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.

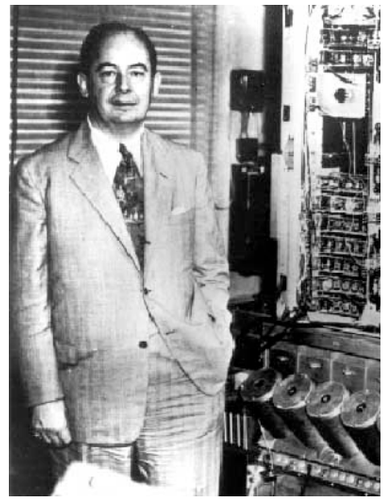
- Find the median.
- Find the closest pair.
- Binary search in a database.
- Identify statistical outliers.
- Find duplicates in a mailing list.

Non-obvious sorting applications.

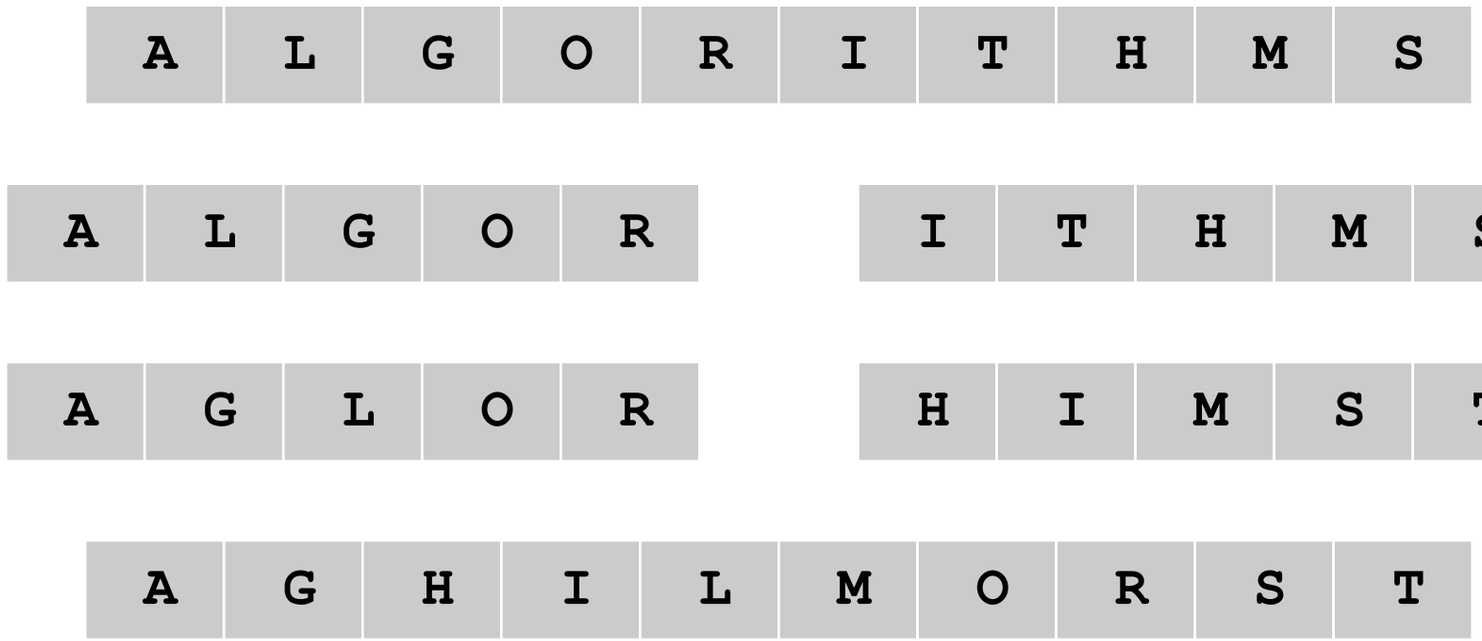
- Data compression.
- Computer graphics.
- Interval scheduling.
- Computational biology.
- Minimum spanning tree.
- Supply chain management.
- Simulate a system of particles.
- Book recommendations on Amazon.
- Load balancing on a parallel computer.
- . . .

