Design and Analysis of Algorithms
Divide-and-Conquer

Si Wu

School of CSE, SCUT

cswusi@scut.edu.cn

TA: 1684350406@qq.com

Topics

- **Divide-and-Conquer Paradigm**
- **Closest Pair of Points**
- **Median and Selection Problems**

Divide-and-Conquer Paradigm

Divide-and-Conquer.

- Divide problem into several subproblems.
- Solve each subproblem recursively.
- Combine solution 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.

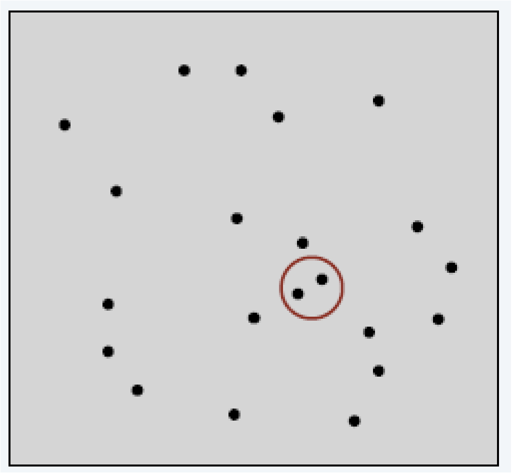
- Brute force: $\Theta(n^2)$.
- Divide-and-conquer: $\Theta(nlogn)$.

Closest Pair of Points

Closest pair problem. Given *n* points in the plane, fine 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.



Closest Pair of Points

Closest pair problem. Given *n* points in the plane, fine a pair of points with the smallest Euclidean distance between them.

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

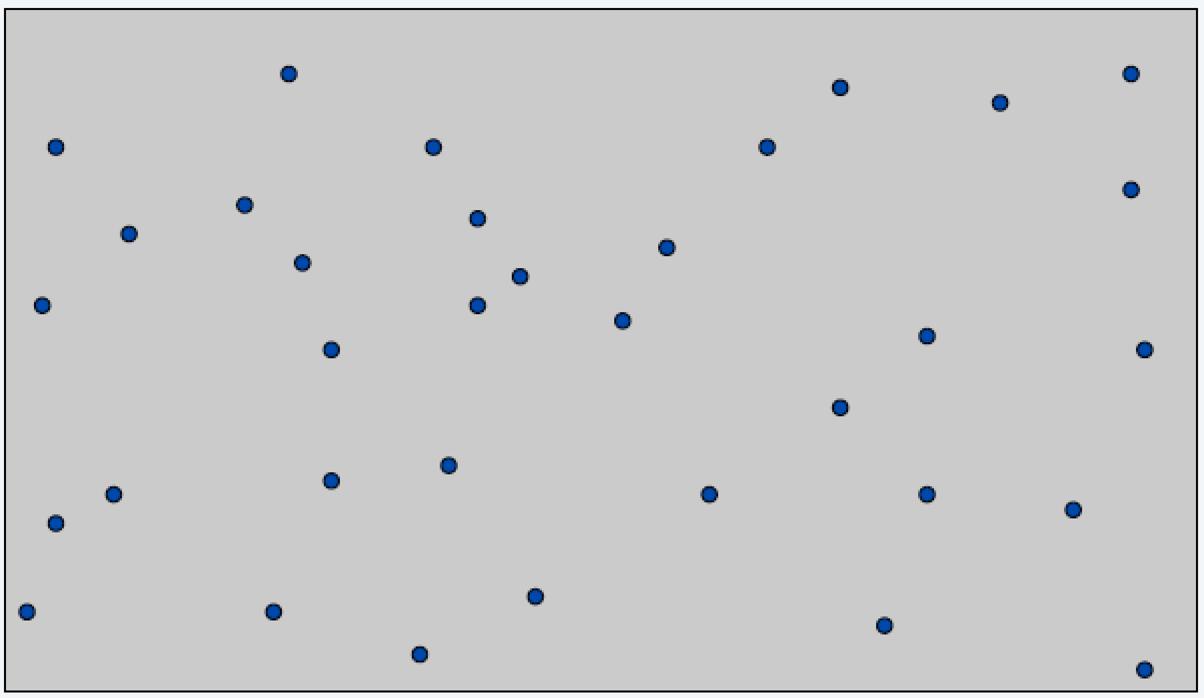
1D version. Easy $O(nlogn)$ algorithm if points are on a line.

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

Closest Pair of Points: First Attempt

Sorting solution.

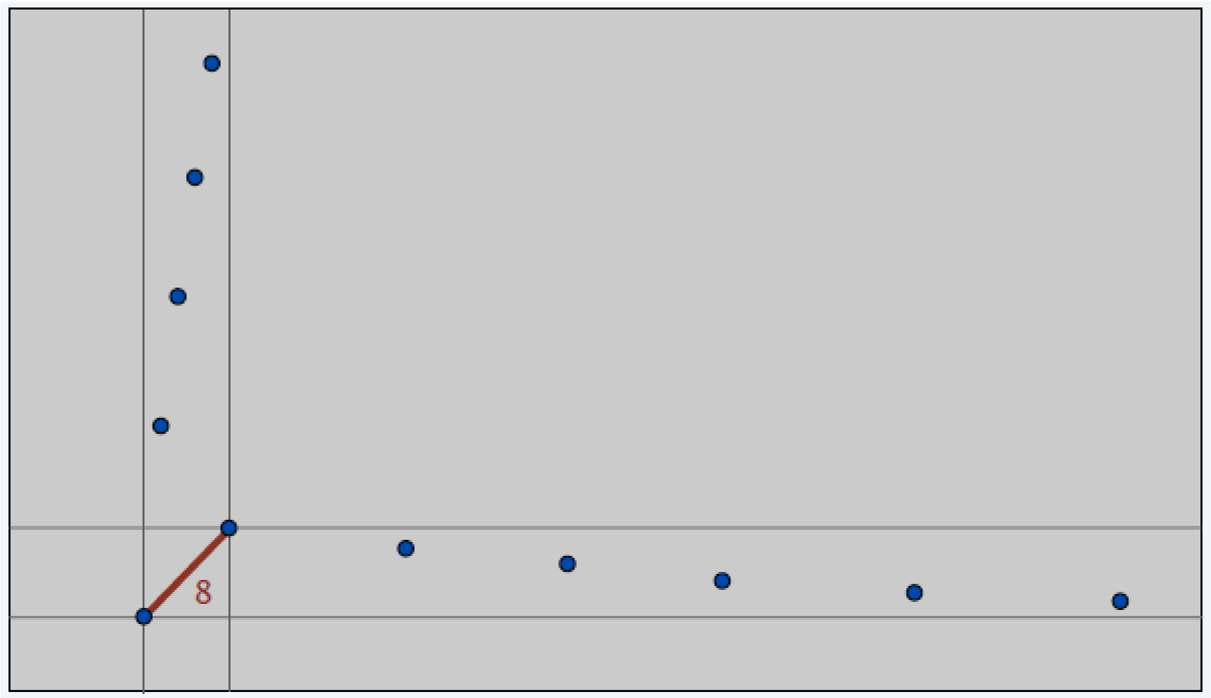
- Sort by x-coordinate and consider nearby points.
- Sort by y-coordinate and consider nearby points.



