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 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.

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

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

Assorted proofs. We describe several ways to 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 \(T(n) = n \log_2 n\).

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

**Proof by induction**

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

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

**Proof by induction**

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

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

**Analysis of mergesort recurrence**

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

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

**Proof.** [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 \(1, 2, \ldots, n - 1\).

\[
T(n) \leq T(n_1) + T(n_2) + n \\
\leq n_1 \lfloor \log_2 n_1 \rfloor + n_2 \lceil \log_2 n_2 \rceil + n \\
= n \lfloor \log_2 n \rfloor + n \\
\leq n \lfloor \log_2 n \rfloor + n \\
= n \lfloor \log_2 n \rfloor.
\]
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.

input

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

for each element $b \in B$:
- binary search in $A$ to find how elements in $A$ are greater than $b$.

output $1 + 3 + 13 = 17$
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 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:

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

5. **DIVIDE AND CONQUER**

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

**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.

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

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: 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 $L$.

Closest pair of points: second attempt

Divide. Subdivide region into 4 quadrants.

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!

Why 11? 

\[ \delta = \min(12, 21) \]

Closest pair of points: divide-and-conquer algorithm

**CLOSEST-PAIR** \((p_1, p_2, \ldots, p_n)\)

Compute separation line \( L \) such that half the points are on each side of the line.
\[ \delta_1 \leftarrow \text{CLOSEST-PAIR} \text{ (points in left half)} \]
\[ \delta_2 \leftarrow \text{CLOSEST-PAIR} \text{ (points in right half)} \]
\[ \delta \leftarrow \min \{ \delta_1, \delta_2 \} \]
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 \( \delta \).

RETURN \( \delta \).

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

**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 \( \geq 2 \left( \frac{1}{2}\delta \right) \).

**Fact.** Claim remains true if we replace 12 with 7.

Closest pair of points: analysis

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

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

**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.

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 $R$.

recur 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.
$(L, M, R) \leftarrow$ Partition-3-Way ($A$, $a$).
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.
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.
  
\[
\text{Pr}[ a_i \text{ and } a_j \text{ are compared } ] = \frac{2}{|j-i+1|}.
\]

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

(2 and 8 are not compared because 3 partitions them)

\[
\text{Pr}(2 \text{ or } 8 \text{ is chosen before } 3, 4, 5, 6, \text{ or } 7)
\]

\[
\frac{2}{7} \leq \frac{2}{|j-i+1|}
\]

\[
\approx 2N \int_1^N \frac{1}{x} \, dx
\]

\[
= 2N \ln N
\]

Remark. Number of compares only decreases if equal elements.
CHAPTER 9

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-3-WAY $(A, p)$.

- If $k \leq |L|$ then return QUICK-SELECT $(L, k)$.
- Else if $k > |L| + |M|$ then return $p$.
- Else return QUICK-SELECT $(R, k - |L| - |M|)$.

Quickselect analysis

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

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

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

Def. $T(n) =$ max$_{k \leq n}$ $T(n, k)$.

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:

$$T(n) \leq n + 2/n \left[ T(n/2) + \ldots + T(n - 3) + T(n - 2) + T(n - 1) \right]$$

$$\leq n + 2/n \left[ 4n/2 + \ldots + 4(n - 3) + 4(n - 2) + 4(n - 1) \right]$$

$$= n + 4(3/4)n$$

$$= 4n.$$
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 $\leq 7/10\ n$ elements.

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

**A.** Recursively compute median of sample of $\leq 2/10\ n$ 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).
- Find median of each group (except extra).
- Use median-of-medians as pivot element.
Median-of-medians selection algorithm

\[ \text{MOM-SELECT} (A, k) \]

\[ n \leftarrow |A|. \]
\[ \text{If } n < 50 \text{ return } k^{th} \text{ smallest of element of } A \text{ via mergesort.} \]

Group \( A \) into \( \left\lfloor \frac{n}{5} \right\rfloor \) groups of 5 elements each (plus extra).
\( B \leftarrow \text{median of each group of 5.} \)
\( p \leftarrow \text{MOM-SELECT}(B, \left\lfloor \frac{n}{10} \right\rfloor) \)  

\((L, M, R) \leftarrow \text{PARTITION-3-WAY} (A, p). \)
\[ \text{If } k \leq |L| \text{ return MOM-SELECT}(L, k). \]
\[ \text{Else if } k > |L| + |M| \text{ return MOM-SELECT}(R, k - |L| - |M|). \]
\[ \text{Else return } p. \]

Analysis of median-of-medians selection algorithm

- At least half of 5-element medians \( \leq p. \)
- At least \( \left\lfloor \frac{n}{5} \rightfloor / 2 = \left\lfloor \frac{n}{10} \right\rfloor \text{ medians } \leq p. \)
Analysis of median-of-medians selection algorithm

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

$$C(n) \leq C(\lfloor n/5 \rfloor) + C(n - 3 \lfloor n/10 \rfloor) + \frac{1}{2} n$$

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!

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

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:

\[
T(n) \leq T\left( \frac{n}{5} \right) + T\left( \frac{n-3}{10} \right) + 11/5n
\leq 44 \left( \frac{n}{5} \right) + 44 \left( \frac{n-3}{10} \right) + 11/5n
\leq 44 \left( \frac{n}{5} \right) + 44n - 44 \left( \frac{n}{4} \right) + 11/5n \\
= 44n.
\]

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 \( O(n) \).