Chapter 5

Divide and Conquer

CLRS 4.3
Divide-and-Conquer

Divide-and-conquer.
- Break up problem into several parts.
- Solve each part recursively.
- Combine solutions to sub-problems into overall solution.

Most common usage.
- Break up problem of size \( n \) into two equal parts of size \( \frac{1}{2}n \).
- Solve two parts recursively.
- Combine two solutions into overall solution in linear time.

Consequence.
- Brute force: \( n^2 \).
- Divide-and-conquer: \( n \log n \).

Divide et impera.
Veni, vidi, vici.
- Julius Caesar
5.1 Mergesort
**Sorting**

Sorting. Given \( n \) elements, rearrange in ascending order.

**Obvious sorting applications.**
- List files in a directory.
- Organize an MP3 library.
- List names in a phone book.
- Display Google PageRank results.

**Problems become easier once sorted.**
- Find the median.
- Find the closest pair.
- Binary search in a database.
- Identify statistical outliers.
- Find duplicates in a mailing list.

**Non-obvious sorting applications.**
- Data compression.
- Computer graphics.
- Interval scheduling.
- Computational biology.
- Minimum spanning tree.
- Supply chain management.
- Simulate a system of particles.
- Book recommendations on Amazon.
- Load balancing on a parallel computer.

...
Mergesort

Mergesort.
- Divide array into two halves.
- Recursively sort each half.
- Merge two halves to make sorted whole.

Jon von Neumann (1945)

<table>
<thead>
<tr>
<th>ALGORM</th>
<th>ITHMS</th>
<th>divide</th>
<th>O(1)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALGOR</td>
<td>ITHMS</td>
<td>sort</td>
<td>2T(n/2)</td>
</tr>
<tr>
<td>AGLOR</td>
<td>HIMST</td>
<td>merge</td>
<td>O(n)</td>
</tr>
<tr>
<td>AGHILMORS</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Merging. Combine two pre-sorted lists into a sorted whole.

How to merge efficiently?
- Linear number of comparisons.
- Use temporary array.

Challenge for the bored. In-place merge. [Kronrod, 1969]

using only a constant amount of extra storage
Merging

Merge.

- Keep track of smallest element in each sorted half.
- Insert smallest of two elements into auxiliary array.
- Repeat until done.

**auxiliary array**
Merging

Merge.

- Keep track of smallest element in each sorted half.
- Insert smallest of two elements into auxiliary array.
- Repeat until done.

\[
\begin{array}{cccccc}
A & G & L & O & R & \\
\end{array}
\]

\[
\begin{array}{cccccc}
H & I & M & S & T & \\
\end{array}
\]

\[
\begin{array}{cccccc}
A & G & & & & \\
\end{array}
\]

auxiliary array
Merging

Merge.

- Keep track of smallest element in each sorted half.
- Insert smallest of two elements into auxiliary array.
- Repeat until done.

![Diagram showing merging process with auxiliary array]
Merging

Merge.
- Keep track of smallest element in each sorted half.
- Insert smallest of two elements into auxiliary array.
- Repeat until done.

![Diagram showing merging process with auxiliary array](image-url)
Merging

**Merge.**

- Keep track of smallest element in each sorted half.
- Insert smallest of two elements into auxiliary array.
- Repeat until done.
Merging

Merge.

- Keep track of smallest element in each sorted half.
- Insert smallest of two elements into auxiliary array.
- Repeat until done.
Merge.

- Keep track of smallest element in each sorted half.
- Insert smallest of two elements into auxiliary array.
- Repeat until done.
Merging

Merge.

- Keep track of smallest element in each sorted half.
- Insert smallest of two elements into auxiliary array.
- Repeat until done.
Merging

Merge.

- Keep track of smallest element in each sorted half.
- Insert smallest of two elements into auxiliary array.
- Repeat until done.

first half exhausted

smallest

A G L O R H I M S T

A G H I L M O R S auxiliary array
Merging

Merge.

- Keep track of smallest element in each sorted half.
- Insert smallest of two elements into auxiliary array.
- Repeat until done.

![Diagram of merging process]

first half exhausted

smallest

auxiliary array
Merging

Merge.

- Keep track of smallest element in each sorted half.
- Insert smallest of two elements into auxiliary array.
- Repeat until done.

```
     A  G  L  O  R
first half exhausted

     H  I  M  S  T
second half exhausted

     A  G  H  I  L  M  O  R  S  T
auxiliary array
```
A Useful Recurrence Relation

Def. \( T(n) = \) number of comparisons to mergesort an input of size \( n \).

Mergesort recurrence.

\[
T(n) \leq \begin{cases} 
0 & \text{if } n = 1 \\
T(\lceil n/2 \rceil) & \text{solve left half} \\
T(\lfloor n/2 \rfloor) & \text{solve right half} \\
n & \text{merging}
\end{cases}
\]

Solution. \( T(n) \in O(n \log_2 n) \).

