5. Divide-and-Conquer

Divide et impera.
Veni, vidi, vici.
- Julius Caesar
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: $n \log n$. 
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

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

Jon von Neumann (1945)

A L G O R I T H M S

A L G O R I T H M S

A G L O R H I M S T

A G H I L M O R S T

divide $O(1)$
sort $2T(n/2)$
merge $O(n)$
Merging

**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. [Kronrud, 1969]

using only a constant amount of extra storage
A Useful Recurrence Relation

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

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. $T(n) = O(n \log_2 n)$.

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

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

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

**Claim.** If \( T(n) \) satisfies this recurrence, then \( T(n) = n \log_2 n \).

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

\( \text{assumes } n \text{ is a power of } 2 \)

\[
\text{Pf. For } n > 1: \\
\frac{T(n)}{n} = \frac{2T(n/2)}{n} + 1 \\
= \frac{T(n/2)}{n/2} + 1 \\
= \frac{T(n/4)}{n/4} + 1 + 1 \\
\cdots \\
= \frac{T(n/n)}{n/n} + 1 + \cdots + 1 \\
= \log_2 n
\]
Proof by Induction

**Claim.** If \( T(n) \) satisfies this recurrence, then \( T(n) = n \log_2 n \).

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

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

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

\[
T(2n) = 2T(n) + 2n = 2n \log_2 n + 2n = 2n(\log_2 (2n) - 1) + 2n = 2n \log_2 (2n)
\]

assumes \( n \) is a power of 2
Claim. If $T(n)$ satisfies the following recurrence, then $T(n) \leq n \lfloor \log n \rfloor$.

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

\[
\text{solve left half} \quad \text{solve right half} \quad \text{merging}
\]

Pf. (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 $1, 2, \ldots, n-1$.

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

\[
n_2 = \lfloor n/2 \rfloor
\]
\[
\leq 2^{\lceil \log n \rceil} / 2
\]
\[
= 2^{\lceil \log n \rceil} / 2
\]
\[
\implies \log n_2 \leq \lfloor \log n \rfloor - 1
\]
5.3 Counting Inversions
**Counting Inversions**

**Music site tries to match your song preferences with others.**

- You rank n songs.
- Music site consults database to find people with similar tastes.

**Similarity metric:** number of inversions between two rankings.

- My rank: 1, 2, ..., n.
- Your rank: $a_1, a_2, ..., a_n$.
- 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

**Brute force:** check all $\Theta(n^2)$ pairs $i$ and $j$. 
Applications

Applications.

- Voting theory.
- Collaborative filtering.
- Measuring the "sortedness" of an array.
- Sensitivity analysis of Google's ranking function.
- Rank aggregation for meta-searching on the Web.
- Nonparametric statistics (e.g., Kendall's Tau distance).
Counting Inversions: Divide-and-Conquer

Divide-and-conquer.
Counting Inversions: Divide-and-Conquer

Divide-and-conquer.

- **Divide**: separate list into two pieces.

\[\begin{array}{cccccccccccc}
1 & 5 & 4 & 8 & 10 & 2 & 6 & 9 & 12 & 11 & 3 & 7 \\
\end{array}\]

\[\begin{array}{cccccccccccc}
1 & 5 & 4 & 8 & 10 & 2 & 6 & 9 & 12 & 11 & 3 & 7 \\
\end{array}\]

Divide: \(O(1)\).
Counting Inversions: Divide-and-Conquer

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

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

5 blue-blue inversions
5-4, 5-2, 4-2, 8-2, 10-2

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

8 green-green inversions
6-3, 9-3, 9-7, 12-3, 12-7, 12-11, 11-3, 11-7

Divide: $O(1)$.

Conquer: $2T(n/2)$
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, and return sum of three quantities.

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

Divide: $O(1)$.

```
1  5  4  8 10  2
5 blue-blue inversions
```

```
6  9 12 11  3  7
8 green-green inversions
```

Conquer: $2T(n/2)$

```
9 blue-green inversions
5-3, 4-3, 8-6, 8-3, 8-7, 10-6, 10-9, 10-3, 10-7
```

Combine: ???

Total = $5 + 8 + 9 = 22$. 

```
**Counting Inversions: Combine**

**Combine:** count blue-green inversions
- Assume each half is **sorted**.
- Count inversions where \( a_i \) and \( a_j \) are in different halves.
- **Merge** two sorted halves into sorted whole.

<table>
<thead>
<tr>
<th>Blue</th>
<th>Green</th>
</tr>
</thead>
<tbody>
<tr>
<td>3 7 10 14 18 19</td>
<td>2 11 16 17 23 25</td>
</tr>
</tbody>
</table>

\[
\text{13 blue-green inversions: } 6 + 3 + 2 + 2 + 0 + 0
\]

Count: \( O(n) \)

<table>
<thead>
<tr>
<th>Blue</th>
<th>Green</th>
</tr>
</thead>
<tbody>
<tr>
<td>2 3 7 10 11 14 16 17 18 19 23 25</td>
<td></td>
</tr>
</tbody>
</table>

Merge: \( O(n) \)

\[
T(n) \leq T\left(\left\lfloor n/2 \right\rfloor \right) + T\left(\left\lceil n/2 \right\rceil \right) + O(n) \quad \Rightarrow \quad T(n) = O(n \log n)
\]
Counting Inversions: Implementation

Pre-condition. [Merge-and-Count] A and B are sorted.
Post-condition. [Sort-and-Count] L is sorted.

Sort-and-Count(L) {
    if list L has one element
        return 0 and the list L

    Divide the list into two halves A and B
    (r_A, A) ← Sort-and-Count(A)
    (r_B, B) ← Sort-and-Count(B)
    (r, L) ← Merge-and-Count(A, B)

    return r = r_A + r_B + r and the sorted list L
}