T}(n) \text{ defined by } \tilde{T}(n) \leq n + \frac{1}{2}\tilde{T}(n) + \frac{1}{2}\tilde{T}(\frac{3}{4}n)$$

$$\rightsquigarrow \tilde{T}(n) \leq 2n + \tilde{T}(\frac{3}{4}n)$$

\blacktriangleright Master Theorem Case 3: $\tilde{T}(n) = \Theta(n)$

Quickselect Discussion

👎 $\Theta(n^2)$ worst case (like Quicksort)

👍 expected cost $\Theta(n)$ (best possible)

👍 no extra space needed

👍 adaptations possible to find several order statistics at once

👍 expected cost can be further improved by choosing pivot from a small sorted sample
 \rightsquigarrow asymptotically optimal randomized cost: $n + \min\{k, n - k\}$ comparisons in expectation
 achieved asymptotically by the *Floyd-Rivest algorithm*

5.3 Linear-Time Selection

Interlude – A recurring conversation

Cast of Characters:



Hi! I'm a *computer science practitioner*.

I love algorithms for the sometimes miraculous applications they enable.

I care for **things** I can implement and **that actually work in practice**.



Hi! I'm a *theoretical computer science researcher*.

I find beauty in elegant and **definitive** answers to questions about complexity.

I care for **eternal truths** and mathematically proven facts;

asymptotically optimal is what counts! (Constant factors are secondary.)

Quickselect Disagreements



For practical purposes, (randomized) Quickselect is perfect.

e.g. used in C++ STL `std::nth_element`



Yeah . . . maybe. But can we select by rank in $O(n)$ deterministic **worst case** time?

Better Pivots

It turns out, we can!

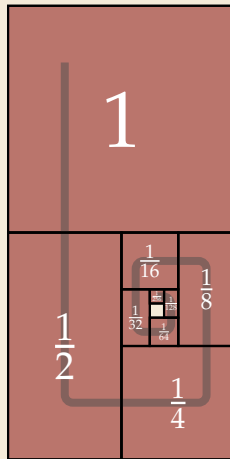
- ▶ All we need is better pivots!
 - ▶ If pivot was the exact median, we would at least halve #elements in each step
 - ▶ Then the total cost of all partitioning steps is $\leq 2n = \Theta(n)$.



But: finding medians is (basically) our original problem!



It totally suffices to find an element of rank αn for $\alpha \in (\varepsilon, 1 - \varepsilon)$ to get overall costs $\Theta(n)$!



The Median-of-Medians Algorithm

```

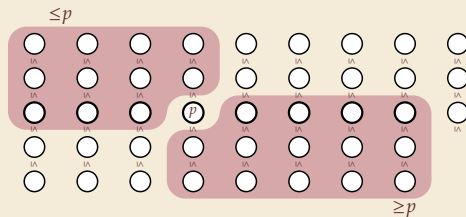
1 procedure choosePivotMoM( $A[l..r]$ )
2    $m := \lfloor n/5 \rfloor$ 
3   for  $i := 0, \dots, m-1$ 
4      $\text{sort}(A[5i..5i+4])$ 
5     // collect median of 5
6      $\text{Swap } A[i] \text{ and } A[5i+2]$ 
7   return  $\text{quickselectMoM}(A[0..m], \lfloor \frac{m-1}{2} \rfloor)$ 
8
9 procedure quickselectMoM( $A[l..r], k$ )
10  if  $r - l \leq 1$  then return  $A[l]$ 
11   $b := \text{choosePivotMoM}(A[l..r])$ 
12   $j := \text{partition}(A[l..r], b)$ 
13  if  $j == k$ 
14    return  $A[j]$ 
15  else if  $j < k$ 
16     $\text{quickselectMoM}(A[j+1..r], k - j - 1)$ 
17  else //  $j > k$ 
18     $\text{quickselectMoM}(A[l..j], k)$ 

```

Analysis:

- Note: 2 mutually recursive procedures
 \rightsquigarrow effectively 2 recursive calls!