Assorted proofs. We describe several ways to prove this recurrence. Initially we assume \( n \) is a power of 2 and replace \( \leq \) with \( = \).
Proof by Recursion Tree

\[ T(n) = \begin{cases} 
0 & \text{if } n = 1 \\
\frac{2T(n/2)}{\text{sorting both halves}} + \frac{n}{\text{merging}} & \text{otherwise}
\end{cases} \]

\[ T(n) \]

\[ T(n/2) \quad \text{and} \quad T(n/2) \]

\[ T(n/4) \quad \text{and} \quad T(n/4) \quad \text{and} \quad T(n/4) \quad \text{and} \quad T(n/4) \]

\[ T(2) \quad \text{and} \quad T(2) \quad \text{and} \quad T(2) \quad \text{and} \quad T(2) \quad \text{and} \quad T(2) \quad \text{and} \quad T(2) \quad \text{and} \quad T(2) \]

\[ \log_2 n \]

\[ n \log_2 n \]
Claim. If $T(n)$ satisfies this recurrence, then $T(n) = n \log_2 n$.

Proof by Induction

Pf. (by induction on $n$)

- **Base case:** $n = 1$.
- **Inductive hypothesis:** $T(n) = n \log_2 n$.
- **Goal:** show that $T(2n) = 2n \log_2 (2n)$.

$$T(2n) = 2T(n) + 2n$$

$$= 2n \log_2 n + 2n$$

$$= 2n \log_2 (2n) - 1 + 2n$$

$$= 2n \log_2 (2n)$$
Claim. If \( T(n) \) satisfies the following recurrence, then \( T(n) \leq n \lceil \lg n \rceil \).

\[
T(n) \leq \begin{cases} 
0 & \text{if } n = 1 \\
T\left(\left\lfloor n/2 \right\rfloor\right) + T\left(\left\lceil n/2 \right\rceil\right) + n & \text{otherwise}
\end{cases}
\]

\( \log_2 n \)

Pf. (by induction on \( n \))

- Base case: \( n = 1 \). \( T(1) = 0 = 1 \lceil \lg 1 \rceil \).
- Define \( n_1 = \lceil n / 2 \rceil \), \( n_2 = \lfloor n / 2 \rfloor \).
- Induction step: Let \( n \geq 2 \), assume true for 1, 2, \ldots, \( n-1 \).

\[
T(n) \leq T(n_1) + T(n_2) + n \\
\leq n_1 \lceil \lg n_1 \rceil + n_2 \lfloor \lg n_2 \rfloor + n \\
\leq n_1 \lceil \lg n_2 \rceil + n_2 \lfloor \lg n_2 \rfloor + n \\
= n \lfloor \lg n_2 \rfloor + n \\
\leq n(\lfloor \lg n \rfloor - 1) + n \\
= n \lceil \lg n \rceil
\]

\[
n_2 = \left\lfloor n/2 \right\rfloor \\
\leq \left\lfloor 2^{\lceil \lg n \rceil} / 2 \right\rfloor \\
= 2^{\lceil \lg n \rceil} / 2 \\
\Rightarrow \lg n_2 \leq \left\lfloor \lg n \right\rfloor - 1
\]
5.5 Integer Multiplication
Integer Arithmetic

**Add.** Given two n-digit integers $a$ and $b$, compute $a + b$.
- $\Theta(n)$ bit operations.

**Multiply.** Given two n-digit integers $a$ and $b$, compute $a \times b$.
- Straightforward solution: $\Theta(n^2)$ bit operations.
Divide-and-Conquer Multiplication: Warmup

To multiply two n-digit integers:

- Multiply four \( \frac{1}{2} n \)-digit integers.
- Add two \( \frac{1}{2} n \)-digit integers, and shift to obtain result.

\[
\begin{align*}
x &= 2^{n/2} \cdot x_1 + x_0 \\
y &= 2^{n/2} \cdot y_1 + y_0 \\
xy &= (2^{n/2} \cdot x_1 + x_0)(2^{n/2} \cdot y_1 + y_0) = 2^n \cdot x_1y_1 + 2^{n/2} \cdot (x_1y_0 + x_0y_1) + x_0y_0
\end{align*}
\]

\[
T(n) = 4T(n/2) + \Theta(n) \quad \Rightarrow \quad T(n) \in \Theta(n^2)
\]

assumes n is a power of 2
Proof by Telescoping

Claim.

\[
T(n) = \begin{cases} 
4T(n/2) & \text{recursive calls} \\
\Theta(n) & \text{add, shift}
\end{cases} \Rightarrow T(n) \in \Theta(n^2)
\]

assumes \( n \) is a power of 2

Pf. For \( n > 1 \):

\[
\frac{T(n)}{n} = \frac{4T(n/2)}{n} + C
\]

\[
= 2 \frac{T(n/2)}{(n/2)} + C
\]

\[
= 2 \left[ 2 \frac{T(n/4)}{(n/4)} + C \right] + C
\]

\[
= 4 \frac{T(n/4)}{(n/4)} + 2C + C
\]

\[
= 4 \left[ 2 \frac{T(n/8)}{(n/8)} + C \right] + 2C + C
\]

\[
= 8 \frac{T(n/8)}{(n/8)} + 4C + 2C + C
\]

\[
\vdots
\]

\[
= n \frac{T(1)}{1} + n/2 C + n/4 C + \ldots + 4C + 2C + C
\]

\[
= C \left( n/2 + n/4 + \ldots + 2 + 1 \right) = C(n-1).
\]
Karatsuba Multiplication

