Chapter 5

Divide and Conquer
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. Given n elements, rearrange in ascending order.

Applications.

* Sort a list of names.
* Organize an MP3 library.
* Display Google PageRank results.
* List RSS news items in reverse chronological order.

Find the median.
Find the closest pair.
Binary search in a database.
Identify statistical outliers.
Find duplicates in a mailing list.

Data compression.
Computer graphics.
Computational biology.
Supply chain management.
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

divide \( O(1) \)

A L G O R

I T H M S

sort \( 2T(n/2) \)

A G L O R

H I M S T

merge \( O(n) \)

A G H I L M O R S T
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. \( T(n) = \) number of comparisons to mergesort an input of size \( n \).

Mergesort recurrence.

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

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

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

sorting both halves

merging

\[ T(n) \]

\[ T(n/2) \quad \quad T(n/2) \]

\[ T(n/4) \quad T(n/4) \quad T(n/4) \quad T(n/4) \]

\[ T(2) \quad T(2) \quad T(2) \quad T(2) \quad T(2) \quad T(2) \quad T(2) \quad T(2) \]

\[ n \]

\[ 2(n/2) \]

\[ 4(n/4) \]

\[ \ldots \]

\[ 2^k(n/2^k) \]

\[ \ldots \]

\[ n/2 (2) \]

\[ n \log_2 n \]
Proof by Telescoping

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

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

assumes \( n \) is a power of 2

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

\[ = \begin{cases} 
0 & \text{if } n = 1 \\
2T(n/2) + \frac{n}{4} & \text{merging}
\end{cases} \]

\[ = \begin{cases} 
0 & \text{if } n = 1 \\
2T(n/2) + \frac{n}{8} & \text{merging}
\end{cases} \]

\[ = \begin{cases} 
0 & \text{if } n = 1 \\
2T(n/2) + \frac{n}{16} & \text{merging}
\end{cases} \]

\[ = \begin{cases} 
0 & \text{if } n = 1 \\
\vdots
\end{cases} \]

\[ = \begin{cases} 
0 & \text{if } n = 1 \\
T(n/n) + \frac{1}{\log_2 n} & \text{merging}
\end{cases} \]

\[ = \log_2 n \]

**Pf.** For \( n > 1 \):

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

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

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

\[ = \frac{T(n/n)}{n/n} + \frac{1 + \cdots + 1}{\log_2 n} \]

\[ = \log_2 n \]
Proof by Induction

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

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

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

\[ \uparrow \text{ assumes } n \text{ is a power of } 2 \]

\[ \begin{aligned}
\text{Pf. (by induction on } n) \\
\text{Base case: } n = 1. \\
\text{Inductive hypothesis: } T(n) = n \log_2 n. \\
\text{Goal: show that } T(2n) = 2n \log_2 (2n).
\end{aligned} \]
Claim. If $T(n)$ satisfies the following recurrence, then $T(n) \leq n \lceil \lg n \rceil$.

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

Pf. (by induction on $n$)

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

$$T(n) \leq T(n_1) + T(n_2) + n$$
$$\leq n_1 \left\lceil \lg n_1 \right\rceil + n_2 \left\lceil \lg n_2 \right\rceil + n$$
$$\leq n_1 \left\lceil \lg n_2 \right\rceil + n_2 \left\lceil \lg n_2 \right\rceil + n$$
$$= n \left\lceil \lg n_2 \right\rceil + n$$
$$\leq n(\left\lfloor \lg n \right\rfloor - 1) + n$$
$$= n \lceil \lg n \rceil$$

$$n_2 = \left\lceil n/2 \right\rceil$$
$$\leq \left\lceil 2 \left\lceil \lg n \right\rceil / 2 \right\rceil$$
$$= \left\lceil \lg n \right\rceil / 2$$
$$\Rightarrow \lg n_2 \leq \left\lfloor \lg n \right\rfloor - 1$$
Two Exercises

- Using recursion tree to guess a result, and then, applying induction to prove.

