Quicksort

Gou Guanglei(苟光磊)
ggl@cqut.edu.cn

Quicksort

- Proposed by C.A.R. Hoare in 1962.
- Divide-and-conquer algorithm.
- Sorts “in place” (pivot)
- Very practical (with tuning)

Divide and conquer

1. **Divide:** Partition the array into two subarrays around a ***pivot*** x such that elements in lower subarray ≤ x ≤ elements in upper subarray.

2. **Conquer:** Recursively sort the two subarrays.

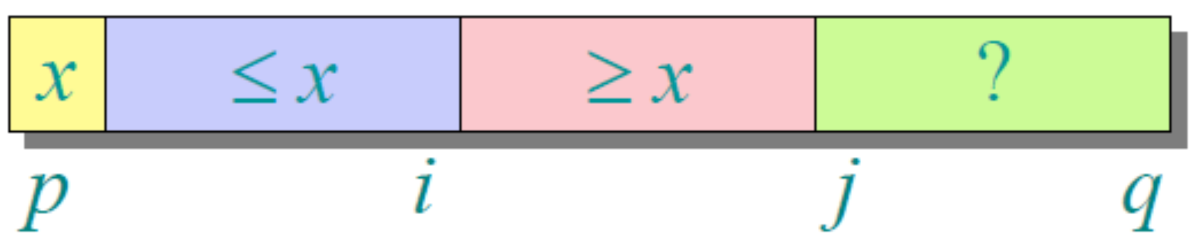
3. **Combine:** Trivial.

- **Partition**

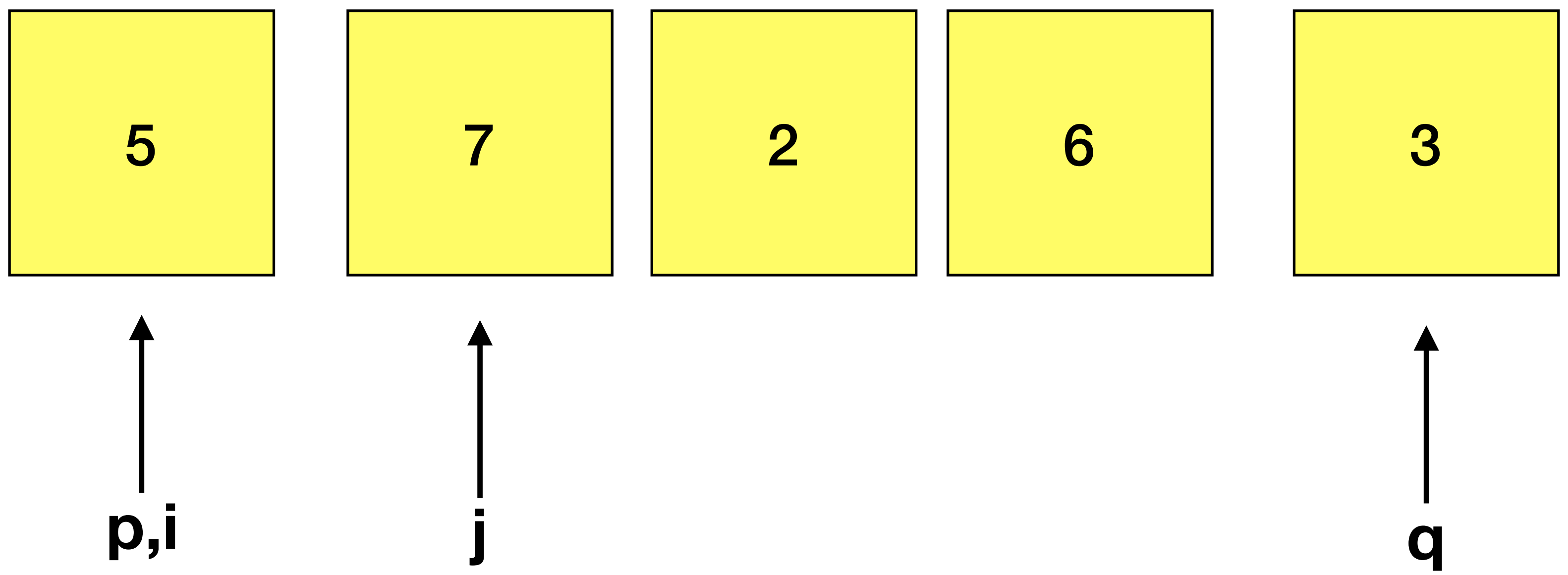
```
PARTITION(A, p, q)  ▷ A[p . . q]
    x ← A[p]        ▷ pivot = A[p]
    i ← p
    for j ← p + 1 to q
        do if A[j] ≤ x
              then  i ← i + 1
                    exchange A[i] ↔ A[j]
    exchange A[p] ↔ A[i]
    return i
```

Running time $= O(n)$ for n elements.

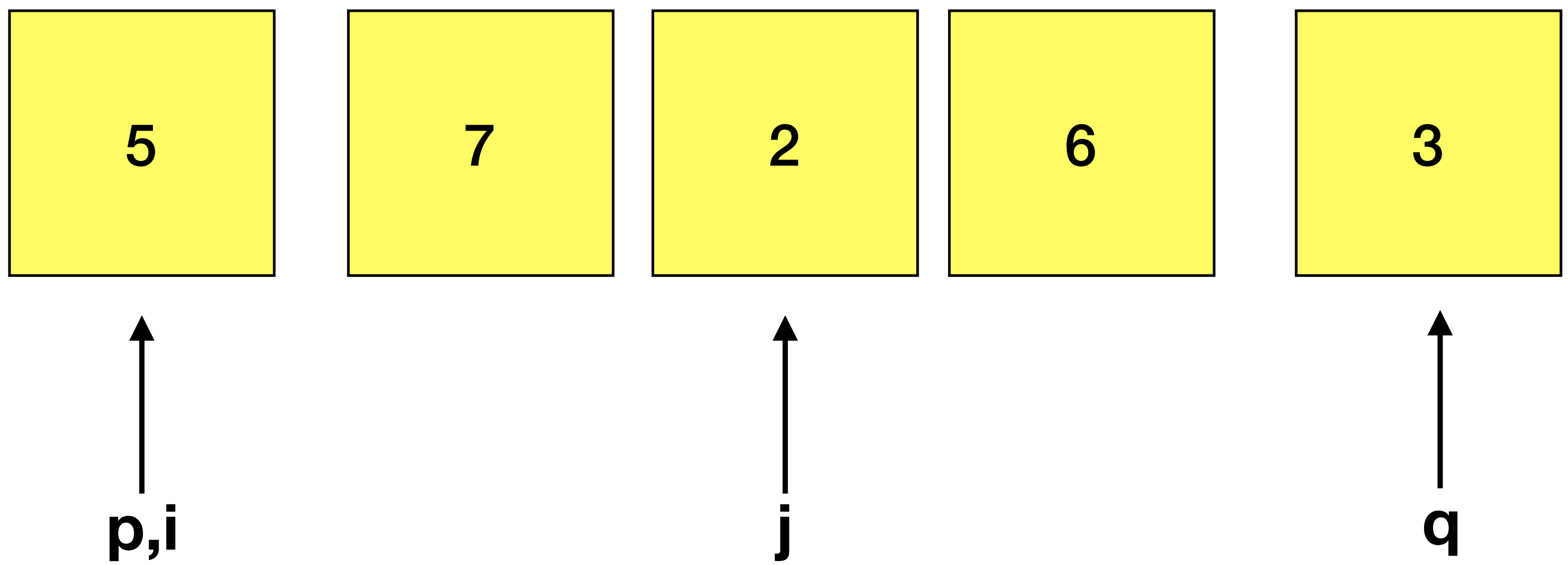
Invariant:



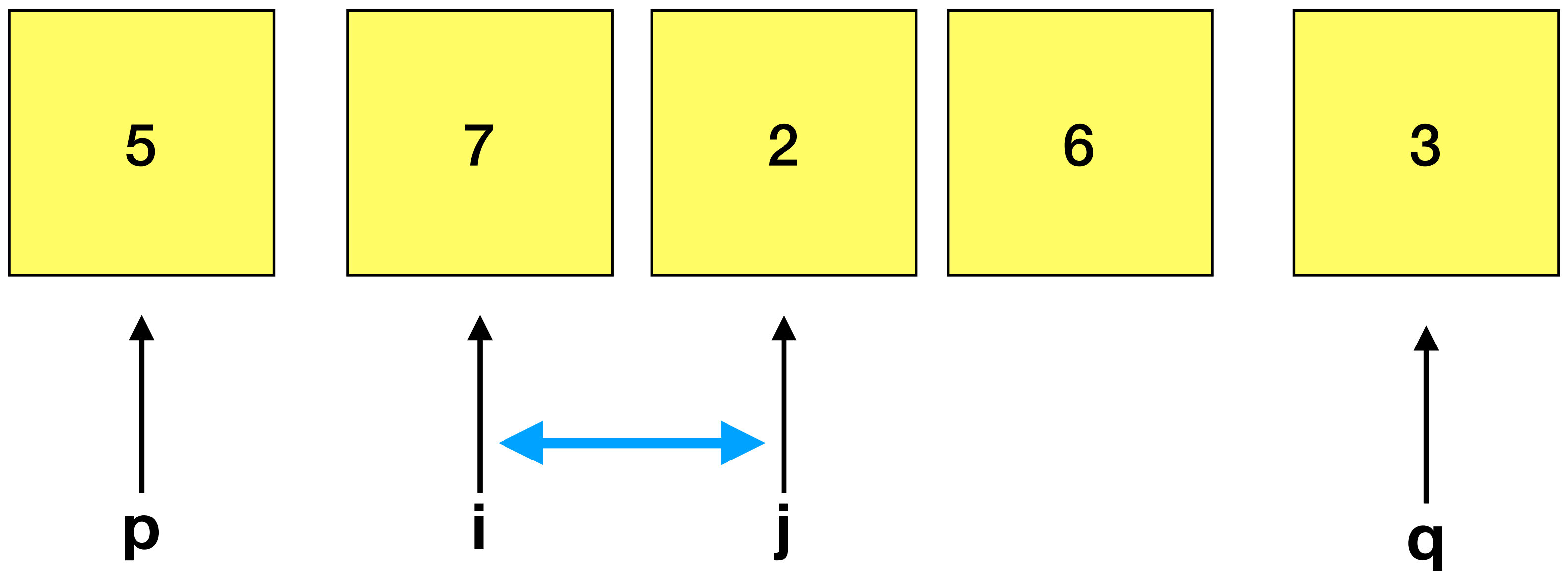
Key:Linear-time partitioning subroutine



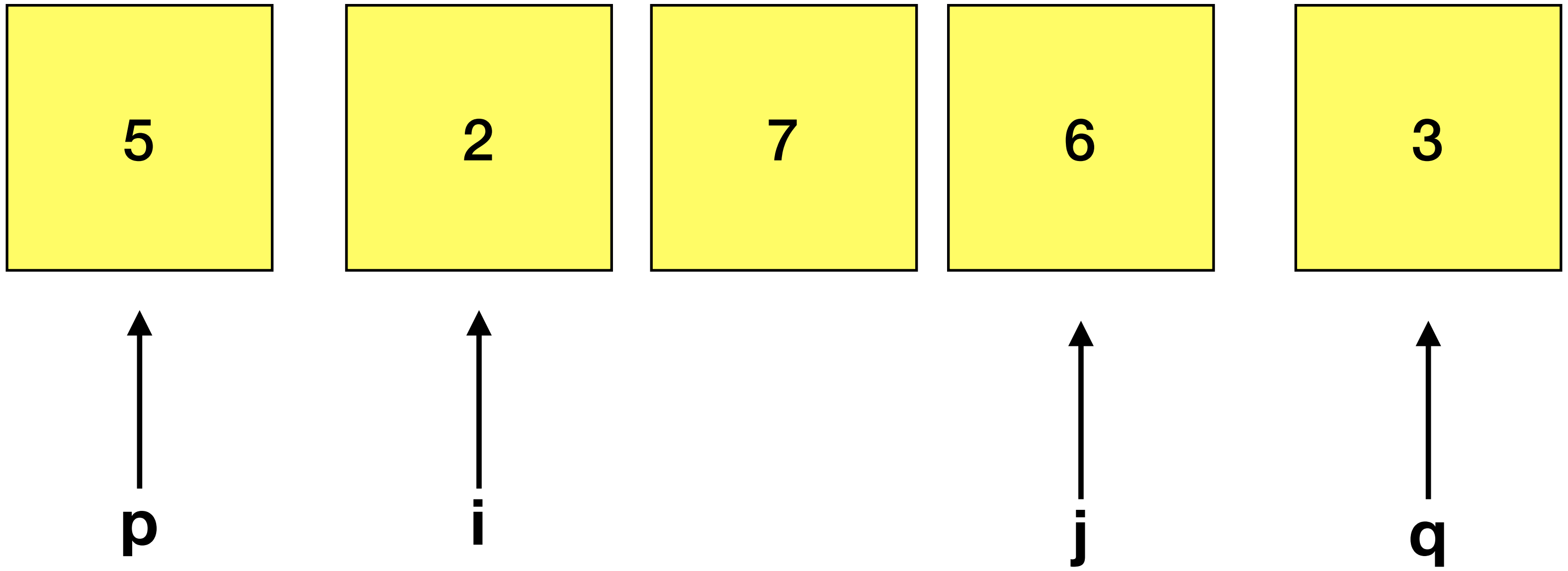

```
x = 5
p = i
for j=p+1 -> q
  if A[j] <= x
    i = i+1
    A[i] <-> A[j]
A[p] = A[i]
```