To multiply two n-digit integers:
- Add two \( \frac{1}{2}n \) digit integers.
- Multiply three \( \frac{1}{2}n \)-digit integers.
- Add, subtract, and shift \( \frac{1}{2}n \)-digit integers to obtain result.

\[
\begin{align*}
x &= 2^{n/2} \cdot x_1 + x_0 \\
y &= 2^{n/2} \cdot y_1 + y_0 \\
x y &= 2^n \cdot x_1 y_1 + 2^{n/2} \cdot (x_1 y_0 + x_0 y_1) + x_0 y_0 \\
&= 2^n \cdot x_1 y_1 + 2^{n/2} \cdot ((x_1 + x_0)(y_1 + y_0) - x_1 y_1 - x_0 y_0) + x_0 y_0
\end{align*}
\]

Theorem. [Karatsuba-Ofman, 1962] Can multiply two n-digit integers in \( O(n^{1.585}) \) bit operations.

\[
T(n) \leq \underbrace{T([n/2]) + T([n/2]) + T(1+[n/2])}_{\text{recursive calls}} + \underbrace{\Theta(n)}_{\text{add, subtract, shift}}
\Rightarrow T(n) \in O(n^{\log_2 3}) \in O(n^{1.585})
\]
Karatsuba: Recursion Tree

\[ T(n) = \begin{cases} 
0 & \text{if } n = 1 \\
3T(n/2) + n & \text{otherwise}
\end{cases} \]

\[ T(n) = \sum_{k=0}^{\log_2 n} n \left(\frac{3}{2}\right)^k = n \left(\frac{3^{1+\log_2 n}}{3^2 - 1}\right) - 1 = 3n^{\log_2 3} - 2 \]

\[ \sum_{k=0}^{n-1} ar^k = a \frac{1 - r^n}{1 - r} \]
**Karatsuba Multiplication**

**Generalization:** $O(n^{1+\varepsilon})$ for any $\varepsilon > 0$.

**Best known:** $(n \log n) 2^{O(\log^* n)}$

where $\log^*(x) = \begin{cases} 0 & \text{if } x \leq 1 \\ 1 + \log^*(\log x) & \text{if } x > 1 \end{cases}$

**Conjecture:** $\Omega(n \log n)$ but not proven.
CLRS 4.3 Master Theorem
Master Theorem from CLRS 4.3

Used for many divide-and-conquer recurrences

\[ T(n) = aT(n/b) + f(n) \]

where \( a \geq 1, b > 1, \) and \( f(n) > 0. \)

\[ a = \text{(constant) number of sub-instances}, \]
\[ b = \text{(constant) size ratio of sub-instances}, \]
\[ f(n) = \text{time used for dividing and recombining.} \]

Based on the \textbf{master theorem} (Theorem 4.1).

Compare \( n^{\log_b a} \) vs. \( f(n) \):
Proof by Recursion Tree

Used for many divide-and-conquer recurrences

\[ T(n) = aT(n/b) + f(n) \]

where \( a \geq 1, b > 1, \) and \( f(n) > 0. \)
\[ T(n) = aT(n/b) + f(n) \]

**Case 1:** \( f(n) \in O(n^L) \) for some constant \( L < \log_b a \).

**Solution:** \( T(n) \in \Theta(n^{\log_b a}) \)

**Case 2:** \( f(n) \in \Theta(n^{\log_b a \log^k n}) \), for some \( k \geq 0 \).

**Solution:** \( T(n) \in \Theta(n^{\log_b a \log^{k+1} n}) \)

**Case 3:** \( f(n) \in \Omega(n^L) \) for some constant \( L > \log_b a \)
and \( f(n) \) satisfies the regularity condition \( af(n/b) \leq cf(n) \) for some \( c<1 \) and all large \( n \).

**Solution:** \( T(n) \in \Theta(f(n)) \)
Master Theorem

\[ T(n) = aT(n/b) + f(n) \]
where \( a \geq 1, b > 1, \) and \( f(n) > 0. \)

**Case 1:** \( f(n) \in O(n^L) \) for some constant \( L < \log_b a. \) (\( f(n) \) is polynomially smaller than \( n^{\log_b a}. \))

**Solution:** \( T(n) \in \Theta(n^{\log_b a}) \)
(Intuitively: cost is dominated by leaves.)
Master Theorem

Case 1: \( f(n) \in O(n^L) \) for some constant \( L < \log_b a \).

Solution: \( T(n) \in \Theta(n^{\log_b a}) \)

\[
T(n) = 5T(n/2) + \Theta(n^2)
\]

Compare \( n^{\log_2 5} \) vs. \( n^2 \).

