# ALGORITHMS & ADVANCED DATA STRUCTURES (#7)

**ORDER STATISTICS**

ADAPTED FROM
CS 146 SJSU (KATERINA POTIKA)

# Order Statistics



- ***i*- th *order statistic*: *i*- th smallest element** in a set of $n$ elements
- 1-st order statistic: minimum
- $n$- th order statistic: maximum
- $n/2$ order statistic: median
  - When $n$ is odd, the median is unique, at $i = (n + 1)/2$.
  - When $n$ is even, there are two medians: ***lower median***, at $i = n/2$, and ***upper median***, at $i = n/2 + 1$.
- “the median: lower median!
- *How can we calculate order statistics? What is the running time?*

# Order Statistics – simple cases



- *How many comparisons are needed to find the minimum element in a set? The maximum?*
- *Can we find the minimum and maximum with less than twice the cost?*
- Yes:
  Walk through elements by pairs
  Compare each element in pair to the other
  Compare the largest to maximum, smallest to minimum
  Total cost: 3 comparisons per 2 elements = O(3n/2)

# Selection Problem



- Selection problem:

  Input: A set $A$ of $n$ **distinct** numbers and a number $i$, with $1 \le i \le n$.

  Output: the element $x \in A$ that is larger than exactly $i - 1$ other elements of $A$.
- Can be solved in $O(n \lg n)$ time. How?
- We will study faster linear-time algorithms.
  - For the special cases when $i = 1$ and $i = n$.
  - For the general problem.

# Finding Order Statistics: The Selection Problem



- First, present a randomized algorithm, runs in $O(n^2)$ worst case and O(n) average case
- An algorithm of theoretical interest only with O(n) worst-case running time (why?)

# Selection in Expected Linear Time



- Modeled after randomized quicksort.
- Uses Randomized-Partition (RP).
  - RP returns the index $k$ of a randomly chosen element (pivot).
    - If the order statistic we are interested in, $i$ equals $k$, then we are done.
    - Else, reduce the problem size using its other ability.
  - RP rearranges the other elements around the random pivot.
    - If $i < k$, selection can be narrowed down to $A[1..k-1]$.
    - Else, select the $(i-k)$th element from $A[k+1..n]$.

  (Assuming RP operates on $A[1..n]$. For A$[p..r]$, change $k$ appropriately.)

# Randomized Select Θ(n) expected time



Randomized-Select($A, p, r, i$) // select $i$-th order statistic.

1. **if** $p = r$ *//base case*
2. **then return** $A[p]$
3. $q \leftarrow$ Randomized-Partition($A, p, r$) *// index of pivot*
4. $k \leftarrow q - p + 1$ //order of pivot
5. **if** $i = k$ *//if the pivot is i-th order statistic*
6. **then** return $A[q]$
7. **elseif** $i < k$ *// ignore bigger else smaller than pivot*
8. **then return** Randomized-Select($A, p, q - 1, i$)
9. **else return** Randomized-Select($A, q+1, r, \boldsymbol{i - k}$)

9. **//look for (i-k)-th since k smallest removed**

# Analysis of RS



- Worst-case Complexity:
  - $\Theta(n^2)$ – As we could get unlucky and always recurse on a subarray that is only one element smaller than the previous subarray.
- Average-case Complexity:
  - $\Theta(n)$ – Intuition: Because the pivot is chosen at random, we expect that we get rid of half of the list each time we choose a random pivot $q$.
  - Why $\Theta(n)$ and not $\Theta(n \lg n)$?

    recursion goes in only one of the two subarrays…

# Select in O(n) worst case



- SELECT recursively partitions the input array.
  • ***Idea:*** Guarantee a good split when the array is partitioned.
  •Use the *deterministic* procedure PARTITION, but with a small modification: Instead of assuming that the last element of the subarray is the pivot, the modified PARTITION procedure is told **which** element to use as the pivot.