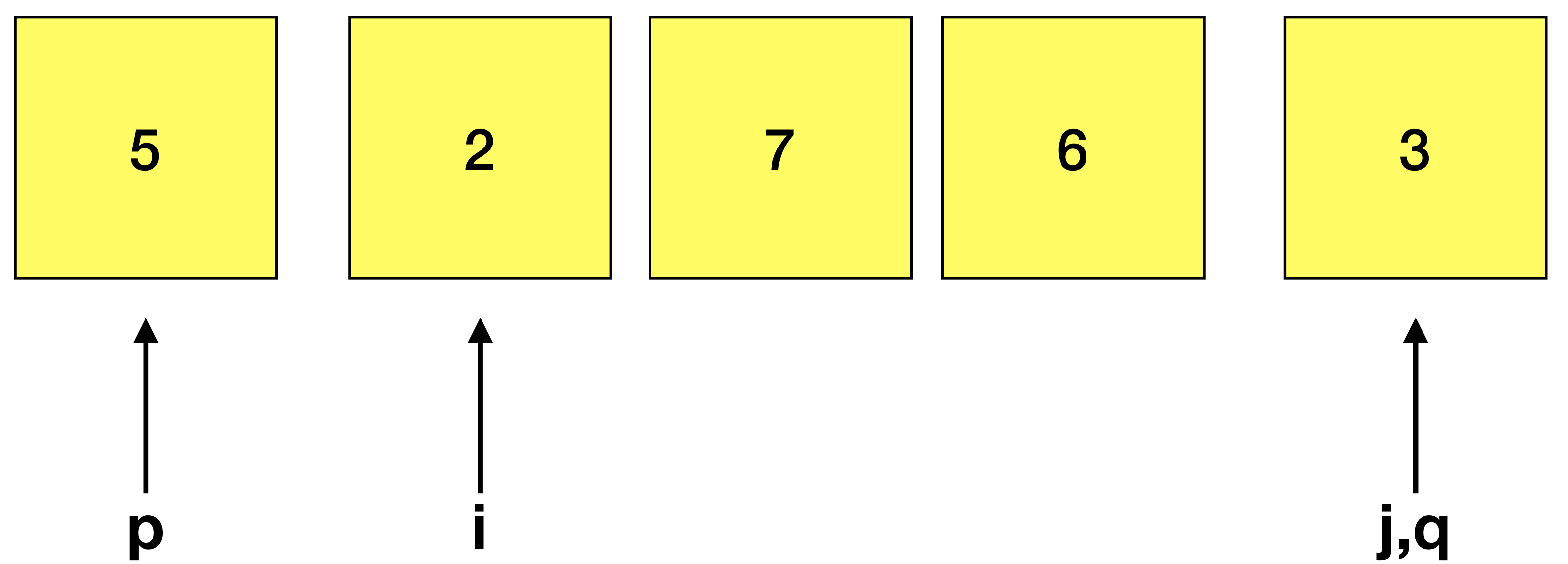

```
x = 5
p = i
for j=p+1 -> q
  if A[j] <= x
    i = i+1
    A[i] <-> A[j]
A[p] = A[i]
```



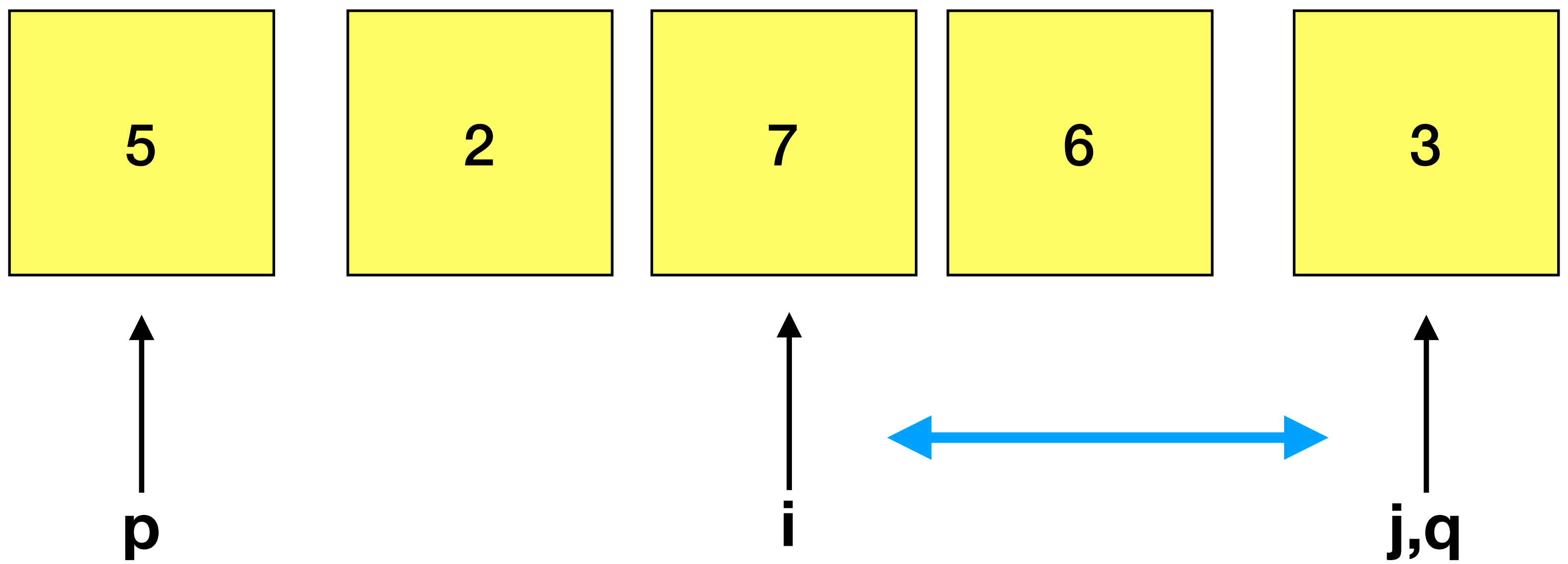

```
x = 5
p = i
for j=p+1 -> q
  if A[j] <= x
    i = i+1
    A[i] <-> A[j]
A[p] = A[i]
```



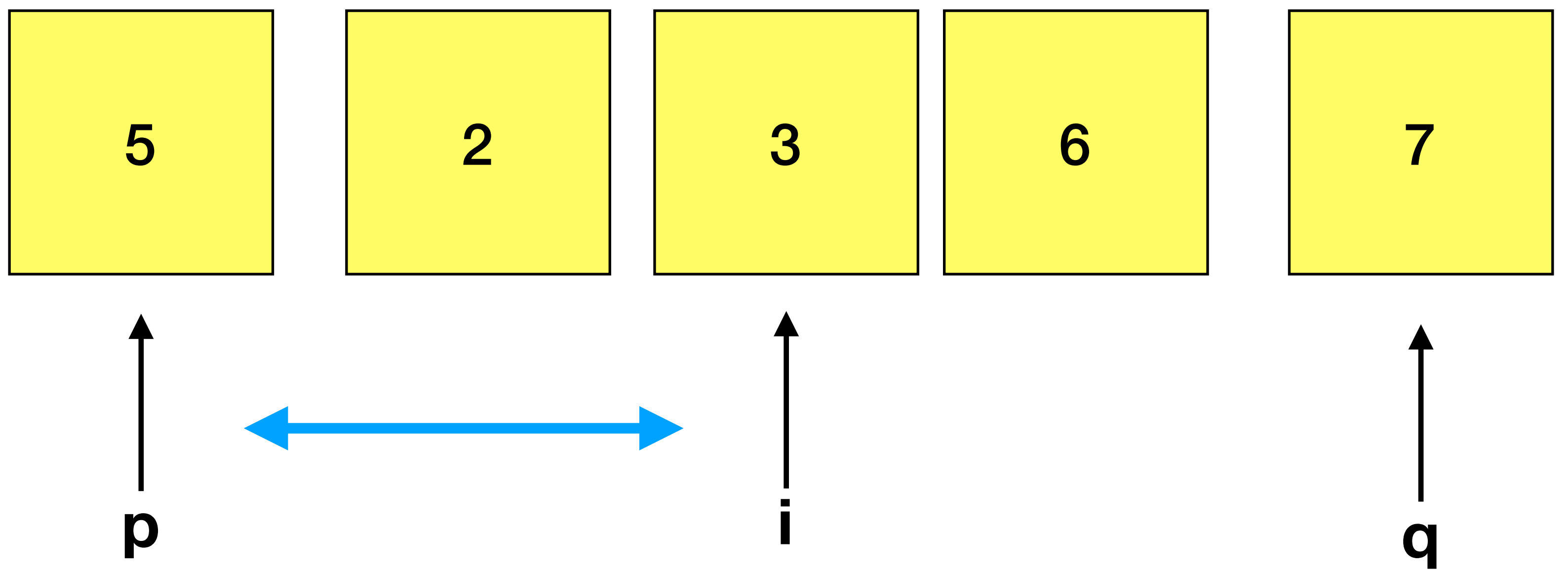

```
x = 5
p = i
for j=p+1 -> q
  if A[j] <= x
    i = i+1
    A[i] <-> A[j]
A[p] = A[i]
```



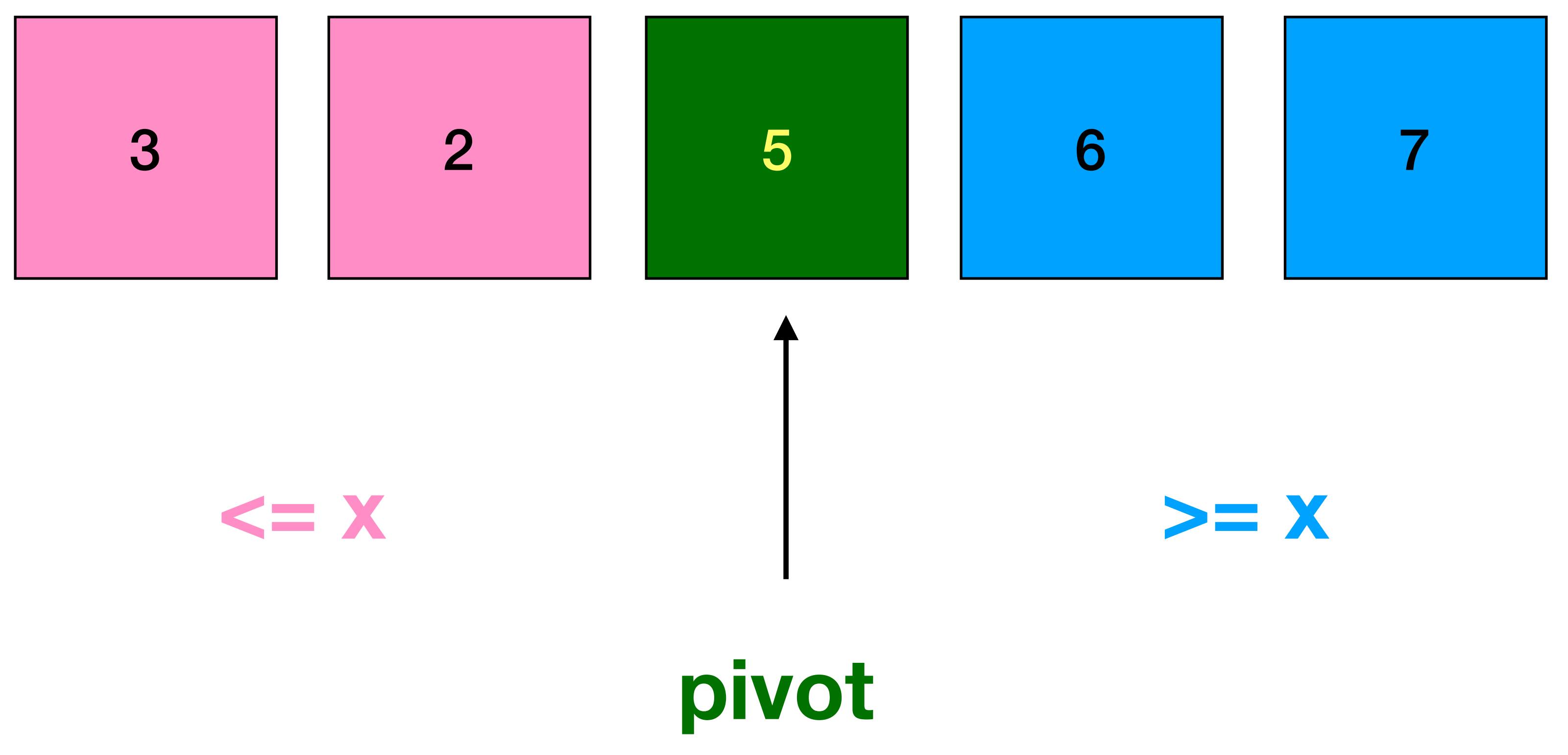

```
x = 5
p = i
for j=p+1 -> q
  if A[j] <= x
    i = i+1
    A[i] <-> A[j]
A[p] = A[i]
```




```
x = 5
p = i
for j=p+1 -> q
  if A[j] <= x
    i = i+1
    A[i] <-> A[j]
A[p] = A[i]
```




```
x = 5
p = i
for j=p+1 -> q
  if A[j] <= x
    i = i+1
    A[i] <-> A[j]
A[p] = A[i]
```


3
2
5
6
7
<= x
>= x
pivot

QUICKSORT(A, p, r)
    if $p < r$
        then $q \leftarrow$ PARTITION(A, p, r)
            QUICKSORT(A, p, $q-1$)
            QUICKSORT(A, $q+1$, r)

Initial call: QUICKSORT(A, 1, n)

Analysis

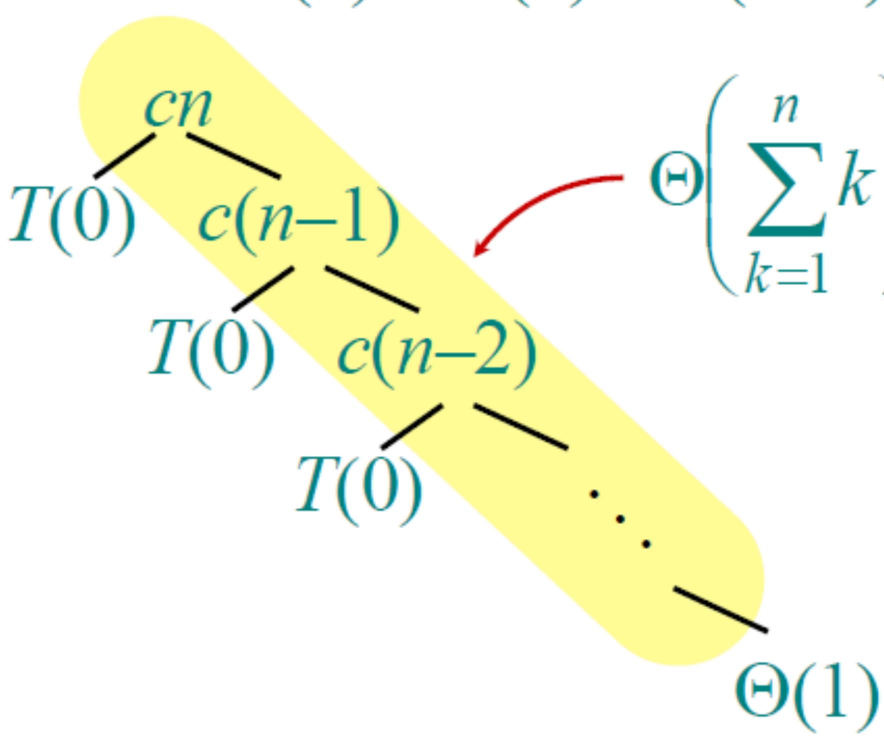
- Best-case

If we're lucky, PARTITION splits the array evenly:

$$\begin{aligned} T(n) &= 2T(n/2) + \Theta(n) \\ &= \Theta(n \lg n) \qquad \text{(same as merge sort)} \end{aligned}$$