Since \( 2 < \log_2 5 \) use Case 1

Solution: \( T(n) \in \Theta(n^{\log_2 5}) \)
Master Theorem

\[ T(n) = aT(n/b) + f(n) \]

where \( a \geq 1, b > 1, \) and \( f(n) > 0. \)

**Simple Case 2:** \( f(n) \in \Theta(n^{\log_b a}). \)

**Solution:** \( T(n) \in \Theta(n^{\log_b a \log n}) \)
Master Theorem

\[ T(n) = aT(n/b) + f(n) \]

where \(a \geq 1\), \(b > 1\), and \(f(n) > 0\).

Case 2: \(f(n) \in \Theta(n^{\log_b a \log^k n})\), for some \(k \geq 0\).

Solution: \(T(n) \in \Theta(n^{\log_b a \log^{k+1} n})\)

(Intuitively: cost is \(n^{\log_b a \log^k n}\) at each level, and there are \(\Theta(\log n)\) levels.)
Case 2: \( f(n) \in \Theta(n^{\log_b a} \log^k n) \), for some \( k \geq 0 \).

Solution: \( T(n) \in \Theta(n^{\log_b a} \log^{k+1} n) \)

\[
T(n) = 27T(n/3) + \Theta(n^3 \log n)
\]

Compare \( n^{\log_3 27} \) vs. \( n^3 \).

Since \( 3 = \log_3 27 \) use Case 2

Solution: \( T(n) \in \Theta(n^3 \log^2 n) \)
Master Theorem

\[ T(n) = aT(n/b) + f(n) \]

where \( a \geq 1, b > 1, \) and \( f(n) > 0. \)

**Case 3:** \( f(n) \in \Omega(n^L) \) for some constant \( L > \log_b a \)
and \( f(n) \) satisfies the regularity condition \( af(n/b) \leq cf(n) \) for some \( c<1 \) and all large \( n. \)

(\( f(n) \) is polynomially greater than \( n^{\log_b a}. \))

**Solution:** \( T(n) \in \Theta(f(n)) \)

(Intuitively: cost is dominated by root.)
Master Theorem

**Case 3:** \( f(n) \in \Omega(n^L) \) for some constant \( L > \log_b a \)
and \( f(n) \) satisfies the regularity condition \( af(n/b) \leq cf(n) \) for some \( c<1 \) and all large \( n \).

**Solution:** \( T(n) \in \Theta(f(n)) \)

What’s with the Case 3 regularity condition?

- Generally not a problem.
- It always holds whenever \( f(n) = n^k \) and \( f(n) \in \Omega(n^{\log_b a+\varepsilon}) \) for constant \( \varepsilon > 0 \).
Master Theorem

**Case 3:** $f(n) \in \Omega(n^L)$ for some constant $L > \log_b a$ and $f(n)$ satisfies the regularity condition $af(n/b) \leq cf(n)$ for some $c<1$ and all large $n$.

**Solution:** $T(n) \in \Theta(f(n))$

\[
T(n) = 5T(n/2) + \Theta(n^3)
\]

Compare $n^\log_2 5$ vs. $n^3$.

Since $3 > \log_2 5$ use **Case 3**

\[
a f(n/b) = 5(n/2)^3 = 5/8 \ n^3 \leq \ c n^3, \text{ for } c = 5/8
\]

**Solution:** $T(n) \in \Theta(n^3)$
Master Theorem

\[ T(n) = 27T(n/3) + \Theta(n^{3}/\log n) \]

Compare \( n^{\log_{3} 27} \) vs. \( n^{3} \).

Since \( 3 = \log_{3} 27 \) use Case 2

but \( n^{3}/\log n \in \textbf{not} \ \Theta(n^{3} \log^{k} n) \) for \( k \geq 0 \)

Cannot use Master Method.
Matrix Multiplication
Matrix Multiplication

Matrix multiplication. Given two n-by-n matrices $A$ and $B$, compute $C = AB$.

Brute force. $\Theta(n^3)$ arithmetic operations.

Fundamental question. Can we improve upon brute force?
Matrix Multiplication: Warmup

Divide-and-conquer.

- Divide: partition A and B into $\frac{1}{2}n$-by-$\frac{1}{2}n$ blocks.
- Conquer: multiply $8\frac{1}{2}n$-by-$\frac{1}{2}n$ recursively.
- Combine: add appropriate products using 4 matrix additions.

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

\[
\begin{align*}
C_{11} &= (A_{11} \times B_{11}) + (A_{12} \times B_{21}) \\
C_{12} &= (A_{11} \times B_{12}) + (A_{12} \times B_{22}) \\
C_{21} &= (A_{21} \times B_{11}) + (A_{22} \times B_{21}) \\
C_{22} &= (A_{21} \times B_{12}) + (A_{22} \times B_{22})
\end{align*}
\]

\[
T(n) = 8T\left(n/2\right) + \Theta(n^2) \implies T(n) \in \Theta(n^3)
\]
Matrix Multiplication: Key Idea

Key idea. multiply 2-by-2 block matrices with only 7 multiplications.