# Choosing a Pivot



- Median-of-Medians:
  - Divide the $n$ elements into $\lceil n/5 \rceil$ groups.
    - $\lfloor n/5 \rfloor$ groups contain 5 elements each - 1 group contains $n \bmod 5 < 5$ elements.
    - Determine the median of each of the groups.
      - Sort each group using Insertion Sort. Pick the median from the sorted list of group elements.
    - Recursively find the median $x$ of the $\lceil n/5 \rceil$ medians.
- Recurrence for running time (of median-of-medians):
  - $T(n) = O(n) + T(\lceil n/5 \rceil) + \ldots.$

# Algorithm Select(A,p,r,i)

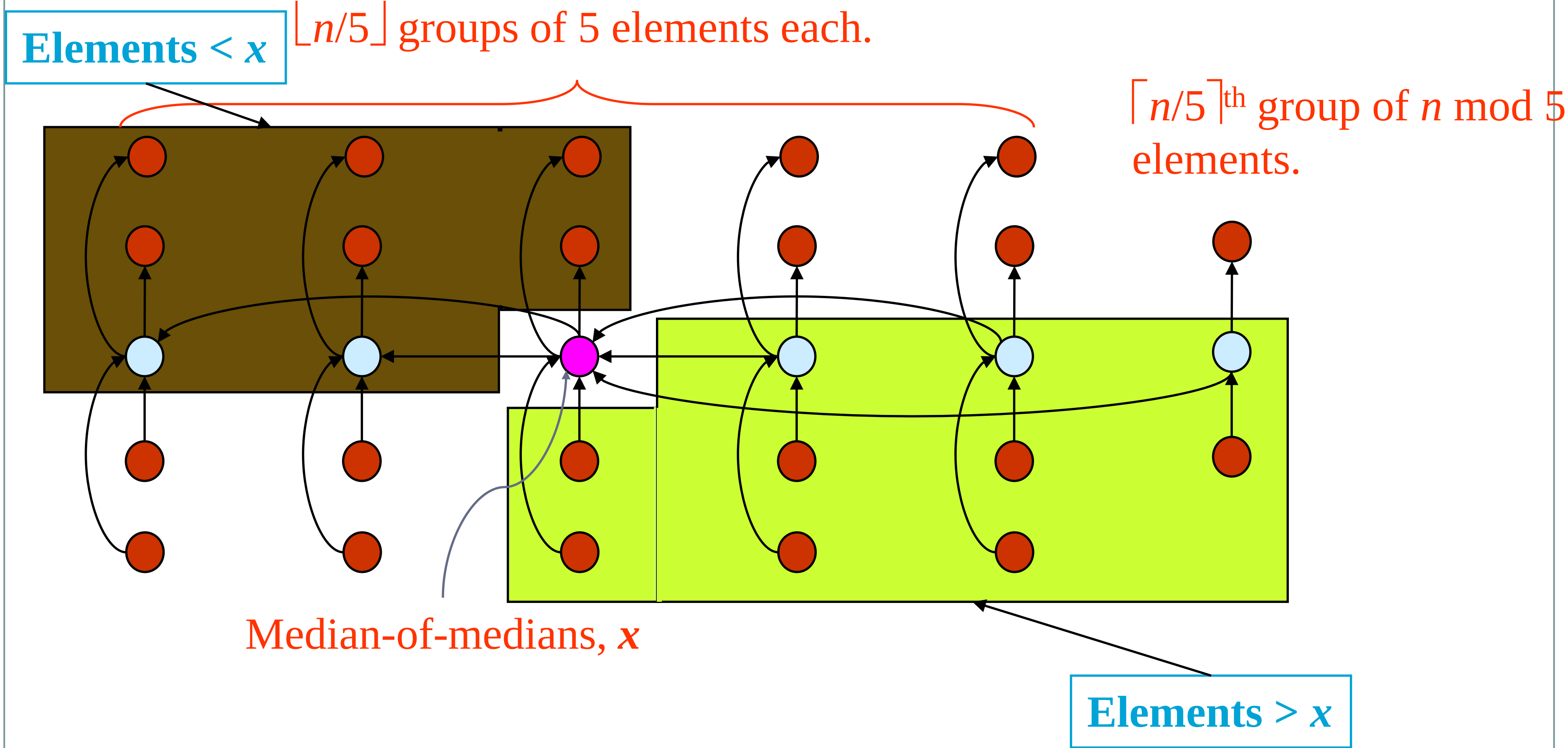


1. Determine the median-of-medians $x$ (using the procedure on the previous slide.)
2. Partition the input array around $x$ using the variant of Partition.
3. Let $k$ be the index of $x$ that Partition returns.
4. If $k = i$, then return $x$.
5. Else if $i < k$, then apply Select recursively to $A[1..k-1]$ to find the $i^{th}$ smallest element.
6. Else if $i > k$, then apply Select recursively to $A[k+1..n]$ to find the $(i-k)^{th}$ smallest element.