- <u>Worst-case</u>

$$T(n) = T(0) + T(n-1) + cn$$



$$\Theta\left(\sum_{k=1}^{n} k\right) = \Theta\left(n^2\right)$$

$$\begin{aligned} T(n) &= T(0) + T(n-1) + \Theta(n) \\ &= \Theta(1) + T(n-1) + \Theta(n) \\ &= T(n-1) + \Theta(n) \\ &= \Theta(n^2) \qquad \textit{(arithmetic series)} \end{aligned}$$

Unbalanced split

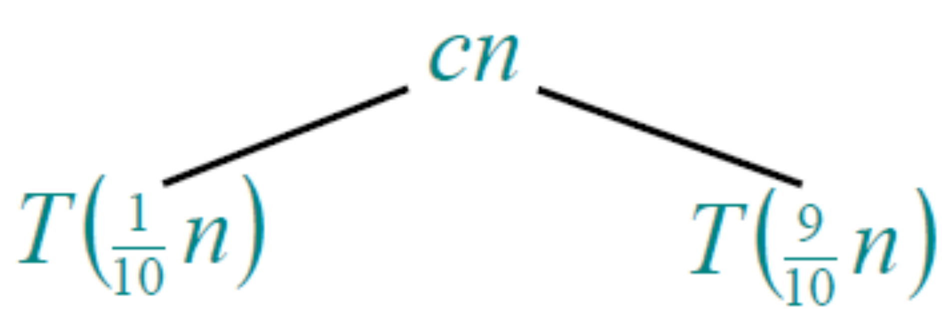
What if the split is always $\frac{1}{10} : \frac{9}{10}$?

$$T(n) = T\left(\tfrac{1}{10}n\right) + T\left(\tfrac{9}{10}n\right) + \Theta(n)$$

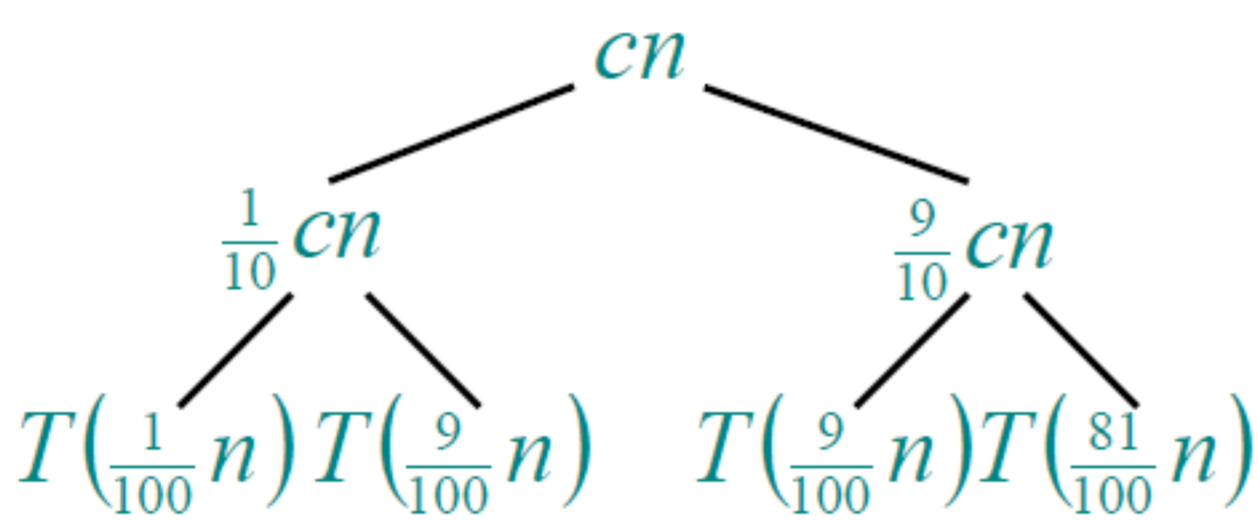
Recursion tree

$$T(n)$$

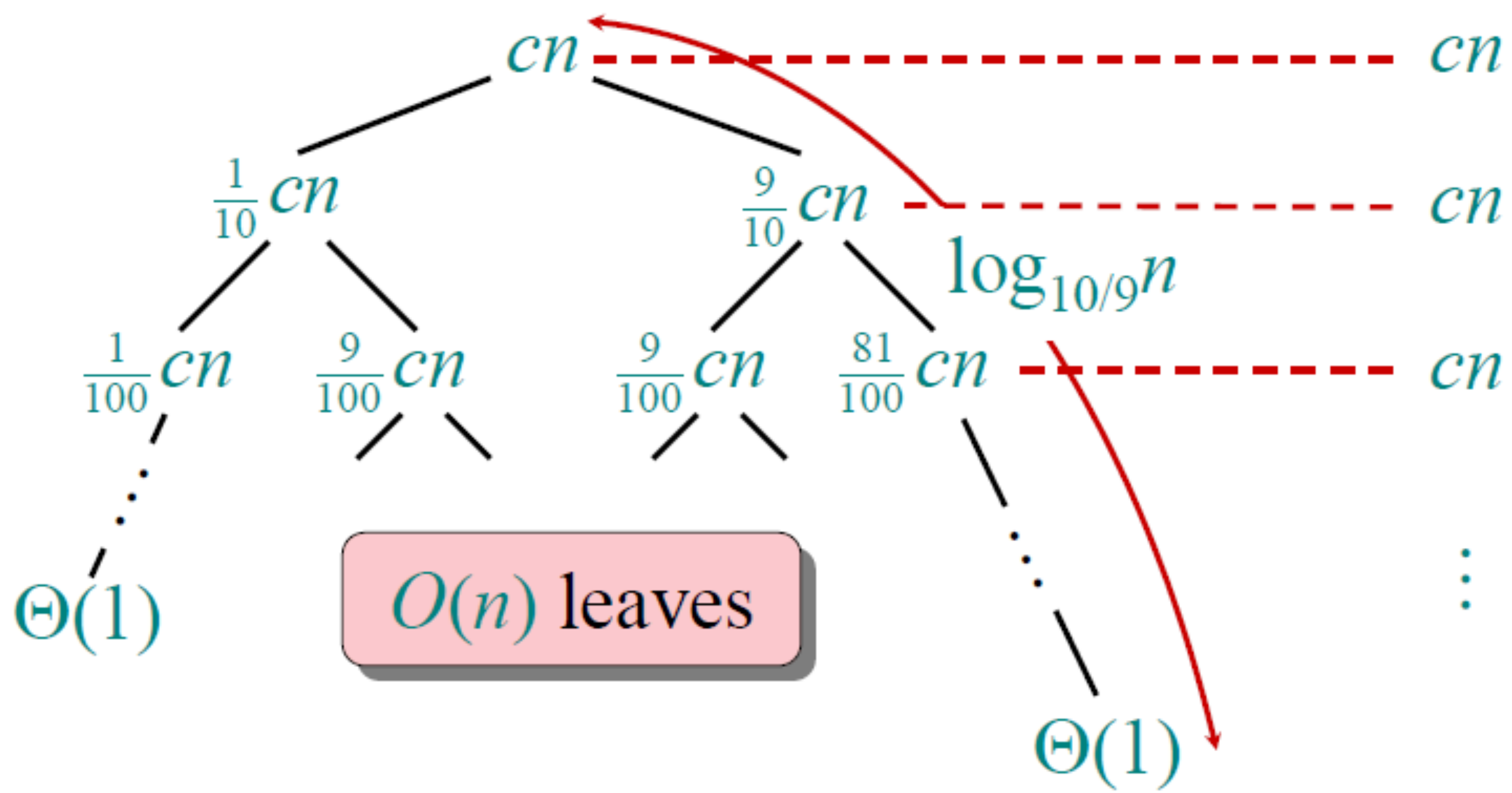
Recursion tree



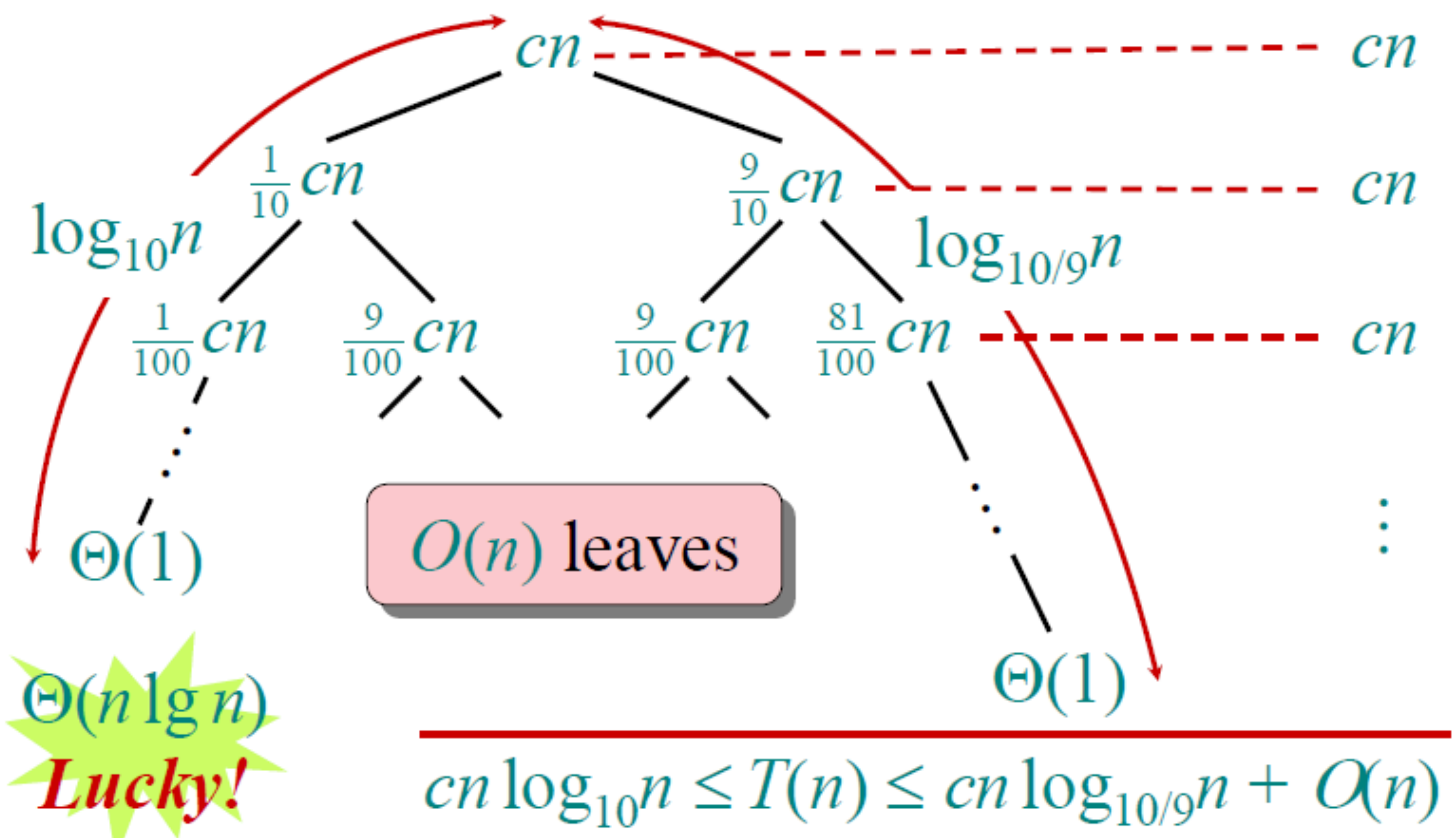
Recursion tree



Recursion tree



- <u>Recursion tree</u>



1.(10 pts) The running time of QUICKSORT depends on both the data being sorted and the partition rule used to select the pivot. Suppose we always pick the pivot element to be the key of the median element of the first three keys of the subarray. On a <u>sorted array</u>, determine whether QUICKSORT now takes $\Theta(n)$, $\Theta(n \log n)$, or $\Theta(n^2)$. Justify your answer.

(4) The running time of QUICKSORT when all elements of array A have the same value is $O(n \log n)$.

Randomized quicksort

IDEA: Partition around a ***random*** element.

- Running time is independent of the input order.
- No assumptions need to be made about the input distribution.
- No specific input elicits the worst-case behavior.
- The worst case is determined only by the output of a random-number generator.

Analysis

Let ***T*(*n*) =** the random variable for the running time of randomized quicksort on an input of size n, assuming random numbers are independent.

For $\boldsymbol{k = 0, 1, \ldots, n-1}$, define the ***indicator random variable***

$$X_k = \begin{cases} 1 & \text{if } \textsc{Partition} \text{ generates a } k : n-k-1 \text{ split,} \\ 0 & \text{otherwise.} \end{cases}$$

$\boldsymbol{E[X_k] = \Pr\{X_k = 1\} = 1/n}$, since all splits are equally likely, assuming elements are distinct.

Analysis (continued)

$$T(n) = \begin{cases} T(0) + T(n-1) + \Theta(n) & \text{if } 0 : n-1 \text{ split,} \\ T(1) + T(n-2) + \Theta(n) & \text{if } 1 : n-2 \text{ split,} \\ \quad\vdots & \\ T(n-1) + T(0) + \Theta(n) & \text{if } n-1 : 0 \text{ split,} \end{cases}$$

$$= \sum_{k=0}^{n-1} X_k \left(T(k) + T(n-k-1) + \Theta(n) \right)$$

Analysis (continued)

$$E[T(n)] = E\left[\sum_{k=0}^{n-1} X_k \left(T(k) + T(n-k-1) + \Theta(n)\right)\right]$$

Take expectations of both sides.

Analysis (continued)

$$E[T(n)] = E\left[\sum_{k=0}^{n-1} X_k \big(T(k) + T(n-k-1) + \Theta(n)\big)\right]$$

$$= \sum_{k=0}^{n-1} E\big[X_k \big(T(k) + T(n-k-1) + \Theta(n)\big)\big]$$

Linearity of expectation.

