ORDER STATISTICS - MEDIAN FINDING, RANK SELECTION

Given n unsorted elements, find the k-th smallest.

We will assume distinct elements.

ORDER STATISTICS - MEDIAN FINDING, RANK SELECTION

Given n unsorted elements, find the k-th smallest.

↳ easy $O(n)$ if $k = O(1)$ or $n - O(1)$.
(median is hardest)

We will assume distinct elements.

ORDER STATISTICS – MEDIAN FINDING, RANK SELECTION

Given n unsorted elements, find the k-th smallest.

↳ easy $O(n)$ if $k = O(1)$ or $n - O(1)$.
(median is hardest)

We will assume distinct elements.

Algorithm by: Blum, Floyd, Pratt, Rivest, Tarjan

1973

we are looking for the element w/ rank r

we are looking for the element w/ rank r

Select(r, 1...n) // find r^{th} smallest # within array[1...n]

we are looking for the element w/ rank r

Select(r, 1…n) // find r^{th} smallest # within array[1…n]

$$\boxed{\text{we will choose an element } x}$$ TBD

Compare all elements to x → compute rank[x] = p

we are looking for the element w/ rank r

Select(r, 1...n) // find r^{th} smallest # within array[1...n]

$$\boxed{\text{we will choose an element } x}$$

Compare all elements to x → compute rank[x] = p

if rank[x] = p = r , DONE

we are looking for the element w/ rank r

Select(r, 1...n) // find r^{th} smallest # within array[1...n]

$$\boxed{\text{we will choose an element } x}$$

Compare all elements to x → compute rank[x] = p

if rank[x] = p = r, DONE, Else use x as pivot to partition input
(set up binary search)

we are looking for the element w/ rank r

Select(r, 1...n) // find r^{th} smallest # within array[1...n]

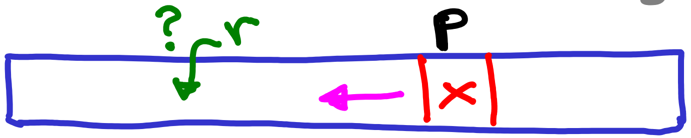
we will choose an element x

Compare all elements to x → compute rank[x] = p

if rank[x] = p = r, DONE, Else use x as pivot to partition input

(set up binary search)

if p > r // rank[x] > r, so search lower



we are looking for the element w/ rank r

Select(r, 1…n) // find r^{th} smallest # within array[1…n]

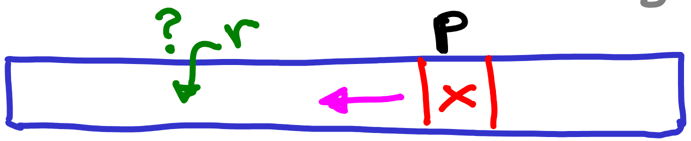
we will choose an element x

Compare all elements to x → compute rank[x] = p

if rank[x] = p = r , DONE, Else use x as pivot to partition input (set up binary search)

if p > r // rank[x] > r, so search lower

Select(r, 1..p-1)



we are looking for the element w/ rank r

Select(r, 1…n) // find r^{th} smallest # within array[1…n]

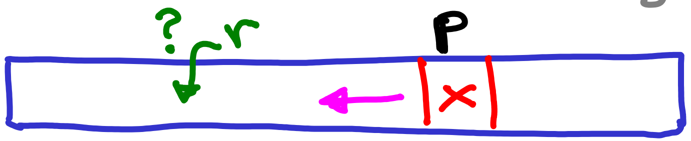
we will choose an element x

Compare all elements to x → compute rank[x] = p

if rank[x] = p = r, DONE, Else use x as pivot to partition input

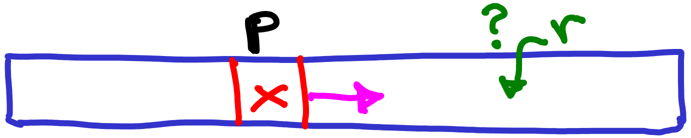
(set up binary search)

if p > r // rank[x] > r, so search lower



Select(r, 1…p-1)

else // p < r, so search higher



we are looking for the element w/ rank r

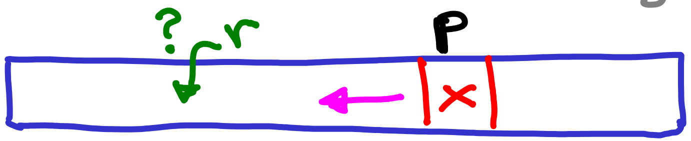
Select(r, 1...n) // find r^{th} smallest # within array[1...n]

we will choose an element x

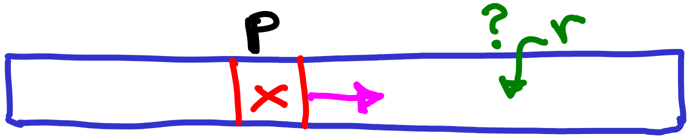
Compare all elements to x → compute rank[x] = p

if rank[x] = p = r , DONE, Else use x as pivot to partition input
(set up binary search)

if p > r // rank[x] > r, so search lower
Select(r, 1...p-1)



else // p < r , so search higher
Select(r-p, p+1...n)



we are looking for the element w/ rank r

Select(r, 1...n) // find r^{th} smallest # within array[1...n]

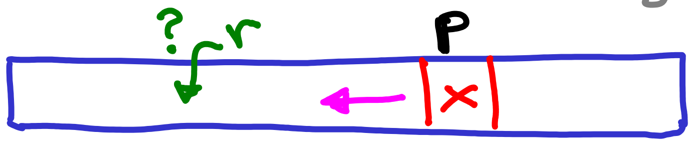
1) Form $\frac{n}{5}$ groups of 5 elements // the last group can have <5

Compare all elements to x → compute rank[x] = p

if rank[x] = p = r , DONE, Else use x as pivot to partition input
(set up binary search)

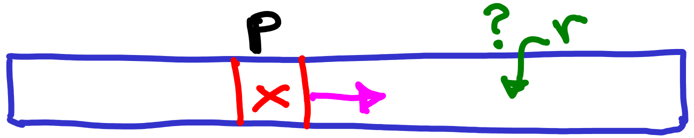
if p > r // rank[x] > r, so search lower

Select(r, 1...p-1)



else // p < r , so search higher

Select(r-p, p+1...n)



we are looking for the element w/ rank r

Select(r, 1…n) // find r^{th} smallest # within array[1…n]

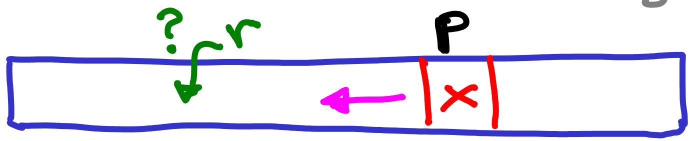
1) Form $\frac{n}{5}$ groups of 5 elements // the last group can have <5
2) Find median in each group // brute force.

Compare all elements to x → compute rank[x] = p

if rank[x] = p = r , DONE, Else use x as pivot to partition input (set up binary search)

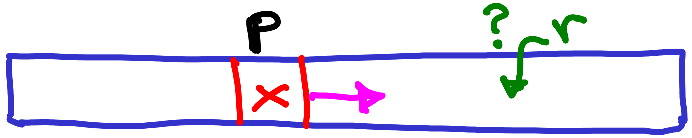
if p > r // rank[x] > r, so search lower

Select(r, 1…p-1)



else // p < r , so search higher

Select(r-p, p+1…n)



we are looking for the element w/ rank r

Select(r, 1...n) // find r^{th} smallest # within array[1...n]

1) Form $\frac{n}{5}$ groups of 5 elements // the last group can have <5

2) Find median in each group // brute force.