(Assumption: Select operates on $A[1..n]$. For subarrays $A[p..r]$, suitably change $k$. )

# Worst-case Split






Arrows point from larger to smaller elements.

# Worst-case Split



- Assumption: Elements are distinct.
- At least half of the $\lceil n/5 \rceil$ medians are greater than $x$.
- Thus, at least half of the $\lceil n/5 \rceil$ groups contribute 3 elements that are greater than $x$.
  - The last group and the group containing $x$ may contribute fewer than 3 elements. Exclude these groups.
- Hence, the no. of elements > $x$ is at least

$$3\left(\left\lceil \frac{1}{2}\left\lceil \frac{n}{5} \right\rceil \right\rceil - 2\right) \geq \frac{3n}{10} - 6$$

- Analogously, the no. of elements < $x$ is at least $3n/10-6$.
- Thus, in the worst case, Select is called recursively on at most $7n/10+6$ elements.

# Recurrence for worst-case running time



- $T$(Select) $\leq$ $T$(Median-of-medians) + $T$(Partition) + $T$(recursive call to select)

- $T(n) \leq O(n) + T(\lceil n/5 \rceil) + O(n) + T(7n/10+6)$

  $T$(Median-of-medians) — $O(n) + T(\lceil n/5 \rceil)$; $T$(Partition) — $O(n)$; $T$(recursive call) — $T(7n/10+6)$

  $$= T(\lceil n/5 \rceil) + T(7n/10+6) + O(n)$$

- Assume $T(n) \leq \Theta(1)$, for $n \leq 140$.

# Solving the recurrence



- To show: $T(n) = O(n) \leq cn$ for suitable $c$ and all $n > 0$.
- Assume: $T(n) \leq cn$ for suitable $c$ and all $n \leq 140$.
- Substituting the inductive hypothesis into the recurrence,
  - $T(n) \leq c\lceil n/5 \rceil + c(7n/10+6)+an$
    $\leq cn/5 + c + 7cn/10 + 6c + an$
    $= 9cn/10 + 7c + an$
    $= cn +(-cn/10 + 7c + an)$
    $\leq cn$, if $-cn/10 + 7c + an \leq 0$.

$-cn/10 + 7c + an \leq 0 \equiv c \geq 10a(n/(n - 70))$, when $n > 70$.

For $n \geq 140$, $c \geq 20a$.

- $n/(n-70)$ is a decreasing function of $n$. Verify.
- Hence, $c$ can be chosen for any $n = n_0 > 70$, provided it can be assumed that $T(n) = O(1)$ for $n \leq n_0$.
- Thus, Select has linear-time complexity in the worst case.
- In practice the constant is to large to be useful.

# Example of Selection Algorithm O(n) $7^{th}$ smallest



- 6, 10, 13, 5, 8, 3, 2, 11 *groups of 5 (only one)*
- 6, 10, 13, 5, 8 | 3, 2, 11 *find median it is 8*
- 6, 10, 13, 5, <u>8 |</u> 3, 2, 11 *swap 8 with 6 (A[0])*
- <u>8</u> | 10, 13, 5, 6 | 3, 2, 11 *pivot is 8, call partition on A*
- ~~5, 6, 3, 2,~~ <u>8,</u> 10, 13, 11 *8 is the $5^{th}$ smallest*
- 10, 13, 11 *look for 7-5=$2^{nd}$ smallest*
- 11, *is the $7^{th}$ since less than 5 (run insertion sort)*