Closest Pair of Points: First Attempt

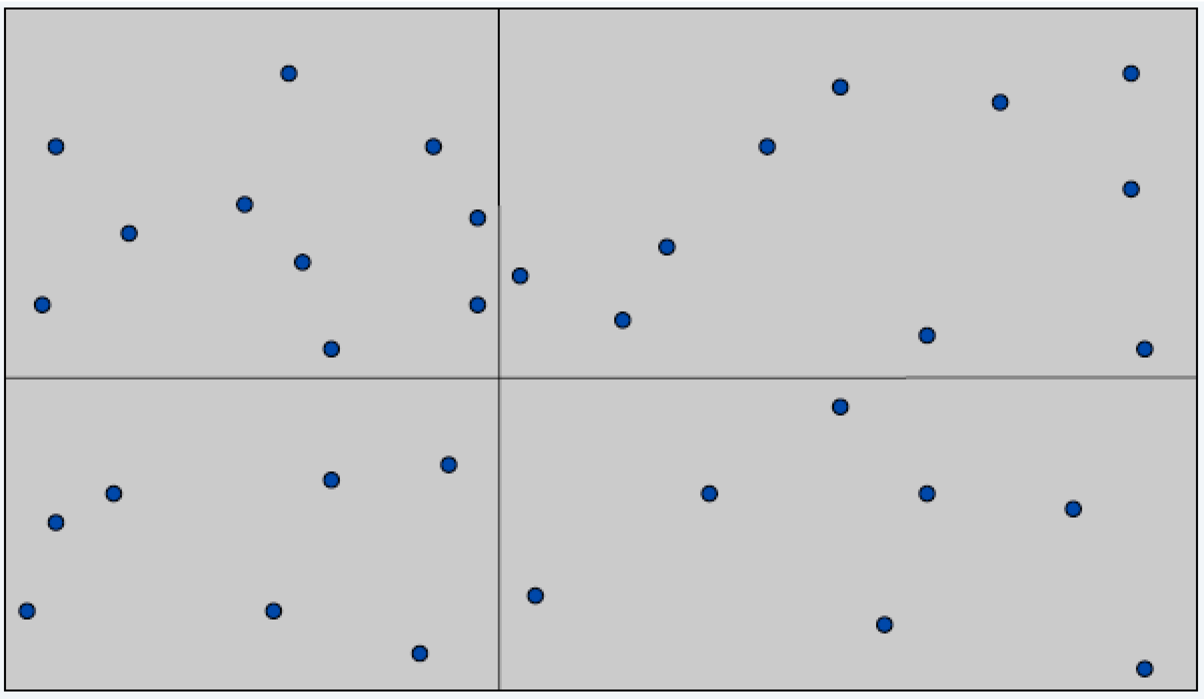
Sorting solution.

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



Closest Pair of Points: Second Attempt

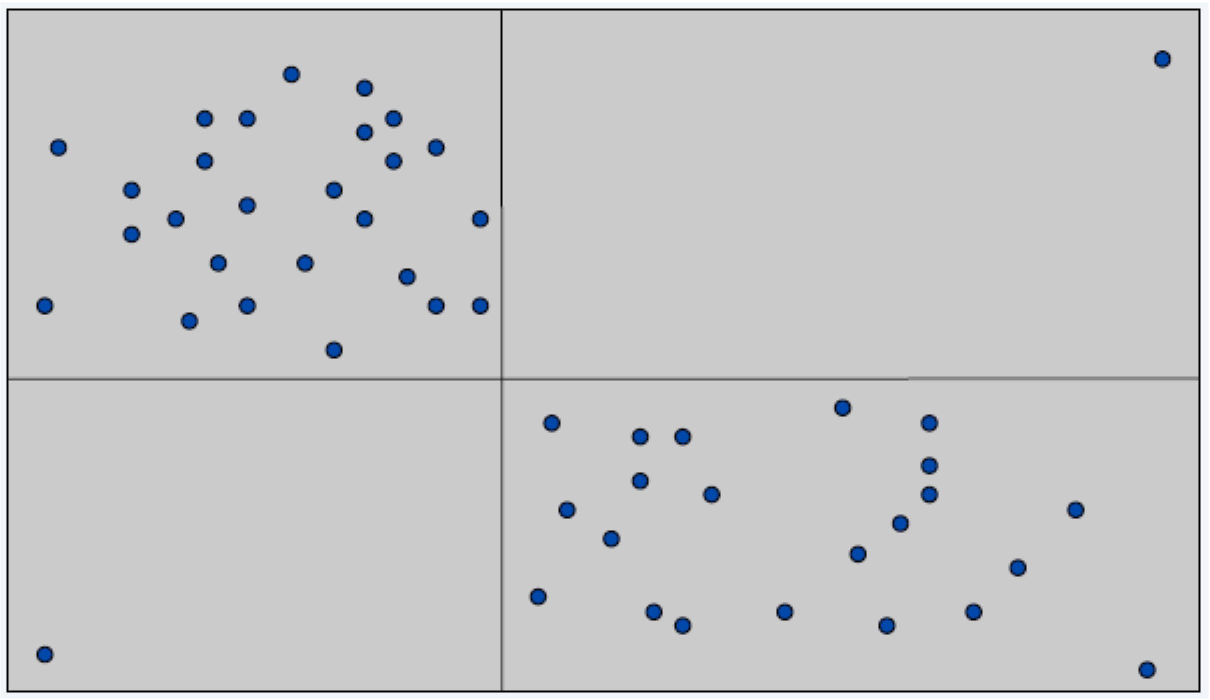
Divide. Subdivide region into 4 quadrants.



Closest Pair of Points: Second Attempt

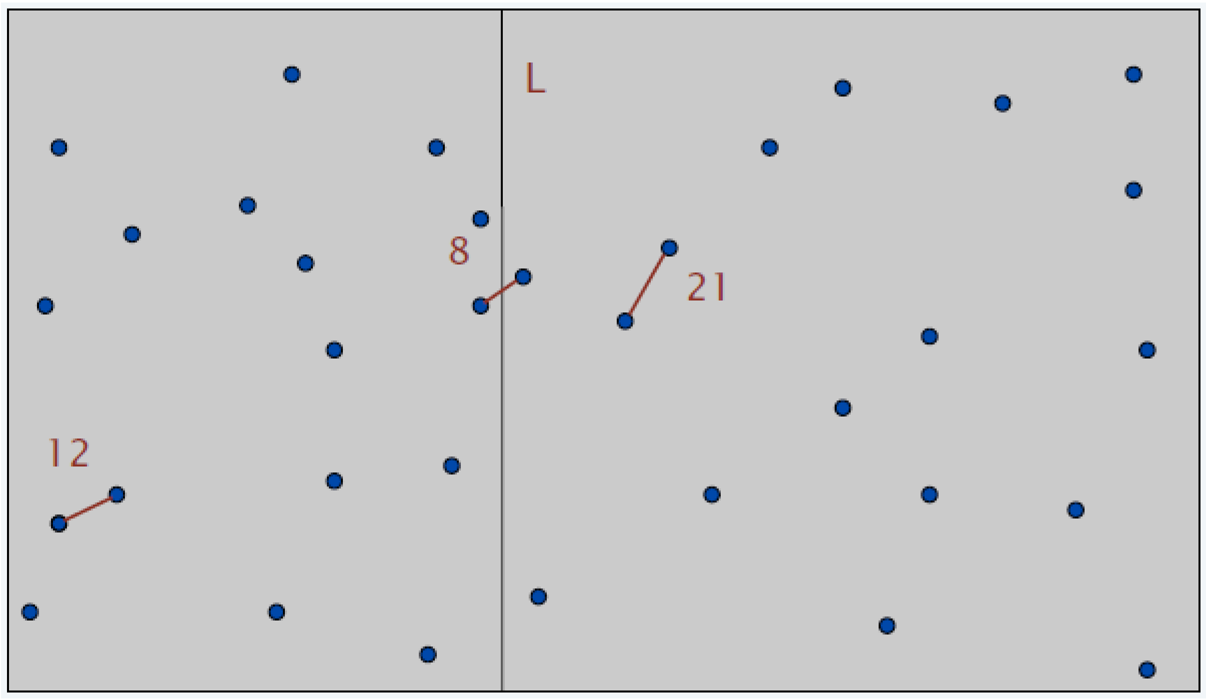
Divide. Subdivide region into 4 quadrants.