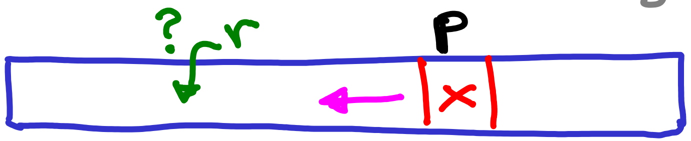
3) Recursively find x = median-of-medians

Compare all elements to x → compute rank[x] = p

if rank[x] = p = r, DONE, Else use x as pivot to partition input (set up binary search)

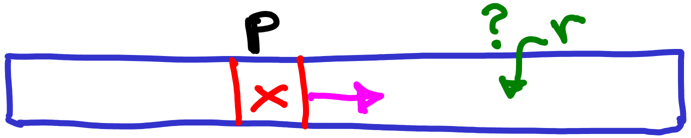
if p > r // rank[x] > r, so search lower

Select(r, 1...p-1)



else // p < r, so search higher

Select(r-p, p+1...n)



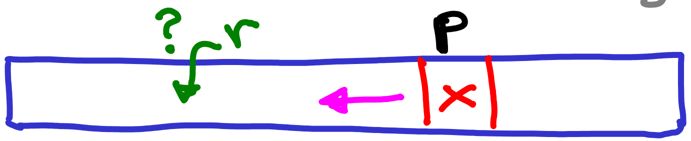
we are looking for the element w/ rank r

Let this run in time T(n)

Select(r, 1…n)   // find r^{th} smallest # within array[1…n]

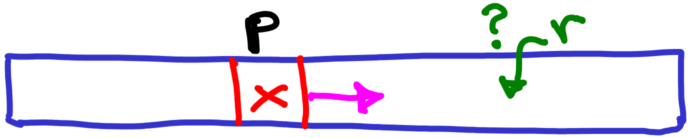
1) Form $\frac{n}{5}$ groups of 5 elements   // the last group can have <5
2) Find median in each group   // brute force.
3) Recursively find x = median-of-medians
4) Compare all elements to x → compute rank[x] = p
5) if rank[x] = p = r , DONE, Else use x as pivot to partition input (set up binary search)
6) if p > r   // rank[x] > r, so search lower

   Select(r, 1…p-1)

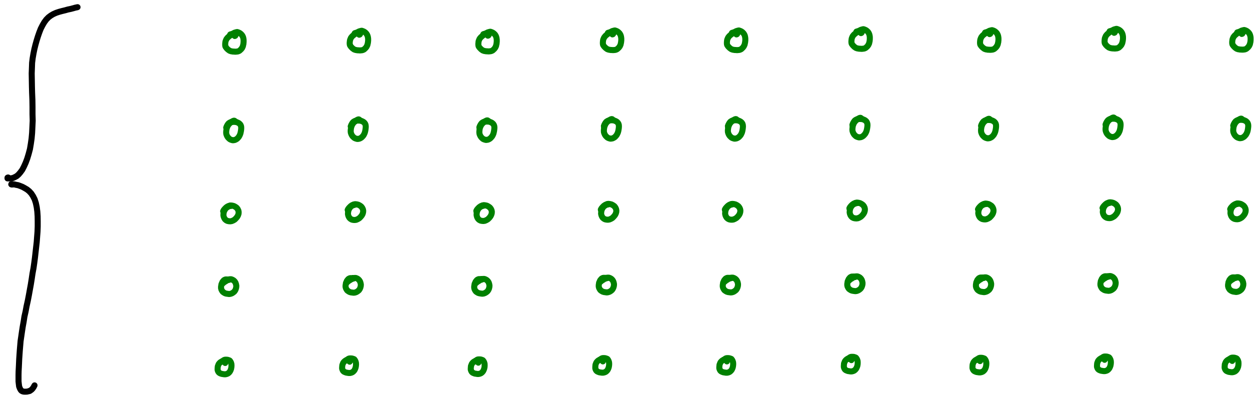


   else   // p < r , so search higher

   Select(r-p, p+1…n)

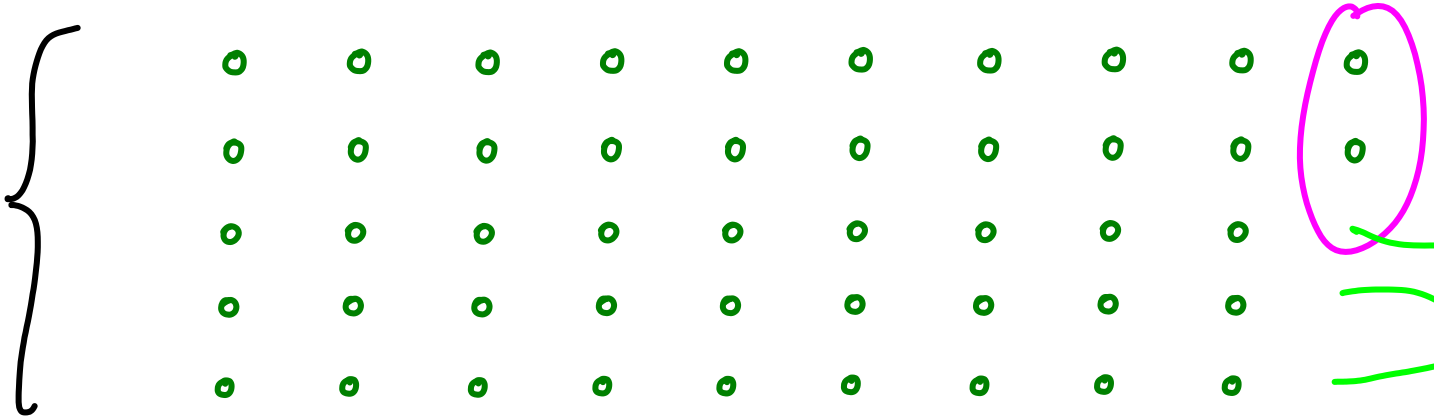


1) Form $\frac{n}{5}$ groups of 5 elements $\Theta(n)$ ... in fact, no work

1) 

1) Form $\frac{n}{5}$ groups of 5 elements $\Theta(n)$ ... in fact, no work

1)