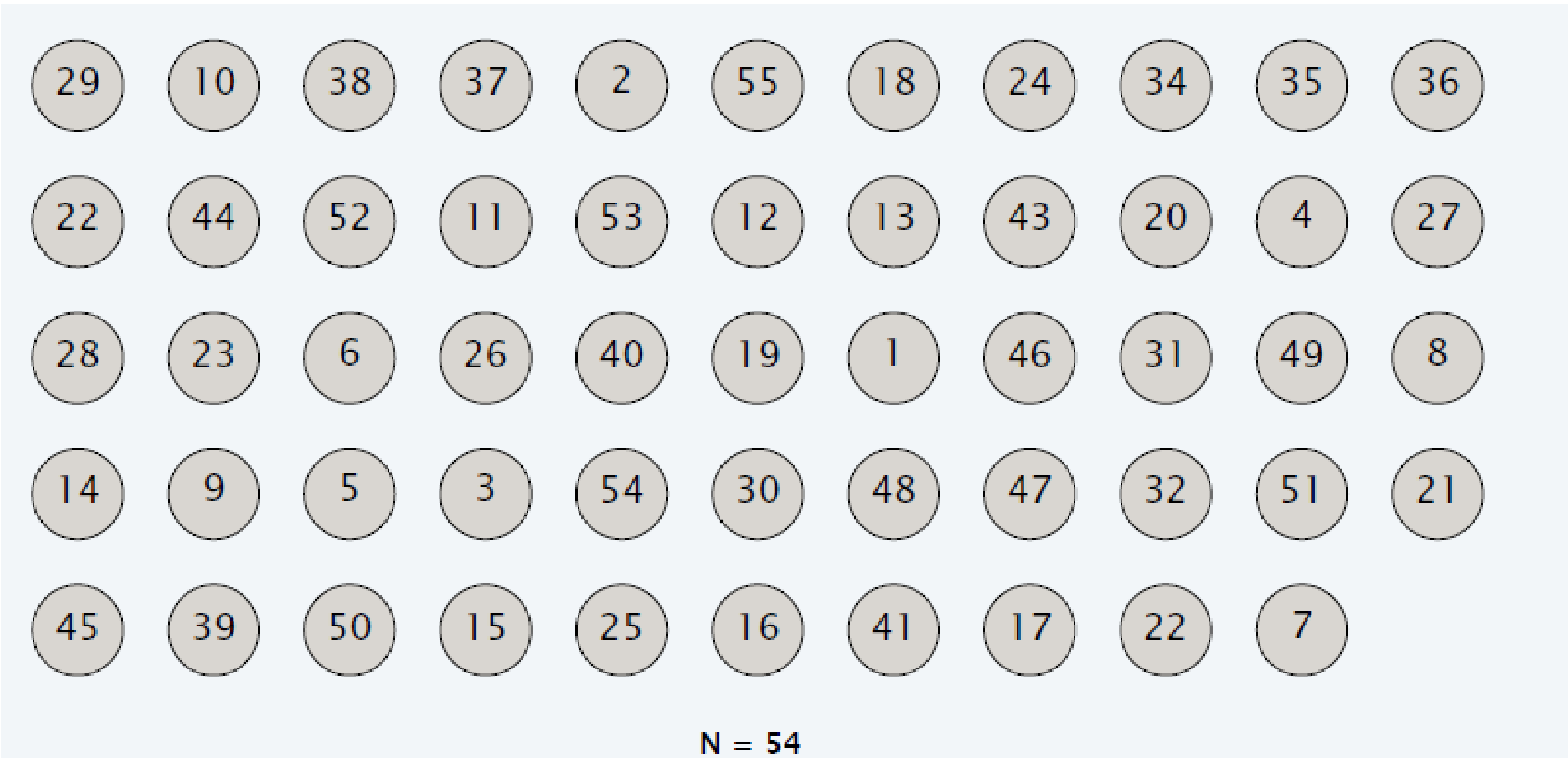
# Linear-Time Median Selection



- Given a "black box" $O(n)$ median algorithm, what can we do?
  - *i*th order statistic:
    - Find median $x$
    - Partition input around $x$
    - if ($i \le (n+1)/2$) recursively find *i*th element of first half
    - else find ($i - (n+1)/2$)th element in second half
    - $T(n) = T(n/2) + O(n) = O(n)$
  - *Can you think of an application to sorting?*

