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: $\Theta(n \log n)$.

attributed to Julius Caesar
5. DIVIDE AND CONQUER

- mergesort
- counting inversions
- closest pair of points
- 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>sorted list B</th>
</tr>
</thead>
<tbody>
<tr>
<td>3 7 10 ( a_i ) 18</td>
<td>2 11 ( b_j ) 17 23</td>
</tr>
<tr>
<td>5 2</td>
<td></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 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 \( T(n) = n \log_2 n \).

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

**Pf 1.**

\[
\begin{aligned}
T(n) & = n \\
2 \times (n/2) & = n \\
4 \times (n/4) & = n \\
8 \times (n/8) & = n \\
\vdots \\
T(n) & = n \log_2 n
\end{aligned}
\]
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 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 $T(2n) = 2n \log_2 (2n)$.

$$
T(2n) = 2 \ T(n) + 2n \\
= 2n \log_2 n + 2n \\
= 2n (\log_2 (2n) - 1) + 2n \\
= 2n \log_2 (2n).
$$
Analysis of mergesort recurrence

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

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

**Pf.** [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 $1, 2, \ldots, n - 1$.

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

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

\[
\leq \left\lceil 2^{\lceil \log_2 n \rceil} / 2 \right\rceil = 2^{\lceil \log_2 n \rceil} / 2
\]

\[
\log_2 n_2 \leq \lceil \log_2 n \rceil - 1
\]

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

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

$\blacksquare$
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></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† Ravi Kumar‡ Moni Naor‡ D. Sivakumar‡

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**

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

**count inversions in left half A**

| 1 | 5 | 4 | 8 | 10 |

\[ 5-4 \]

**count inversions in right half B**

| 2 | 6 | 9 | 3 | 7 |

\[ 6-3 \quad 9-3 \quad 9-7 \]

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

| 1 | 5 | 4 | 8 | 10 |

\[ 4-2 \quad 4-3 \quad 5-2 \quad 5-3 \quad 8-2 \quad 8-3 \quad 8-6 \quad 8-7 \quad 10-2 \quad 10-3 \quad 10-6 \quad 10-7 \quad 10-9 \]

**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>17 23 2 11 16</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>sort A</th>
<th>sort B</th>
</tr>
</thead>
<tbody>
<tr>
<td>3 7 10 14 18</td>
<td>2 11 16 17 23</td>
</tr>
</tbody>
</table>

| binary search to count inversions \((a, b)\) with \(a \in A\) and \(b \in B\) |
|--------|--------|
| 3 7 10 14 18 | 2 11 16 17 23 |
| 5 2 1 1 0 |
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 \(C\).

<table>
<thead>
<tr>
<th>(a_i)</th>
<th>3</th>
<th>7</th>
<th>10</th>
<th>18</th>
</tr>
</thead>
<tbody>
<tr>
<td>(b_j)</td>
<td>2</td>
<td>11</td>
<td>17</td>
<td>23</td>
</tr>
<tr>
<td>5</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

merge to form sorted list \(C\)

| 2 | 3 | 7 | 10 | 11 | ... |
Counting inversions: divide-and-conquer algorithm implementation

**Input.** List \( L \).

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

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

\[
\text{IF} \text{ list } L \text{ has one element}
\]

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

\[
\text{DIVIDE} \text{ the list into two halves } A \text{ and } B.
\]

\[
(r_A, A) \leftarrow \text{SORT-AND-COUNT}(A).
\]

\[
(r_B, B) \leftarrow \text{SORT-AND-COUNT}(B).
\]

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

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

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

**Fact.** Claim remains true if we replace 12 with 7.
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** \(\delta\).
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.

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

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

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

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

not subject to lower bound since it uses the floor function
5. Divide and Conquer

- quicksort
- counting inversions
- closest pair of points
- 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** **Quick-Select** $(L, k)$.

**ELSE IF** $k > |L| + |M|$ **RETURN** **Quick-Select** $(R, k - |L| - |M|)$

**ELSE** **RETURN** $p$.  

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

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

\[
T(n) \leq T\left(\frac{3}{4} n\right) + n \quad \Rightarrow \quad T(n) \leq 4n
\]

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

**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:

\[
T(n) \leq n + \frac{2}{n} \left[ T\left(\frac{n}{2}\right) + \ldots + T(n - 3) + T(n - 2) + T(n - 1) \right]
\]

\[
\leq n + \frac{2}{n} \left[ 4 \frac{n}{2} + \ldots + 4(n - 3) + 4(n - 2) + 4(n - 1) \right]
\]

\[
= n + 4 \left(\frac{3}{4} n\right)
\]

\[
= 4n. \quad \blacksquare
\]

- Tiny cheat: sum should start at \( T(\lfloor n/2 \rfloor) \)

- Can assume we always recur on largest subarray
- Since \( T(n) \) is monotonic and
- We are trying to get an upper bound
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.

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

two subproblems of different sizes!
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.
Median-of-medians selection algorithm

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

\begin{align*}
n & \leftarrow |A|.
\text{IF} \ n < 50 \ \text{RETURN} \ k^{th} \ \text{smallest of element of} \ A \ \text{via mergesort.} \\
\text{Group} \ A \ \text{into} \ \lfloor n / 5 \rfloor \ \text{groups of} \ 5 \ \text{elements each (plus extra).} \\
B & \leftarrow \text{median of each group of} \ 5.
\end{align*}

\begin{align*}
p & \leftarrow \text{MOM-SELECT}(B, \lfloor n / 10 \rfloor) \quad \text{median of medians} \\
(L, M, R) & \leftarrow \text{PARTITION-3-WAY} \ (A, p).
\end{align*}

\begin{align*}
\text{IF} \ & k \leq |L| \quad \text{RETURN} \ \text{MOM-SELECT} \ (L, k). \\
\text{ELSE IF} \ & k > |L| + |M| \quad \text{RETURN} \ \text{MOM-SELECT} \ (R, k - |L| - |M|) \\
\text{ELSE} \ & \quad \text{RETURN} \ p.
\end{align*}
Analysis of median-of-medians selection algorithm

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

$N = 54$
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 $\leq p$. 

Median of medians $p$
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 $\leq p$. 

$N = 54$
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 $\lfloor n / 10 \rfloor$ medians $\geq p$. 

$N = 54$
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 $\geq p$. 

N = 54
Median-of-medians selection algorithm recurrence.

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

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

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

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.**

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

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

**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:

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

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

---

**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

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

**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 $i$.

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

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