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



Closest Pair of Points: Divide-and-Conquer Algorithm

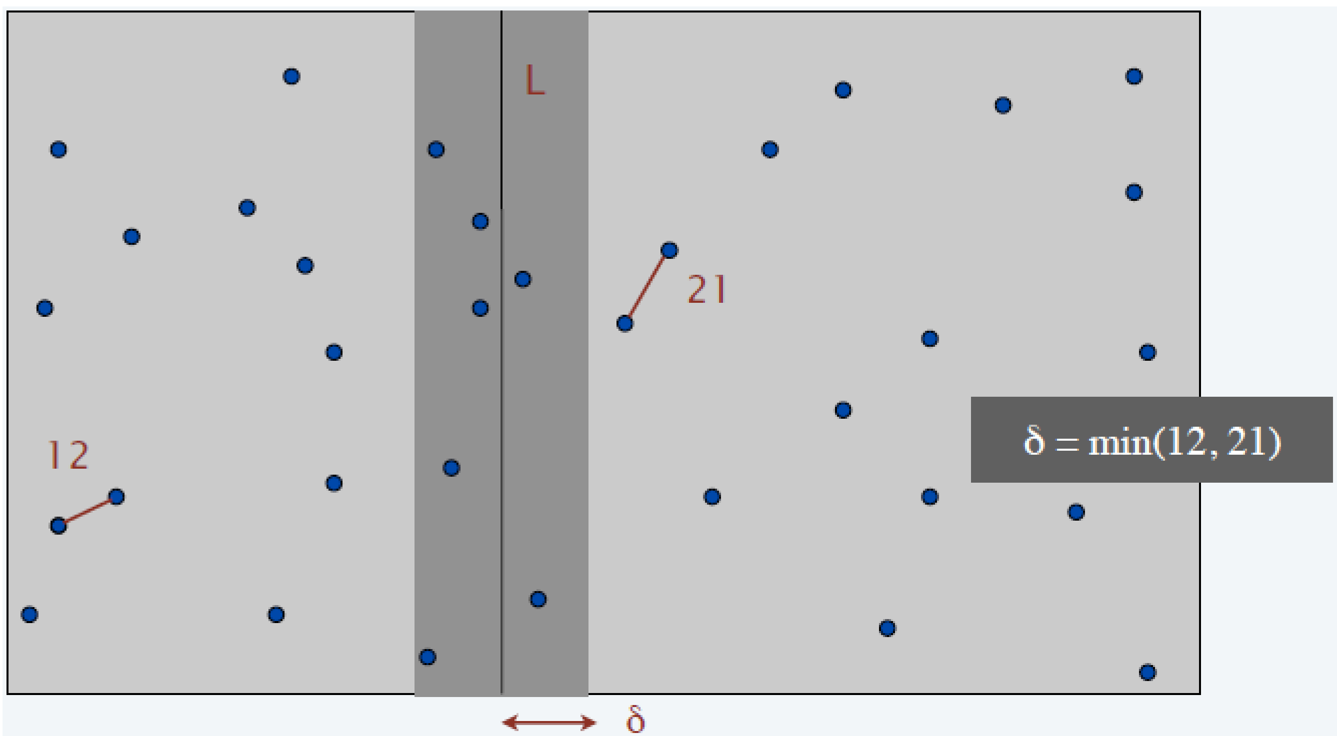
- Divide: draw vertical line L so that n/2 points on each side.
- Conquer: find closet pair in each side recursively.
- Combine: find closet pair with one point in each side.
- Return best of 3 solutions.



How to Find Closest Pair with One Point in Each Side?

Find closest pair with one point in each side, assuming that distance $< \delta$.

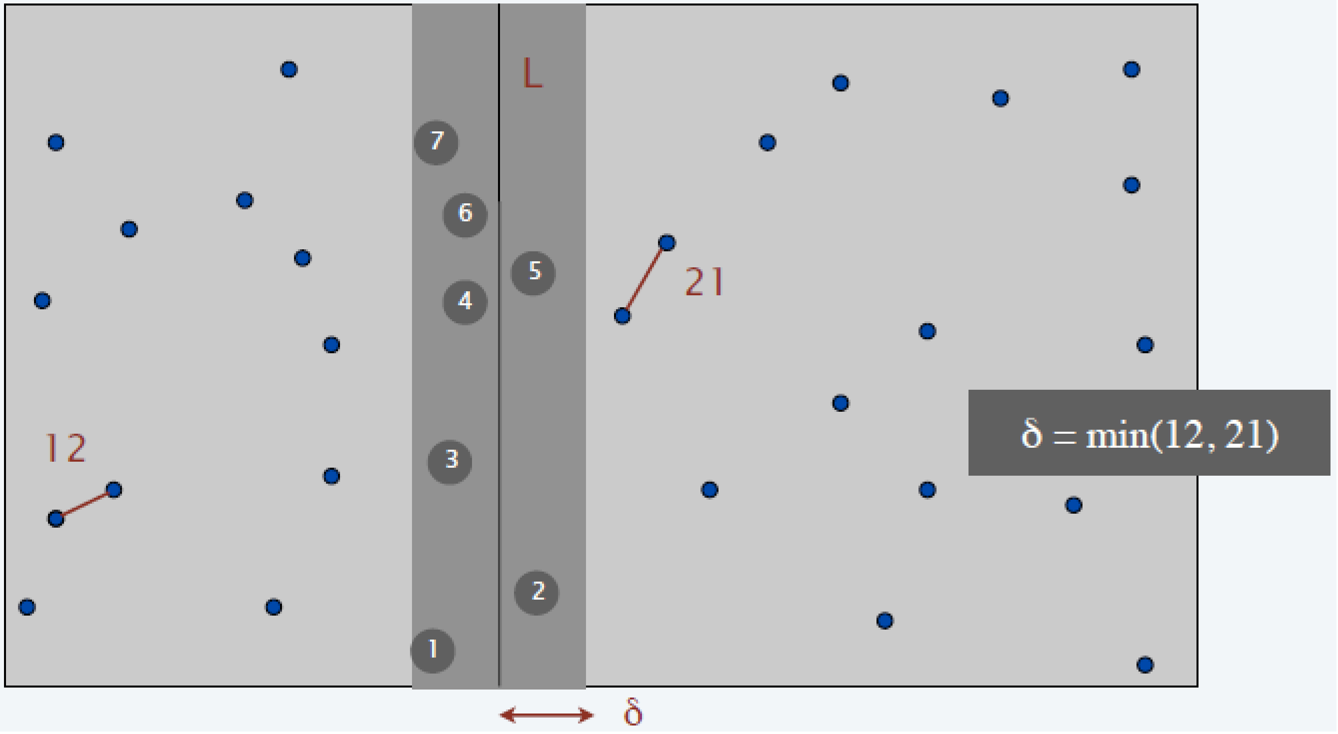
- Observation: only need to consider points within δ of line L.



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 δ of line L.
- Sort points in 2δ-strip by their y-coordinate.
- Only check distances of those within 15 positions in sorted list.



How to Find Closest Pair with One Point in Each Side?

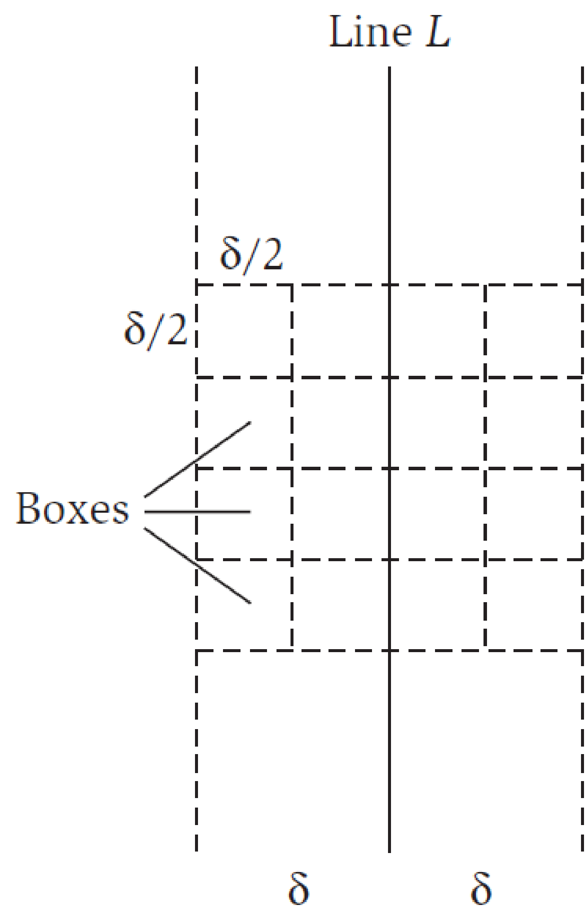
Each box can contain at most one input point.

Def. Let s_i be the point in the 2δ-strip, with the i-th smallest y-coordinate.

Claim. If $|i - j| \geq 16$, then the distance between s_i and s_j is at least $\frac{3}{2}\delta$.