1. recursive call inside choosePivotMoM on $m \leq \frac{n}{5}$ elements
2. recursive call inside quickselectMoM



\rightsquigarrow partition excludes $\sim 3 \cdot \frac{m}{2} \sim \frac{3}{10}n$ elem.

$$\begin{aligned}
 \rightsquigarrow C(n) &\leq \Theta(n) + C\left(\frac{1}{5}n\right) + C\left(\frac{7}{10}n\right) \\
 &\leq \Theta(n) + C\left(\frac{1}{5}n + \frac{7}{10}n\right) \\
 &= \Theta(n) + C\left(\frac{9}{10}n\right) \rightsquigarrow C(n) = \Theta(n)
 \end{aligned}$$

ansatz: overall cost linear

5.4 Fast Multiplication

Integer Multiplication

- ▶ What's the cost of computing $x \cdot y$ for two integers x and y ?

↪ depends on how big the numbers are!

- ▶ If x and y have $O(w)$ bits, multiplication takes $O(1)$ time on word-RAM
- ▶ otherwise, need a dedicated algorithm!

Long multiplication (»Schulmethode«)

- ▶ Given $x = \sum_{i=0}^{n-1} x_i 2^i$ and $y = \sum_{i=0}^{n-1} y_i 2^i$, want $z = \sum_{i=0}^{2n-1} z_i 2^i$

```
1 for i := 0, ..., n - 1
2   c := 0
3   for j := 0, ..., n - 1
4      $z_{i+j} := z_{i+j} + c + x_i \cdot y_j$ 
5      $c := \lfloor z_{i+j} / 2 \rfloor$ 
6      $z_{i+j} := z_{i+j} \bmod 2$ 
7   end for
8    $z_{i+n} := c$ 
9 end for
```

- ▶ $\Theta(n^2)$ bit operations
- ▶ could work with base 2^w instead of 2
↪ $\Theta((n/w)^2)$ time
- ▶ here: count bit operations for simplicity
can be generalized

Example:

easier in binary!
("shift and add")

```
1001010101 * 101101
-----
1001010101
0000000000
1001010101
1001010101
0000000000
1001010101
-----
110100011110001
```


Divide & Conquer Multiplication

- ▶ assume n is power of 2 (fill up with 0-bits otherwise)

- ▶ We can write

- ▶ $x = a_1 2^{n/2} + a_2$ and

- ▶ $y = b_1 2^{n/2} + b_2$

- ▶ for a_1, a_2, b_1, b_2 integers with $n/2$ bits

$$\rightsquigarrow x \cdot y = (a_1 2^{n/2} + a_2) \cdot (b_1 2^{n/2} + b_2) = a_1 b_1 2^n + (a_1 b_2 + a_2 b_1) 2^{n/2} + a_2 b_2$$

- ▶ recursively compute 4 smaller products
 - ▶ combine with shifts and additions ($O(n)$ bit operations)

- ▶ ...but is this any good?

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

\rightsquigarrow Master Theorem Case 1: $T(n) = \Theta(n^2)$... just like the primary school method!?

- ▶ but Master Theorem gives us a hint: cost is dominated by the leaves

\rightsquigarrow try to do more work in conquer step!

Karatsuba Multiplication

- ▶ how can we do “less divide and more conquer”?

Recall: $x \cdot y = a_1b_12^n + (a_1b_2 + a_2b_1)2^{n/2} + a_2b_2$

💡 Let's do some algebra.

$$\begin{aligned}c &:= (a_1 + a_2) \cdot (b_1 + b_2) \\&= a_1b_1 + (a_1b_2 + a_2b_1) + a_2b_2\end{aligned}$$

$$\rightsquigarrow (a_1b_2 + a_2b_1) = c - a_1b_1 - a_2b_2$$

this can be computed with **3** recursive multiplications

$a_1 + a_2$ and $b_1 + b_2$ still have roughly $n/2$ bits

```
1 procedure karatsuba(x, y):
2   // Assume x and y are  $n = 2^k$  bit integers
3    $a_1 := \lfloor x/2^{n/2} \rfloor$ ;  $a_2 := x \bmod 2^{n/2}$  // implemented by shifts
4    $b_1 := \lfloor y/2^{n/2} \rfloor$ ;  $b_2 := y \bmod 2^{n/2}$ 
5    $c_1 := \text{karatsuba}(a_1, b_1)$ 
6    $c_2 := \text{karatsuba}(a_2, b_2)$ 
7    $c := \text{karatsuba}(a_1 + a_2, b_1 + b_2) - c_1 - c_2$ 
8   return  $c_12^n + c2^{n/2} + c_2$  // shifts and additions
```