Don't worry about extras

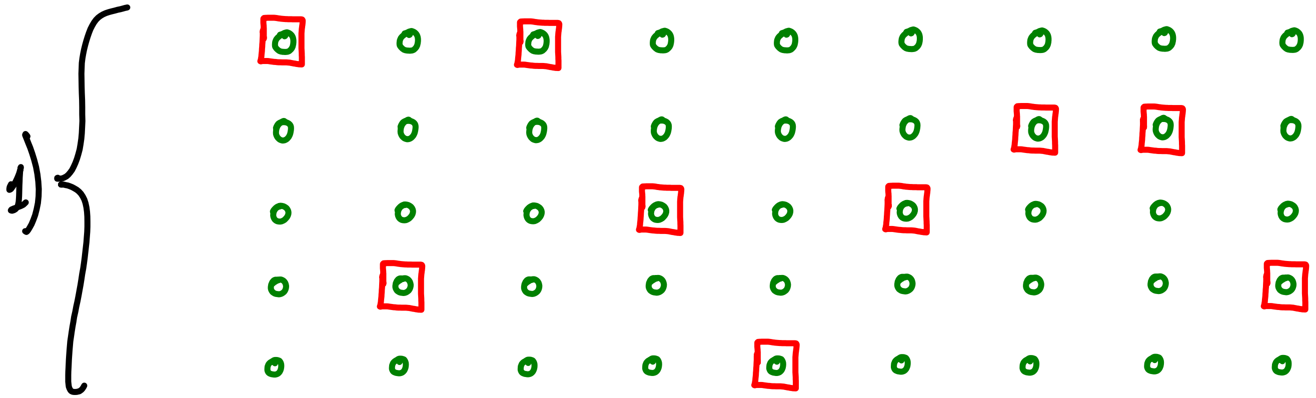
could add three elements $= \infty$

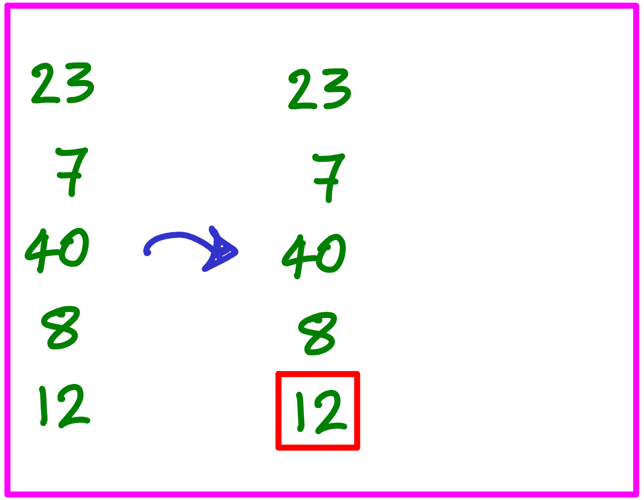
OR

remove MAX & MAX-1 $\Theta(n)$

1) Form $\frac{n}{5}$ groups of 5 elements $\Theta(n)$

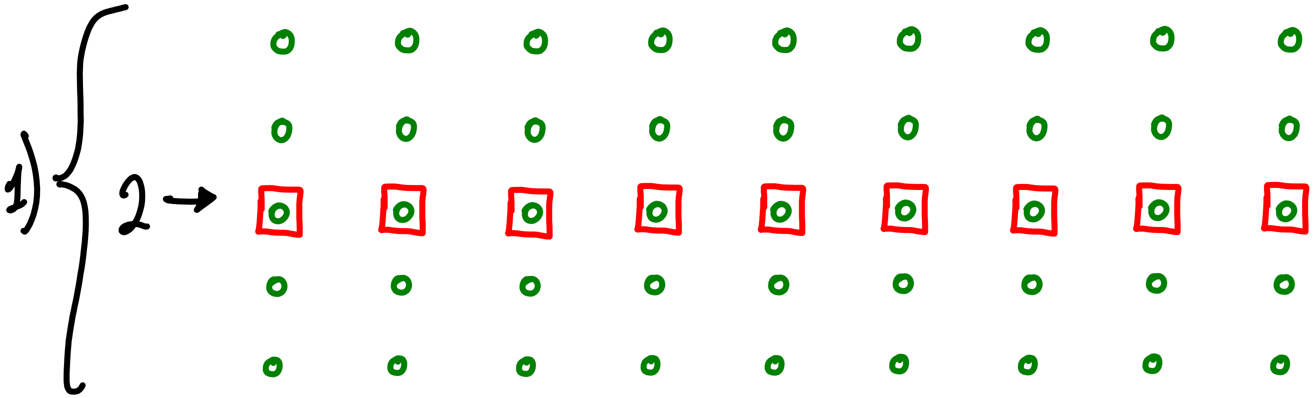
2) Find median in each group $\frac{n}{5} \cdot \Theta(1) = \Theta(n)$

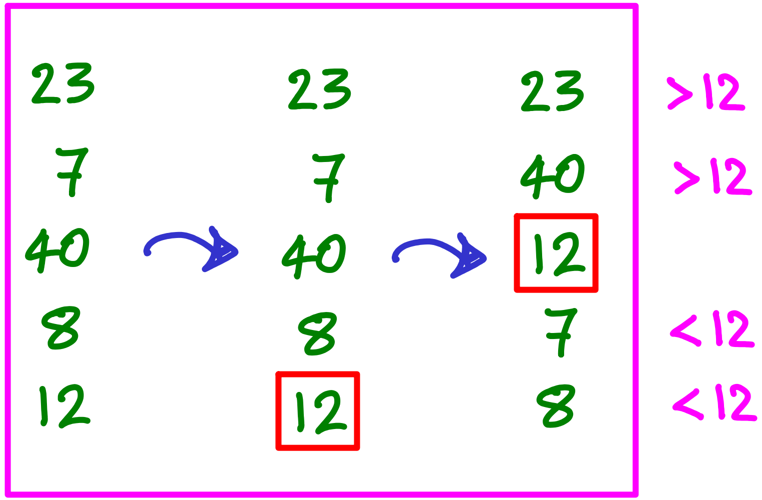




1) Form $\frac{n}{5}$ groups of 5 elements $\Theta(n)$

2) Find median in each group (and re-organize) $\frac{n}{5} \cdot \Theta(1) = \Theta(n)$

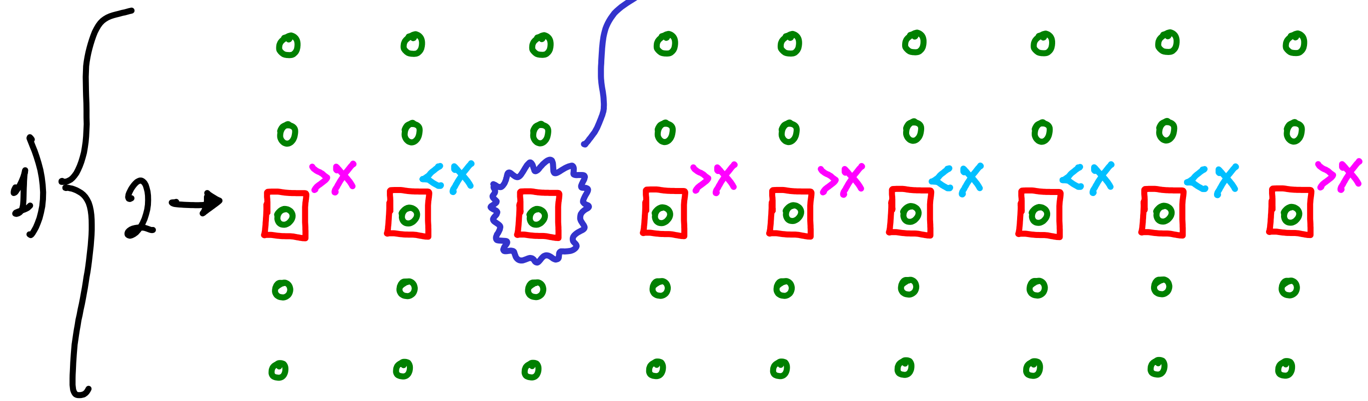




1) Form $\frac{n}{5}$ groups of 5 elements $\Theta(n)$

2) Find median in each group (and re-organize) $\frac{n}{5}\cdot\Theta(1)=\Theta(n)$