Pf.

- No two points lie in same $\frac{1}{2}\delta$ by $\frac{1}{2}\delta$ box.
- Two points at least 3 rows apart
- have distance $\geq 3(\frac{1}{2}\delta)$.



Note. *The value of 15 can be reduced. The important thing is that it is an absolute constant.*

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.
- $\delta_1 \leftarrow$ Closest-Pair (points in left half).
- $\delta_2 \leftarrow$ Closest-Pair (points in right half).
- $\delta \leftarrow \min\{\delta_1, \delta_2\}$.
- 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 15 neighbors. If any of these distances is less than δ, update δ.

Return δ.

$$\boldsymbol{T(n) = ?}$$

Closest Pair of Points: Divide-and-Conquer Algorithm

Closest-Pair $(p_1, p_2, \dots, p_n)$

- Compute separation line L such that half the points are on each side of the line. ⟵ $O(nlogn)$
- $\delta_1 \leftarrow$ Closest-Pair (points in left half). ⟵ $2T(n/2)$
- $\delta_2 \leftarrow$ Closest-Pair (points in right half).
- $\delta \leftarrow \min\{\delta_1, \delta_2\}$.
- Delete all points further than δ from Line L. ⟵ $O(n)$
- Sort remaining points by y-coordinate. ⟵ $O(nlogn)$
- Scan points in y-order and compare distance between each point and next 15 neighbors. If any of these distances is less than δ, update δ. ⟵ $O(n)$

Return δ.

$$\boldsymbol{T(n) = ?}$$

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 ? time.

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

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 $O(nlog^2n)$ time.

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

Master Theorem - Case 2. If $f(n) = \Theta(n^k log^p n)$ for $p \geq 0$ and $k = \log_b a$, then $T(n) = \Theta\left(n^k log^{p+1} n\right)$.

Median and Selection Problems

Selection. Given *n* elements, find *k*-th smallest.

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

Applications. Find the “top k”…

Can we do it with $O(n)$ compares?

Quick-Select

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 $k \leq |L|$ 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 n-1 compares)

An Example of Quick-Select

Quick-Select (A, k)
Pick pivot $p \in A$ uniformly at random.
$(L, M, R) \leftarrow$ Partition-3-Way (A, p).
if $k \leq |L|$ Return Quick-Select (L, k).
else if $k > |L| + |M|$ Return Quick-Select $(R, k - |L| - |M|)$.
else Return p.

Example: select the 8-th smallest element

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
65	28	59	33	21	56	22	95	50	12	90	53	28	77	39

k = 8^{th} smallest

An Example of Quick-Select

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.

select the $k = 8^{th}$ smallest

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
65	28	59	33	21	56	22	95	50	12	90	53	28	77	39

$k = 8^{th}$ smallest

An Example of Quick-Select

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.

choose a pivot element at random and partition

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
65	28	59	33	21	56	22	95	50	12	90	53	28	77	39

$k = 8^{th}$ smallest

An Example of Quick-Select

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.

partitioned array

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
28	33	21	56	22	50	12	53	28	39	59	65	95	90	77

k = 8^{th} smallest

An Example of Quick-Select

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.

recursively select 8th smallest element in left subarray

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
28	33	21	56	22	50	12	53	28	39	59	65	95	90	77

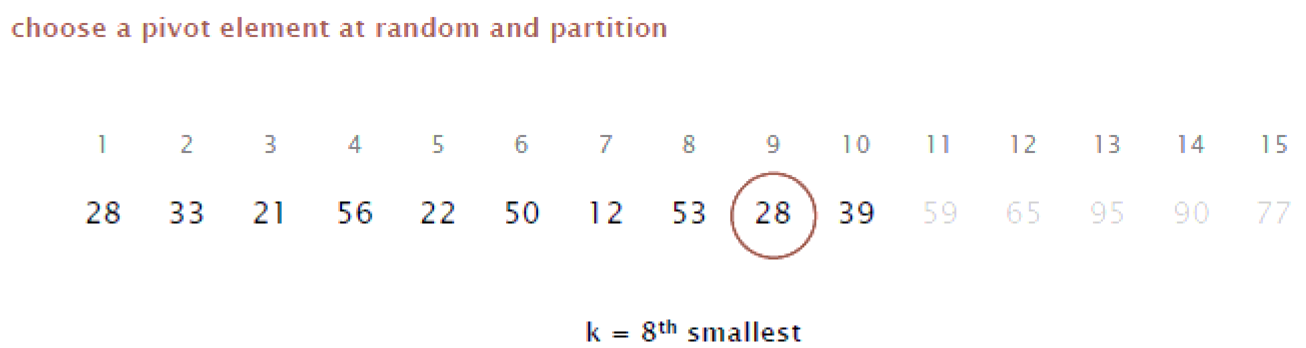
k = 8th smallest

An Example of Quick-Select

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.



An Example of Quick-Select

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.

partitioned array

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
21	22	12	28	28	33	56	50	53	39	59	65	95	90	77

k = 8^{th} smallest

An Example of Quick-Select

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.

recursively select the 3rd smallest element in right subarray

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
21	22	12	28	28	33	56	50	53	39	59	65	95	90	77

k = 3rd smallest

An Example of Quick-Select

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.

choose a pivot element at random and partition

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
21	22	12	28	28	33	56	50	53	39	59	65	95	90	77

k = 3^{rd} smallest

An Example of Quick-Select

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.

partitioned array

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
21	22	12	28	28	33	39	50	53	56	59	65	95	90	77

k = 3rd smallest

An Example of Quick-Select

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.

stop: desired element is in middle subarray

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
21	22	12	28	28	33	39	50	53	56	59	65	95	90	77

Quick-Select Analysis

Intuition. Split candy bar uniformly ➡ expected size of larger piece is ?

Quick-Select Analysis

Intuition. Split candy bar uniformly ➡ expected size of larger piece is ¾.

$$T(n) \le T\left(\frac{3}{4}n\right) + ?$$

Quick-Select Analysis

Intuition. Split candy bar uniformly ➡ expected size of larger piece is ¾.

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

Quick-Select Analysis

Intuition. Split candy bar uniformly ➡ expected size of larger piece is ¾.

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

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

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

Quick-Select Analysis

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

Pf.

- Assume true for 1,2,…,n-1.
- $T(n)$ satisfies for the following recurrence:

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

Quick-Select Analysis

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

Pf.

- Assume true for 1,2,…,n-1.
- $T(n)$ satisfies for the following recurrence:

$$
\begin{aligned}
T(n) &\leq n + \frac{2}{n}[T\left(\frac{n}{2}\right) + \cdots + T(n-3) + T(n-2) + T(n-1)] \\
&\leq n + \frac{2}{n}[\frac{4n}{2} + \cdots + 4(n-3) + 4(n-2) + 4(n-1)] \\
&\leq n + 4(\frac{3n}{4}) \\
&= 4n.
\end{aligned}
$$

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.

How to find approximate median in linear time?

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.

How to find approximate median in linear time?

