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 (of the same kind).
- Solve (conquer) 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 (conquer) two subproblems recursively.
- Combine two solutions into overall solution in linear time.

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

- `mergesort`
- `counting inversions`
- `closest pair of points`
- `randomized quicksort`
- `median and selection`
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>3</th>
<th>7</th>
<th>10</th>
<th>$a_i$</th>
<th>18</th>
</tr>
</thead>
<tbody>
<tr>
<td>sorted list B</td>
<td>2</td>
<td>11</td>
<td>$b_j$</td>
<td>20</td>
<td>23</td>
</tr>
</tbody>
</table>

merge to form sorted list $C$

| 2 | 3 | 7 | 10 | 11 |
A useful recurrence relation

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

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

Mergesort recurrence.

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

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: recursion tree

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

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

assuming \( n \) is a power of 2
Proof by induction

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

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

**Pf.** [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 $T(2n) = 2n \log_2 (2n)$.

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

\[\blacksquare\]
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 \( 1, 2, \ldots, n - 1 \).

\[
\begin{align*}
T(n) & \leq T(n_1) + T(n_2) + n \\
& \leq n_1 \lfloor \log_2 n_1 \rfloor + n_2 \lceil \log_2 n_2 \rceil + n \\
& \leq n_1 \lfloor \log_2 n_2 \rfloor + n_2 \lceil \log_2 n_2 \rceil + n \\
& = n \lfloor \log_2 n_2 \rfloor + n \\
& \leq n \lfloor \log_2 n \rfloor - 1 + n \\
& = n \lfloor \log_2 n \rfloor.
\end{align*}
\]
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 $a_i > a_j$.

```
<table>
<thead>
<tr>
<th></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>
```

2 inversions: 3–2, 4–2

**Brute force:** check all $\Theta(n^2)$ pairs.
Counting inversions: 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).

---

**Rank Aggregation Methods for the Web**

Cynthia Dwork\1
Ravi Kumar\2
Moni Naor\1
D. Sivakumar\2

**ABSTRACT**

We consider the problem of combining ranking results from various sources. In the context of the Web, the main applications include building meta-search engines, combining ranking functions, selecting documents based on multiple criteria, and improving search precision through word associations. We develop a set of techniques for the rank aggregation problem and compare their performance to that of well-known methods. A primary goal of our work is to design rank aggregation techniques that can effectively combat “spam,” a serious problem in Web searches. Experiments show that our methods are simple, efficient, and effective.

*Keywords:* rank aggregation, ranking functions, meta-search, multi-word queries, spam
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**

<table>
<thead>
<tr>
<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>
</table>

**Count inversions in left half \( A \)**

<table>
<thead>
<tr>
<th>1</th>
<th>5</th>
<th>4</th>
<th>8</th>
<th>10</th>
<th>5-4</th>
</tr>
</thead>
</table>

**Count inversions in right half \( B \)**

<table>
<thead>
<tr>
<th>2</th>
<th>6</th>
<th>9</th>
<th>3</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>6-3</td>
<td>9-3</td>
<td>9-7</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<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>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>
</tbody>
</table>

**Output** \( 1 + 3 + 13 = 17 \)
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>list B</th>
</tr>
</thead>
<tbody>
<tr>
<td>7 10 18 3 14</td>
<td>20 23 2 11 16</td>
</tr>
</tbody>
</table>

sort A

<table>
<thead>
<tr>
<th>sort A</th>
</tr>
</thead>
<tbody>
<tr>
<td>3 7 10 14 18</td>
</tr>
</tbody>
</table>

sort B

<table>
<thead>
<tr>
<th>sort B</th>
</tr>
</thead>
<tbody>
<tr>
<td>2 11 16 20 23</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>count</th>
</tr>
</thead>
<tbody>
<tr>
<td>3 7 10 14 18</td>
</tr>
<tr>
<td>2 11 16 20 23</td>
</tr>
<tr>
<td>5 2 1 0 0</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 \(C\).

\[
\begin{array}{cccc}
3 & 7 & 10 & a_i \\
2 & 11 & b_j & 20 & 23 \\
5 & 2
\end{array}
\]

merge to form sorted list \(C\)

\[
\begin{array}{cccc}
2 & 3 & 7 & 10 & 11
\end{array}
\]
Counting inversions: divide-and-conquer algorithm implementation

Input. List $L$.

Output. Number of inversions in $L$ and sorted list of elements $L'$.

\[ \text{SORT-AND-COUNT}(L) \]

If list $L$ has one element

\[ \text{RETURN } (0, L). \]

Divide the list into two halves $A$ and $B$.

\[ (r_A, A) \leftarrow \text{SORT-AND-COUNT}(A). \]
\[ (r_B, B) \leftarrow \text{SORT-AND-COUNT}(B). \]
\[ (r_{AB}, L') \leftarrow \text{MERGE-AND-COUNT}(A, B). \]

\[ \text{RETURN } (r_A + r_B + r_{AB}, L'). \]
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
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.

fast closest pair inspired fast algorithms for these problems
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.

**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

**Sorting solution.**

- Sort by $x$-coordinate and consider nearby points.
- Sort by $y$-coordinate and consider nearby points.
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.
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.

seems like $\Theta(N^2)$
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 < $\delta$.
- 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!

\[ \delta = \min(12, 21) \]
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** \((p_1, p_2, \ldots, p_n)\)

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** \(\delta\).
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.

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

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

$$(x_1 - x_2)^2 + (y_1 - y_2)^2$$
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.
5. **Divide and Conquer**

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

*Chapter 7*
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.

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

**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.

![Diagram of BST representation of partitioning elements]

- First partitioning element in left subarray
- First partitioning element (chosen uniformly at random)
- 3 and 6 are compared (when 3 is partitioning element)
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.

