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?

<table>
<thead>
<tr>
<th>Run time (nanoseconds)</th>
<th>1.3 $N^3$</th>
<th>$10 N^2$</th>
<th>$47 N \log_2 N$</th>
<th>$48 N$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time to solve a problem of size</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1000</td>
<td>1.3 seconds</td>
<td>10 msec</td>
<td>0.4 msec</td>
<td>0.048 msec</td>
</tr>
<tr>
<td>10,000</td>
<td>22 minutes</td>
<td>1 second</td>
<td>6 msec</td>
<td>0.48 msec</td>
</tr>
<tr>
<td>100,000</td>
<td>15 days</td>
<td>1.7 minutes</td>
<td>78 msec</td>
<td>4.8 msec</td>
</tr>
<tr>
<td>million</td>
<td>41 years</td>
<td>2.8 hours</td>
<td>0.94 seconds</td>
<td>48 msec</td>
</tr>
<tr>
<td>10 million</td>
<td>41 millennia</td>
<td>1.7 weeks</td>
<td>11 seconds</td>
<td>0.48 seconds</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Max size problem solved in one time</th>
</tr>
</thead>
<tbody>
<tr>
<td>second</td>
</tr>
<tr>
<td>minute</td>
</tr>
<tr>
<td>hour</td>
</tr>
<tr>
<td>day</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$N$ multiplied by 10, time multiplied by</th>
</tr>
</thead>
<tbody>
<tr>
<td>1,000</td>
</tr>
</tbody>
</table>

Orders of Magnitude

<table>
<thead>
<tr>
<th>Seconds</th>
<th>Equivalent</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1 second</td>
</tr>
<tr>
<td>10</td>
<td>10 seconds</td>
</tr>
<tr>
<td>$10^2$</td>
<td>1.7 minutes</td>
</tr>
<tr>
<td>$10^3$</td>
<td>17 minutes</td>
</tr>
<tr>
<td>$10^4$</td>
<td>2.8 hours</td>
</tr>
<tr>
<td>$10^5$</td>
<td>1.1 days</td>
</tr>
<tr>
<td>$10^6$</td>
<td>1.6 weeks</td>
</tr>
<tr>
<td>$10^7$</td>
<td>3.8 months</td>
</tr>
<tr>
<td>$10^8$</td>
<td>3.1 years</td>
</tr>
<tr>
<td>$10^9$</td>
<td>3.1 decades</td>
</tr>
<tr>
<td>$10^{10}$</td>
<td>3.1 centuries</td>
</tr>
</tbody>
</table>

... forever

$10^{21}$ age of universe

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

Powers of 2

- $2^{10}$ thousand
- $2^{20}$ million
- $2^{30}$ billion
A Useful Recurrence Relation

\[ T(N) = \text{worst case running time on input of size } N. \]

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

\[ T(N) \leq \begin{cases} O(1) & \text{if } N \leq n_0 \\ T(\lceil N/2 \rceil) + T(\lfloor N/2 \rfloor) + O(N) & \text{otherwise} \end{cases} \]

- 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.

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

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, \ldots, 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 \( a_i > a_j \)

<table>
<thead>
<tr>
<th>Songs</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>Me</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>You</td>
<td>1</td>
<td>3</td>
<td>4</td>
<td>2</td>
<td>5</td>
</tr>
</tbody>
</table>

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

\[ 1 \quad 5 \quad 4 \quad 8 \quad 10 \quad 2 \quad 6 \quad 9 \quad 12 \quad 11 \quad 3 \quad 7 \]

\[ 1 \quad 5 \quad 4 \quad 8 \quad 10 \quad 2 \quad 6 \quad 9 \quad 12 \quad 11 \quad 3 \quad 7 \]

O(1)

5 blue-blue inversions
8 green-green inversions

2T(N / 2)

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.

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}  
O(N^2)

Total = 5 + 8 + 9 = 22.

Can we do this step in sub-quadratic time?

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.

13 blue-green inversions: $6 + 3 + 2 + 2 + 0 + 0$

\[ T(N) = T\left(\left\lfloor N/2 \right\rfloor\right) + T\left(\left\lceil N/2 \right\rceil\right) + O(N) \Rightarrow T(N) = O(N \log N) \]

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.
- Conquer: find closest pair in each side recursively.
- Combine: find closest pair with one point in each side.