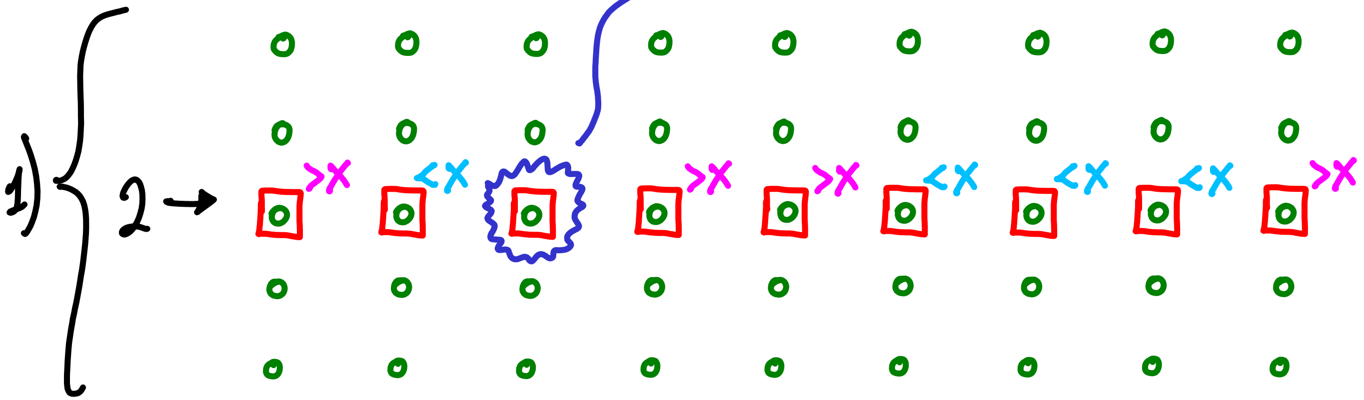
3) Recursively find x = median-of-medians TIME ?



1) Form $\frac{n}{5}$ groups of 5 elements $\Theta(n)$

2) Find median in each group (and re-organize) $\frac{n}{5} \cdot \Theta(1) = \Theta(n)$

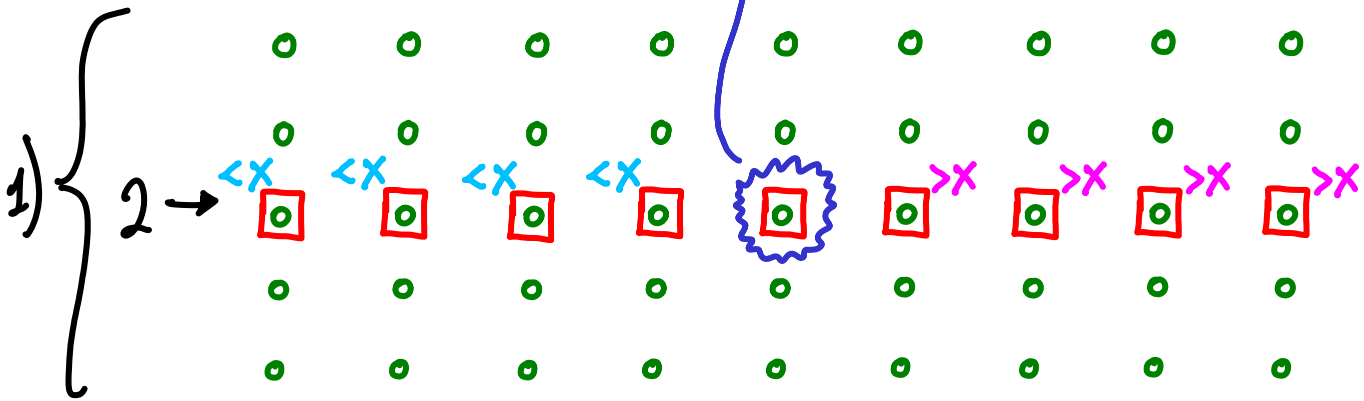
3) Recursively find x = median-of-medians TIME $\longrightarrow T(\frac{n}{5})$



1) Form $\frac{n}{5}$ groups of 5 elements $\Theta(n)$

2) Find median in each group (and re-organize) $\frac{n}{5} \cdot \Theta(1) = \Theta(n)$

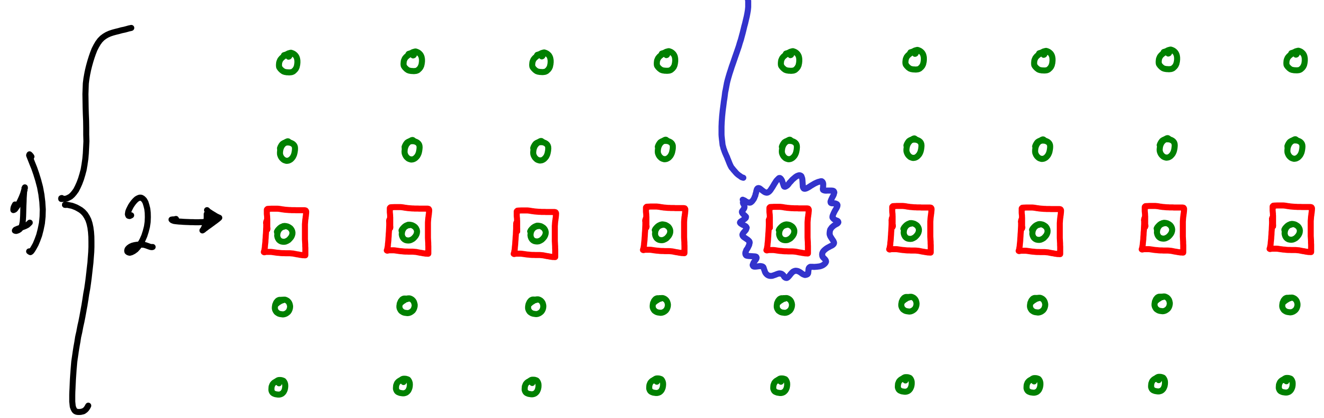
3) Recursively find x = median-of-medians (and re-organize) $T(\frac{n}{5}) + \Theta(n)$



1) Form $\frac{n}{5}$ groups of 5 elements $\Theta(n)$

2) Find median in each group (and re-organize) $\frac{n}{5} \cdot \Theta(1) = \Theta(n)$

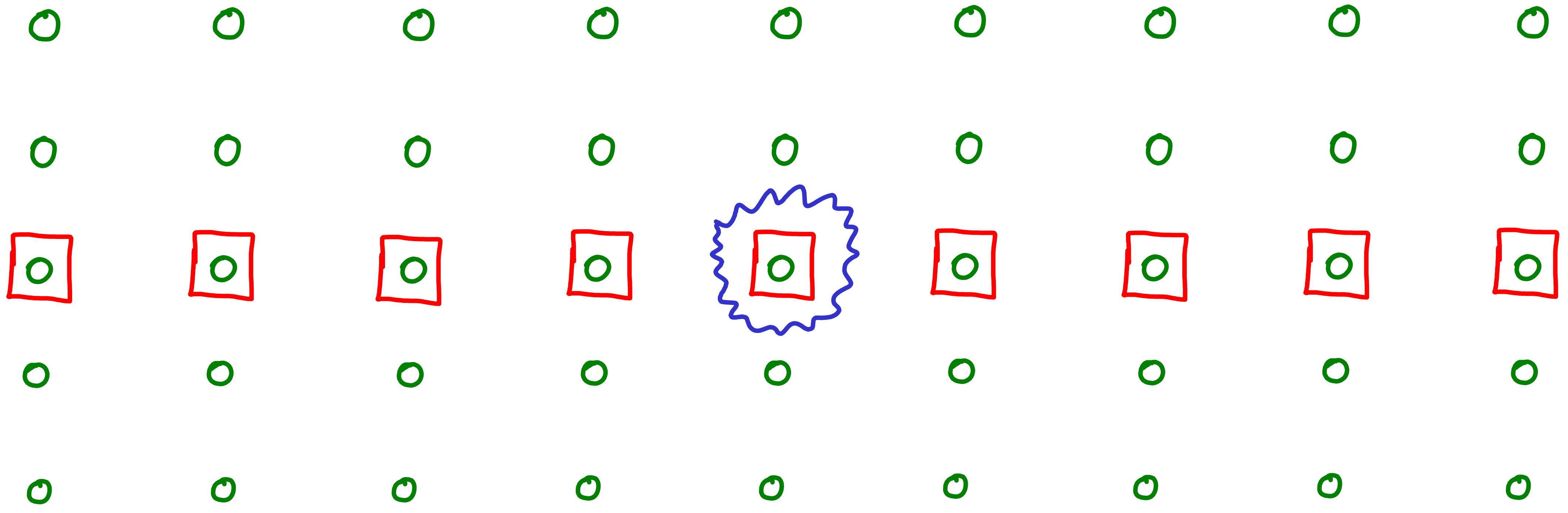
3) Recursively find x = median-of-medians (and re-organize) $T(\frac{n}{5}) + \Theta(n)$