Analysis:

- ▶ nonrecursive cost: only additions and shifts
 - ▶ all numbers $O(n)$ bits
- \rightsquigarrow conquer cost $f(n) = \Theta(n)$

Recurrence:

- ▶ $T(n) = 3T(n/2) + \Theta(n)$
 - ▶ Master Theorem Case 1
- $\rightsquigarrow T(n) = \Theta(n^{\lg 3}) = O(n^{1.585})$

much cheaper (for large n)!

Integer Multiplication

- ▶ until 1960, integer multiplication was conjectured to take $\Omega(n^2)$ bit operations

↪ Karatsuba's algorithm was a big breakthrough

- ▶ which he discovered as a student!
 - ▶ idea can be generalized to breaking numbers into $k \geq 2$ parts (*Toom-Cook algorithm*)
 - ▶ asymptotically *much* better algorithms are now known!
 - ▶ e. g., the *Schönhage-Strassen algorithm* with $O(n \log n \log \log n)$ bit operations (!)
 - ▶ these are based on the *Fast Fourier Transform* (FFT) algorithm
 - ▶ numbers = polynomials evaluated at base (e. g., $z = 2$)
- ↪ multiplication of numbers = convolution of polynomials
- ▶ FFT makes computation of this convolution cheap by computing the polynomial via interpolation
 - ▶ Schönhage-Strassen adds careful finite-field algebra to make computations efficient

Matrix Multiplication

- ▶ The same trick can also be used for faster matrix multiplication

▶ Recall: For $A, B \in \mathbb{R}^{n \times n}$ we define $C = A \cdot B$ via $c_{i,j} = \sum_{k=1}^n a_{i,k} b_{k,j}$ entry of A in row i and column k

↪ Naive cost: n^2 sums with n terms each ↪ $\Theta(n^3)$ arithmetic operations

- ▶ Can use D&C as follows (assuming n is a power of 2 again)

▶ Decompose (cut in half hor. & vert.) $A = \begin{pmatrix} A_{1,1} & A_{1,2} \\ A_{2,1} & A_{2,2} \end{pmatrix}, \quad B = \begin{pmatrix} B_{1,1} & B_{1,2} \\ B_{2,1} & B_{2,2} \end{pmatrix}, \quad C = \begin{pmatrix} C_{1,1} & C_{1,2} \\ C_{2,1} & C_{2,2} \end{pmatrix}$

↪ We get C as $C_{1,1} = A_{1,1} \cdot B_{1,1} + A_{1,2} \cdot B_{2,1}$
 $C_{1,2} = A_{1,1} \cdot B_{1,2} + A_{1,2} \cdot B_{2,2}$ (note “ \cdot ” and “ $+$ ” operate on matrices here)
 $C_{2,1} = A_{2,1} \cdot B_{1,1} + A_{2,2} \cdot B_{2,1}$
 $C_{2,2} = A_{2,1} \cdot B_{1,2} + A_{2,2} \cdot B_{2,2}$

4 matrix sums with $(\frac{n}{2})^2$ entries each

- ▶ 8 recursive matrix multiplications on two $\frac{n}{2} \times \frac{n}{2}$ matrices + $\Theta(n^2)$ summations

- ▶ # operations $T(n) = 8T(n/2) + \Theta(n^2)$

↪ Master Theorem Case 1: $T(n) = \Theta(n^3)$ 😞

(but: still useful for better memory locality!)

Strassen Algorithm for Matrix Multiplication

► Observation (again): Can do more conquer for less divide!

