5. **DIVIDE AND CONQUER I**

- mergesort
- counting inversions
- closest pair of points
- randomized quicksort
- median and selection

---

### Divide-and-conquer paradigm

**Divide-and-conquer.**
- Divide up problem into several subproblems.
- Solve each subproblem recursively.
- Combine solutions to subproblems into overall solution.

**Most common usage.**
- Divide problem of size $n$ into two subproblems (of the same kind) of size $n/2$ in linear time.
- Solve two subproblems recursively.
- Combine two solutions into overall solution in linear time.

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

---

### Sorting problem

**Problem.** Given a list of $n$ elements from a totally-ordered universe, rearrange them in ascending order.
Sorting applications

Obvious applications.
- Organize an MP3 library.
- Display Google PageRank results.
- List RSS news items in reverse chronological order.

Some problems become easier once elements are sorted.
- Identify statistical outliers.
- Binary search in a database.
- Remove duplicates in a mailing list.

Non-obvious applications.
- Convex hull.
- Closest pair of points.
- Interval scheduling / interval partitioning.
- Minimum spanning trees (Kruskal’s algorithm).
- Scheduling to minimize maximum lateness or average completion time.
- ...

Mergesort

- Recursively sort left half.
- Recursively sort right half.
- Merge two halves to make sorted whole.

input

| A | L | G | O | R | I | T | H | M | S |

sort left half

| A | G | L | O | R | I | T | H | M | S |

sort right half

| A | G | L | O | R | H | I | M | S | T |

merge results

| A | G | H | I | L | M | O | R | S | T |

Merging

Goal. Combine two sorted lists A and B into a sorted whole C.
- Scan A and B from left to right.
- Compare \( a_i \) and \( b_j \).
- If \( a_i \leq b_j \), append \( a_i \) to C (no larger than any remaining element in B).
- If \( a_i > b_j \), append \( b_j \) to C (smaller than every remaining element in A).

<table>
<thead>
<tr>
<th>sorted list A</th>
<th>sorted list B</th>
</tr>
</thead>
<tbody>
<tr>
<td>3 7 10 ( a_i ) 18</td>
<td>2 11 ( b_j ) 20 23</td>
</tr>
</tbody>
</table>

merge to form sorted list C

| 2 3 7 10 11 |

A useful recurrence relation

Def. \( T(n) = \) max number of compares to mergesort a list of size \( \leq n \).

Note. \( T(n) \) is monotone nondecreasing.

Mergesort recurrence.

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

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

Assorted proofs. We describe several ways to solve this recurrence. Initially we assume \( n \) is a power of 2 and replace \( \leq \) with = in the recurrence.
Divide-and-conquer recurrence: proof by recursion tree

Proposition. If \( T(n) \) satisfies the following recurrence, then \( T(n) = n \log_2 n \).

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

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

Pf 1.

\[
\begin{array}{c}
T(n) \\
\downarrow \\
\frac{n}{2} \\
\downarrow \\
T(n/4) \\
\downarrow \\
\frac{n}{8} \\
\vdots \\
T(n/8^n) = n \\
\end{array}
\]

Proof by induction

Proposition. If \( T(n) \) satisfies the following recurrence, then \( T(n) = n \log_2 n \).

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

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

Pf 2. [by induction on \( n \)]

- Base case: when \( n = 1 \), \( T(1) = 0 = n \log_2 n \).
- Inductive hypothesis: assume \( T(n) = n \log_2 n \).
- Goal: show that \( T(2n) = 2n \log_2 (2n) \).

\[
T(2n) = 2 T(n) + 2n \\
= 2 n \log_2 n + 2n \\
= 2 n (\log_2 (2n) – 1) + 2n \\
= 2 n \log_2 (2n). \quad \blacksquare
\]

Analysis of mergesort recurrence

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

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

Pf. [by strong induction on \( n \)]

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

\[
T(n) \leq T(n_1) + T(n_2) + n \leq 2^\lceil \log_2 n \rceil / 2 \\
\leq n_1 \lceil \log_2 n_1 \rceil + n_2 \lceil \log_2 n_2 \rceil + n \\
\leq n_1 \lceil \log_2 n_1 \rceil + n_2 \lceil \log_2 n_2 \rceil + n \\
= n_2 \lceil \log_2 n_2 \rceil + n \\
\leq n \lfloor \log_2 n \rfloor + n \\
= n \lfloor \log_2 n \rfloor. \quad \blacksquare
\]