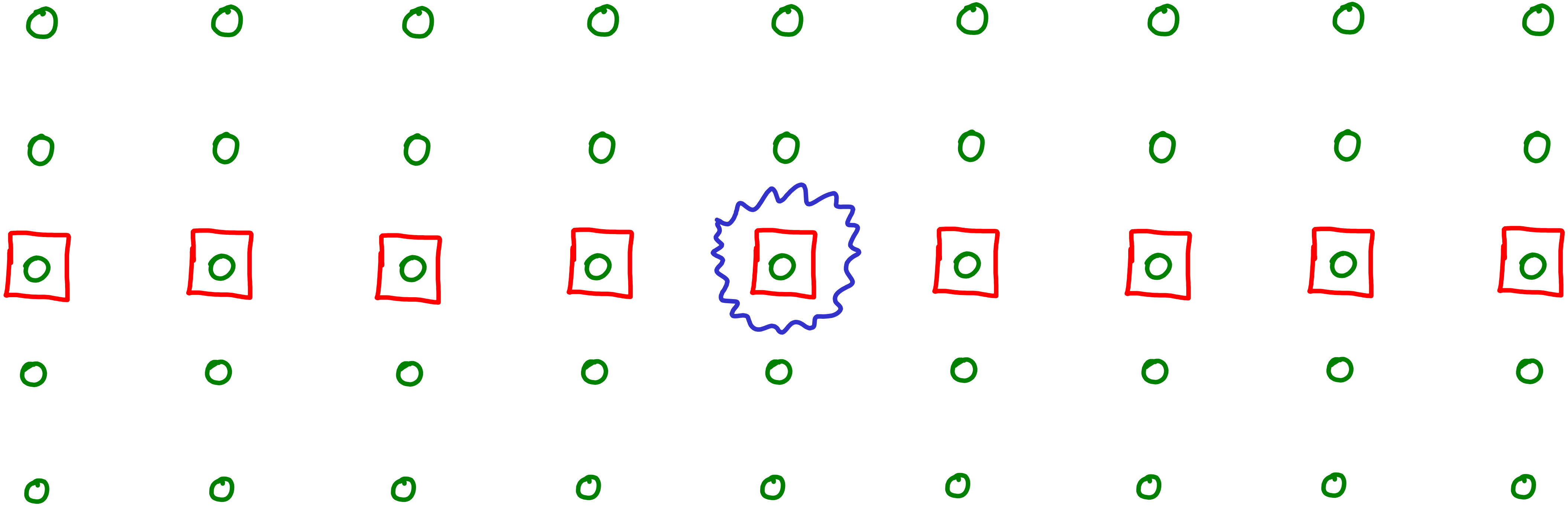
Re-organizing is not part of the algorithm. It's part of the proof.

(although we could afford it)

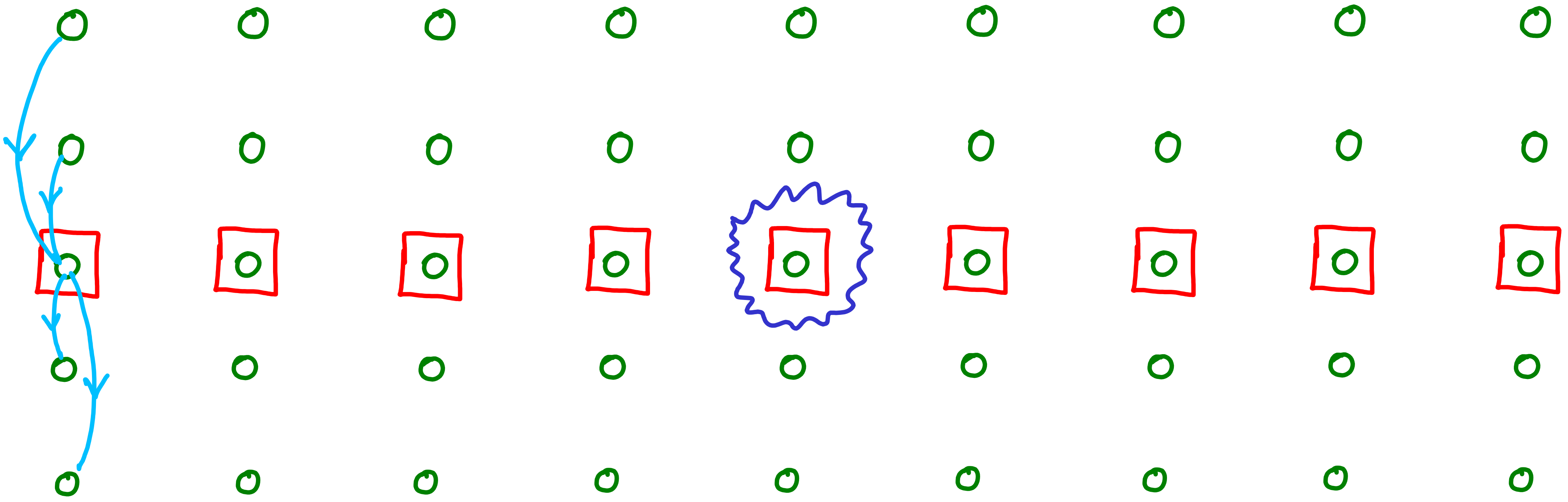
That's the algorithm. Now to find $T(n)$