Mergesort

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



Jon von Neumann (1945)



Merging

Merging. Combine two pre-sorted lists into a sorted whole.

How to merge efficiently?

- Linear number of comparisons.
- Use temporary array.

Challenge f

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.

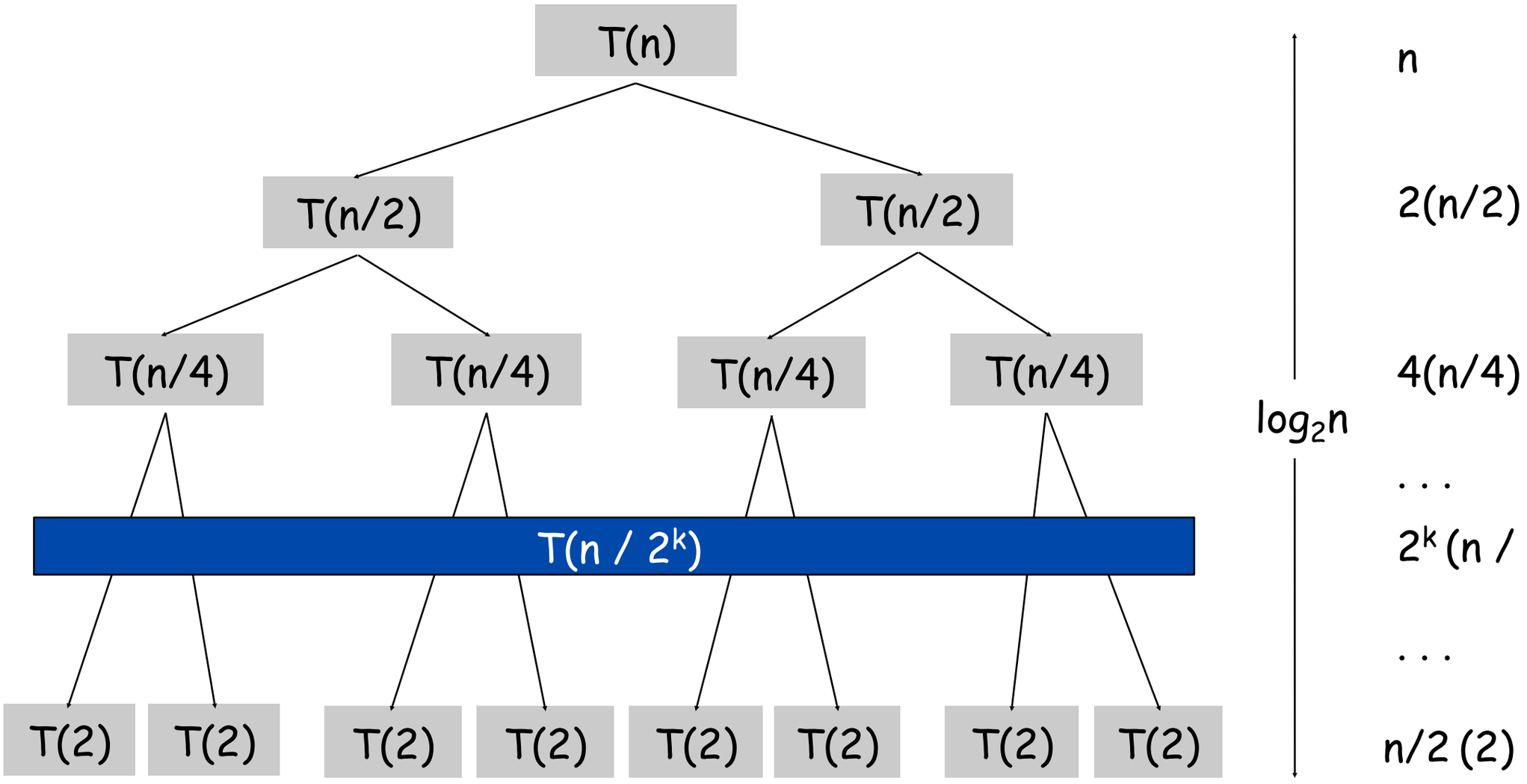
$$\mathrm{T}(n) \le \begin{cases} 0 & \text{if } n = 1 \\ \underbrace{T(\lceil n/2 \rceil)}_{\text{solve left half}} + \underbrace{T(\lfloor n/2 \rfloor)}_{\text{solve right half}} + \underbrace{n}_{\text{merging}} & \text{otherwise} \end{cases}$$

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 ≤ with =.