5. Divide and Conquer

- mergesort
- counting inversions
- closest pair of points
- randomized quicksort
- median and selection
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: \( 1, 2, \ldots, n \).
- Your rank: \( a_1, a_2, \ldots, a_n \).
- Songs \( i \) and \( j \) are inverted if \( i < j \), but \( a_i > a_j \).

\[
\begin{array}{cccccc}
  & A & B & C & D & E \\
 me & 1 & 2 & 3 & 4 & 5 \\
 you & 1 & 3 & 4 & 2 & 5 \\
\end{array}
\]

2 inversions: 3-2, 4-2

Brute force: check all \( \Theta(n^2) \) pairs.

Counting inversions: divide-and-conquer

- Divide: separate list into two halves \( A \) and \( B \).
- Conquer: recursively count inversions in each list.
- Combine: count inversions \((a, b)\) with \( a \in A \) and \( b \in B \).
- Return sum of three counts.

<table>
<thead>
<tr>
<th>input</th>
<th>1 5 4 8 10 2 6 9 3 7</th>
</tr>
</thead>
<tbody>
<tr>
<td>count inversions in left half A</td>
<td>1 5 4 8 10</td>
</tr>
<tr>
<td>count inversions in right half B</td>
<td>2 6 9 3 7</td>
</tr>
<tr>
<td>count inversions ((a, b)) with ( a \in A ) and ( b \in B )</td>
<td>4-2 4-3 5-2 5-3 8-2 8-3 8-6 8-7 10-2 10-3 10-6 10-7 10-9</td>
</tr>
<tr>
<td>output</td>
<td>1 + 3 + 13 = 17</td>
</tr>
</tbody>
</table>

Counting inversions: how to combine two subproblems?

Q. How to count inversions \((a, b)\) with \( a \in A \) and \( b \in B \)?
A. Easy if \( A \) and \( B \) are sorted!

Warmup algorithm.
- Sort \( A \) and \( B \).
- For each element \( b \in B \),
  - binary search in \( A \) to find how elements in \( A \) are greater than \( b \).

<table>
<thead>
<tr>
<th>list A</th>
<th>7 10 18 3 14</th>
</tr>
</thead>
<tbody>
<tr>
<td>list B</td>
<td>20 23 2 11 16</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>sort A</th>
<th>3 7 10 14 18</th>
</tr>
</thead>
<tbody>
<tr>
<td>sort B</td>
<td>2 11 16 20 23</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>binary search to count inversions ((a, b)) with ( a \in A ) and ( b \in B )</th>
<th>3 7 10 14 18</th>
</tr>
</thead>
<tbody>
<tr>
<td>2 11 16 20 23</td>
<td></td>
</tr>
<tr>
<td>5 2 1 0 0</td>
<td></td>
</tr>
</tbody>
</table>
Counting inversions: how to combine two subproblems?

Count inversions \((a, b)\) with \(a \in A\) and \(b \in B\), assuming \(A\) and \(B\) are sorted.

- Scan \(A\) and \(B\) from left to right.
- Compare \(a_i\) and \(b_j\).
- If \(a_i < b_j\), then \(a_i\) is not inverted with any element left in \(B\).
- If \(a_i > b_j\), then \(b_j\) is inverted with every element left in \(A\).
- Append smaller element to sorted list \(C\).

count inversions \((a, b)\) with \(a \in A\) and \(b \in B\)

\[
\begin{array}{c|c|c|c|c}
3 & 7 & 10 & a_i & 18 \\
\hline
2 & 11 & b_j & 20 & 23
\end{array}
\]

merge to form sorted list \(C\)

2 3 7 10 11

---

Counting inversions: divide-and-conquer algorithm analysis

**Proposition.** The sort-and-count algorithm counts the number of inversions in a permutation of size \(n\) in \(O(n \log n)\) time.