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

\[
P_1 = A_{11} \times (B_{12} - B_{22})
\]
\[
P_2 = (A_{11} + A_{12}) \times B_{22}
\]
\[
P_3 = (A_{21} + A_{22}) \times B_{11}
\]
\[
P_4 = A_{22} \times (B_{21} - B_{11})
\]
\[
P_5 = (A_{11} + A_{22}) \times (B_{11} + B_{22})
\]
\[
P_6 = (A_{12} - A_{22}) \times (B_{21} + B_{22})
\]
\[
P_7 = (A_{11} - A_{21}) \times (B_{11} + B_{12})
\]

- 7 multiplications.
- 18 = 10 + 8 additions (or subtractions).
Strassen: Recursion Tree

\[ T(n) = \begin{cases} 
0 & \text{if } n = 1 \\
7T(n/2) + n^2 & \text{otherwise}
\end{cases} \]

\[ T(n) = \sum_{k=0}^{\log_2 n} n^2 \left( \frac{7}{4} \right)^k = n^2 \frac{\left( \frac{7}{4} \right)^{1+\log_2 n} - 1}{\frac{7}{4} - 1} \approx \frac{7}{3} n^{\log_2 7}. \]
Fast Matrix Multiplication

**Fast matrix multiplication.** (Strassen, 1969)
- **Divide:** partition A and B into $\frac{1}{2}n$-by-$\frac{1}{2}n$ blocks.
- **Compute:** 14 $\frac{1}{2}n$-by-$\frac{1}{2}n$ matrices via 10 matrix additions.
- **Conquer:** multiply 7 $\frac{1}{2}n$-by-$\frac{1}{2}n$ matrices recursively.
- **Combine:** 7 products into 4 terms using 18 matrix additions.

**Analysis.**
- Assume $n$ is a power of 2.
- $T(n) = \#$ arithmetic operations.

$$T(n) = 7T\left(\frac{n}{2}\right) + \Theta(n^2) \quad \Rightarrow \quad T(n) \in \Theta(n^{\log_2 7}) \in O(n^{2.81})$$
Fast Matrix Multiplication in Practice

Implementation issues.
- Sparsity.
- Caching effects.
- Numerical stability.
- Odd matrix dimensions.
- Crossover to classical algorithm around $n = 128$.

Common misperception: "Strassen is only a theoretical curiosity."
- Advanced Computation Group at Apple Computer reports 8x speedup on G4 Velocity Engine when $n \sim 2,500$.
- Range of instances where it's useful is a subject of controversy.

Remark. Can "Strassenize" $Ax=b$, determinant, eigenvalues, and other matrix ops.
Q. Multiply two 2-by-2 matrices with only 7 scalar multiplications?
A. Yes! [Strassen, 1969] \( \Theta(n^\log_2 7) \in O(n^{2.81}) \)

Q. Multiply two 2-by-2 matrices with only 6 scalar multiplications?
A. Impossible. [Hopcroft and Kerr, 1971] \( \Theta(n^\log_2 6) \in O(n^{2.59}) \)

Q. Two 3-by-3 matrices with only 21 scalar multiplications?
A. Also impossible. \( \Theta(n^\log_3 21) \in O(n^{2.77}) \)

Q. Two 70-by-70 matrices with only 143,640 scalar multiplications?
A. Yes! [Pan, 1980] \( \Theta(n^\log_{70} 143640) \in O(n^{2.80}) \)

Decimal wars.
- December, 1979: \( O(n^{2.521813}) \).
- January, 1980: \( O(n^{2.521801}) \).
Fast Matrix Multiplication in Theory

**Best known.** $O(n^{2.376})$ [Coppersmith-Winograd, 1987-2010.]

In 2010, Andrew Stothers gave an improvement to the algorithm $O(n^{2.374})$. In 2011, Virginia Williams combined a mathematical short-cut from Stothers' paper with her own insights and automated optimization on computers, improving the bound $O(n^{2.3728642})$. In 2014, François Le Gall simplified the methods of Williams and obtained an improved bound of $O(n^{2.3728639})$.

**Conjecture.** $O(n^{2+\varepsilon})$ for any $\varepsilon > 0$.

**Caveat.** Theoretical improvements to Strassen are progressively less practical (hidden constant gets worse).
5.4 Closest Pair of Points
Closest Pair of Points

Closest pair. Given n points in the plane, find a pair with smallest Euclidean distance between them.

Fundamental geometric primitive.

- Graphics, computer vision, geographic information systems, molecular modeling, air traffic control.
- Special case of nearest neighbor, Euclidean MST, Voronoi.

Brute force. Check all pairs of points p and q with $\Theta(n^2)$ comparisons.

1-D version. $O(n \log n)$ easy if points are on a line.

Assumption. No two points have same x coordinate.

to make presentation cleaner
Closest Pair of Points

Algorithm.
- **Divide**: draw vertical line $L$ so that roughly $\frac{1}{2}n$ points on each side.
Closest Pair of Points

Algorithm.

