5. **DIVIDE AND CONQUER I**

- mergesort
- counting inversions
- closest pair of points
- 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 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.

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 $A$).

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.

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

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

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

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

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

assuming \( n \) is a power of 2

**Proof by induction**

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

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

assuming \( n \) is a power of 2

**Pf 1.**

![Recursion tree diagram]

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

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

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

\[
= 2n \log_2 n + 2n
\]

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

\[
= 2n \log_2 (2n) \quad \blacksquare
\]

**Analysis of mergesort recurrence**

**Claim.** If \( T(n) \) satisfies the following recurrence, then \( T(n) \leq n \lceil \log_2 n \rceil \).

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

**Proof.** [by strong induction on \( n \)]

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

\[
n_2 = \lceil n/2 \rceil
\]

\[
T(n) \leq T(n_1) + T(n_2) + n
\]

\[
\leq n_1 \lceil \log_2 n_1 \rceil + n_2 \lfloor \log_2 n_2 \rfloor + n
\]

\[
\leq n_1 \lceil \log_2 n_1 \rceil + n_2 \lfloor \log_2 n_2 \rfloor + n
\]

\[
= n \lceil \log_2 n \rceil + n
\]

\[
\leq n (\lceil \log_2 n \rceil - 1) + n
\]

\[
= n \lceil \log_2 n \rceil \quad \blacksquare
\]

**5. Divide and Conquer**

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

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

<table>
<thead>
<tr>
<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>
</tr>
<tr>
<td>you</td>
<td>1</td>
<td>3</td>
<td>4</td>
<td>2</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>list A</th>
<th>list B</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>10</td>
</tr>
<tr>
<td>sort A</td>
<td>sort B</td>
</tr>
<tr>
<td>17</td>
<td>23</td>
</tr>
</tbody>
</table>

binary search to count inversions $(a, b)$ with $a \in A$ and $b \in B$

<table>
<thead>
<tr>
<th>list A</th>
<th>list B</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>7</td>
</tr>
<tr>
<td>2</td>
<td>11</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>binary search to count inversions $(a, b)$ with $a \in A$ and $b \in B</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>5</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 \(C\).

Counting inversions: divide-and-conquer algorithm implementation

**Input.** List \(L\).
**Output.** Number of inversions in \(L\) and sorted list of elements \(L'\).

```
SORT-AND-COUNT \((L)\)
  If list \(L\) has one element
    RETURN \((0, L)\).

DIVIDE the list into two halves \(A\) and \(B\).
  \((r_A, A) \leftarrow \text{SORT-AND-COUNT}(A)\).
  \((r_B, B) \leftarrow \text{SORT-AND-COUNT}(B)\).
  \((r_{AB}, L') \leftarrow \text{MERGE-AND-COUNT}(A, B)\).
  RETURN \((r_A + r_B + r_{AB}, L')\).
```

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:

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

Section 5.4
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.

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

Brute force. Check all pairs with \( \Theta(n^2) \) distance calculations.

1d version. Easy \( O(n \log n) \) algorithm if points are on a line.

Nondegeneracy assumption. No two points have the same \( x \)-coordinate.

Closest pair of points: first attempt

Sorted 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.

---

**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 $L$.

---

**Obstacle.** Impossible to ensure $n/4$ points in each piece.
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 \( L \).
- Sort points in \( 2\delta \)-strip by their \( y \)-coordinate.
- Only check distances of those within 11 positions in sorted list!

Compute separation line \( L \) such that half the points are on each side of the line.
\( \delta_1 \leftarrow \texttt{CLOSEST-PAIR} \) (points in left half).
\( \delta_2 \leftarrow \texttt{CLOSEST-PAIR} \) (points in right half).
\( \delta \leftarrow \min \{ \delta_1, \delta_2 \} \).
Delete all points further than \( \delta \) 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 \( \delta \), update \( \delta \).
\textbf{RETURN} \( \delta \).

---

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

\textbf{Def.} Let \( s_j \) be the point in the \( 2\delta \)-strip, with the \( j^{th} \) smallest \( y \)-coordinate.

\textbf{Claim.} If \( |i - j| \geq 12 \), then the distance between \( s_i \) and \( s_j \) is at least \( \delta \).

\textbf{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 \( \geq 2 (\frac{1}{2} \delta) \).

\textbf{Fact.} Claim remains true if we replace 12 with 7.

---

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

\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.

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

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

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

---

**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**

**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** ($A$, $k$)

**Algorithm**

1. Pick pivot $p \in A$ uniformly at random.
2. $(L,M,R) \leftarrow$ Partition-3-Way ($A$, $p$).
3. If $k \leq |L|$ return Quick-select ($L$, $k$).
4. Else if $k > |L| + |M|$ return Quick-select ($R$, $k - |L| - |M|$).
5. Else return $p$. 