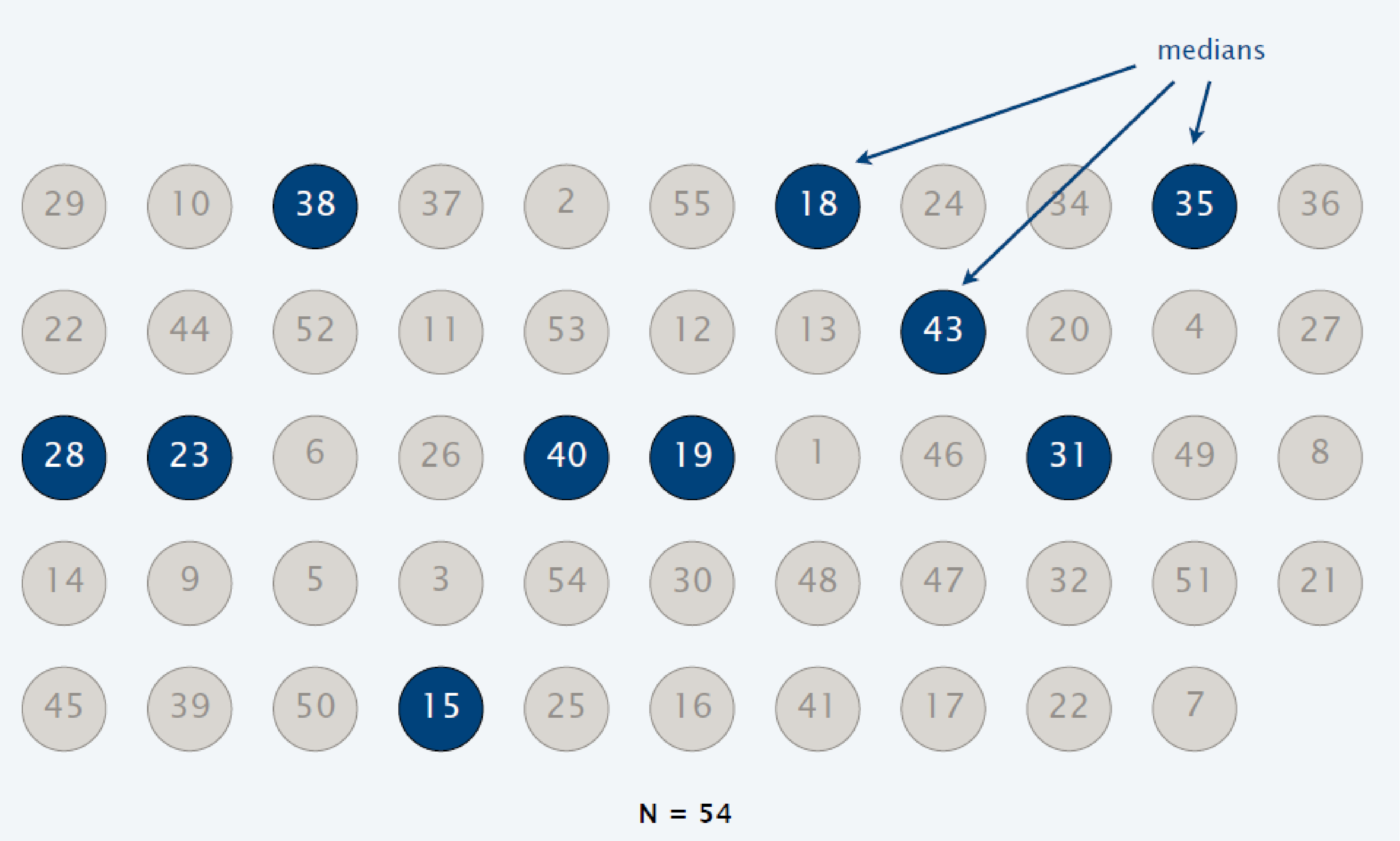
# Example




29 10 38 37 2 55 18 24 34 35 36
22 44 52 11 53 12 13 43 20 4 27
28 23 6 26 40 19 1 46 31 49 8
14 9 5 3 54 30 48 47 32 51 21
45 39 50 15 25 16 41 17 22 7
N = 54