- **Divide**: draw vertical line $L$ so that roughly $\frac{1}{2}n$ points on each side.
- **Conquer**: find closest pair in each side recursively.
Closest Pair of Points

Algorithm.

- **Divide:** draw vertical line $L$ so that roughly $\frac{1}{2} n$ points on each side.
- **Conquer:** find closest pair in each side recursively.
- **Combine:** find closest pair with one point in each side. — seems like $\Theta(n^2)$
- Return best of 3 solutions.
Closest Pair of Points

Find closest pair with one point in each side, assuming that distance < $\delta$.

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

Find closest pair with one point in each side, assuming that distance < δ.

- Observation: only need to consider points within δ of line L.

δ = min(12, 21)
Find closest pair with one point in each side, assuming that distance < $\delta$.

- Observation: only need to consider points within $\delta$ of line $L$.
- Sort points in $2\delta$-strip by their $y$ coordinate.

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

Find closest pair with one point in each side, assuming that distance < \(\delta\).

- Observation: only need to consider points within \(\delta\) of line \(L\).
- Sort points in \(2\delta\)-strip by their \(y\) coordinate.
- Only check distances of those within 11 positions in sorted list!

\[\delta = \min(12, 21)\]
Def. Let $s_i$ be the point in the $2\delta$-strip, with the $i^{th}$ smallest $y$-coordinate.

Claim. If $|i - j| \geq 12$, then the distance between $s_i$ and $s_j$ is at least $\delta$.

Pf.
- No two points lie in same $\frac{1}{2}\delta$-by-$\frac{1}{2}\delta$ box.
- Two points at least 2 rows apart have distance $\geq 2(\frac{1}{2}\delta) = \delta$.

Fact. Still true if we replace 12 with 7.

Scan points in $y$-order and compare distance between each point and next 11 neighbours. If any of these distances is less than $\delta$, update $\delta$. 

Closest Pair of Points
Closest Pair of Points

Smallest-Dist(p₁, ..., pₙ) {

    if n=2 then return dist(p₁,p₂)

    Compute separation line L such that half the points are on one side and half on the other side.

    δ’ = Smallest-Dist(left half)
    δ’’ = Smallest-Dist(right half)
    δ = min(δ’,δ’’)

    Delete all points further than δ from separation line L

    Sort remaining points by y-coordinate.

    Scan points in y-order and compare distance between each point and next 11 neighbours. If any of these distances is less than δ, update δ.

    return δ.
}
Closest-Pair\((p_1, ..., p_n)\) {

  if \(n=2\) then return \(dist(p_1, p_2), p_1, p_2\)

  Compute separation line \(L\) such that half the points
  are on one side and half on the other side.

  \(\delta', p', q' = \text{Closest-Pair(left half)}\)
  \(\delta'', p'', q'' = \text{Closest-Pair(right half)}\)
  \(\delta, p, q = \min(\delta', \delta'') (p', q', p'', q'')\)

  Delete all points further than \(\delta\) from separation line \(L\)

  Sort remaining points by y-coordinate.

  Scan points in y-order and compare distance between
  each point and next 11 neighbours. If any of these
  distances is less than \(\delta\), update \(\delta, p, q\).

  return \(\delta, p, q\).
}
Closest Pair of Points: Analysis

Running time.

\[ T(n) \leq 2T\left(\frac{n}{2}\right) + O(n \log n) \implies T(n) \in O(n \log^2 n) \]

Q. Can we achieve \( O(n \log n) \)?

A. Yes. First sort all points according to \( x \) coordinate before algo. Don’t sort points in strip from scratch each time.
   - Each recursion returns a list: all points sorted by \( y \) coordinate.
   - Sort by merging two pre-sorted lists.

\[ T(n) \leq 2T\left(\frac{n}{2}\right) + O(n) \implies T(n) \in O(n \log n) \]
Beyond the Master Theorem
Median Finding
Median Finding

Median Finding. Given \( n \) distinct numbers \( a_1, \ldots, a_n \), find \( i \) such that

\[
|\{ j : a_j < a_i \}| = \lfloor n-1 / 2 \rfloor \quad \text{and} \quad |\{ j : a_j > a_i \}| = \lceil n-1 / 2 \rceil.
\]

<table>
<thead>
<tr>
<th>22</th>
<th>31</th>
<th>44</th>
<th>7</th>
<th>12</th>
<th>19</th>
<th>20</th>
<th>35</th>
<th>3</th>
<th>40</th>
<th>27</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>7</td>
<td>12</td>
<td>19</td>
<td>20</td>
<td>22</td>
<td>27</td>
<td>31</td>
<td>35</td>
<td>40</td>
<td>44</td>
</tr>
<tr>
<td>9</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>1</td>
<td>11</td>
<td>2</td>
<td>8</td>
<td>10</td>
<td>3</td>
</tr>
</tbody>
</table>
Selection

Given $n$ distinct numbers $a_1, \ldots, a_n$, and index $k$, find $i$ such that

$$\left|\{ j : a_j < a_i \}\right| = k-1 \text{ and } \left|\{ j : a_j > a_i \}\right| = n-k.$$ 