Proof by Recursion Tree

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



Proof by Telescoping

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

assumes n is a power of 2

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

Pf. For n > 1:

$$\begin{aligned} \frac{T(n)}{n} &= \frac{2T(n/2)}{n} + 1 \\ &= \frac{T(n/2)}{n/2} + 1 \\ &= \frac{T(n/4)}{n/4} + 1 + 1 \\ &\cdots \\ &= \frac{T(n/n)}{n/n} + \underbrace{1 + \cdots + 1}_{\log_2 n} \\ &= \log_2 n \end{aligned}$$

Proof by Induction

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

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

assumes n is a power of 2

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).

$$\begin{aligned} T(2n) &= 2T(n) + 2n \\ &= 2n\log_2 n + 2n \\ &= 2n\left(\log_2(2n) - 1\right) + 2n \\ &= 2n\log_2(2n) \end{aligned}$$

Analysis of Mergesort Recurrence

Claim. If T(n) satisfies the following recurrence, then $T(n) \le n \lceil \lg n \rceil$.

$\lceil \lg n \rceil$ = $\log_2 n$

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

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, ... , n–1.

$$\begin{aligned} T(n) &\le T(n_1) + T(n_2) + n \\ &\le n_1 \lceil \lg n_1 \rceil + n_2 \lceil \lg n_2 \rceil + n \\ &\le n_1 \lceil \lg n_2 \rceil + n_2 \lceil \lg n_2 \rceil + n \\ &= n \lceil \lg n_2 \rceil + n \\ &\le n(\lceil \lg n \rceil - 1) + n \\ &= n \lceil \lg n \rceil \end{aligned}$$

$$\begin{aligned} n_2 &= \lceil n/2 \rceil \\ &\le \lceil 2^{\lceil \lg n \rceil} / 2 \rceil \\ &= 2^{\lceil \lg n \rceil} / 2 \\ \Rightarrow \lg n_2 &\le \lceil \lg n \rceil - 1 \end{aligned}$$

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.

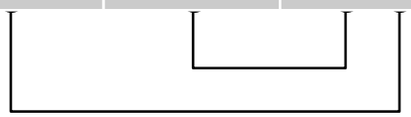
Similarity metric: number of inversions between two rankings.

- My rank: 1, 2, …, n.
- Your rank: $a_1, a_2, \ldots, a_n$.
- Songs i and j inverted if $i < j$, but $a_i > a_j$.

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

Songs

	A	B	C	D	E
Me	1	2	3	4	5
You	1	3	4	2	5



Inversions

3-2, 4-2

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.

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.

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

1	5	4	8	10	2

6	9	12	11	3	7

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

Divide: $O(1)$.

1	5	4	8	10	2

5 blue-blue inversions

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

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

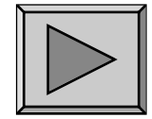
Combine: ???

Total = 5 + 8 + 9 = 22.

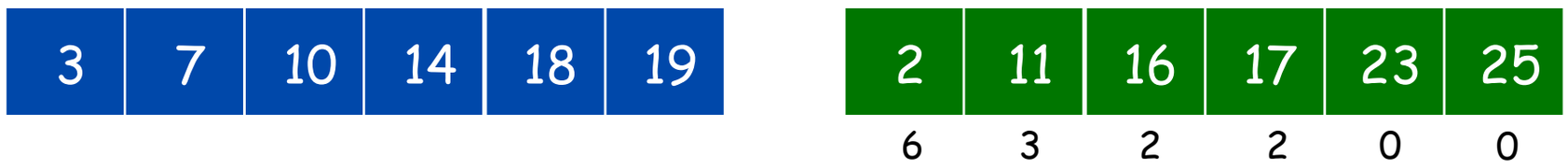
Counting Inversions: Combine

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.



to maintain sorted invariant



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

Count: O(n)

Merge: O(n)

$$T(n) \leq T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + O(n) \Rightarrow T(n) = O(n \log n)$$

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
}
```