Analysis (continued)

$$
\begin{aligned}
E[T(n)] &= E\left[\sum_{k=0}^{n-1} X_k\big(T(k)+T(n-k-1)+\Theta(n)\big)\right] \\
&= \sum_{k=0}^{n-1} E\big[X_k\big(T(k)+T(n-k-1)+\Theta(n)\big)\big] \\
&= \sum_{k=0}^{n-1} E[X_k]\cdot E[T(k)+T(n-k-1)+\Theta(n)]
\end{aligned}
$$

Independence of X_k from other random choices.

Analysis (continued)

$$E[T(n)] = E\left[\sum_{k=0}^{n-1} X_k \big(T(k) + T(n-k-1) + \Theta(n)\big)\right]$$

$$= \sum_{k=0}^{n-1} E\big[X_k \big(T(k) + T(n-k-1) + \Theta(n)\big)\big]$$

$$= \sum_{k=0}^{n-1} E[X_k] \cdot E[T(k) + T(n-k-1) + \Theta(n)]$$

$$= \frac{1}{n}\sum_{k=0}^{n-1} E[T(k)] + \frac{1}{n}\sum_{k=0}^{n-1} E[T(n-k-1)] + \frac{1}{n}\sum_{k=0}^{n-1} \Theta(n)$$

Linearity of expectation; $E[X_k] = 1/n$.

Analysis (continued)

$$
\begin{aligned}
E[T(n)] &= E\left[\sum_{k=0}^{n-1} X_k\left(T(k) + T(n-k-1) + \Theta(n)\right)\right] \\
&= \sum_{k=0}^{n-1} E\left[X_k\left(T(k) + T(n-k-1) + \Theta(n)\right)\right] \\
&= \sum_{k=0}^{n-1} E[X_k] \cdot E[T(k) + T(n-k-1) + \Theta(n)] \\
&= \frac{1}{n}\sum_{k=0}^{n-1} E[T(k)] + \frac{1}{n}\sum_{k=0}^{n-1} E[T(n-k-1)] + \frac{1}{n}\sum_{k=0}^{n-1} \Theta(n) \\
&= \frac{2}{n}\sum_{k=1}^{n-1} E[T(k)] + \Theta(n)
\end{aligned}
$$

Summations have identical terms.

Analysis (continued)

$$E[T(n)] = \frac{2}{n}\sum_{k=2}^{n-1} E[T(k)] + \Theta(n)$$

(The $k = 0, 1$ terms can be absorbed in the $\Theta(n)$.)

Prove: $E[T(n)] \le a\,n \lg n$ for constant $a > 0$.

- Choose a large enough so that $a\,n \lg n$ dominates $E[T(n)]$ for sufficiently small $n \ge 2$.

Use fact: $\sum_{k=2}^{n-1} k \lg k \le \frac{1}{2}n^2 \lg n - \frac{1}{8}n^2$

Substitution method

$$
\begin{aligned}
E[T(n)] &\le \frac{2}{n}\sum_{k=2}^{n-1} ak\lg k + \Theta(n) \\
&= \frac{2a}{n}\left(\frac{1}{2}n^2\lg n - \frac{1}{8}n^2\right) + \Theta(n) \\
&= an\lg n - \left(\frac{an}{4} - \Theta(n)\right) \\
&\le an\lg n\,,
\end{aligned}
$$

if a is chosen large enough so that $an/4$ dominates the $\Theta(n)$.

Quicksort in practice

- Quicksort is a great general-purpose sorting algorithm.
- Quicksort is typically over twice as fast as merge sort.
- Quicksort can benefit substantially from ***code tuning***.
- Quicksort behaves well even with caching and virtual memory.

Selection

- Input : A list of numbers S; an integer k
- Output : The **kth** smallest element of S

- Naïve algorithm : Sort S. O(n log n) time.
- Can we do better?

Randomized divide-and-conquer algorithm

RAND-SELECT(A, p, q, i) ▷ ith smallest of $A[p \,..\, q]$

    if $p = q$ **then return** $A[p]$

    $r \leftarrow$ RAND-PARTITION(A, p, q)

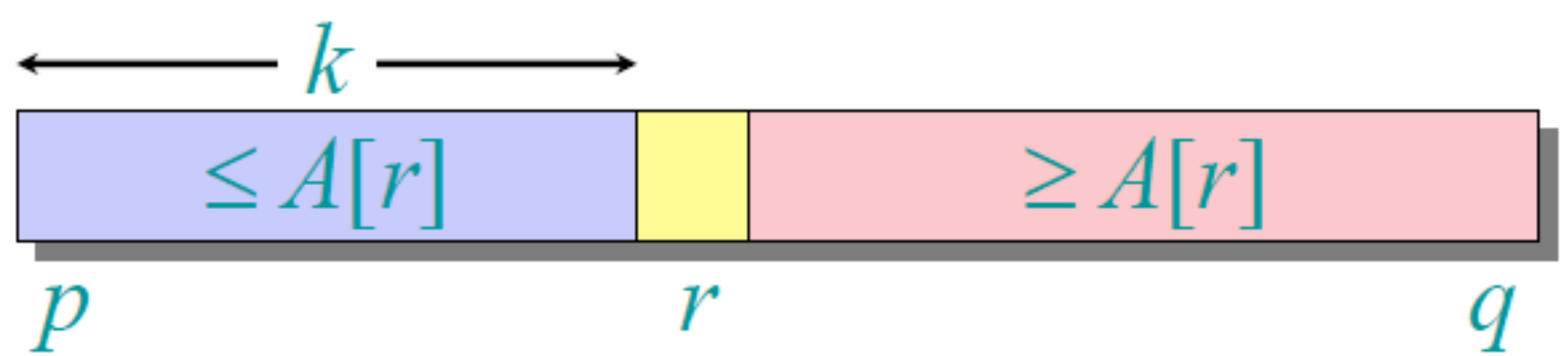
    $k \leftarrow r - p + 1$ ▷ $k = \text{rank}(A[r])$

    if $i = k$ **then return** $A[r]$

    if $i < k$

        then return RAND-SELECT$(A, p, r - 1, i)$

        else return RAND-SELECT$(A, r + 1, q, i - k)$



Example

Select the $i = 7$th smallest:

6	10	13	5	8	3	2	11

$i = 7$

pivot

Partition:

2	5	3	6	8	13	10	11

$k = 4$

Select the $7 - 4 = 3$rd smallest recursively.

worst-case

$$\begin{aligned} T(n) &= T(n-1) + \Theta(n) \\ &= \Theta(n^2) \end{aligned}$$

arithmetic series

Worse than sorting!

- Best-case

$$T(n) = T(n/2) + O(n) = O(n)$$

What if 9/10 : 1/10 split?

$$T(n) = T(9n/10) + O(n) = O(n)$$

Analysis of expected time

- The analysis follows that of randomized quicksort, but it's a little different.

$$X_k = \begin{cases} 1 & \text{if PARTITION generates a } k : n-k-1 \text{ split,} \\ 0 & \text{otherwise.} \end{cases}$$

$$T(n) = \begin{cases} T(\max\{0, n-1\}) + \Theta(n) & \text{if } 0 : n-1 \text{ split,} \\ T(\max\{1, n-2\}) + \Theta(n) & \text{if } 1 : n-2 \text{ split,} \\ \quad\vdots & \\ T(\max\{n-1, 0\}) + \Theta(n) & \text{if } n-1 : 0 \text{ split,} \end{cases}$$

$$= \sum_{k=0}^{n-1} X_k \left(T(\max\{k, n-k-1\}) + \Theta(n)\right).$$

$$
\begin{aligned}
E[T(n)] &= E\left[\sum_{k=0}^{n-1} X_k\big(T(\max\{k, n-k-1\}) + \Theta(n)\big)\right] \\
&= \sum_{k=0}^{n-1} E\left[X_k\big(T(\max\{k, n-k-1\}) + \Theta(n)\big)\right] \\
&= \sum_{k=0}^{n-1} E[X_k] \cdot E[T(\max\{k, n-k-1\}) + \Theta(n)] \\
&= \frac{1}{n}\sum_{k=0}^{n-1} E[T(\max\{k, n-k-1\})] + \frac{1}{n}\sum_{k=0}^{n-1} \Theta(n) \\
&\le \frac{2}{n}\sum_{k=\lfloor n/2 \rfloor}^{n-1} E[T(k)] + \Theta(n)
\end{aligned}
$$

Upper terms appear twice.

$$E[T(n)] = \frac{2}{n} \sum_{k=\lfloor n/2 \rfloor}^{n-1} E[T(k)] + \Theta(n)$$

Prove: $E[T(n)] \le cn$ for constant $c > 0$.

- The constant c can be chosen large enough so that $E[T(n)] \le cn$ for the base cases.

Use fact: $\displaystyle\sum_{k=\lfloor n/2 \rfloor}^{n-1} k \le \tfrac{3}{8} n^2$

Substitution method

$$
\begin{aligned}
E[T(n)] &\le \frac{2}{n} \sum_{k=\lfloor n/2 \rfloor}^{n-1} ck + \Theta(n) \\
&\le \frac{2c}{n}\left(\frac{3}{8}n^2\right) + \Theta(n) \\
&= cn - \left(\frac{cn}{4} - \Theta(n)\right) \\
&\le cn\,,
\end{aligned}
$$

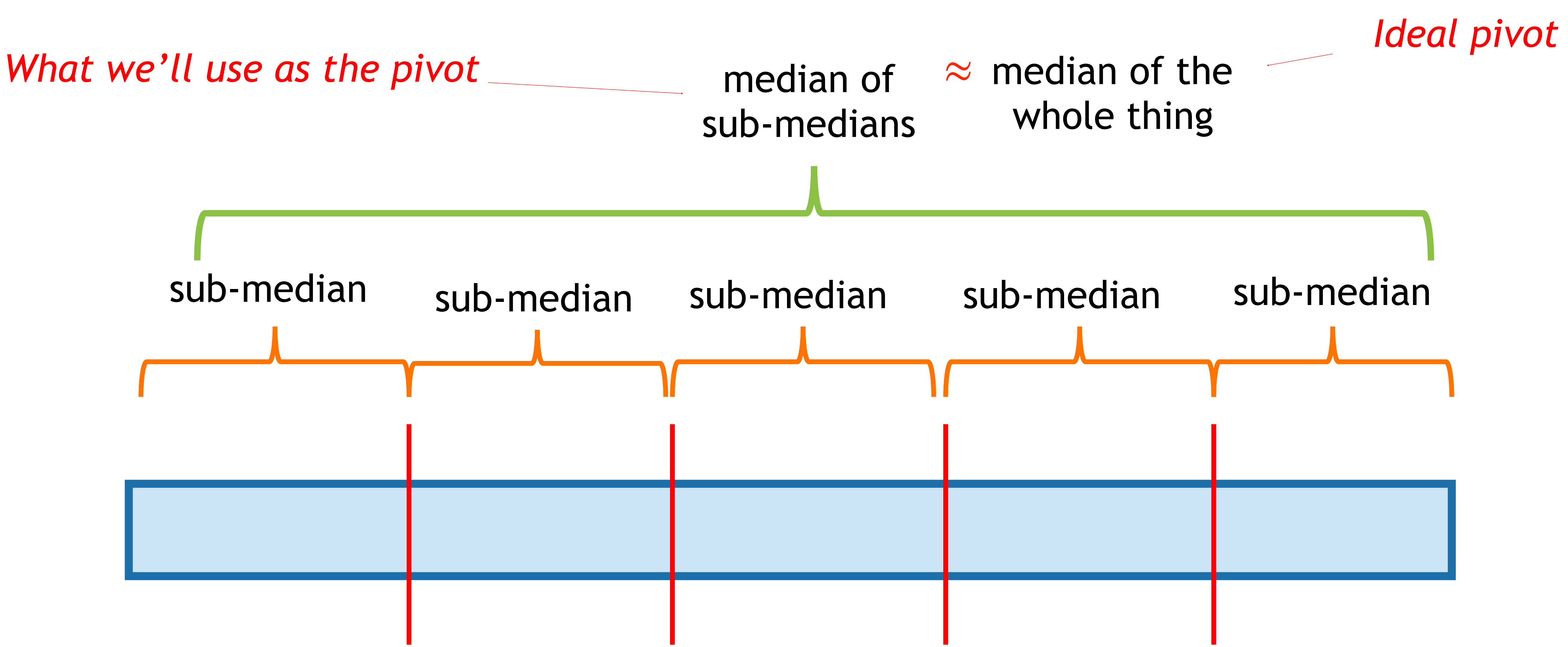
if c is chosen large enough so that $cn/4$ dominates the $\Theta(n)$.

Randomized selection algorithm

- Works fast: linear expected time.
- Excellent algorithm in practice.
- But, the worst case is ***very bad: $\Theta(n^2)$***.
- Is there an algorithm that runs in ***linear time in the worst case***?
- Yes, due to Blum, Floyd, Pratt, Rivest, and Tarjan [1973].
- IDEA: Generate a *good* pivot *recursively*.

Another divide-and-conquer alg!

- We can't solve SELECT(A,n/2) (yet)
- But we can divide and conquer and solve SELECT(B,m/2) for smaller values of m (where len(B) = m).
- **Lemma*:** The median of sub-medians is close to the median.



*we will make this a bit more precise.

How to pick the pivot

- CHOOSEPIVOT(A):
  - Split A into m = groups, of size <=5 each.
  - **For** i=1, .., m
    - Find the median within the i'th group, call it p_i
  - p = SELECT([p_1, p_2, p_3, …, p_m] , m/2)
  - **return** p

This takes time O(1), for each group, since each group has size 5. So that's O(m)=O(n) total in the for loop.