► We recursively compute the following 7 products:

$$M_1 := (A_{1,2} - A_{2,2}) \cdot (B_{2,1} + B_{2,2})$$

$$M_2 := (A_{1,1} + A_{2,2}) \cdot (B_{1,1} + B_{2,2})$$

$$M_3 := (A_{1,1} - A_{2,1}) \cdot (B_{1,1} + B_{1,2})$$

$$M_4 := (A_{1,1} + A_{1,2}) \cdot B_{2,2}$$

$$M_5 := A_{1,1} \cdot (B_{1,2} - B_{2,2})$$

$$M_6 := A_{2,2} \cdot (B_{2,1} - B_{1,1})$$

$$M_7 := (A_{2,1} + A_{2,2}) \cdot B_{1,1}$$

↪ We then obtain the 4 parts of C as

$$C_{1,1} = M_1 + M_2 - M_4 + M_6$$

$$C_{1,2} = M_4 + M_5$$

$$C_{2,1} = M_6 + M_7$$

$$C_{2,2} = M_2 - M_3 + M_5 - M_7$$

(Proof: left as exercise 😊)

Analysis:

► **conquer step:** larger but still
 $O(1)$ # matrix add/subtract

↪ $\Theta(n^2)$ operations for conquer

↪ total # arithmetic operations
 $T(n) = 7T(n/2) + \Theta(n^2)$

↪ Master Theorem Case 1:
 $T(n) = \Theta(n^{\lg 7}) = O(n^{2.808})$

Open Problems

*Multiplication is extremely fundamental, but its **computational complexity** is an open problem and subject of active research!*

Integer multiplication:

- ▶ **conjectured** to require $\Omega(n \log n)$ bit operations (no proof known!)
- ▶ Harvey & van der Hoeven **2021**: $O(n \log n)$ algorithm possible!

Matrix multiplication (MM):

- ▶ more relevant than it might seem since complexity identical to
 - ▶ computing inverse matrices, determinants
 - ▶ Gaussian elimination (\rightsquigarrow solving systems of linear equations)
 - ▶ recognition of context free languages

\rightsquigarrow best exponent even has standard notation:
smallest $\omega \in [2, 3)$ so that MM takes $O(n^\omega)$ operations

- ▶ Big open question: Is $\omega > 2$?
- ▶ best known bound: $\omega \leq 2.371339$ (from 2024!)

Timeline of matrix multiplication exponent		
Year	Bound on omega	Authors
1969	2.8074	Strassen ^[1]
1978	2.796	Pan ^[10]
1979	2.780	Bini, Capovani [8], Roman ^[11]
1981	2.522	Schönhage ^[12]
1981	2.517	Roman ^[13]
1981	2.496	Coppersmith, Winograd ^[14]
1986	2.479	Strassen ^[15]
1990	2.3755	Coppersmith, Winograd ^[16]
2010	2.3737	Stothers ^[17]
2012	2.3729	Williams ^{[18][19]}
2014	2.3728639	Le Gall ^[20]
2020	2.3728596	Alman, Williams ^{[21][22]}
2022	2.371866	Duan, Wu, Zhou ^[23]
2024	2.371552	Williams, Xu, Xu, and Zhou ^[2]
2024	2.371339	Alman, Duan, Williams, Xu, Xu, and Zhou ^[24]

5.5 Majority

Majority

- ▶ **Given:** Array $A[0..n)$ of objects
- ▶ **Goal:** Check if there is an object x that occurs at $> \frac{n}{2}$ positions in A
if so, return x
- ▶ Naive solution: check each $A[i]$ whether it is a majority $\rightsquigarrow \Theta(n^2)$ time

Majority – Divide & Conquer

Can be solved faster using a simple Divide & Conquer approach:

- If A has a majority, that element must also be a majority of at least one half of A .

↪ Can find majority (if it exists) of left half and right half recursively

↪ Check these ≤ 2 candidates.

- Costs similar to mergesort $\Theta(n \log n)$

```
1 procedure majority(A[0..n])
2   if  $n == 1$  then return  $A[0]$  end if
3    $k := \lfloor \frac{n}{2} \rfloor$ 
4    $M_\ell := \text{majority}(A[0..k])$ 
5    $M_r := \text{majority}(A[k..n])$ 
6   if  $M_\ell == M_r$  then return  $M_\ell$  end if
7    $m_\ell := 0$ ;  $m_r := 0$ 
8   for  $i := 0, \dots, n - 1$ 
9     if  $A[i] == M_\ell$  then  $m_\ell = m_\ell + 1$  end if
10    if  $A[i] == M_r$  then  $m_r = m_r + 1$  end if
11  end for
12  if  $m_\ell \geq k + 1$ 
13    return  $M_\ell$ 
14  else if  $m_r \geq k + 1$ 
15    return  $M_r$ 
16  else
17    return NO_MAJORITY_ELEMENT
```