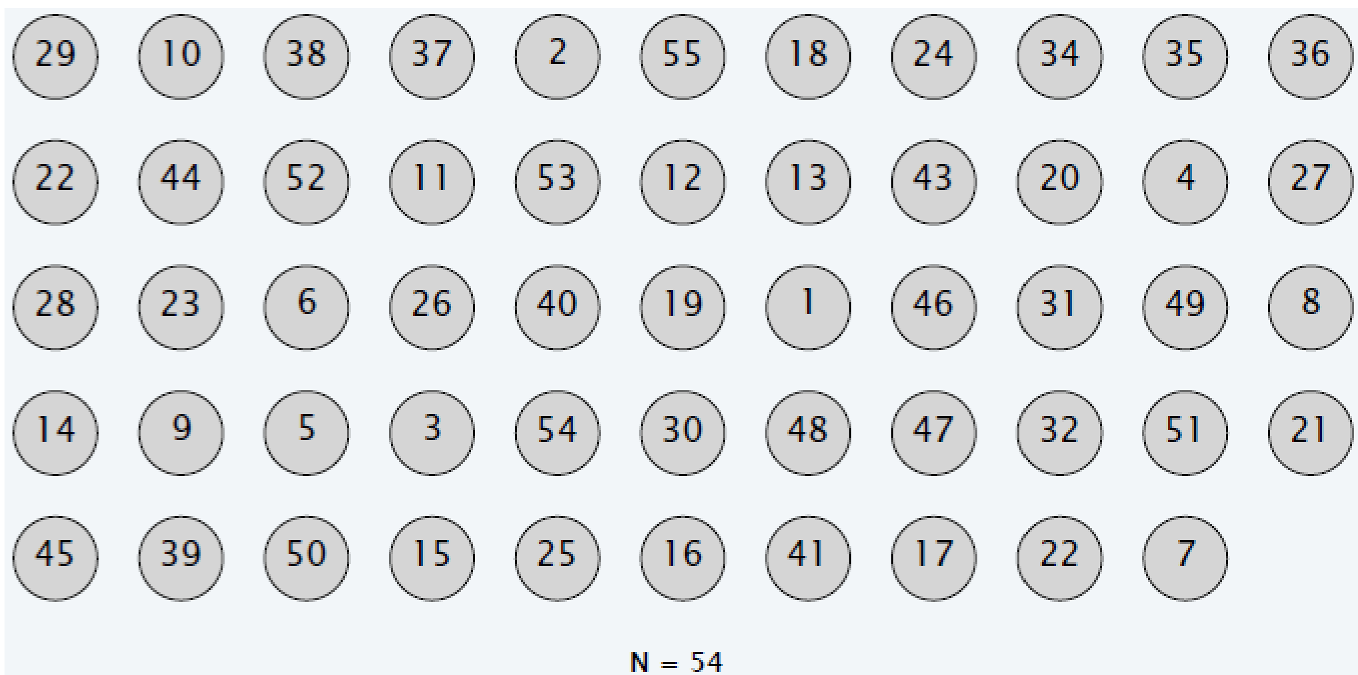
Recursively compute median of $\leq \frac{2}{10}n$ elements.

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

two sub-problems of
different sizes

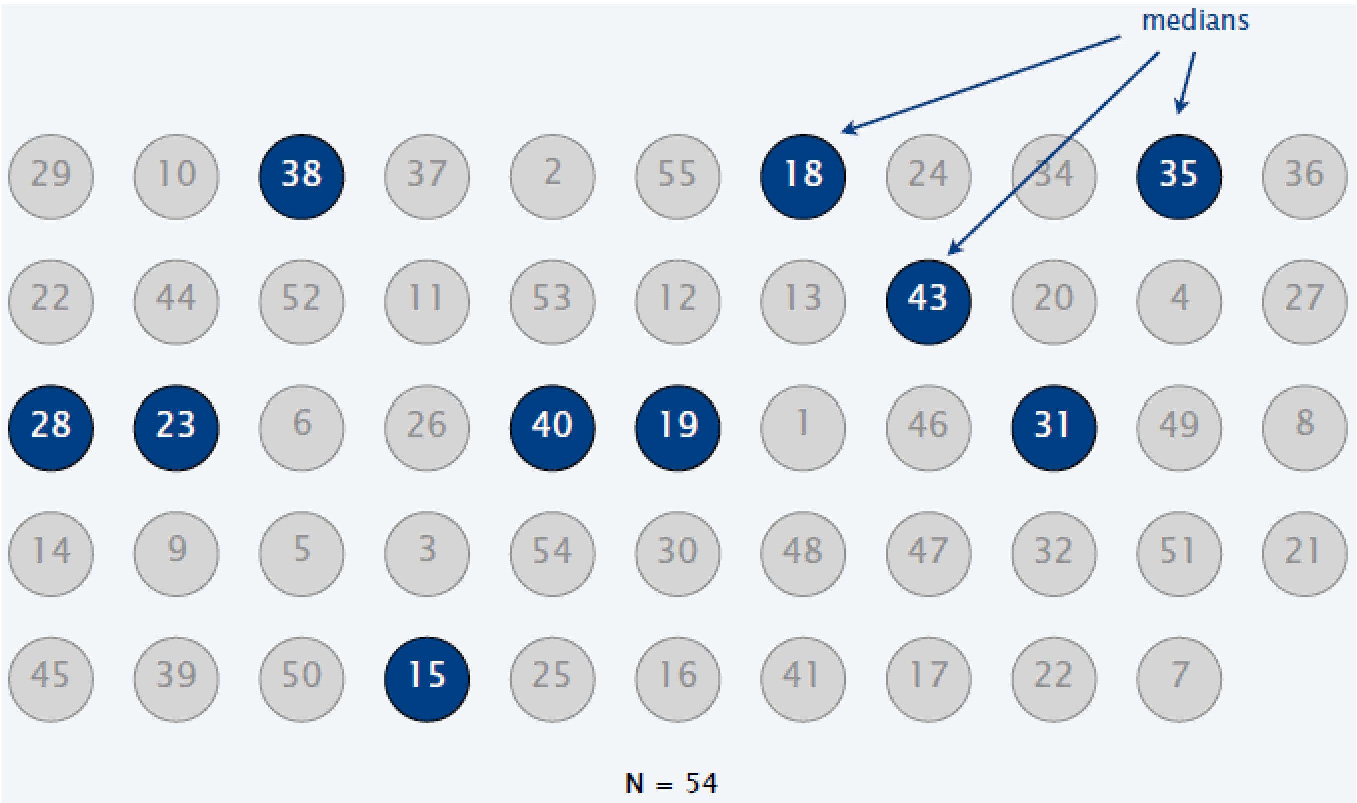
Choosing the Pivot Element

- Divide n elements into $\lfloor n/5 \rfloor$ groups of 5 elements each.