Key step: find closest pair with one point in each side.
- Extra information: closest pair entirely in one side had distance $\delta$.

$\delta = \min(12, 21)$
**Closest Pair**

Key step: find closest pair with one point in each side.
- Extra information: closest pair entirely in one side had distance \( \delta \).
- Observation: only need to consider points \( S \) within \( \delta \) of line.

\[ \delta = \min(12, 21) \]

\[ \delta = \min(12, 21) \]

**Extra information:**
- Closest pair entirely in one side had distance \( \delta \).

**Observation:**
- Only need to consider points \( S \) within \( \delta \) of line.

- Sort points in strip \( S \) by their y coordinate.
  - Sufficient to compute distances for pairs within constant number of positions of each other in sorted list!

- Crucial fact: if \( p \) and \( q \) are in \( S \), and if \( d(p, q) < \delta \), 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 \( \geq 2\delta / 2 \).

**Closest Pair**

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

\[ \delta = \text{ClosestPair}(p_1, p_2, \ldots, p_N) \]

\[ O(N \log N) \]

\[ 2T(N/2) \]

\[ O(N) \]

\[ O(N \log N) \]

\[ T(N) = T\left(\left\lfloor N/2 \right\rfloor \right) + T\left(\left\lceil N/2 \right\rceil \right) + O(N \log N) \Rightarrow T(N) = O(N \log^2 N) \]
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.

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

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.

<table>
<thead>
<tr>
<th>1</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>+</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

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.

**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.

$$123,456 \times 987,654 = (10^3 w + x) \times (10^3 y + z)$$

$$= 10^6 (wy) + 10^3 (wz + xy) + 10^0 (xz) = 10^6 (121,401) + 10^3 (80,442 + 450,072) + 10^0 (298,224)$$

$$= 121,401,299,224$$

<table>
<thead>
<tr>
<th>$w$</th>
<th>123</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x$</td>
<td>456</td>
</tr>
<tr>
<td>$y$</td>
<td>987</td>
</tr>
<tr>
<td>$z$</td>
<td>654</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$w$</th>
<th>123</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x$</td>
<td>456</td>
</tr>
<tr>
<td>$y$</td>
<td>987</td>
</tr>
<tr>
<td>$z$</td>
<td>654</td>
</tr>
</tbody>
</table>

**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.

$$123,456 \times 987,654 = (10^3 w + x) \times (10^3 y + z)$$

$$= 10^6 (wy) + 10^3 (wz + xy) + 10^0 (xz) = 10^6 (p) + 10^3 (r - p - q) + 10^0 (q)$$

$$= 10^6 (121,401) + 10^3 (950,139 - 121,401 - 298,224) + 10^0 (298,224)$$

$$= 121,401,299,224$$

<table>
<thead>
<tr>
<th>$w$</th>
<th>123</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x$</td>
<td>456</td>
</tr>
<tr>
<td>$y$</td>
<td>987</td>
</tr>
<tr>
<td>$z$</td>
<td>654</td>
</tr>
</tbody>
</table>

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

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

\[
p = wy \\
q = xz \\
r = (w + x)(y + z)
\]

\[
T(N) \leq T(\lceil N/2 \rceil) + T(\lceil N/2 \rceil) + T(1 + \lceil N/2 \rceil) + \Theta(N) \\
\]

\[
\Rightarrow T(N) = O(N \log_2 3)
\]

Matrix Multiplication:

Given two \(N \times N\) matrices \(A\) and \(B\), compute \(C = AB\).

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

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

- Hard to imagine naïve algorithm can be improved upon.

Matrix Multiplication: Warmup

Warmup: divide-and-conquer.
- Divide: partition \(A\) and \(B\) into \(N/2 \times N/2\) blocks.
- Conquer: multiply \(8 N/2 \times N/2\) recursively.
- Combine: add appropriate products using 4 matrix additions.

\[
C_{11} = (A_{11} \times B_{11}) + (A_{12} \times B_{21}) \\
C_{12} = (A_{11} \times B_{12}) + (A_{12} \times B_{22}) \\
C_{21} = (A_{21} \times B_{11}) + (A_{22} \times B_{21}) \\
C_{22} = (A_{21} \times B_{12}) + (A_{22} \times B_{22})
\]

\[
T(N) = 8T(N/2) + \Theta(N^2) \Rightarrow T(N) = \Theta(N^3)
\]

Matrix Multiplication: Idea

Idea: multiply 2\(\times\)2 matrices with only 7 scalar multiplications.