8									4										6		12			
1	8	9	3	15	5	9	1	3	4	12	2	1	5	20	15	13	2	4	6	12	1	15	22	3

Pivot is SELECT(8 4 5 6 12 , 3) = 6:

1	8	9	3	15	5	9	1	3	4	12	2	1	5	20	15	13	2	4	6	12	1	15	22	3

PARTITION around that 6:

1	3	5	1	3	4	2	1	2	4	1	3	5	6	8	9	15	9	12	20	15	13	12	15	22

This part is L

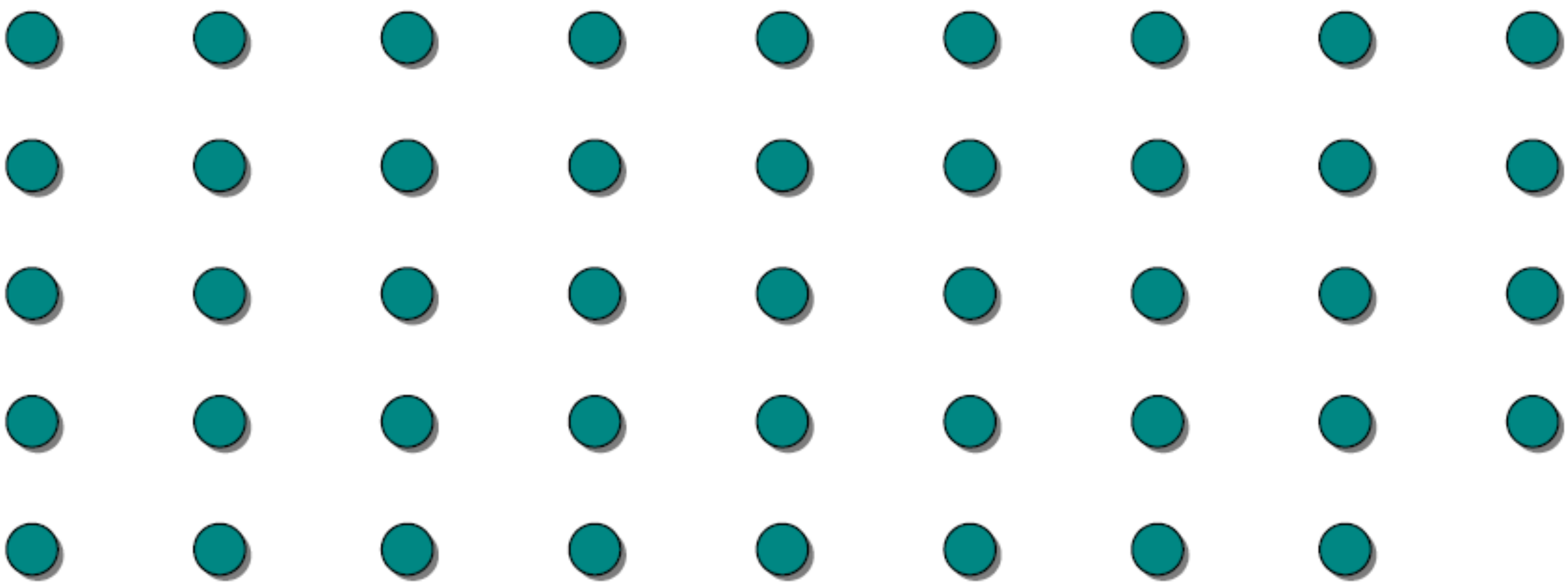
This part is R: it's almost the same size as L.

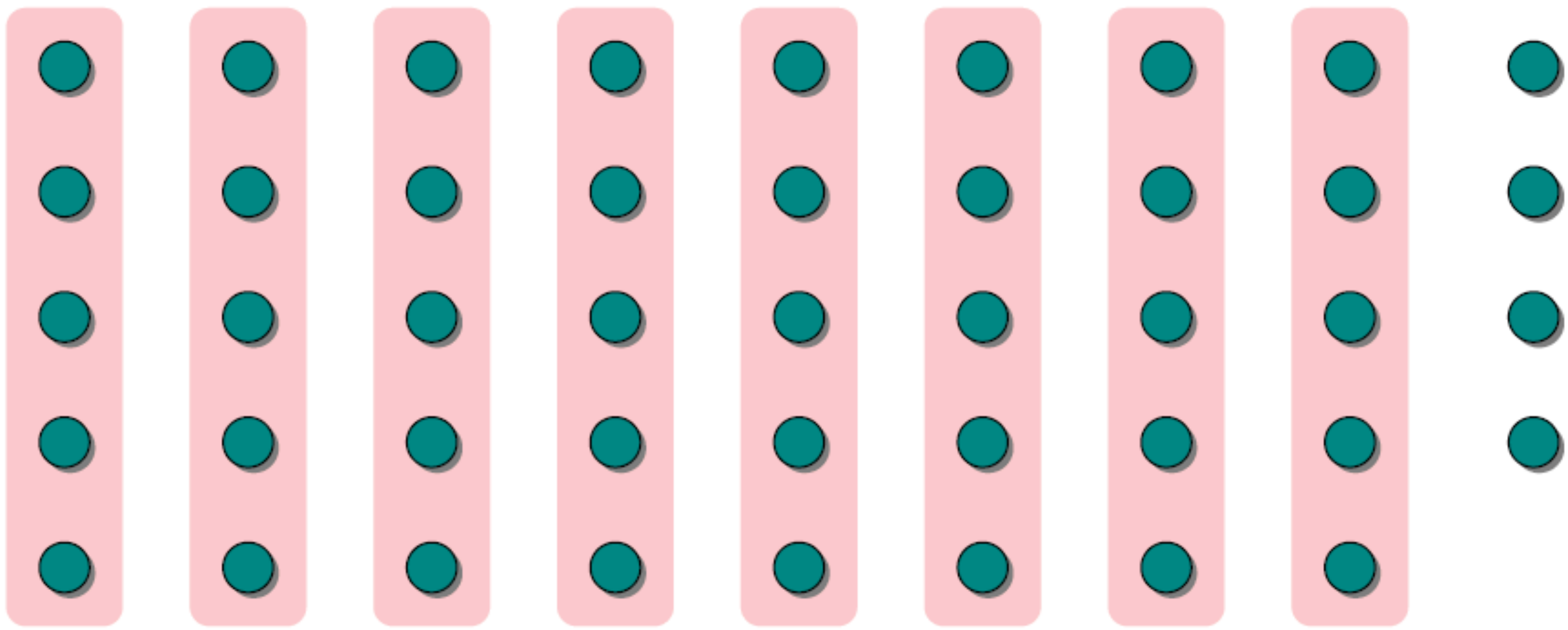
Worst-case linear-time selection algorithm

```
SELECT(A[1 .. n], k):
  if n ≤ 25
      use brute force
  else
      m ← ⌈n/5⌉
      for i ← 1 to m
          B[i] ← SELECT(A[5i − 4 .. 5i], 3)          ⟨⟨Brute force!⟩⟩
      mom ← SELECT(B[1 .. m], ⌊m/2⌋)                 ⟨⟨Recursion!⟩⟩
      r ← PARTITION(A[1 .. n], mom)
      if k < r
          return SELECT(A[1 .. r − 1], k)            ⟨⟨Recursion!⟩⟩
      else if k > r
          return SELECT(A[r + 1 .. n], k − r)        ⟨⟨Recursion!⟩⟩
      else
          return mom
```

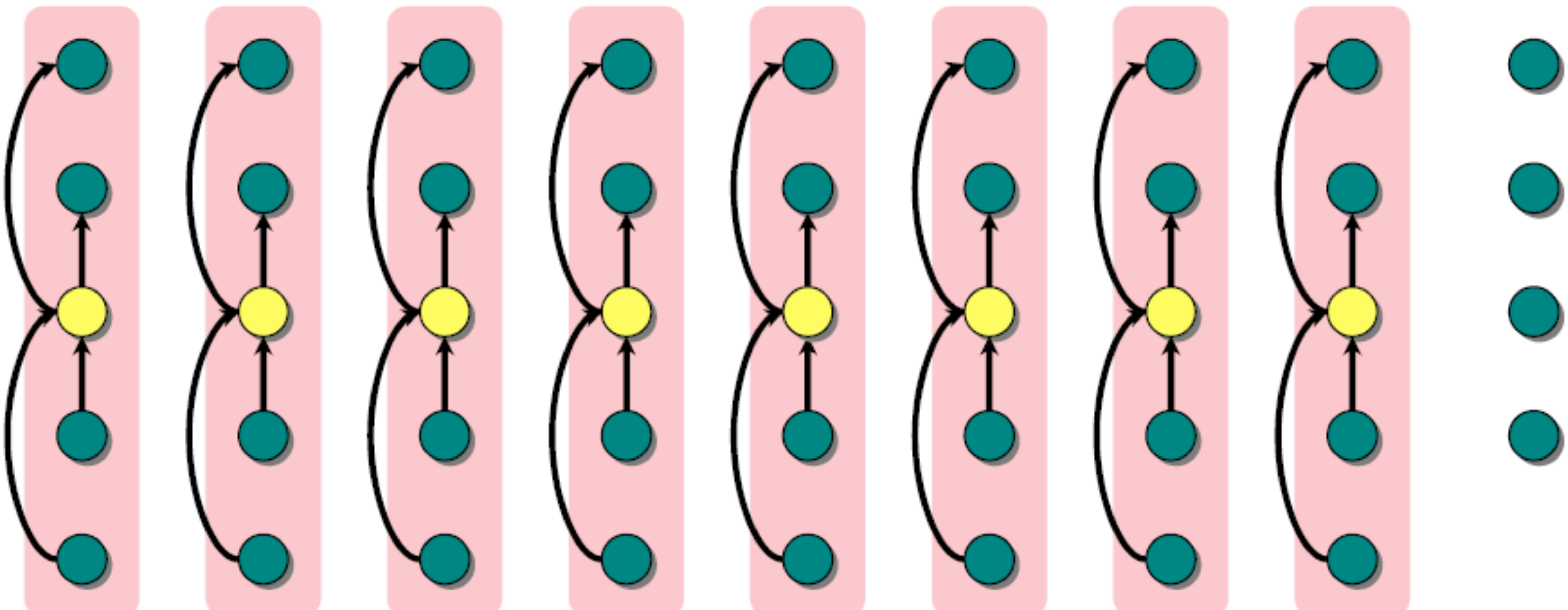
Algorithm

- Divide the n elements into groups of 5.
- Find the median of each 5-element group by rote.
- Recursively SELECT the median *mom of the* $\lfloor n/5 \rfloor$ group medians to be the pivot.
- Partition around the pivot *mom*. Let $r = rank(mom)$.
- if $k < r$ then recursively SELECT the kth smallest element in the lower part
- else if k > r then recursively SELECT the(k–r)th smallest element in the upper part
- else return *mom*