(1) \( T(n) = 3 \ T\left(\frac{n}{4}\right) + \Theta(n^2) \)

  Use \( cn^2 \) to replace \( \Theta(n^2) \) for \( c > 0 \) in recursion tree
  Apply \( T(n) \leq dn^2 \) for \( d > 0 \), the guess result, in induction prove
  Determine the constraint associated with \( d \) and \( c \)

(2) \( T(n) = T(n/3) + T(2n/3) + O(n) \)

  Use \( c \) to represent the constant factor in \( O(n) \) in recursion tree
  Apply \( T(n) \leq dn \lg n \) for \( d > 0 \), the guess result, in induction prove
  Determine the constraint associated with \( d \) and \( c \)
Theorem 4.1 (Master theorem)
Let \( a \geq 1 \) and \( b > 1 \) be constants, let \( f(n) \) be a function, and let \( T(n) \) be defined on the nonnegative integers by the recurrence

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

where we interpret \( n/b \) to mean either \( \lfloor n/b \rfloor \) or \( \lceil n/b \rceil \). Then \( T(n) \) has the following asymptotic bounds:

1. If \( f(n) = O(n^{\log_b a - \epsilon}) \) for some constant \( \epsilon > 0 \), then \( T(n) = \Theta(n^{\log_b a}) \).
2. If \( f(n) = \Theta(n^{\log_b a}) \), then \( T(n) = \Theta(n^{\log_b a} \log n) \).
3. If \( f(n) = \Omega(n^{\log_b a + \epsilon}) \) for some constant \( \epsilon > 0 \), and if \( af(n/b) \leq cf(n) \) for some constant \( c < 1 \) and all sufficiently large \( n \), then \( T(n) = \Theta(f(n)) \). 

\[
\begin{align*}
T(n) = 9T(n/3) + n, & \quad T(n) = \Theta(n^2); \\
T(n) = T(2n/3) + 1, & \quad T(n) = \Theta(\log n); \\
T(n) = 8T(n/2) + \Theta(n^2), & \quad T(n) = \Theta(n^3); \\
T(n) = 3T(n/4) + n \log n, & \quad T(n) = \Theta(n \log n) \\
T(n) = 2T(n/2) + \Theta(n), & \quad T(n) = \Theta(n \log n) \\
T(n) = 7T(n/2) + \Theta(n2), & \quad T(n) = \Theta(n^{\log 7})
\end{align*}
\]
Parallel Merge Sort

Merge two sorted subsequences $p$ and $q$ in parallel with $k$ processors

**Speedup:** seq. time / para. time, **Efficiency:** # of processors ($k$)/speed up

**Cost:** # of processors x parallel time, **Cost-optimal:** efficiency = 1

Other parallel sorts: bitonic, quick, radix, and sample sort


5.3 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: a₁, a₂, ..., aₙ.
Songs i and j inverted if i < j, but aᵢ > aⱼ.

<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

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.
Divide-and-conquer.

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

<table>
<thead>
<tr>
<th>1</th>
<th>5</th>
<th>4</th>
<th>8</th>
<th>10</th>
<th>2</th>
<th>6</th>
<th>9</th>
<th>12</th>
<th>11</th>
<th>3</th>
<th>7</th>
</tr>
</thead>
</table>

Divide: $O(1)$. 

| 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.

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

Divide: \(O(1)\).

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

Conquer: \(2T(n/2)\)

5 blue-blue inversions

5-4, 5-2, 4-2, 8-2, 10-2

8 green-green inversions

6-3, 9-3, 9-7, 12-3, 12-7, 12-11, 11-3, 11-7
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, and return sum of three quantities.

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

Divide: $O(1)$.

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

5 blue-blue inversions
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.

to maintain sorted invariant

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

Count: \( O(n) \)

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

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

```plaintext
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
}
```
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: First Attempt

**Divide.** Sub-divide region into 4 quadrants.
Closest Pair of Points: First Attempt

**Divide.** Sub-divide region into 4 quadrants.

**Obstacle.** Impossible to ensure $n/4$ points in each piece.
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.

![Diagram showing the closest pair of points](image-url)
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. \( \rightarrow \) 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 < $\delta$.
Observation: only need to consider points within $\delta$ of line L.
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.
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 $\geq 2(\frac{1}{2}\delta)$. 

**Fact.** Still true if we replace 12 with 7.
Closest Pair Algorithm

Closest-Pair(p₁, …, pₙ) {

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

    \[ \delta_1 = \text{Closest-Pair(left half)} \]
    \[ \delta_2 = \text{Closest-Pair(right half)} \]
    \[ \delta = \min(\delta_1, \delta_2) \]

    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 neighbors. If any of these distances is less than \( \delta \), update \( \delta \).

    return \( \delta \).

}
Closest Pair of Points: Analysis

Running time.

\[ T(n) \leq 2T(n/2) + O(n \log n) \Rightarrow T(n) = O(n \log^2 n) \]

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

**A.** Yes. Don't sort points in strip from scratch each time. Each recursive returns two lists: all points sorted by \( y \) coordinate, and all points sorted by \( x \) coordinate. Sort by merging two pre-sorted lists.

\[ T(n) \leq 2T(n/2) + O(n) \Rightarrow T(n) = O(n \log n) \]