3-way partitioning can be done in-place (using at most $n$ compares)
Quickselect analysis

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

$$T(n) \leq T\left(\frac{3}{4} n\right) + n \Rightarrow T(n) \leq 4n$$

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

**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:
  $$T(n) \leq n + 2/n \left[ T(n/2) + \ldots + T(n-3) + T(n-2) + T(n-1) \right]$$
  $$\leq n + 2/n \left[ 4n/2 + \ldots + 4(n-3) + 4(n-2) + 4(n-1) \right]$$
  $$= n + 4\left(\frac{3}{4} n\right)$$
  $$= 4n.$$  

- 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 $\lfloor n/5 \rfloor$ groups of 5 elements each (plus extra).

Choosing the pivot element

- Divide $n$ elements into $\lfloor n/5 \rfloor$ groups of 5 elements each (plus extra).
- Find median of each group (except extra).
Choosing the pivot element

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

Median-of-medians selection algorithm

\textbf{MOM-SELECT} (A, k)

$n \leftarrow |A|.$

\textbf{If:} $n < 50$ \textbf{Return} $k$th smallest of element of $A$ via mergesort.

Group $A$ into $\lfloor n/5 \rfloor$ groups of 5 elements each (plus extra).

$B \leftarrow$ median of each group of 5.

\[ p \leftarrow \text{MOM-SELECT}(B, \lfloor n/10 \rfloor) \]

\textbf{Algorithm} (L, M, R) $\leftarrow$ \text{PARTITION-3-WAY} (A, p).

\textbf{If:} $k \leq |L|$ \textbf{Return} MOM-SELECT (L, k).

\textbf{Else If:} $k > |L| + |M|$ \textbf{Return} MOM-SELECT (R, $k - |L| - |M|$)

\textbf{Else:} \textbf{Return} $p$.

Analysis of median-of-medians selection algorithm

- At least half of 5-element medians $\leq p.$
- At least $\lfloor n/5 \rfloor$ medians $\leq p.$

Analysis of median-of-medians selection algorithm

- At least half of 5-element medians $\leq p.$
- At least $\lfloor n/5 \rfloor$ medians $\leq p.$
Analysis of median-of-medians selection algorithm

• At least half of $5$-element medians $\leq p$.
• At least $\lceil n/5 \rceil = \lceil n/10 \rceil$ medians $\leq p$.
• At least $3 \lceil n/10 \rceil$ elements $\leq p$.

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 $3 \lceil n/10 \rceil$ elements $\geq p$. 
Median-of-medians selection algorithm recurrence

Median-of-medians selection algorithm recurrence.

- Select called recursively with \(\lceil n/5 \rceil\) elements to compute MOM \(p\).
- At least \(3 \lceil n/10 \rceil\) elements \(\leq p\).
- At least \(3 \lceil n/10 \rceil\) elements \(\geq p\).
- Select called recursively with at most \(n - 3 \lceil n/10 \rceil\) elements.

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

\[
C(n) \leq C(\lceil n/5 \rceil) + C(n - 3 \lceil n/10 \rceil) + \frac{11}{5} n
\]

Now, solve recurrence.

- Assume \(n\) is both a power of 5 and a power of 10?
- Assume \(C(n)\) is monotone nondecreasing?

Analysis of selection algorithm recurrence.

- \(T(n) = \max \#\) compares on an array of \(\leq n\) elements.
- \(T(n)\) is monotone, but \(C(n)\) is not!

\[
T(n) = \begin{cases} 
6n & \text{if } n < 50 \\
T(\lfloor n/5 \rfloor) + T(n - 3 \lceil n/10 \rceil) + \frac{11}{5} n & \text{otherwise}
\end{cases}
\]

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:

\[
\begin{align*}
T(n) & \leq T(\lfloor n/5 \rfloor) + T(n - 3 \lceil n/10 \rceil) + 11/5 n \\
& \leq 44 \lfloor n/5 \rfloor + 44 (n - 3 \lceil n/10 \rceil) + 11/5 n \\
& \leq 44 n - 44 (n/4) + (11/5 n \quad \text{for } n \geq 50, 3 \lceil n/10 \rceil \approx n/4 \\
& = 44 n
\end{align*}
\]

Linear-time selection postmortem

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

Theory.

- Optimized version of BFPrT: \(\leq 5.4305 n\) compares.
- Best known upper bound [Dor-Zwick 1995]: \(\leq 2.95 n\) compares.
- Best known lower bound [Dor-Zwick 1999]: \(\geq (2+\varepsilon) n\) compares.

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

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