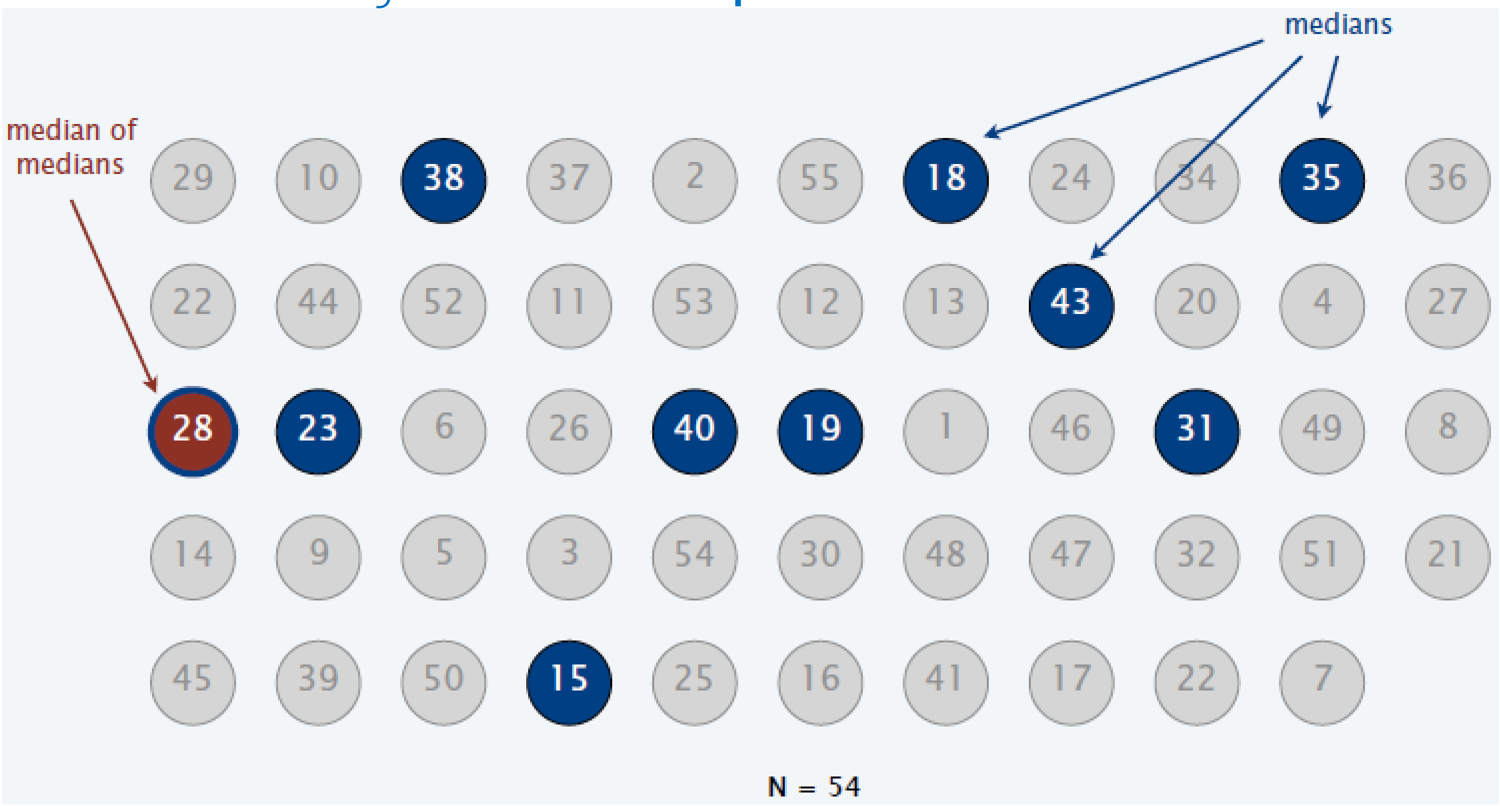
Choosing the Pivot Element

- Divide n elements into $\lfloor n/5 \rfloor$ groups of 5 elements each.
- Find median of each group.



Choosing the Pivot Element

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



Median-of-Medians Selection Algorithm

MoM-Select (A, k)

$n \leftarrow |A|$.

if $n < 50$ **Return** k-th smallest of element of A via Merge-Sort.

Group A into $\lceil n/5 \rceil$ groups of 5 elements each.

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

$p \leftarrow$ MoM-Select $(\mathrm{B}, \lfloor n/10 \rfloor)$. ← median of medians

$(L, M, R) \leftarrow$ Partition-3-Way (A, p).

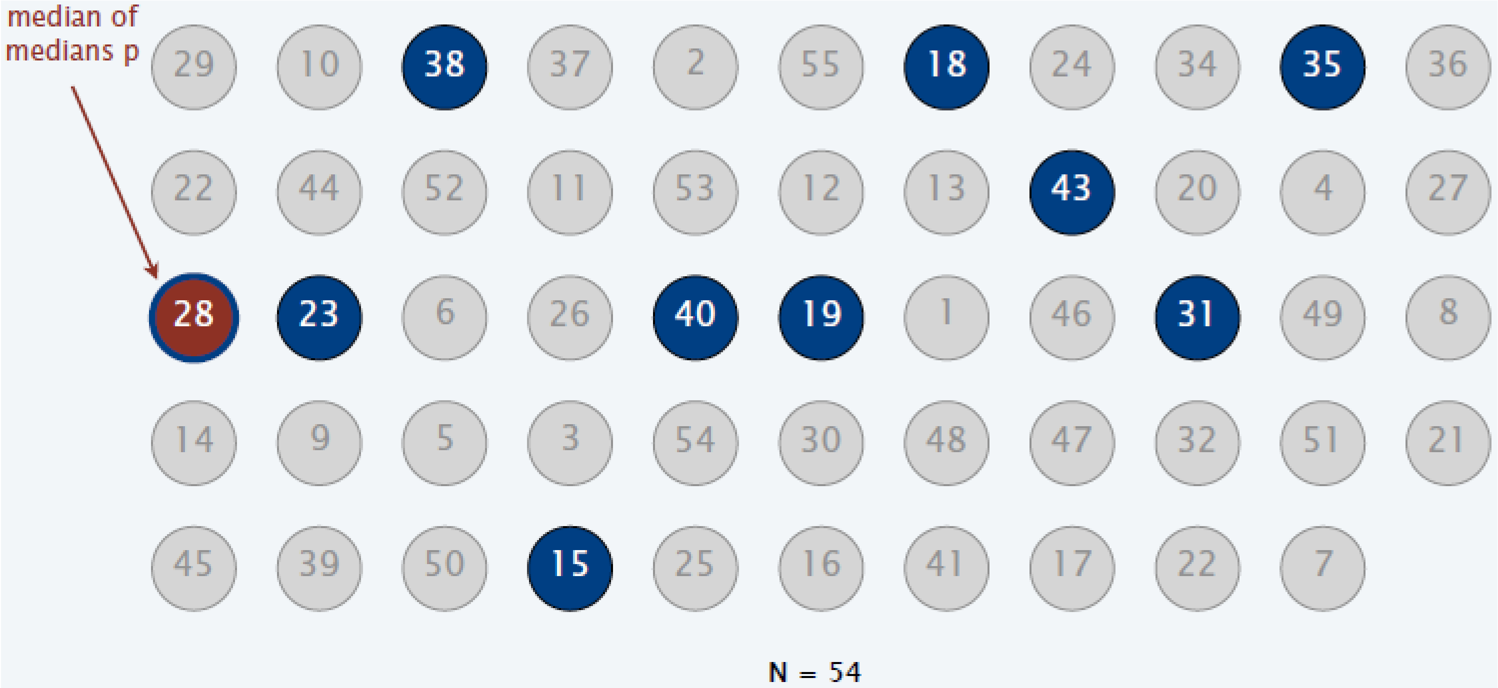
if $k \le |L|$ Return MoM-Select (L, k).

else if $k > |L| + |M|$ Return MoM-Select $(R, k - |L| - |M|)$.

else Return p.

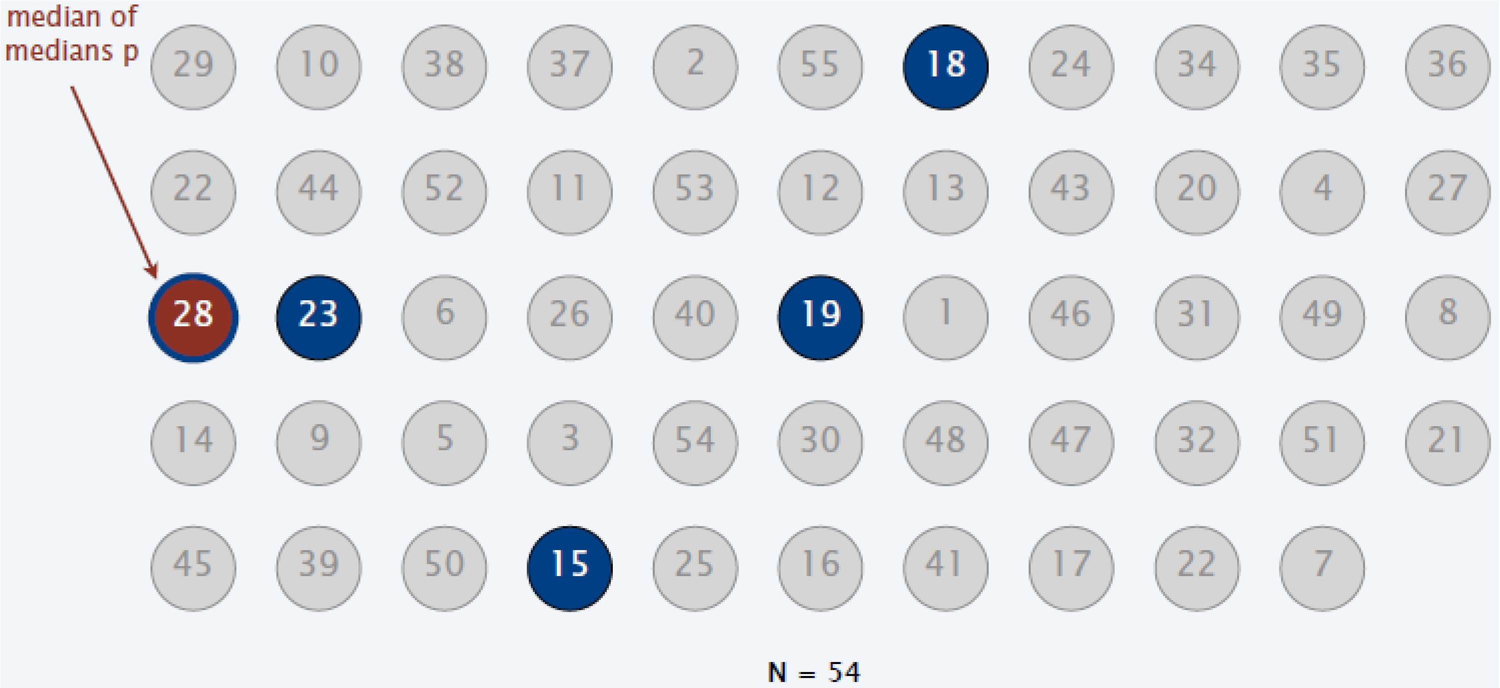
Analysis of Median-of-Medians Selection Algorithm