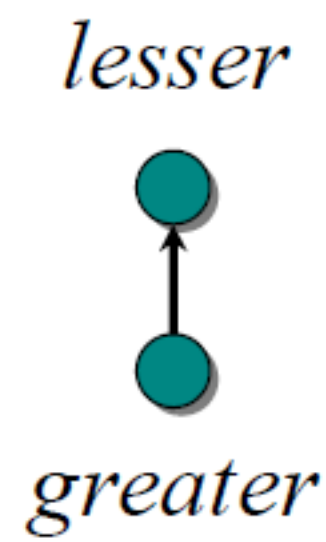


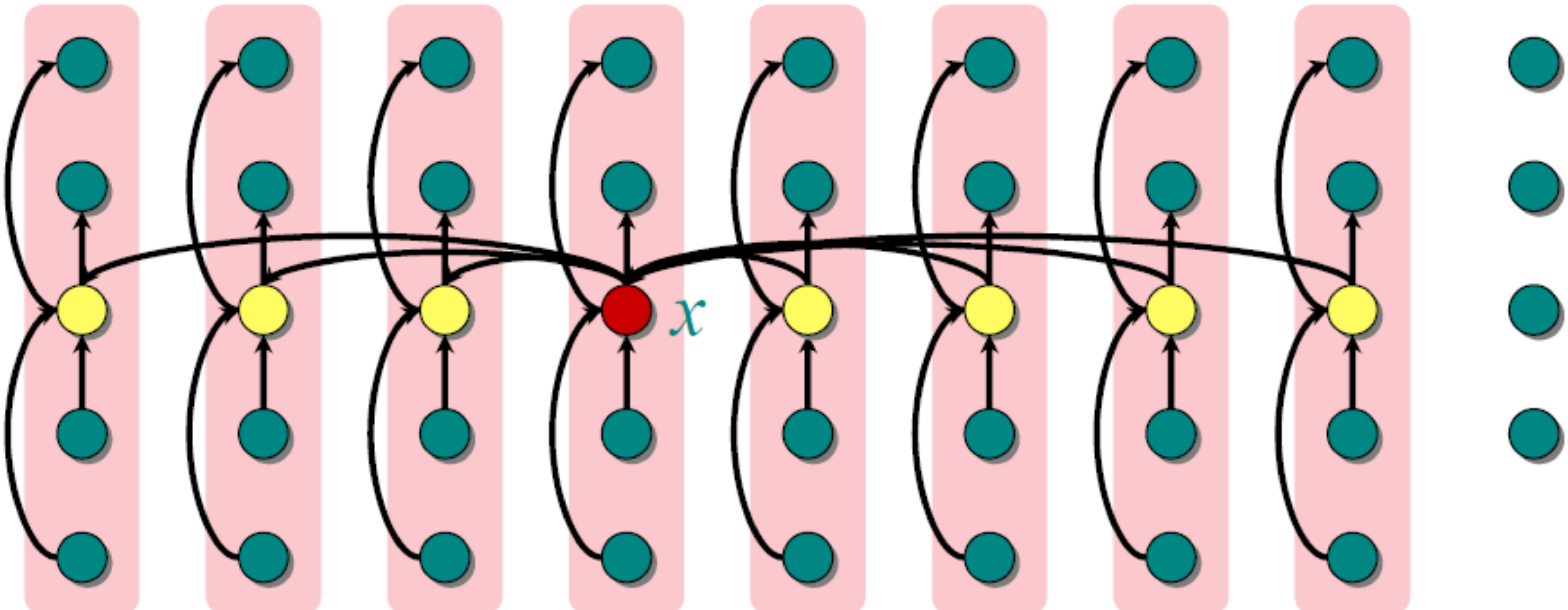


1. Divide the n elements into groups of 5.

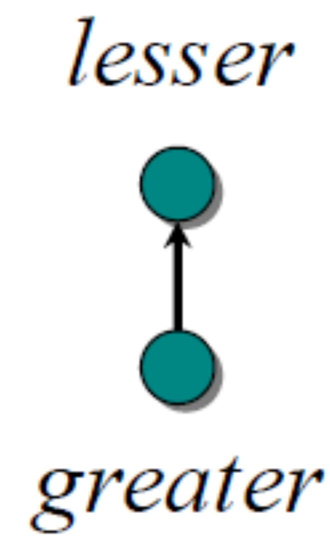


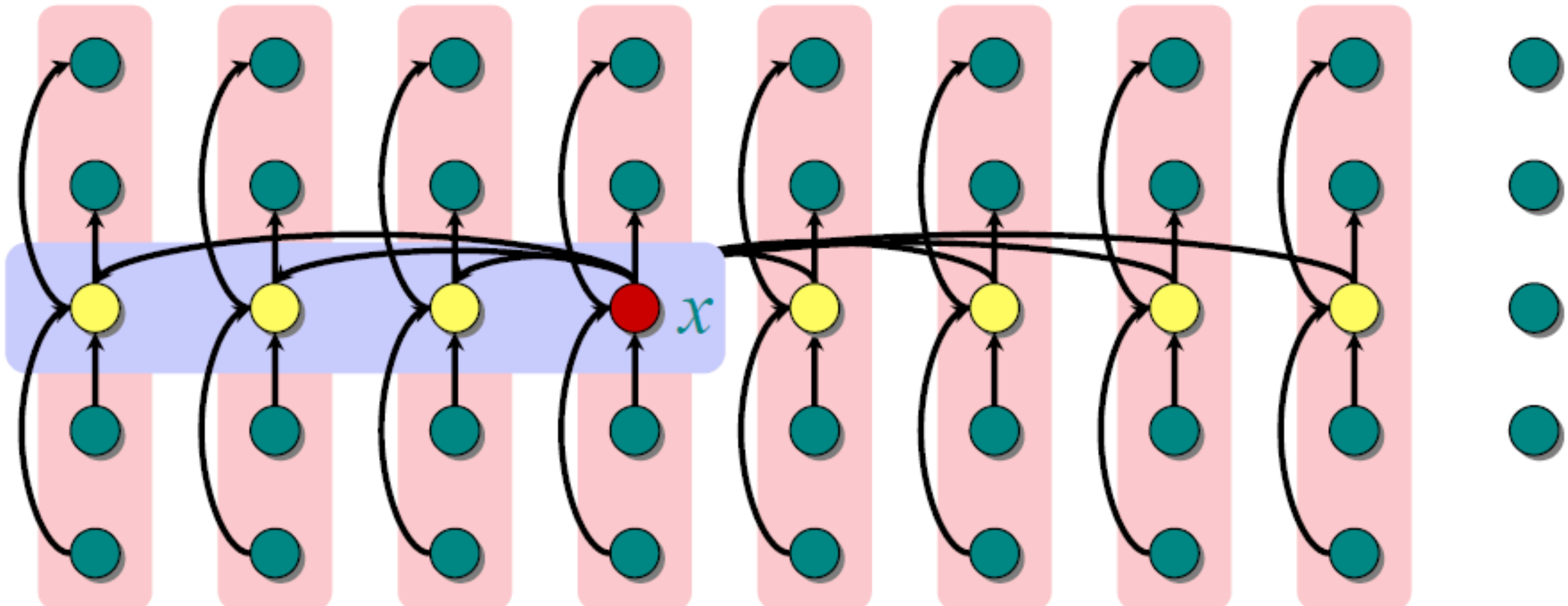
1. Divide the n elements into groups of 5. Find the median of each 5-element group by rote.



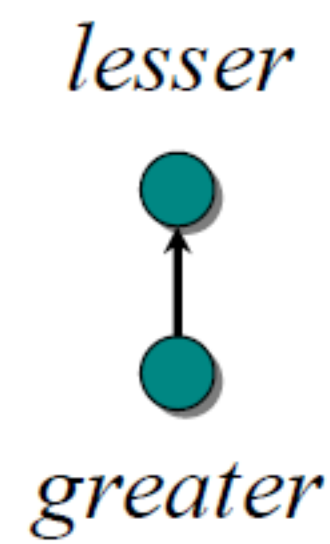


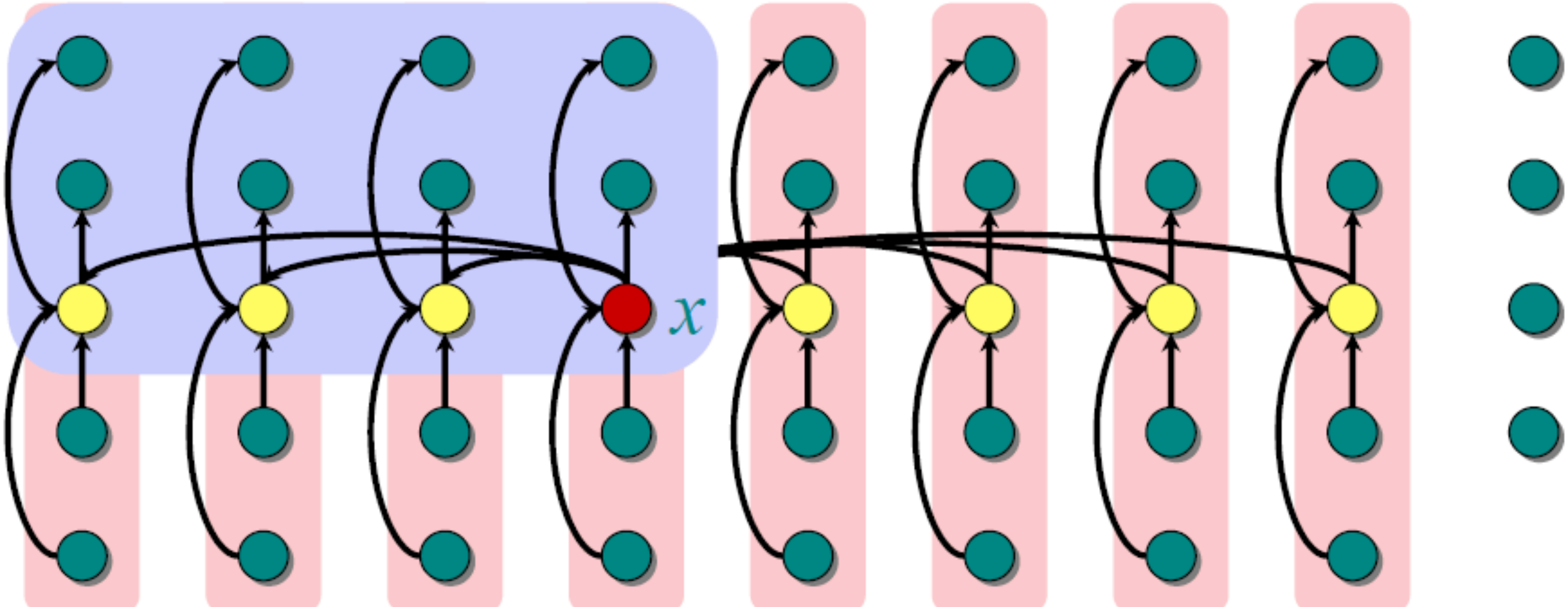
1. Divide the n elements into groups of 5. Find the median of each 5-element group by rote.
2. Recursively SELECT the median x of the $\lfloor n/5 \rfloor$ group medians to be the pivot.





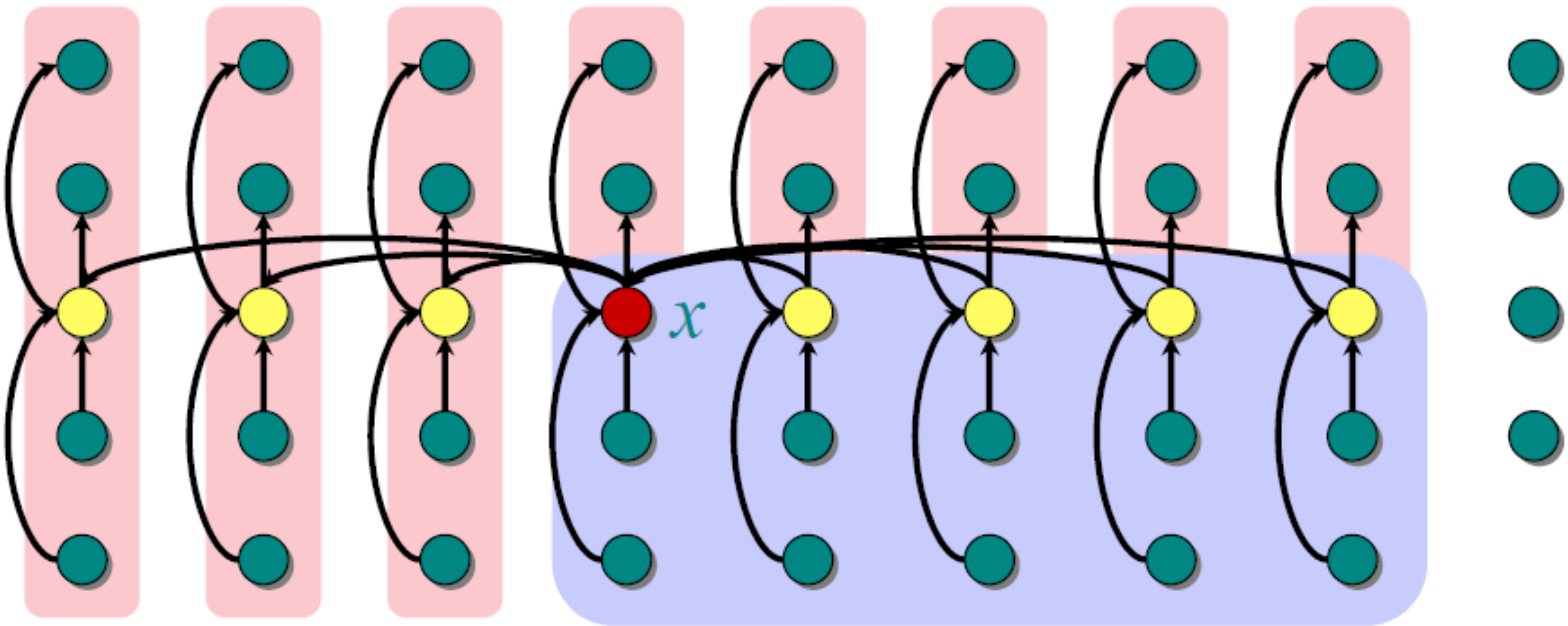
At least half the group medians are $\leq x$, which is at least $\lfloor \lfloor n/5 \rfloor /2 \rfloor = \lfloor n/10 \rfloor$ group medians.





At least half the group medians are $\le x$, which is at least $\lfloor \lfloor n/5 \rfloor /2 \rfloor = \lfloor n/10 \rfloor$ group medians.

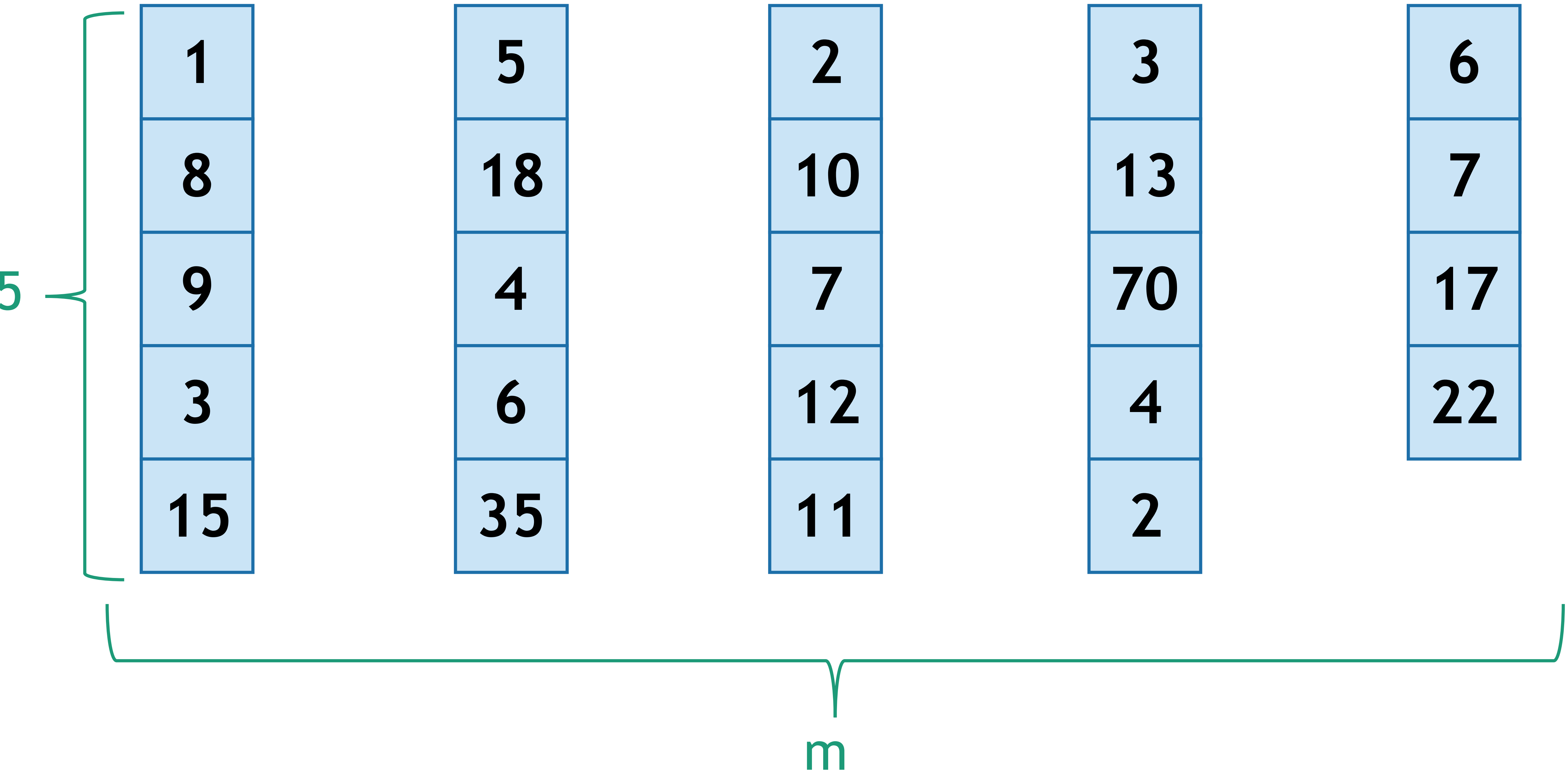
- Therefore, at least $3\lfloor n/10 \rfloor$ elements are $\le x$.



At least half the group medians are $\leq x$, which is at least $\lfloor \lfloor n/5 \rfloor /2 \rfloor = \lfloor n/10 \rfloor$ group medians.