**Pf.** The worst-case running time \(T(n)\) satisfies the recurrence:

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

---

5. Divide and Conquer

- mergesort
- counting inversions
- closest pair of points
- randomized quicksort
- median and selection
**Closest pair of points**

**Closest pair problem.** Given \( n \) points in the plane, find a pair of points with the 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.

\[ \text{fast closest pair inspired fast algorithms for these problems} \]

**Closest pair of points: first attempt**

**Sorting solution.**
- Sort by \( x \)-coordinate and consider nearby points.
- Sort by \( y \)-coordinate and consider nearby points.

\[ \]
Closest pair of points: second attempt

Divide. Subdivide region into 4 quadrants.

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

Closest pair of points: divide-and-conquer algorithm

- Divide: draw vertical line $L$ so that $n/2$ points on each side.
- Conquer: find closest pair in each side recursively.
- Combine: find closest pair with one point in each side.
- Return best of 3 solutions.

How to find closest pair with one point in each side?

Find closest pair with one point in each side, assuming that distance $< \delta$.
- Observation: only need to consider points within $\delta$ of line $L$. 

$\delta = \min(12, 21)$
How to find closest pair with one point in each side?

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

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

Why 11?

δ = min(12, 21)

Closest pair of points: divide-and-conquer algorithm

\text{CLOSEST-PAIR} (p_1, p_2, ..., p_n)

Compute separation line L such that half the points are on each side of the line.

δ_1 \leftarrow \text{CLOSEST-PAIR} (points in left half).
δ_2 \leftarrow \text{CLOSEST-PAIR} (points in right half).
δ \leftarrow \min \{ δ_1, δ_2 \}.
Delete all points further than δ from 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.
\text{RETURN} δ.

Closest pair of points: analysis

\textbf{Theorem.} The divide-and-conquer algorithm for finding the closest pair of points in the plane can be implemented in O(n log^2 n) time.

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

\[ (x_1 - x_2)^2 + (y_1 - y_2)^2 \]

\textbf{Lower bound.} In quadratic decision tree model, any algorithm for closest pair (even in 1D) requires Ω(n log n) quadratic tests.
Improved closest pair algorithm

Q. How to improve to \(O(n \log n)\)?

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

Theorem. [Shamos 1975] The divide-and-conquer algorithm for finding the closest pair of points in the plane can be implemented in \(O(n \log n)\) time.

Pf.

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

Note. See Section 13.7 for a randomized \(O(n)\) time algorithm.

Randomized quicksort

3-way partition array so that:

- Pivot element \(p\) is in place.
- Smaller elements in left subarray \(L\).
- Equal elements in middle subarray \(M\).
- Larger elements in right subarray \(R\).

Recurs in both left and right subarrays.

Randomized-Quicksort \((A)\)

If list \(A\) has zero or one element
RETURN.
Pick pivot \(p \in A\) uniformly at random.
\((L, M, R) \leftarrow \text{Partition-3-way} \ (A, a_i)\).
Randomized-Quicksort \((L)\).
Randomized-Quicksort \((R)\).

Analysis of randomized quicksort

Proposition. The expected number of compares to quicksort an array of \(n\) distinct elements is \(O(n \log n)\).

Pf. Consider BST representation of partitioning elements.

The original array of elements \(A\)

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

3-way partitioning can be done in-place (using \(n-1\) compares)
Analysis of randomized quicksort

**Proposition.** The expected number of compares to quicksort an array of $n$ distinct elements is $O(n \log n)$.

**Pf.** Consider BST representation of partitioning elements.
- An element is compared with only its ancestors and descendants.

\[
\Pr \left[ a_i \text{ and } a_j \text{ are compared} \right] = \frac{2}{i + j}.
\]

Pr[2 and 8 compared] = $\frac{2}{7}$
(compared if either 2 or 8 are chosen as partition before 3, 4, 5, 6 or 7)

\[
\Pr\{2 \text{ and } 8 \text{ compared}\} = \frac{2}{7}.
\]

Analysis of randomized quicksort

**Proposition.** The expected number of compares to quicksort an array of $n$ distinct elements is $O(n \log n)$.

**Pf.** Consider BST representation of partitioning elements.
- An element is compared with only its ancestors and descendants.