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



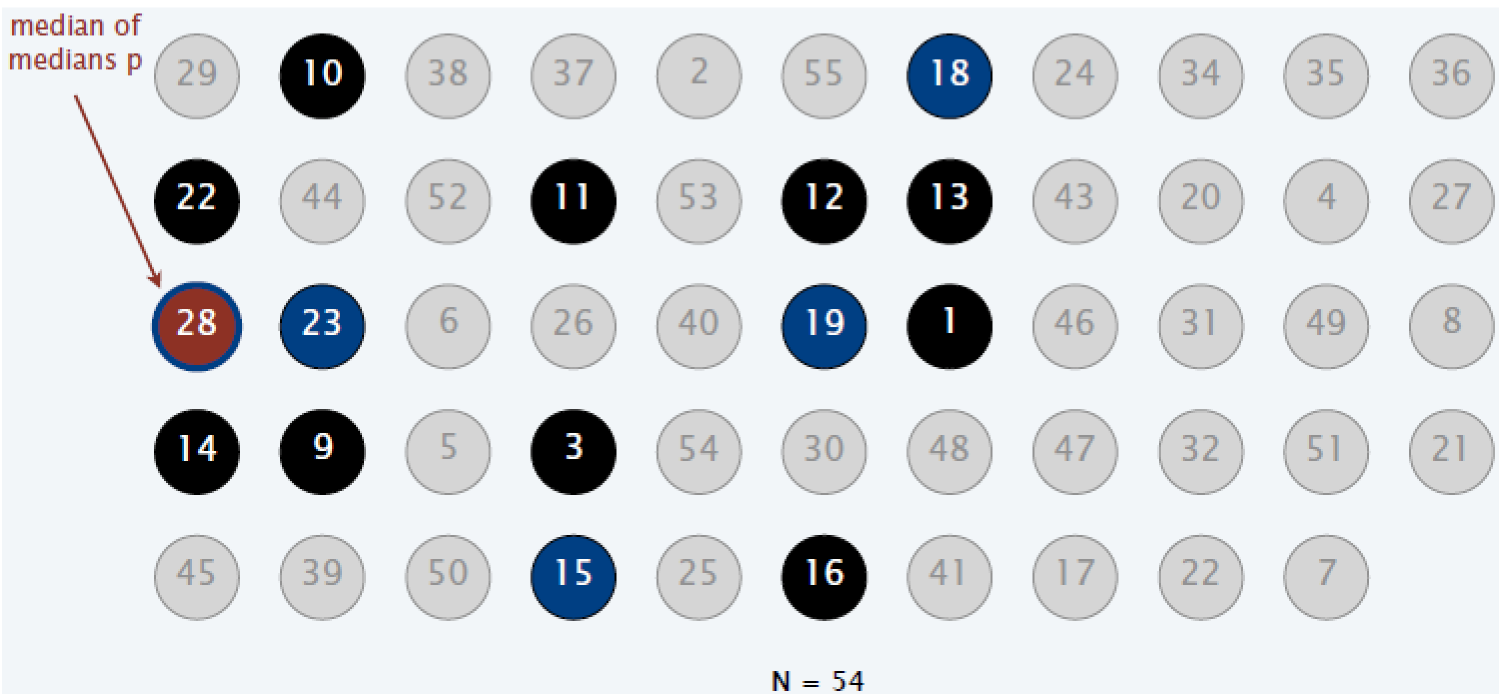
Analysis of Median-of-Medians Selection Algorithm

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



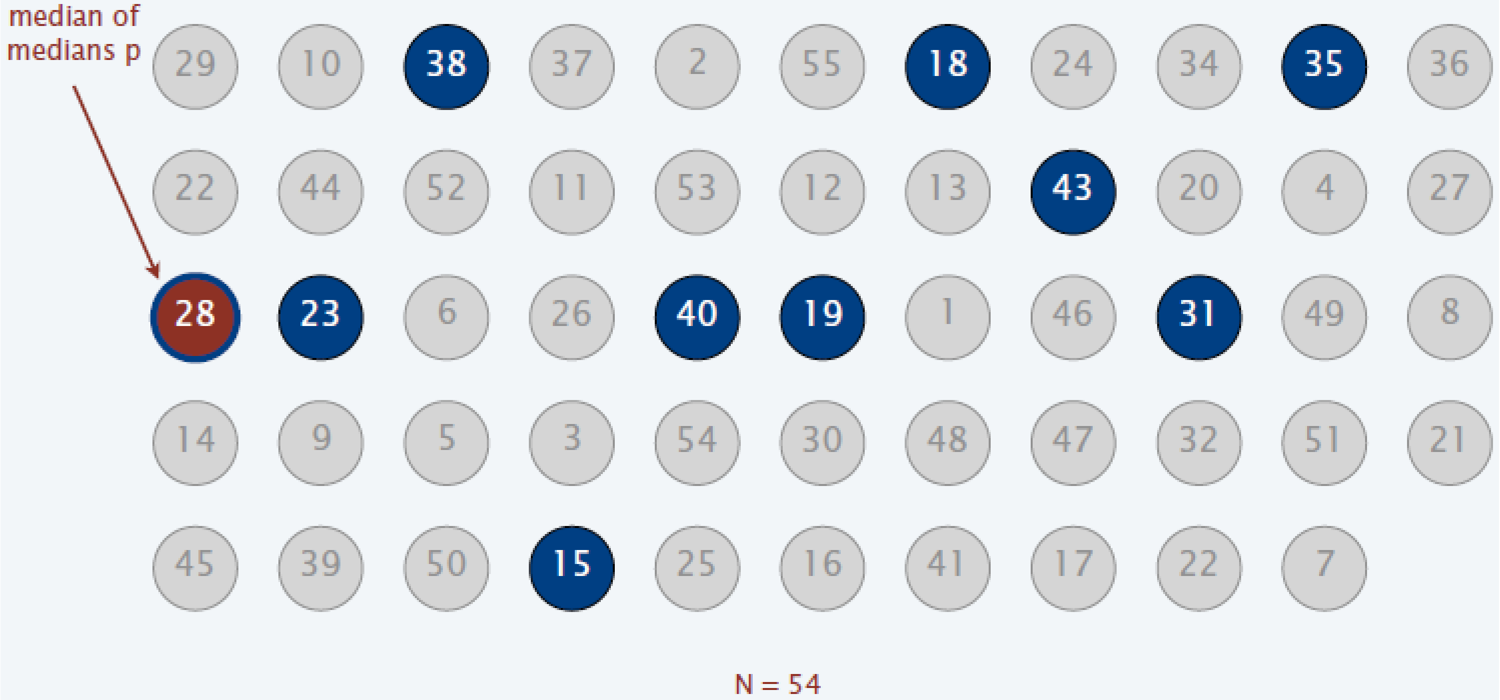
Analysis of Median-of-Medians Selection Algorithm