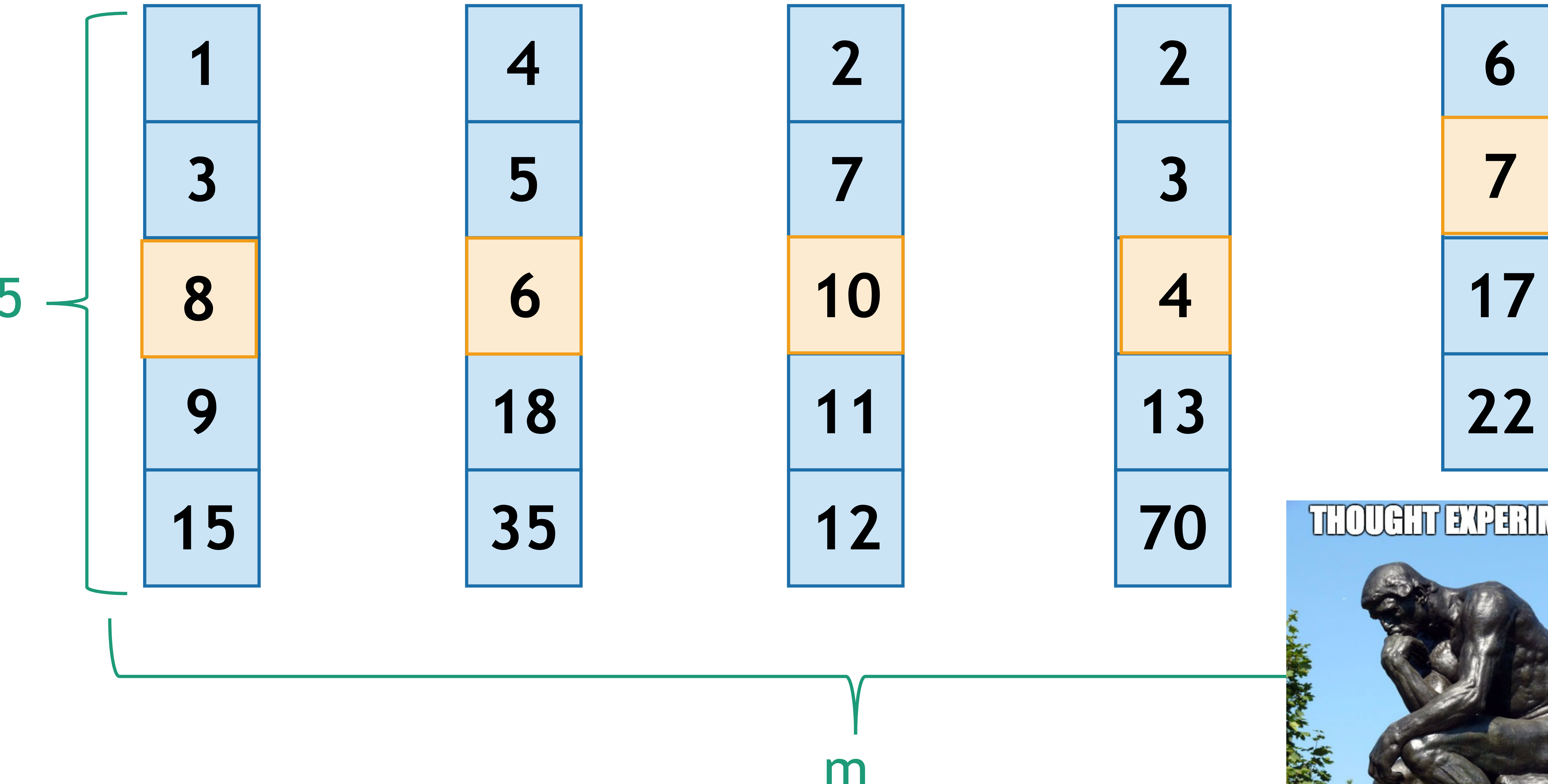
- Therefore, at least $3\lfloor n/10 \rfloor$ elements are $\leq x$.
- Similarly, at least $3\lfloor n/10 \rfloor$ elements are $\geq x$.

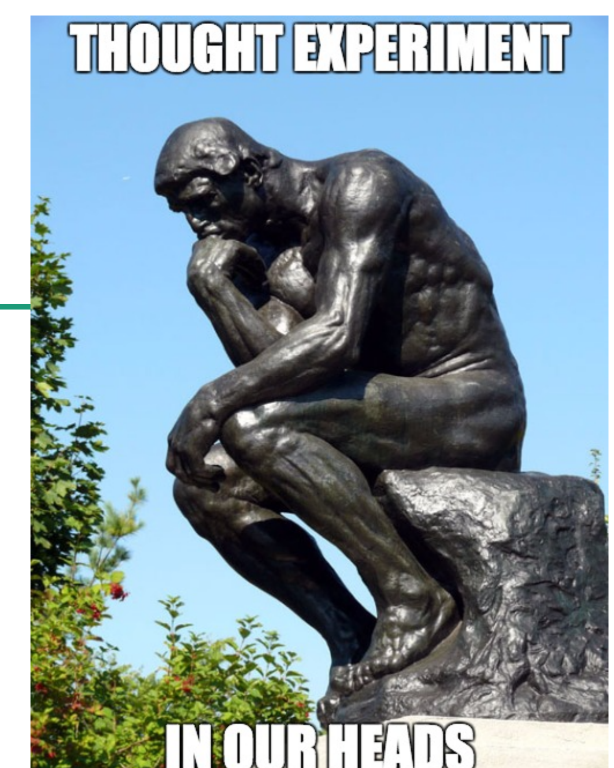
Proof by picture



Say these are our m = [n/5] sub-arrays of size at most 5.

Proof by picture

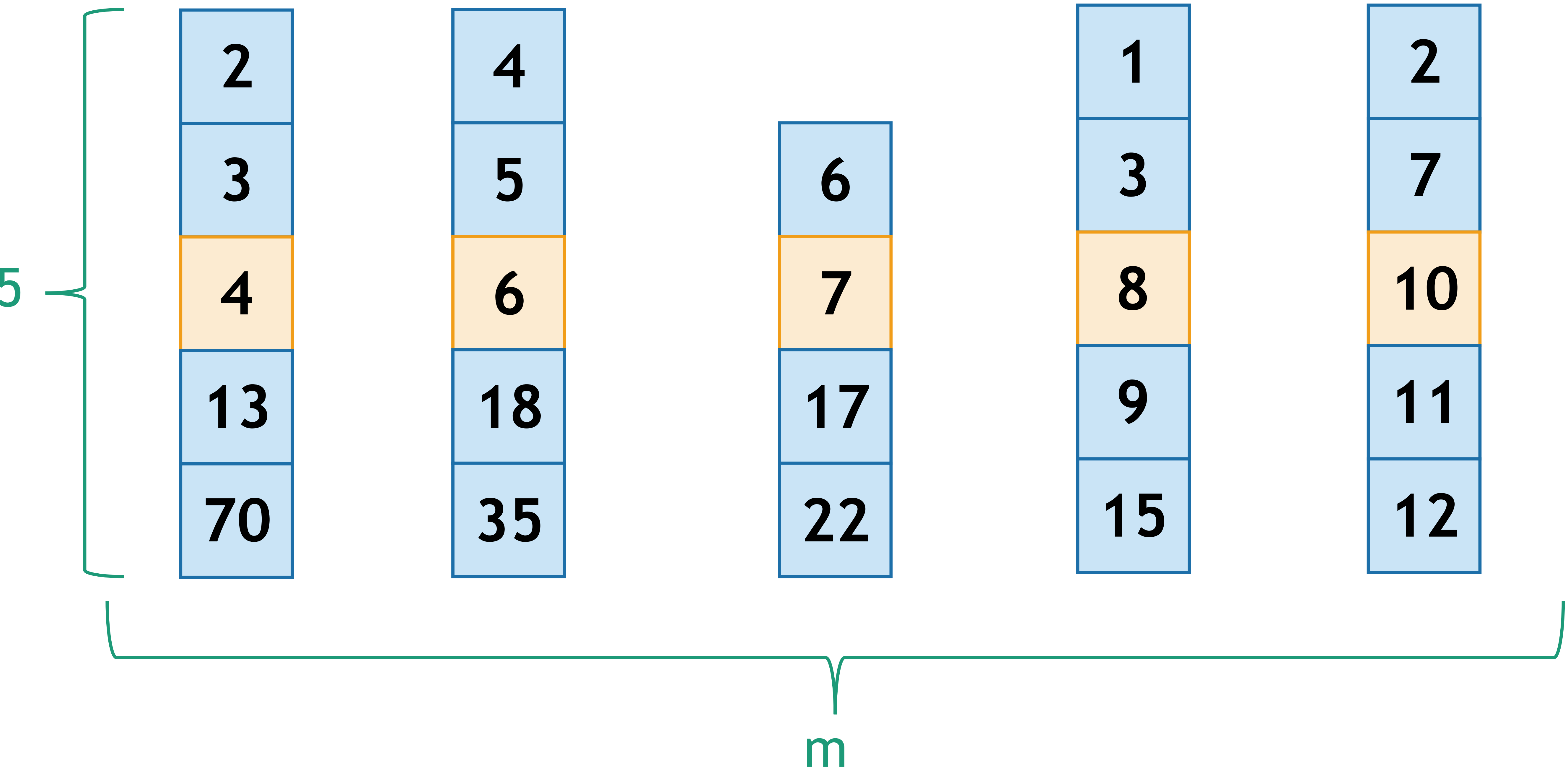




In our head, let's sort them.

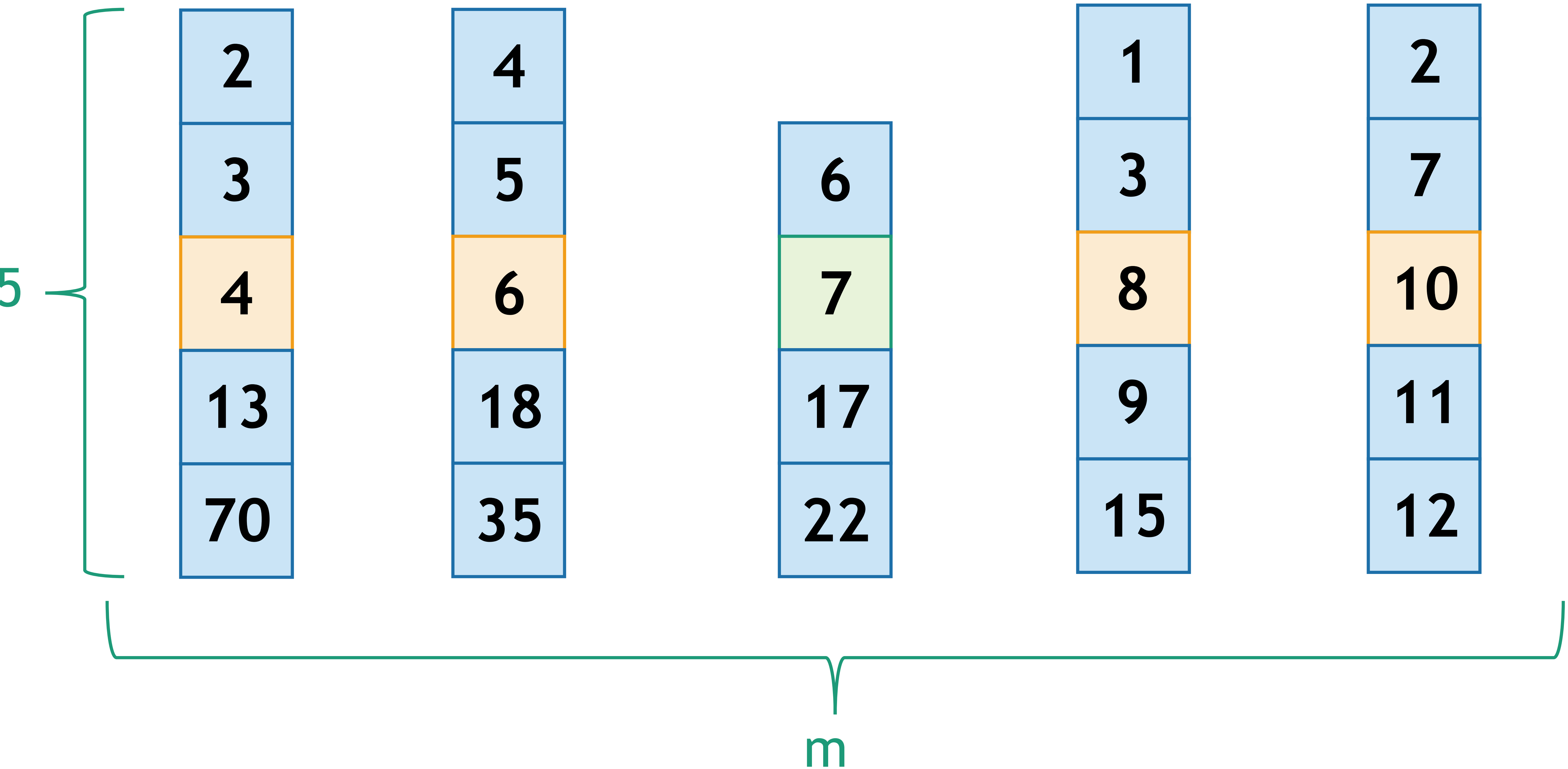
Then find medians T(n/5).