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

- 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.

\[
\begin{bmatrix}
C_{11} & C_{12} \\
C_{21} & C_{22}
\end{bmatrix}
= 
\begin{bmatrix}
A_{11} & A_{12} \\
A_{21} & A_{22}
\end{bmatrix}
\begin{bmatrix}
B_{11} & B_{12} \\
B_{21} & B_{22}
\end{bmatrix}
\]

Analysis.
- Assume N is a power of 2.
- \( T(N) = \# \text{arithmetic operations} \)

\[
T(N) = 7T(N/2) + \Theta(N^2) 
\Rightarrow T(N) = \Theta(N^{\log_2 7}) = O(N^{2.81})
\]

Beyond Strassen

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

Can you multiply two 2 x 2 matrix with only 6 scalar multiplications?
- Impossible (Hopcroft and Kerr, 1971).

Two 3 x 3 matrices with only 21 scalar multiplications?
- Also impossible.

Two 70 x 70 matrices with only 143,640 scalar multiplications?
- Yes! (Pan, 1980).

Decimal wars.
- December, 1979: \( O(N^{2.52113}) \).
- 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 \( O(N \log N) \) 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 $a_1, a_2, \ldots, a_N$.

```plaintext
FastSelect (i, N, a_1, a_2, \ldots, a_N)
```

$x \leftarrow \text{FastPartition}(N, a_1, a_2, \ldots, a_N)$  
$k \leftarrow \text{rank}(x)$

- if ($i == k$)  
  return $x$
- else if ($i < k$)  
  $b[] \leftarrow \text{all items of } a[] \text{ less than } x$  
  return $\text{FastSelect}(i^{th}, k-1, b_1, b_2, \ldots, b_{k-1})$
- else if ($i > k$)  
  $c[] \leftarrow \text{all items of } a[] \text{ greater than } x$  
  return $\text{FastSelect}((i-k)^{th}, N-k, c_1, c_2, \ldots, c_{N-k})$
```

**Fast Partition**

`FastPartition()`.
- Divide $N$ elements into $\lfloor N/5 \rfloor$ groups of 5 elements each, plus extra.
- Brute force sort each of the 5-element groups.

Find $x = \text{"median of medians" using } \text{FastSelect()}$ recursively.
Fast Selection and Fast Partition

**FastPartition().**
- Divide N elements into \( \lfloor N/5 \rfloor \) 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 \( \leq x \)
  - at least \( \lfloor N/5 \rfloor / 2 = \lfloor N/10 \rfloor \) medians \( \leq x \)
- At least 3 \( \lfloor N/10 \rfloor \) elements \( \leq x \).

**Median of medians**

```
14  9  05  03  02  12  01  17  20  04  36
22 10  06  11  25  16  13  24  31  07  27
28 23  38  15  40  19  18  43  32  35  08
29 39  50  26  53  30  41  46  33  49  21
45 44  52  37  54  53  48  47  34  51
```
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 \( \lfloor N/5 \rfloor / 2 = \lfloor N/10 \rfloor \) medians ≤ x
- At least 3 \( \lfloor N/10 \rfloor \) elements ≤ x.
- At least 3 \( \lfloor N/10 \rfloor \) elements ≥ x.
  ⇒ FastSelect() called recursively with at most \( N - 3 \lfloor N/10 \rfloor \) elements in last step

\[
T(N) \leq \frac{c}{2} T(\lfloor N/5 \rfloor) + \frac{c}{2} T(\lfloor N/10 \rfloor) + cN \quad \text{if } N < 50
\]

Claim: \( T(N) \leq 20cN \).
- Base case: \( N < 50 \).
- Inductive step: assume true for 1, 2, \ldots, N-1.

\[
T(N) \leq T(\lfloor N/5 \rfloor) + T(\lfloor N-3 \rfloor/10) + cN
\]

\[
\leq 20c \lfloor N/5 \rfloor + 20c \lfloor N-3 \rfloor/10) + cN
\]

\[
\leq 20c(N/5) + 20c(N) - 20c(N/4) + cN
\]

\[
= 20cN
\]

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.