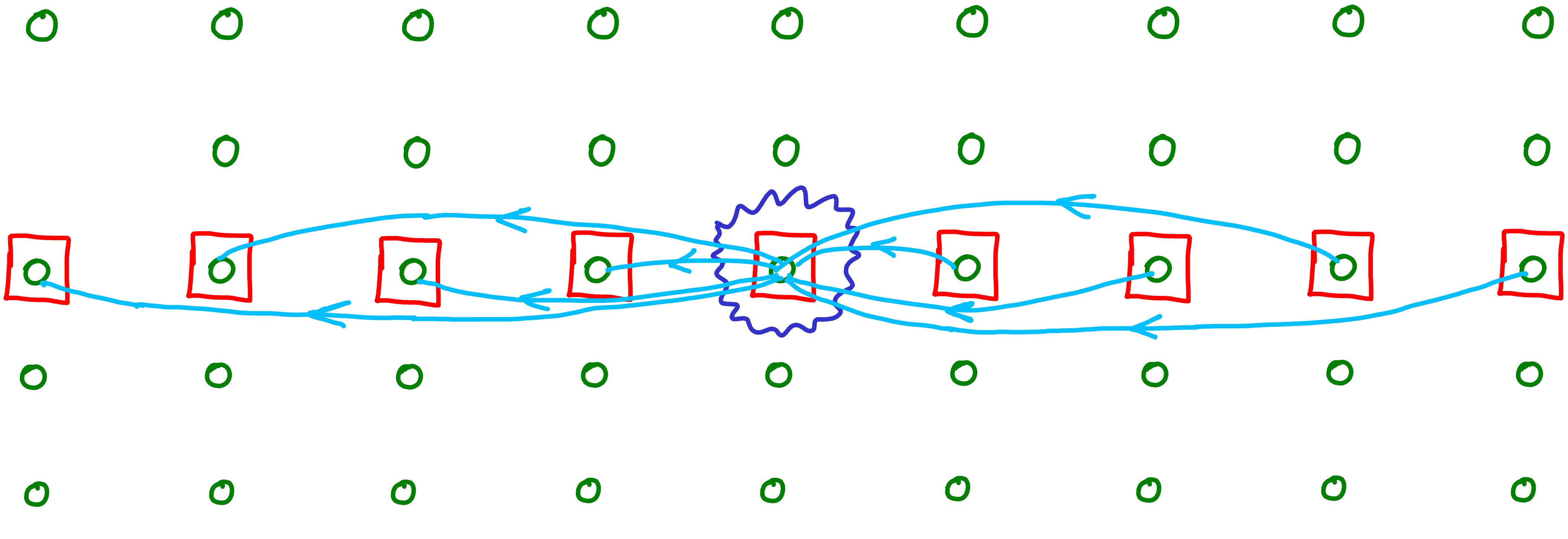
Recap:
- in each column, $\boxed{\circ}$ ranks 3rd : median of 5 : $\Theta(n)$
- among all $\boxed{\circ}$ median is $\boxed{\circ}$: $T(\frac{n}{5})$