![BST representation](image)
**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.
- \( \Pr \left[ a_i \text{ and } a_j \text{ are compared} \right] = \frac{2}{j - i + 1}, \text{ where } i < j. \)

![BST representation of partitioning elements](image-url)

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

(compared if either 2 or 8 are chosen as partition before 3, 4, 5, 6 or 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.
- \( \Pr[ a_i \text{ and } a_j \text{ are compared} ] = \frac{2}{j - i + 1} \), where \( i < j \).

- Expected number of compares

\[
\sum_{i=1}^{n} \sum_{j=i+1}^{n} \frac{2}{j - i - 1} = 2 \sum_{i=1}^{n} \sum_{j=2}^{n-i+1} \frac{1}{j} \\
\leq 2n \sum_{j=1}^{n} \frac{1}{j} \\
\sim 2n \int_{x=1}^{n} \frac{1}{x} dx \\
= 2n \ln n
\]

**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
**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)
```

Pick pivot \( p \in A \) uniformly at random.

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

**IF** \( k \leq |L| \) **RETURN** \( \text{QUICK-SELECT}(L, k) \).

**ELSE IF** \( k > |L| + |M| \) **RETURN** \( \text{QUICK-SELECT}(R, k - |L| - |M|) \).

**ELSE** \( \text{RETURN } p \).

3-way partitioning can be done in-place (using \( n-1 \) 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.** \( T(n, k) = \text{expected \# compares to select } k\text{th smallest in an array of size } \leq n \).

**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:

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

can assume we always recur on largest subarray since \( T(n) \) is monotonic and we are trying to get an upper bound
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 \( \leq 7/10 \) \( n \) elements.

**Q.** How to find approximate median in linear time?
**A.** Recursively compute median of sample of \( \leq 2/10 \) \( n \) elements.

\[
T(n) = \begin{cases} 
\Theta(1) & \text{if } n = 1 \\
T(7/10 \ n) + T(2/10 \ n) + \Theta(n) & \text{otherwise}
\end{cases}
\]
Choosing the pivot element

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

\( N = 54 \)
Median-of-medians selection algorithm

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

\(n \leftarrow |A|\).

**IF** \(n < 50\) **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.

\(p \leftarrow\) **MOM-SELECT** \((B, \lfloor n / 10 \rfloor)\) median of medians

\((L, M, R) \leftarrow\) **PARTITION-3-WAY** \((A, p)\).

**IF** \(k \leq |L|\) \(\text{RETURN}\) **MOM-SELECT** \((L, k)\).

**ELSE IF** \(k > |L| + |M|\) \(\text{RETURN}\) **MOM-SELECT** \((R, k - |L| - |M|)\)

**ELSE** \(\text{RETURN} p\).
Analysis of median-of-medians selection algorithm

- At least half of 5-element medians ≤ $p$.
Analysis of median-of-medians selection algorithm

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

- At least half of 5-element medians $\leq p$.
- At least $\lceil n/5 \rceil / 2 = \lfloor n/10 \rfloor$ medians $\leq p$.
- At least $3 \lfloor n/10 \rfloor$ elements $\leq p$. 

$N = 54$
Analysis of median-of-medians selection algorithm

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

<table>
<thead>
<tr>
<th>29</th>
<th>10</th>
<th>38</th>
<th>37</th>
<th>2</th>
<th>55</th>
<th>18</th>
<th>24</th>
<th>34</th>
<th>35</th>
<th>36</th>
</tr>
</thead>
<tbody>
<tr>
<td>22</td>
<td>44</td>
<td>52</td>
<td>11</td>
<td>53</td>
<td>12</td>
<td>13</td>
<td>43</td>
<td>20</td>
<td>4</td>
<td>27</td>
</tr>
<tr>
<td>28</td>
<td>23</td>
<td>6</td>
<td>26</td>
<td>40</td>
<td>19</td>
<td>1</td>
<td>46</td>
<td>31</td>
<td>49</td>
<td>8</td>
</tr>
<tr>
<td>14</td>
<td>9</td>
<td>5</td>
<td>3</td>
<td>54</td>
<td>30</td>
<td>48</td>
<td>47</td>
<td>32</td>
<td>51</td>
<td>21</td>
</tr>
<tr>
<td>45</td>
<td>39</td>
<td>50</td>
<td>15</td>
<td>25</td>
<td>16</td>
<td>41</td>
<td>17</td>
<td>22</td>
<td>7</td>
<td></td>
</tr>
</tbody>
</table>
Analysis of median-of-medians selection algorithm

- At least half of 5-element medians ≥ \( p \).
- Symmetrically, at least \( \left\lceil n/10 \right\rceil \) medians ≥ \( p \).

N = 54
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 $\geq p$. 

N = 54
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 $3 \lfloor n / 10 \rfloor$ 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.** $C(n) = \max \# \text{ compares on an array of } n \text{ elements.}$

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

median of medians recursive select computing median of 5 partitioning (n compares per group)

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.

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

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

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:

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


---

**Abstract**

The number of comparisons required to select the $i$-th smallest of $n$ numbers is shown to be at most a linear function of $n$ by analysis of a new selection algorithm -- PICK. Specifically, no more than $5.4305n$ comparisons are ever required. This bound is improved for extreme values of $i$, and a new lower bound on the requisite number of comparisons is also proved.

---

**Theory.**

- Optimized version of BFPRT: $\leq 5.4305n$ compares.
- Best known upper bound [Dor–Zwick 1995]: $\leq 2.95n$ compares.
- Best known lower bound [Dor–Zwick 1999]: $\geq (2 + \varepsilon)n$ compares.
Linear-time selection postmortem


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

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