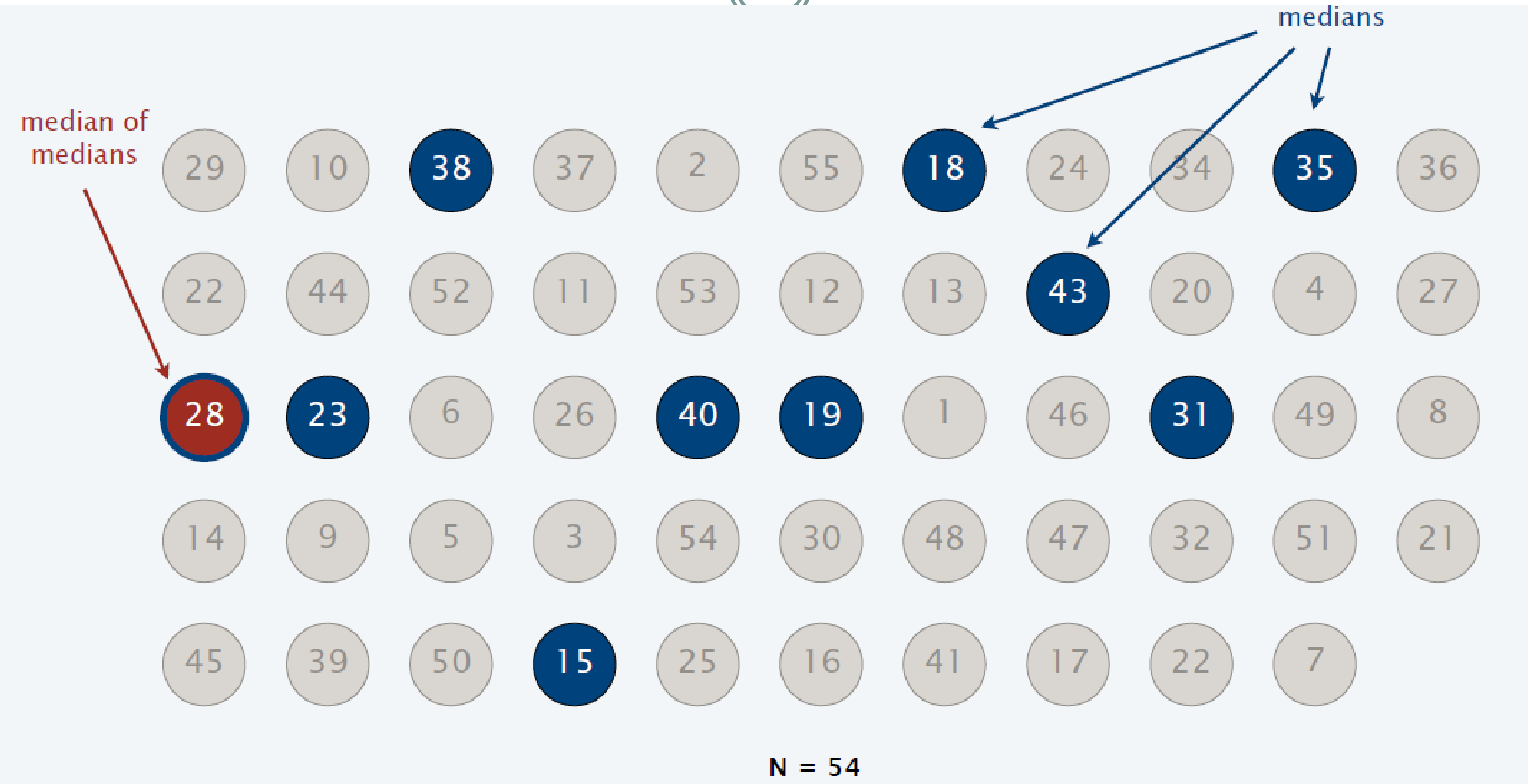
# Medians of 5




medians
29
10
38
37
2
55
18
24
34
35
36
22
44
52
11
53
12
13
43
20
4
27
28
23
6
26
40
19
1
46
31
49
8
14
9
5
3
54
30
48
47
32
51
21
45
39
50
15
25
16
41
17
22
7
N = 54