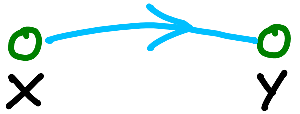


Let $\underset{x}{\circ} \longrightarrow \underset{y}{\circ}$ mean $x > y$

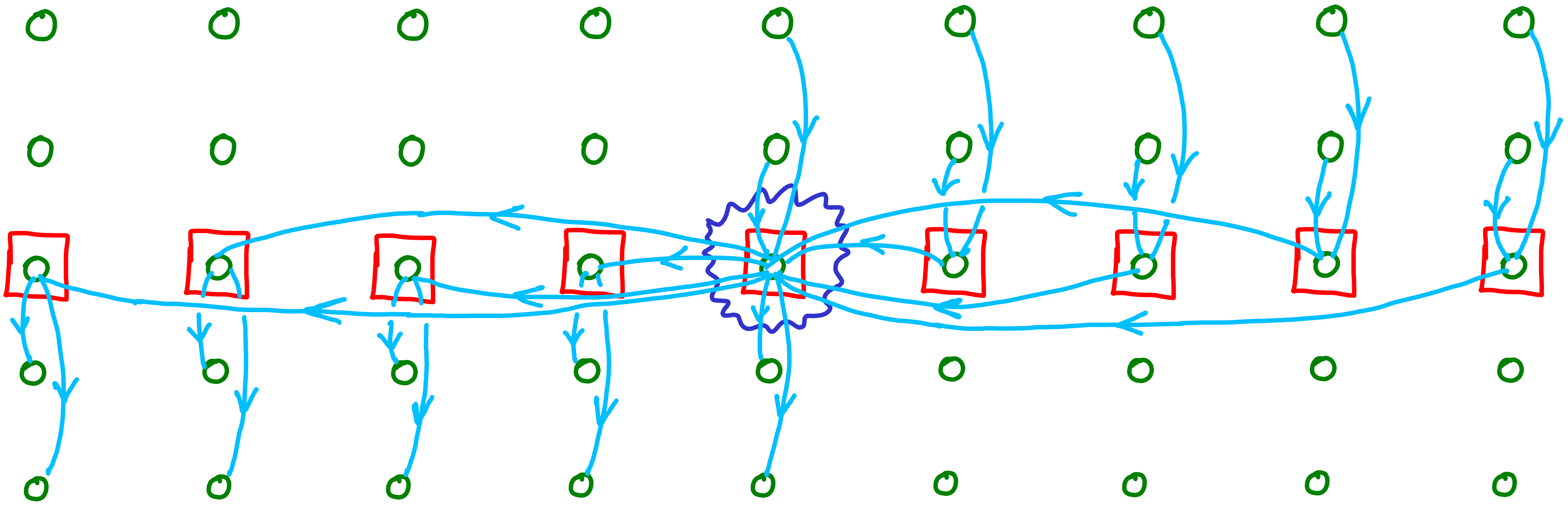


Let $\underset{X}{\circ} \longrightarrow \underset{Y}{\circ}$ mean $X > Y$

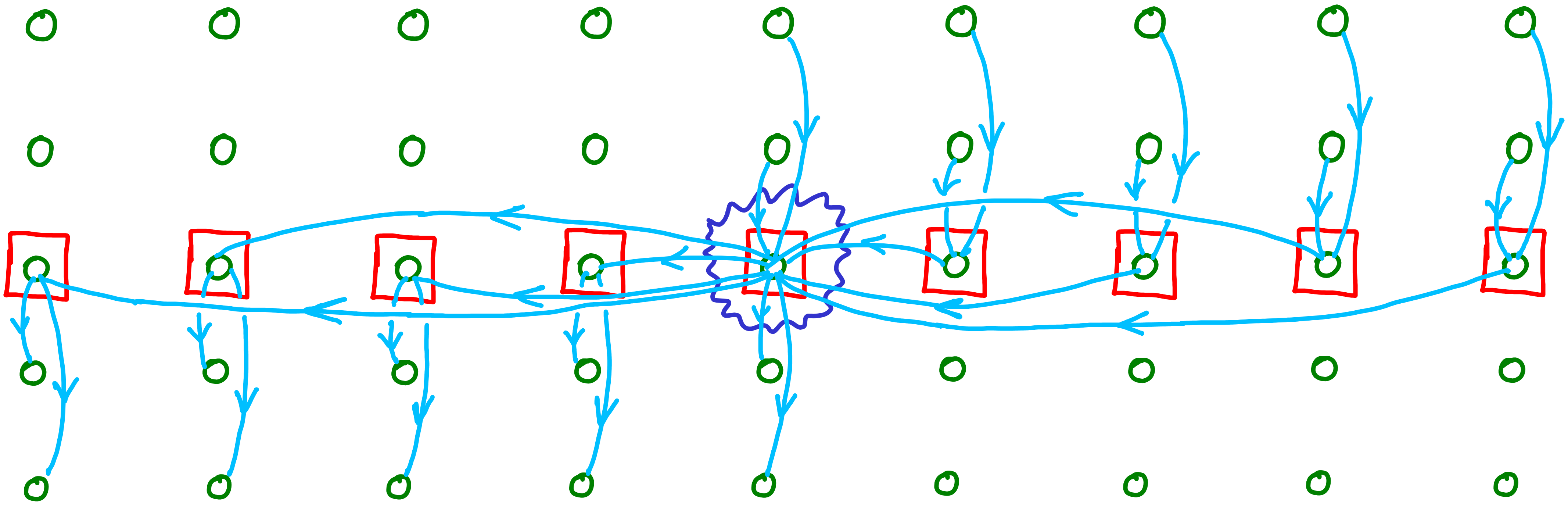


Let 

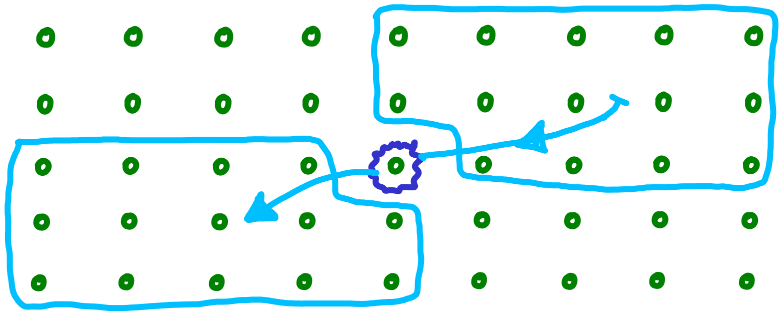
mean $X > Y$

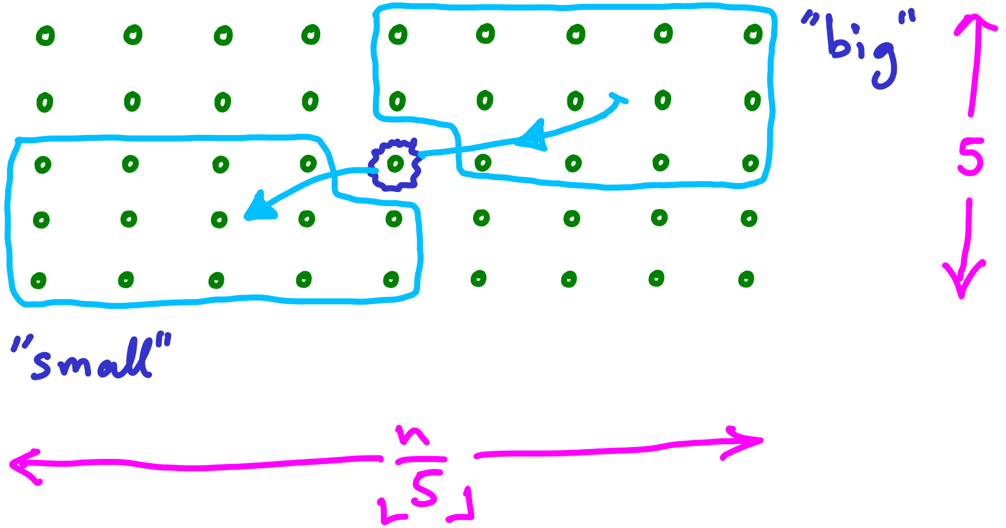


Let $X \longrightarrow Y$ mean $X > Y$

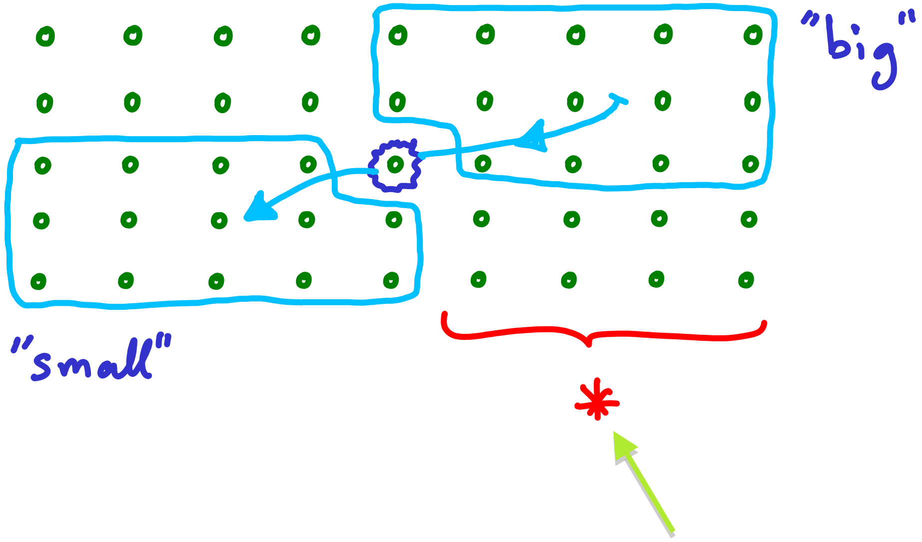


Let $X \to Y$ mean $X > Y$





"big"
5
"small"
$\lfloor \frac{n}{5} \rfloor$

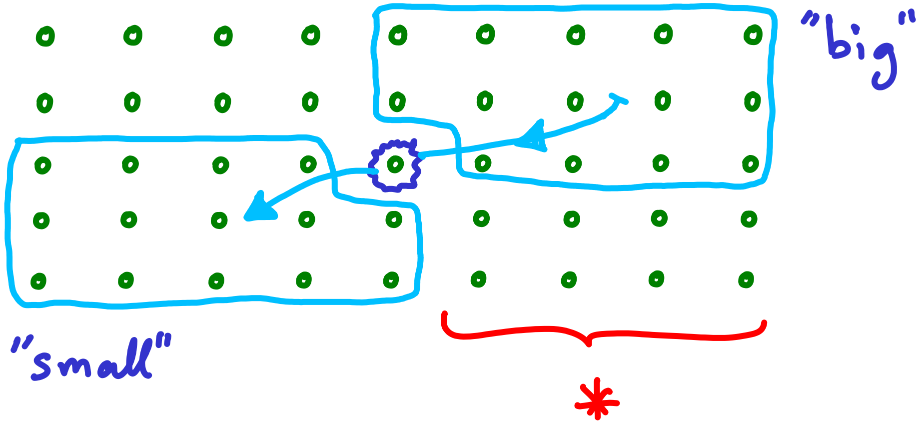


columns containing big elements

$$\#\text{"big"} = \#\text{"small"} \geq 3 \cdot \left\lfloor \frac{\lfloor n/5 \rfloor}{2} \right\rfloor$$

*

big items per column



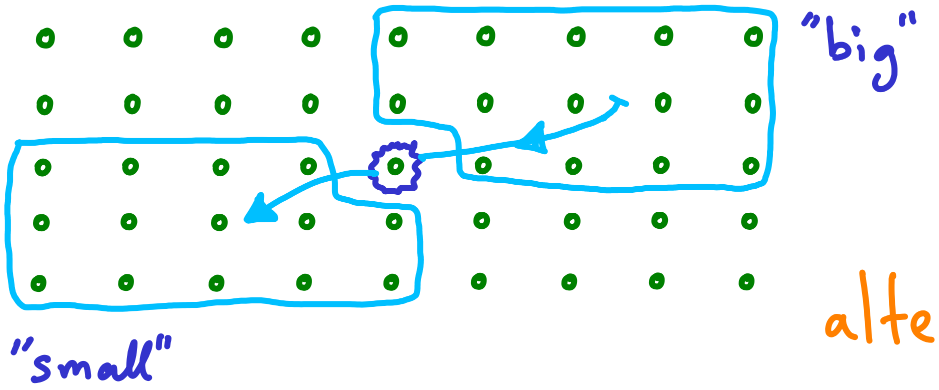
$$\#\text{"big"} = \#\text{"small"} \geq 3 \cdot \left\lfloor \frac{\lfloor n/5 \rfloor}{2} \right\rfloor \geq 3 \cdot \left\lfloor \frac{n}{10} \right\rfloor$$

*

$\left\lfloor \frac{n}{5} \right\rfloor \rightarrow \frac{n}{5}$ if we ignore incomplete column