Proof by picture



Then let’s sort them by the median

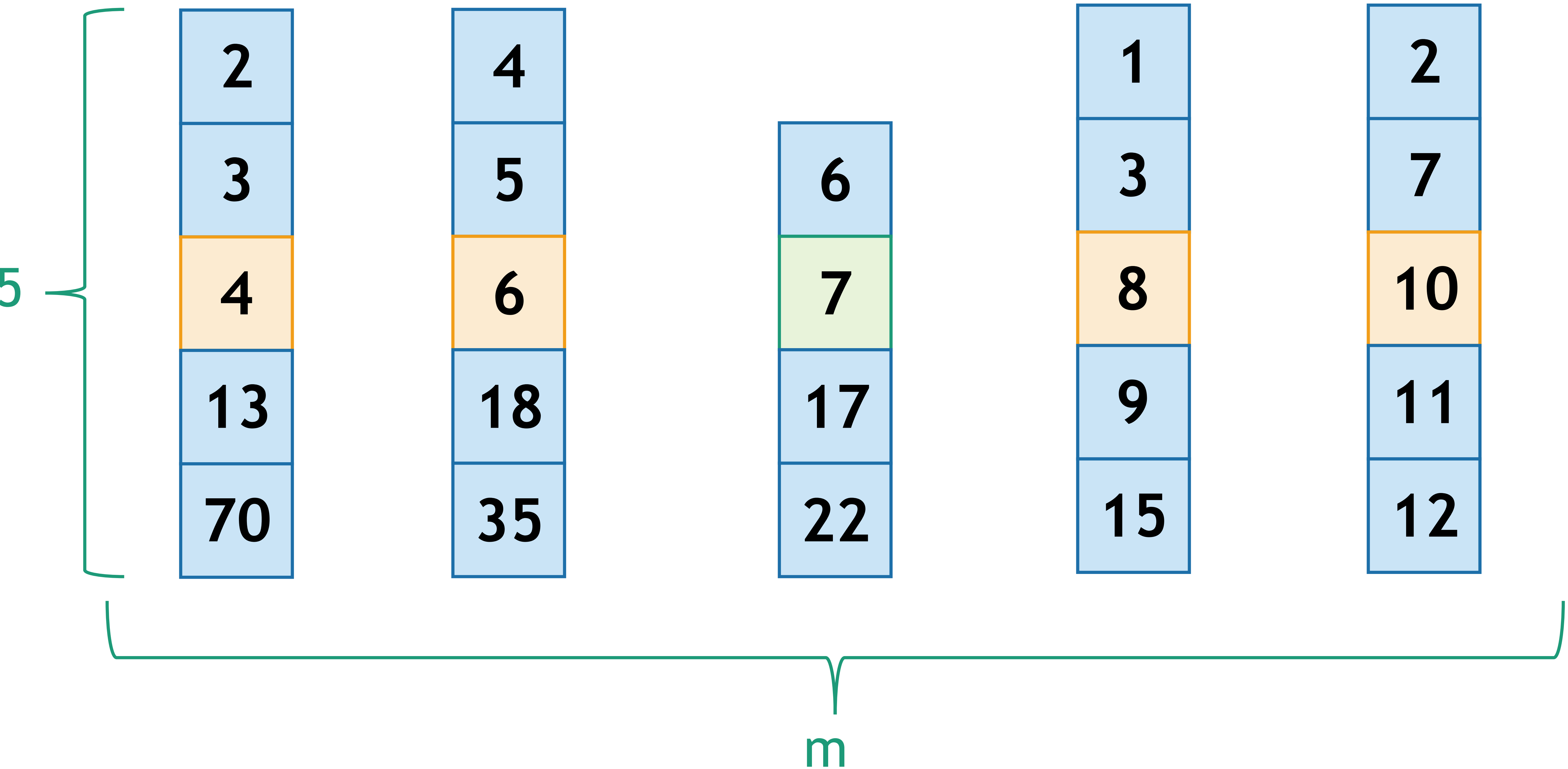
Proof by picture



The median of the medians is 7. That's our pivot!

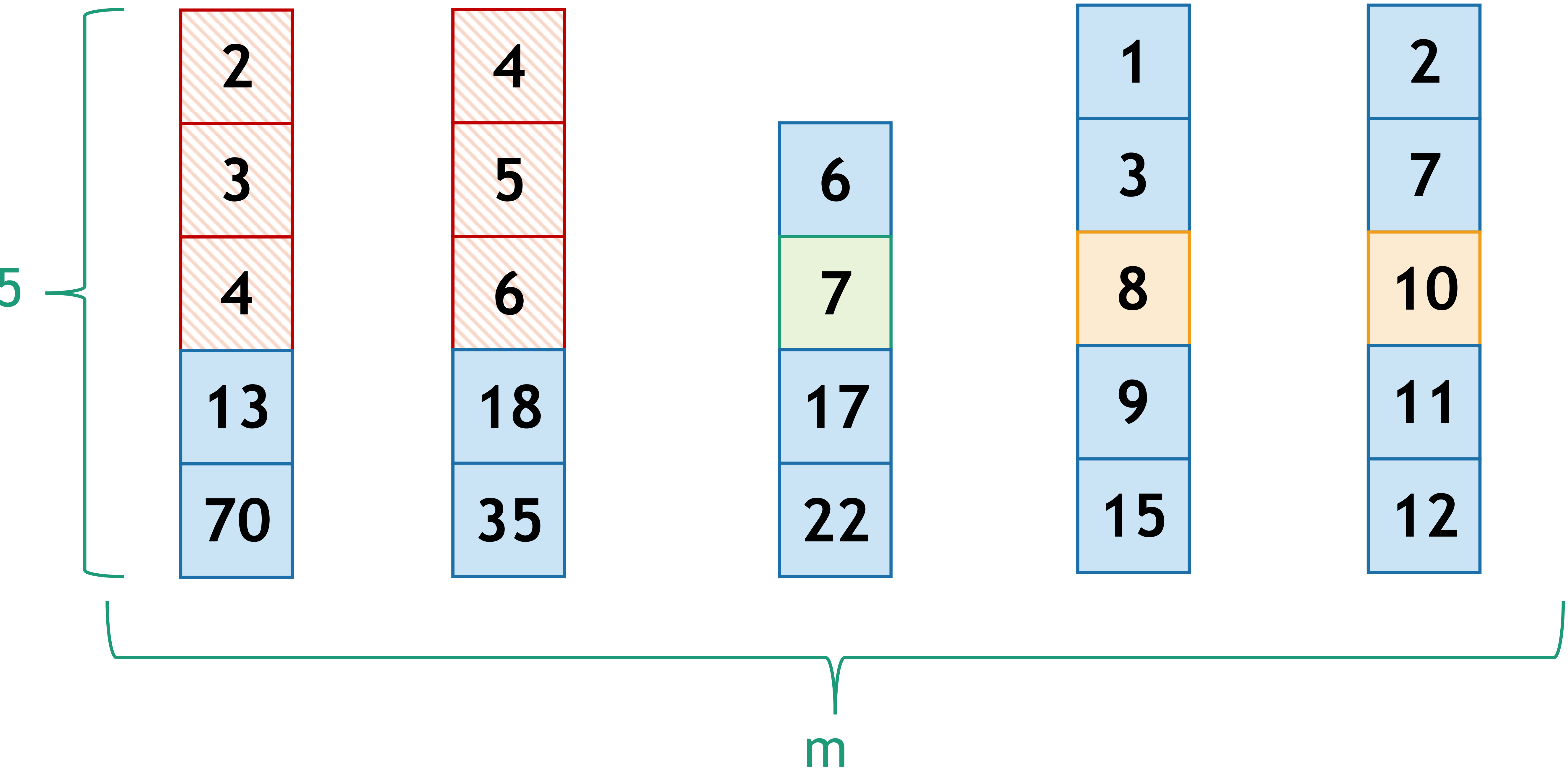
Proof by picture

We will show that lots of elements are smaller than the pivot, hence not too many are larger than the pivot.



How many elements are SMALLER than the pivot?

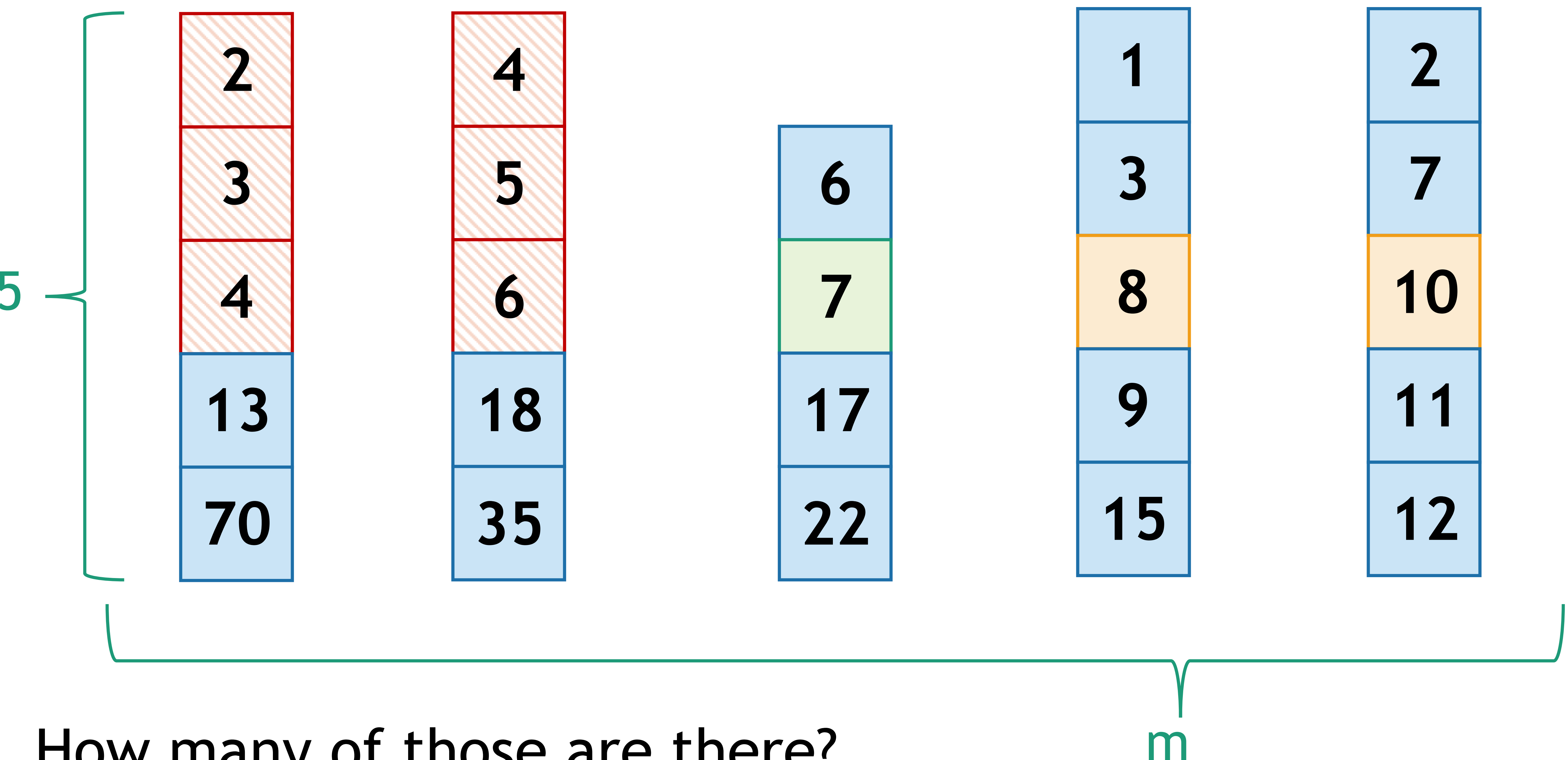
Proof by picture



At least these ones: everything above and to the left.

Proof by picture

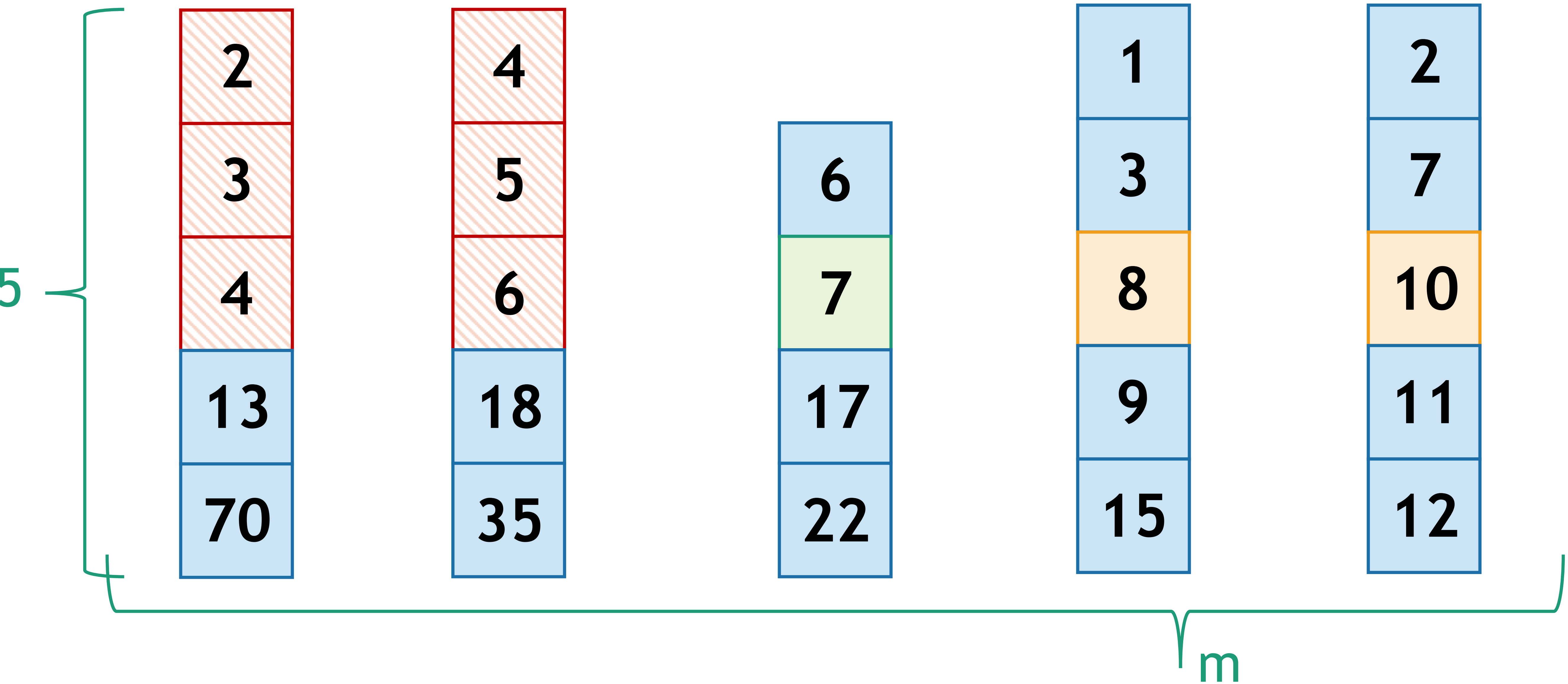
($\lceil m/2 \rceil$-1) of these, but then one of them could have been the "leftovers" group.



How many of those are there?

at least $3 \cdot (\lceil m/2 \rceil - 2)$

Proof by picture



So how many are LARGER than the pivot? At most…

(derivation on board)

$$n - 1 - 3(\lceil m/2 \rceil - 2) \leq 7n/10 + 5$$

Remember

$m = \lceil n/5 \rceil$

Analysis

- The two subarrays partitoned cannot be too large or too small.
- The median of group medians (*mom*) is larger than $\lceil\lceil n/5\rceil/2\rceil - 1 \approx n/10$ group medians.
- Thus *mom* is larger than 3n/10 elements in the input array.
- So, in the worst case, we recusively search at most 7n/10 elements array.
- $T(n) \leq T(n/5) + T(7n/10) + O(n) = O(n)$ (Prove it using substitution method!)