$k=4$

<table>
<thead>
<tr>
<th>22</th>
<th>31</th>
<th>44</th>
<th>7</th>
<th>12</th>
<th>19</th>
<th>20</th>
<th>35</th>
<th>3</th>
<th>40</th>
<th>27</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>3</th>
<th>7</th>
<th>12</th>
<th>19</th>
<th>20</th>
<th>22</th>
<th>27</th>
<th>31</th>
<th>35</th>
<th>40</th>
<th>44</th>
</tr>
</thead>
<tbody>
<tr>
<td>9</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>1</td>
<td>11</td>
<td>2</td>
<td>8</td>
<td>10</td>
<td>3</td>
</tr>
</tbody>
</table>
Algorithm partition(A, start, stop)

Input: An array A, indices start and stop.


pivot ← A[stop]
left ← start
right ← stop - 1

while left ≤ right do
    while (left ≤ right and A[left] < pivot) do left ← left + 1
    while (left ≤ right and A[right] ≥ pivot) do right ← right - 1
    if (left < right) then exchange A[left] ↔ A[right]


return left
Example of execution of partition

\[ A = [6 \ 3 \ 7 \ 3 \ 2 \ 5 \ 7 \ 5] \quad \text{pivot} = 5 \]

\[ A = [6 \ 3 \ 7 \ 3 \ 2 \ 5 \ 7 \ 5] \quad \text{swap 6, 2} \]

\[ A = [2 \ 3 \ 7 \ 3 \ 6 \ 5 \ 7 \ 5] \]

\[ A = [2 \ 3 \ 7 \ 3 \ 6 \ 5 \ 7 \ 5] \quad \text{swap 7,3} \]

\[ A = [2 \ 3 \ 3 \ 7 \ 6 \ 5 \ 7 \ 5] \]

\[ A = [2 \ 3 \ 3 \ 7 \ 6 \ 5 \ 7 \ 5] \quad \text{swap 7, pivot} \]

\[ A = [2 \ 3 \ 3 \ 5 \ 6 \ 5 \ 7 \ 7] \]
Selection from Median

Selection. Given \( n \) distinct numbers \( a_1, \ldots, a_n \), and index \( k \), find \( i \) such that:

\[
\left| \{ j : a_j < a_i \} \right| = k-1 \quad \text{and} \quad \left| \{ j : a_j > a_i \} \right| = n-k.
\]

\[
\text{Selection}(A, \text{start}, \text{stop}, k) \quad (\text{where start} \leq k \leq \text{stop})
\]

\[\begin{align*}
\text{Partition} & \quad \small{a_j < a_i} & \quad m & \quad i & \quad a_j > a_i \\
\text{if } k = m \text{ return } m
\end{align*}\]

\[\begin{align*}
\text{Selection}(A, \text{start}, m-1, k) & \quad \text{if } k < m \\
\text{Selection}(A, m+1, \text{stop}, k) & \quad \text{if } k > m
\end{align*}\]

\[\star\text{Median}(A, \text{start}, \text{stop}) = \text{Selection}(A, \text{start}, \text{stop}, \left\lceil \frac{\text{stop}-\text{start}}{2} \right\rceil)\]
Selection from Median

Selection. Given $n$ distinct numbers $a_1, \ldots, a_n$, and index $k$, find $i$ such that $|\{ j : a_j < a_i \}| = k-1$ and $|\{ j : a_j > a_i \}| = n-k$.

\[ \text{Selection}(A, \text{start}, \text{stop}, k) \quad \text{(where } \text{start} \leq k \leq \text{stop}) \]

*Median*( $A, \text{start}, \text{stop}$ ) $= \text{Selection}(A, \text{start}, \text{stop}, \lceil \text{stop}-\text{start} / 2 \rceil)$
Selection from Median

Selection. Given \( n \) distinct numbers \( a_1, \ldots, a_n \), and index \( k \), find \( i \) such that
\[
|\{ j : a_j < a_i \}| = k-1 \quad \text{and} \quad |\{ j : a_j > a_i \}| = n-k.
\]

\[
\text{Selection}(A, \text{start}, \text{stop}, k) \quad (\text{where} \, \text{start} \leq k \leq \text{stop})
\]

if \( k = m \) return \( m \)

\[
\text{Selection}(A, \text{start}, m-1, k) \quad \text{Selection}(A, m+1, \text{stop}, k)
\]

\[
* \text{Median}(A, \text{start}, \text{stop}) = \text{Selection}(A, \text{start}, \text{stop}, \lceil \text{stop-start} / 2 \rceil)
\]
Selection from Median

Select\( (a_1, \ldots, a_n, k) \) \{

    copy \( a_1, \ldots, a_n \) into \( A[1] \ldots A[n] \)
    \textbf{return} \ SelectREC(A,1,n,k)
\}

SelectREC(A,start,stop,k) \{

    if start=stop then \textbf{return} \ stop

    i = \text{Median}(A,start,stop)
    \textbf{\{\textit{* = Selection}(A,start,stop,\lceil stop-start / 2 \rceil)\}}

    exchange \( A[i] \) with \( A[stop] \)
    m = \text{partition}(A,start,stop)

    if k=m then \textbf{return} \ m
    if k<m then \textbf{return} \ SelectREC(A,start,m-1,k)
    \textbf{else return} \ SelectREC(A,m+1,stop,k).
\}
Selection

Selection. Given n distinct numbers \( a_1, \ldots, a_n \), and index \( k \), find \( i \) such that \(|\{ j : a_j < a_i \}| = k-1 \) and \(|\{ j : a_j > a_i \}| = n-k\).