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



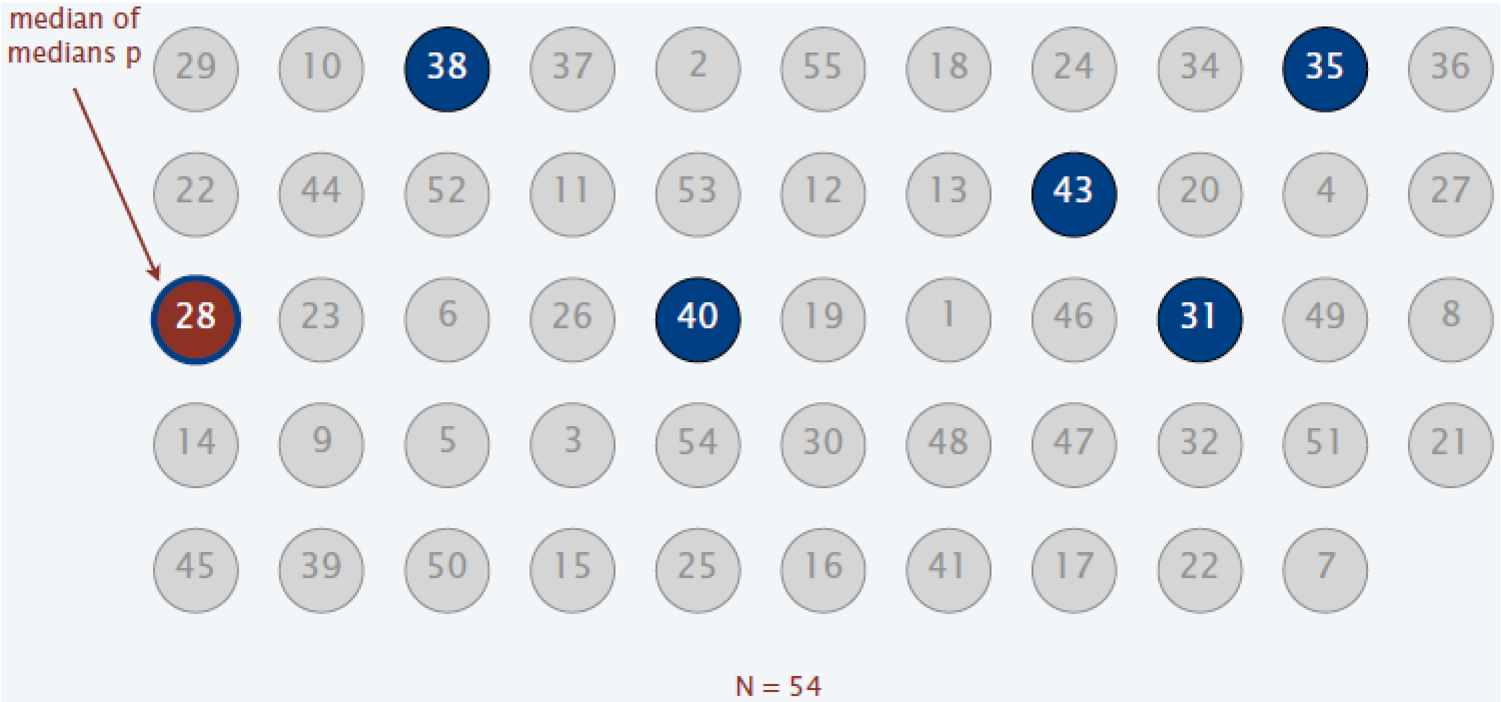
Analysis of Median-of-Medians Selection Algorithm

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



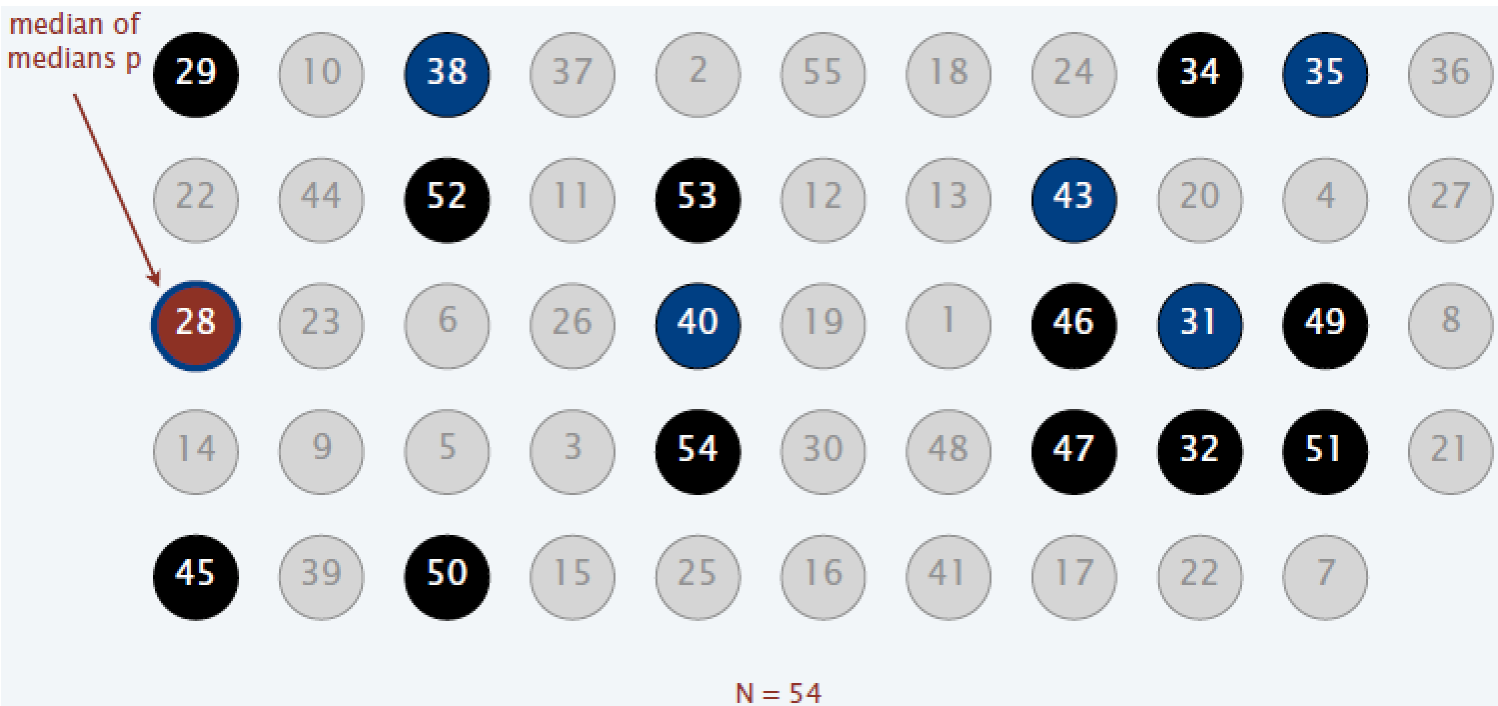
Analysis of Median-of-Medians Selection Algorithm

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

Def. $T(n) =$ max # compares on an array of $\leq n$ elements.

$$T(n) \leq \begin{cases} 6n, & if\ n < 50 \\ T\left(\left\lfloor \frac{n}{5} \right\rfloor\right) + T\left(n - 3\left\lfloor \frac{n}{10} \right\rfloor\right) + \frac{11}{5}n, & otherwise \end{cases}$$

median of medians

recursive select

computing median of 5 (6 compares per group)
partitioning (n compares)

Median-of-Medians Selection Algorithm Recurrence

$$T(n) \le \begin{cases} 6n, & if\ n < 50 \\ T\left(\left\lfloor \frac{n}{5} \right\rfloor\right) + T\left(n - 3\left\lfloor \frac{n}{10} \right\rfloor\right) + \frac{11}{5}n, & otherwise \end{cases}$$

Claim. $T(n) \le 44n$.

- Base case: $T(n) \le 6n$ for $n < 50$ (Merge-Sort).
- Inductive hypothesis: assume true for 1,2,…, n-1.
- Inductive step: for $n \ge 50$, we have:

$$\begin{aligned} T(n) &\le T\left(\left\lfloor \frac{n}{5} \right\rfloor\right) + T\left(n - 3\left\lfloor \frac{n}{10} \right\rfloor\right) + \frac{11}{5}n \\ &\le 44\left(\left\lfloor \frac{n}{5} \right\rfloor\right) + 44\left(n - 3\left\lfloor \frac{n}{10} \right\rfloor\right) + \frac{11}{5}n \\ &\le 44\left(\frac{n}{5}\right) + 44n - 44\left(\frac{n}{4}\right) + \frac{11}{5}n \\ &= 44n. \end{aligned}$$

for $n \ge 50, 3\left\lfloor \frac{n}{10} \right\rfloor \ge n/4$