\[
\Pr\{a_i \text{ and } a_j \text{ are compared}\} = \frac{2}{i + j - 1},
\]

\[
\sum_{i=1}^{n} \sum_{j=i+1}^{n} \frac{2}{j - i - 1} = 2 \sum_{i=1}^{n} \sum_{j=i+1}^{n} \frac{1}{j} \leq 2n \sum_{j=1}^{n} \frac{1}{j}.
\]

\[
\approx 2n \int_{x=1}^{n} \frac{1}{x} dx = 2n \ln n.
\]

Remark. Number of compares only decreases if equal elements.
5. **Divide and Conquer**

- mergesort
- counting inversions
- closest pair of points
- randomized quicksort
- median and selection

**Quickselect**

3-way partition array so that:
- Pivot element $p$ is in place.
- Smaller elements in left subarray $L$.
- Equal elements in middle subarray $M$.
- Larger elements in right subarray $R$.

Recur in one subarray—the one containing the $k$th smallest element.

**Quick-select** $(A, k)$

Pick pivot $p \in A$ uniformly at random.

$(L, M, R) \leftarrow$ **Partition-three-way** $(A, p)$.

If $k \leq |L|$

```
RETURN **Quick-select** $(L, k)$.
```

Else if $k > |L| + |M|$

```
RETURN **Quick-select** $(R, k - |L| - |M|)$.
```

Else RETURN $p$.

**Median and selection problems**

**Selection.** Given $n$ elements from a totally ordered universe, find $k$th smallest.

- Minimum: $k = 1$; maximum: $k = n$.
- Median: $k = \lfloor (n + 1) / 2 \rfloor$.
- $O(n)$ compares for min or max.
- $O(n \log n)$ compares by sorting.
- $O(n \log k)$ compares with a binary heap.

**Applications.** Order statistics; find the “top $k$”; bottleneck paths, ...

**Q.** Can we do it with $O(n)$ compares?
**A.** Yes! Selection is easier than sorting.

**Quickselect analysis**

**Intuition.** Split candy bar uniformly $\Rightarrow$ expected size of larger piece is $\frac{3}{4}$.

$T(n) \leq T(\lfloor \frac{3}{4}n \rfloor) + n \Rightarrow T(n) \leq 4n$

**Def.** $T(n, k) =$ expected # compares to select $k$th smallest in an array of size $\leq n$.

**Def.** $T(n) = \max_k T(n, k)$.

**Proposition.** $T(n) \leq 4n$.

**Pf.** [by strong induction on $n$]

- Assume true for $1, 2, \ldots, n - 1$.
- $T(n)$ satisfies the following recurrence:
  - $T(n) \leq n + 2 / n [ T(n/2) + \ldots + T(n-3) + T(n-2) + T(n-1) ]$
  - $\leq n + 2 / n [ 4n/2 + \ldots + 4(n-3) + 4(n-2) + 4(n-1) ]$
  - $= n + 4(3/4)n$
  - $= 4n$.

tiny cheat: sum should start at $T(\lfloor n/2 \rfloor)$
**Selection in worst case linear time**

**Goal.** Find pivot element $p$ that divides list of $n$ elements into two pieces so that each piece is guaranteed to have $\leq \frac{7}{10} n$ elements.

**Q.** How to find approximate median in linear time?

**A.** Recursively compute median of sample of $\leq \frac{2}{10} n$ elements.

$$T(n) = \begin{cases} 
\Theta(1) & \text{if } n = 1 \\
T\left(\frac{7}{10} n\right) + T\left(\frac{2}{10} n\right) + \Theta(n) & \text{otherwise}
\end{cases}$$

**Choosing the pivot element**

- Divide $n$ elements into $\lceil n/5 \rceil$ groups of 5 elements each (plus extra).
- Find median of each group (except extra).

- Use median-of-medians as pivot element.

**Choosing the pivot element**

- Divide $n$ elements into $\lceil n/5 \rceil$ groups of 5 elements each (plus extra).
- Find median of each group (except extra).
- Find median of $\lceil n/5 \rceil$ medians recursively.
- Use median-of-medians as pivot element.