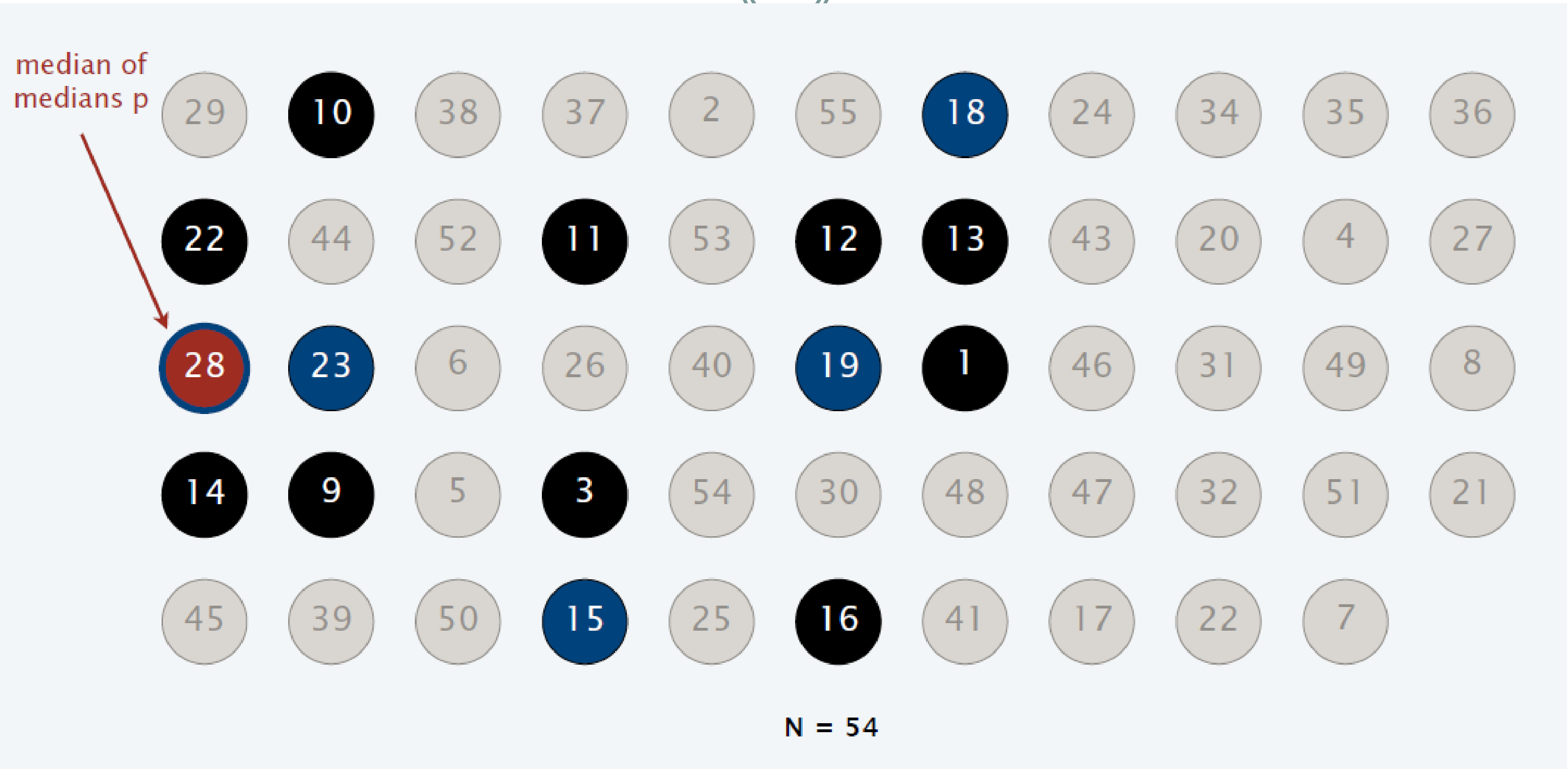
# Median of Medians





# Smaller than median of medians





# Larger than median of medians