Selection(\( A, \text{start}, \text{stop}, k \)) (where \( \text{start} \leq k \leq \text{stop} \))

Partition

if \( k = m \) return \( m \)

if \( k < m \) Selection(\( A, \text{start}, m-1, k \))

if \( k > m \) Selection(\( A, m+1, \text{stop}, k \))
Selection from Pseudo-Median

Select\((a_1, \ldots, a_n, k)\) 

\[
\text{copy } a_1, \ldots, a_n \text{ into } A[1] \ldots A[n]
\]

\[
\text{return } \text{SelectREC}(A,1,n,k)
\]

SelectREC\((A,\text{start},\text{stop},k)\) 

\[
\text{if start}=\text{stop} \text{ then return stop}
\]

\[
i = \text{PseudoMedian}(A,\text{start},\text{stop})
\]

\[
\text{exchange } A[i] \text{ with } A[\text{stop}]
\]

\[
m = \text{partition}(A,\text{start},\text{stop})
\]

\[
\text{if } k=m \text{ then return } m
\]

\[
\text{if } k<m \text{ then return } \text{SelectREC}(A,\text{start},m-1,k)
\]

\[
\text{else return } \text{SelectREC}(A,m+1,\text{stop},k).
\]
PseudoMedian(A,start,stop) {

    for i=0 to ⌈(stop-start / 5)⌉-1
                       A[start+5*i+1],
                       A[start+5*i+2],
                       A[start+5*i+3],
                       A[start+5*i+4]) ]

    m = SelectREC(B,1, ⌈(stop-start / 5)⌉, ⌈(stop-start / 10)⌉)

    find j such that A[j]=B[m]
    return j
}

Selection from Pseudo-Median
Selection. Given \( n \) distinct numbers \( a_1, \ldots, a_n \), and index \( k \), find \( i \) such that
\[
|\{ j : a_j < a_i \}| = k-1 \quad \text{and} \quad |\{ j : a_j > a_i \}| = n-k.
\]
Selection

Given \( n \) distinct numbers \( a_1, \ldots, a_n \), and index \( k \), find \( i \) such that

\[
\left| \{ j : a_j < a_i \} \right| = k-1 \quad \text{and} \quad \left| \{ j : a_j > a_i \} \right| = n-k.
\]

\[
T(n) \leq T\left( \left\lfloor \frac{n}{5} \right\rfloor \right) + \max \{ T\left( \left\lfloor \frac{3n}{10} \right\rfloor \right) \ldots T\left( \left\lfloor \frac{7n}{10} \right\rfloor \right) \} + \Theta(n)
\]

Solution: \( T(n) \in \Theta(n) \)
Selection

Given $n$ distinct numbers $a_1, \ldots, a_n$, and index $k$, find $i$ such that

$$\left| \{ j : a_j < a_i \} \right| = k-1 \text{ and } \left| \{ j : a_j > a_i \} \right| = n-k.$$ 

Solution: $T(n) \in \Theta(n)$

Assuming $T(i) \leq d$ for $1 \leq i \leq n$, $\Theta(n) \leq cn$

$$T(n+1) \leq T(\frac{n+1}{5}) + T(\frac{7(n+1)}{10}) + c(n+1)$$

$$\leq d(n+1)/5 + 7d(n+1)/10 + c(n+1)$$

$$= (2d+7d+10c)/10 (n+1)$$

$$= (9d+10c)/10 (n+1)$$

$$\leq d (n+1) \text{ as long as } (9d+10c)/10 \leq d, \text{ or equivalently } 10c \leq d.$$
Selection

**Selection.** Given n distinct numbers $a_1, ..., a_n$, and index $k$, find $i$ such that

$$|\{ j : a_j < a_i \}| = k-1 \text{ and } |\{ j : a_j > a_i \}| = n-k.$$ 

$$T(n) \leq T\left( \frac{n}{5} \right) + T\left( \frac{7n}{10} \right) + \Theta(n)$$

**Solution:** $T(n) \in \Theta(n)$

**example:** $d=10c,$

Assuming $T(i) \leq 10c$ i for $1 \leq i \leq n$, $\Theta(n) \leq cn$

$T(n+1) \leq T(n+1 /5) + T(7/10 (n+1)) + c(n+1)$

$\leq 10c/5 (n+1) + 7\cdot 10c/10 (n+1) + c(n+1)$

$= (2c+7c+c)(n+1)$

$= 10c (n+1)$
Chapter 5

Divide and Conquer

CLRS 4.3