**Choosing the pivot element**

- Divide $n$ elements into $\lceil n/5 \rceil$ groups of 5 elements each (plus extra).
- Find median of each group (except extra).
- Find median of $\lceil n/5 \rceil$ medians recursively.
- Use median-of-medians as pivot element.
Median-of-medians selection algorithm

**MOM-SELECT** (A, k)

\[ n \leftarrow 1A1. \]

If \( n < 50 \) Return \( k^{th} \) smallest of element of \( A \) via mergesort.

Group \( A \) into \( \lceil n / 5 \rceil \) groups of 5 elements each (plus extra).
\( B \leftarrow \) median of each group of 5.
\( p \leftarrow \) **MOM-SELECT**(B, \( \lfloor n/10 \rfloor \))

\((L, M, R) \leftarrow \text{Partition-3-way} \ (A, p).\)

If \( k \leq |L| \) Return **MOM-SELECT** (L, k).

Else if \( k > |L| + |M| \) Return **MOM-SELECT** (R, \( k-|L|-|M| \))

Else Return \( p \).

Analysis of median-of-medians selection algorithm

- At least half of 5-element medians \( \leq p \).
- At least \( \lfloor n/5 \rfloor / 2 = \lfloor n/10 \rfloor \) medians \( \leq p \).

Analysis of median-of-medians selection algorithm

- At least half of 5-element medians \( \leq p \).
- At least \( \lfloor n/5 \rfloor / 2 = \lfloor n/10 \rfloor \) medians \( \leq p \).
- At least 3 \( \lfloor n/10 \rfloor \) elements \( \leq p \).
Analysis of median-of-medians selection algorithm

- At least half of 5-element medians $\geq p$.

Analysis of median-of-medians selection algorithm

- At least half of 5-element medians $\geq p$.
- Symmetrically, at least $\lceil n/10 \rceil$ medians $\geq p$.
- At least $3 \lfloor n/10 \rfloor$ elements $\geq p$.

Median-of-medians selection algorithm recurrence

**Median-of-medians selection algorithm recurrence.**
- Select called recursively with $\lfloor n/5 \rfloor$ elements to compute MOM $p$.
- At least $3 \lfloor n/10 \rfloor$ elements $\leq p$.
- At least $3 \lceil n/10 \rceil$ elements $\geq p$.
- Select called recursively with at most $n - 3 \lfloor n/10 \rfloor$ elements.

**Def.** $C(n) =$ max # compares on an array of $n$ elements.

$$C(n) \leq C(\lfloor n/5 \rfloor) + C(n - 3\lfloor n/10 \rfloor) + \frac{11}{5} n$$

**median of medians**

Now, solve recurrence.
- Assume $n$ is both a power of 5 and a power of 10?
- Assume $C(n)$ is monotone nondecreasing?
Median-of-medians selection algorithm recurrence

Analysis of selection algorithm recurrence.
- \( T(n) = \max \# \text{ compares on an array of } \leq n \text{ elements.} \)
- \( T(n) \) is monotone, but \( C(n) \) is not!

\[
T(n) \leq \begin{cases} 
6n & \text{if } n < 50 \\
T(\lfloor n/5 \rfloor) + T(n - 3 \lfloor n/10 \rfloor) + \frac{1}{2}n & \text{otherwise}
\end{cases}
\]

Claim. \( T(n) \leq 44n \).
- Base case: \( T(n) \leq 6n \) for \( n < 50 \) (mergesort).
- Inductive hypothesis: assume true for \( 1, 2, \ldots, n - 1 \).
- Induction step: for \( n \geq 50 \), we have:

\[
T(n) \leq T(\lfloor n/5 \rfloor) + T(n - 3 \lfloor n/10 \rfloor) + 11/5n
\leq 44(\lfloor n/5 \rfloor) + 44(n - 3 \lfloor n/10 \rfloor) + 11/5n
\leq 44(n/5) + 44n - 44(n/4) + 11/5n \quad \text{for } n \geq 50, 3 \lfloor n/10 \rfloor \geq n/4
= 44n. \quad \Box
\]

Linear-time selection postmortem

Proposition. [Blum–Floyd–Pratt–Rivest–Tarjan 1973] There exists a compare-based selection algorithm whose worst-case running time is \( O(n) \).

Practice. Constant and overhead (currently) too large to be useful.

Open. Practical selection algorithm whose worst-case running time is \( O(n) \).