Chapter 5
Divide and Conquer

5.1 Mergesort

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 $\frac{n}{2}$.
- 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

Sorting

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.

obvious applications
problems become easy once items are in sorted order
non-obvious applications
Mergesort

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

Jon von Neumann (1945)

Mergesort. 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.

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

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
**Proof by Telescoping**

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

**Pf.** For $n > 1$:

\[
T(n) = \begin{cases} 
0 & \text{if } n = 1 \\
\frac{2T(n/2)}{n} + \frac{n}{\log_2 n} & \text{otherwise}
\end{cases}
\]

This assumes $n$ is a power of 2.

- $T(1) = 0$ (base case)
- For $n > 1$:
  \[
  T(n) = \frac{2T(n/2)}{n} + \frac{n}{\log_2 n}
  = \frac{T(n/2)}{n/2} + 1
  = \frac{T(n/4)}{n/4} + 1 + 1
  = \cdots
  = \frac{T(n/n)}{n/n} + 1 + \cdots + 1
  = \log_2 n
  \]

**Analysis of Mergesort Recurrence**

**Claim.** If $T(n)$ satisfies the following recurrence, then $T(n) \leq n \lfloor \log_2 n \rfloor$.

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

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

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

**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>

**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.*

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

<table>
<thead>
<tr>
<th>Divide: separate list into two pieces.</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Divide: ( O(1) ).</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
</tr>
</tbody>
</table>

**Counting Inversions: Divide-and-Conquer**

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

<table>
<thead>
<tr>
<th>Divide: ( O(1) ).</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
</tr>
</tbody>
</table>
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.

```
Divide: O(1).
Conquer: 2T(n/2)
Combine: ???
```

```
5-3, 4-3, 8-6, 8-3, 8-7, 10-6, 10-9, 10-3, 10-7
```

Total = 5 + 8 + 9 = 22.

```
T(n) ≤ T(⌈n/2⌉) + T(⌊n/2⌋) + 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.

```
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.

Closest Pair of Points: First Attempt

Divide. Sub-divide region into 4 quadrants.

Obstacle. Impossible to ensure \( n/4 \) points in each piece.

to make presentation cleaner

Fast closest pair inspired fast algorithms for these problems
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, assuming that distance $\leq \delta$.
- Return best of 3 solutions.
Find closest pair with one point in each side, assuming that distance < $\delta$.
- Observation: only need to consider points within $\delta$ of line $L$.

**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(p_1, ..., p_n) {
    Compute separation line L such that half the points are on one side and half on the other side.
    δ_1 = Closest-Pair(left half)
    δ_2 = Closest-Pair(right half)
    δ = min(δ_1, δ_2)
    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 neighbors. If any of these distances is less than δ, update δ.
    return δ.
}

Running time.

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.

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