Majority – Linear Time

We can actually do much better!

```
1 def MJRTY(A[0..n])
2   c := 0
3   for i := 1, ..., n - 1
4     if c == 0
5       x := A[i]; c := 1
6     else
7       if A[i] == x then c := c + 1 else c := c - 1
8   return x
```

► MJRTY(A[0..n]) returns *candidate* majority element

► either that candidate is the majority element or none exists(!)

👍 Clearly $\Theta(n)$ time



5.6 Closest Pair of Points in the Plane

Closest Pair of Points in the Plane

- ▶ **Given:** Array $P[0..n]$ of points in the plane (\mathbb{R}^2)
each has x and y coordinates: $P[i].x$ and $P[i].y$
- ▶ **Goal:** Find pair $P[i], P[j]$ that is closest in (Euclidean) distance
i. e., i and j that minimize $\sqrt{(P[i].x - P[j].x)^2 + (P[i].y - P[j].y)^2}$
- ▶ Naive solution: compute distance of each pair $\rightsquigarrow \Theta(n^2)$ time
 - ▶ cost here = # arithmetic operations
 - ▶ ignore numerical accuracy
- \rightsquigarrow formally work on the *real RAM*
 - ▶ like word-RAM, but words contain **exact** real numbers
 - ▶ support arithmetic operations and comparisons,
but **not** bitwise operations or $\lfloor \cdot \rfloor$ and $\lceil \cdot \rceil$

Closest Pair – Divide & Conquer

Closest Pair – Refined Conquer

Closest Pair – Code

```
1 procedure closestDist( $P[0..n]$ ,  $byX[0..n]$ ,  $byY[0..n]$ )
2   //  $P$  contains  $n$  points with distinct  $x$ -coordinates
3   //  $P[byX[0]].x \leq P[byX[1]].x \leq \dots \leq P[byX[n]].x$ 
4   //  $P[byY[0]].y \leq P[byY[1]].y \leq \dots \leq P[byY[n]].y$ 
5   if  $n == 2$  return  $d_2(P[0], P[1])$ 
6   if  $n == 3$  return  $\min\{d_2(P[0], P[1]),$ 
7        $d_2(P[1], P[2]), d_2(P[0], P[2])\}$ 
8   // 1. Split by median  $x$  and recurse
9    $k := \lfloor n/2 \rfloor$ ;  $m := byX[k]$ 
10   $byX_L := byX[0..m]$ ;  $byX_R := byX[m..n]$ 
11   $P_L, P_R, byY_L, byY_R :=$  new empty array
12  for  $i := 0, \dots, n-1$ 
13    if  $P[byY[i]].x \leq m$ 
14       $P_L.append(P[byY[i]])$ 
15       $byY_L.append(byY[i])$ 
16    else
17       $P_R.append(P[byY[i]])$ 
18       $byY_R.append(byY[i])$ 
19    end if
20  end for
21  // ...
```

```
22  // ... closestDist continued
23   $\delta_L :=$  closestDist( $P_L, byX_L, byY_L$ )
24   $\delta_R :=$  closestDist( $P_R, byX_R, byY_R$ )
25   $\delta := \min\{\delta_L, \delta_R\}$ 
26  // 2. Check pairs straddling  $x = m$  line
27  // Find points close to  $m$ 
28  for  $i := 0, \dots, n-1$ 
29    if  $|P[byY[i]].x - m| \leq \delta$ 
30       $C.append(byY[i])$ 
31    end if
32  end for
33  // Distance  $\leq \delta$  implies within 8 positions in  $C$ 
34  for  $i := 0, \dots, C.size()$ 
35    for  $j := i+1, \dots, i+7$ 
36       $\delta := \min\{\delta, d_2(P[C[i]], P[C[j]])\}$ 
37    end for
38  end for
39  return  $\delta$ 
40
41  procedure  $d_2(P, Q)$ 
42  return  $\sqrt{(P.x - Q.x)^2 + (P.y - Q.y)^2}$ 
```