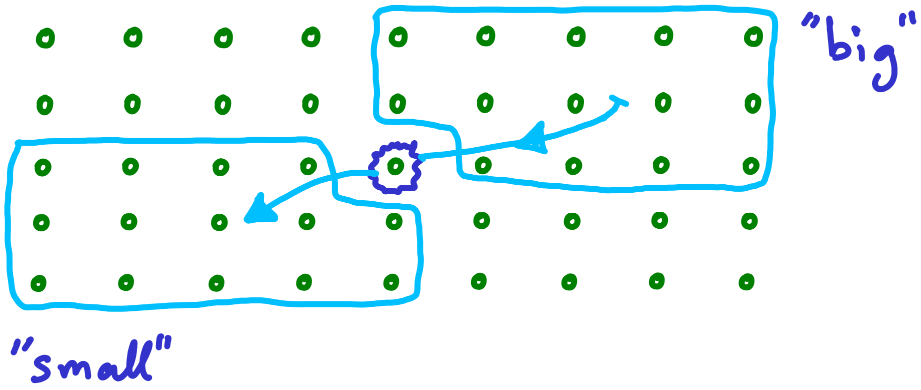
$\Theta(n)$ work takes care of this

$\left\lfloor \frac{n/5}{2} \right\rfloor \rightarrow \frac{n/5}{2}$ if n: even



$$\#\text{"big"} = \#\text{"small"} \geq 3 \cdot \left\lfloor \frac{\lfloor n/5 \rfloor}{2} \right\rfloor \geq 3 \cdot \left\lfloor \frac{n}{10} \right\rfloor$$

alternate analysis: $\geq \frac{1}{4} n$ for $n \geq 50$

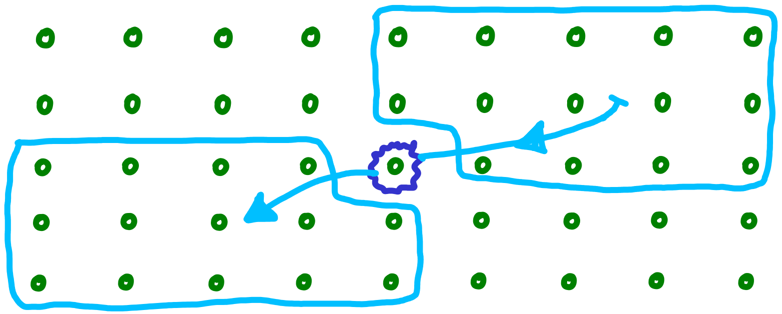


$$\#\text{"big"} = \#\text{"small"} \geq 3 \cdot \left\lfloor \frac{\lfloor n/5 \rfloor}{2} \right\rfloor \geq 3 \cdot \left\lfloor \frac{n}{10} \right\rfloor$$

$$\geq \frac{1}{4} n \quad \text{for } n \geq 50$$

if x is not at the target rank/index, and we need to search lower (i.e., rank(x) > target), then recurse on all elements except "big"

(in the worst case)

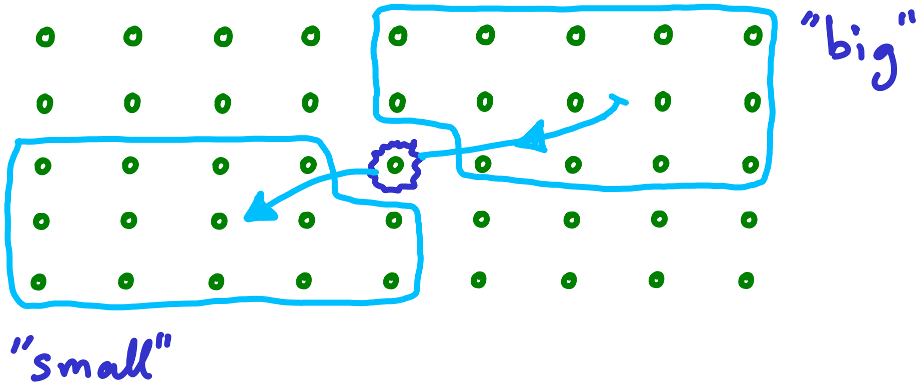


$\#\text{"big"} = \#\text{"small"} \geqslant 3 \cdot \left\lceil \frac{\lfloor n/5 \rfloor}{2} \right\rceil \geqslant 3 \cdot \left\lfloor \frac{n}{10} \right\rfloor$

$\geqslant \frac{1}{4} n$ for $n \geqslant 50$

if x is not at the target rank/index, and we need to search lower (i.e., rank(x) > target), then recurse on all elements except "big"

[symmetrically, if searching for target > rank(x), recurse on all except "small"]

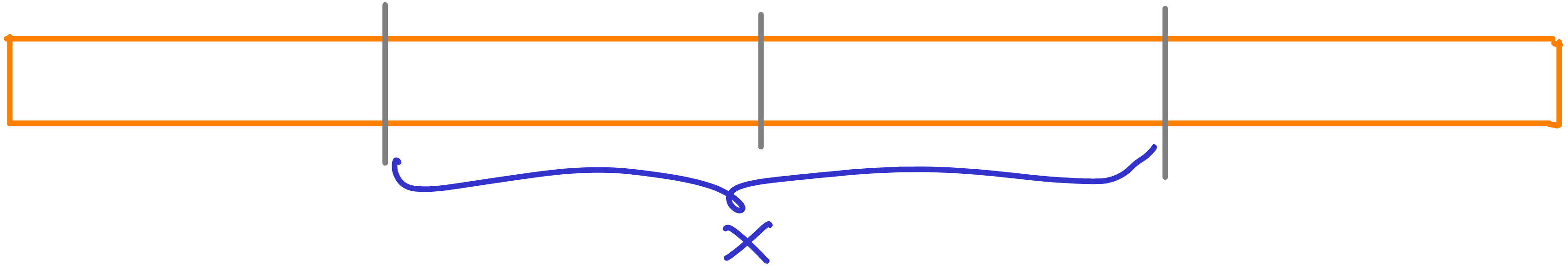


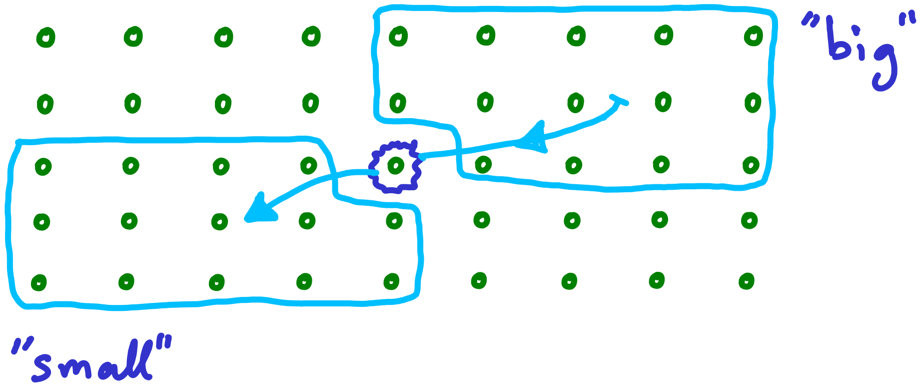
$$\#\text{"big"} = \#\text{"small"} \geq 3 \cdot \left\lfloor \frac{\lfloor n/5 \rfloor}{2} \right\rfloor \geq 3 \cdot \left\lfloor \frac{n}{10} \right\rfloor$$

$$\geq \frac{1}{4} n \quad \text{For } n \geq 50$$

if x is not at the target rank/index, and we need to search lower (i.e., rank(x) > target), then recurse on all elements except "big"

[symmetrically, if searching for target > rank(x), recurse on all except "small"]



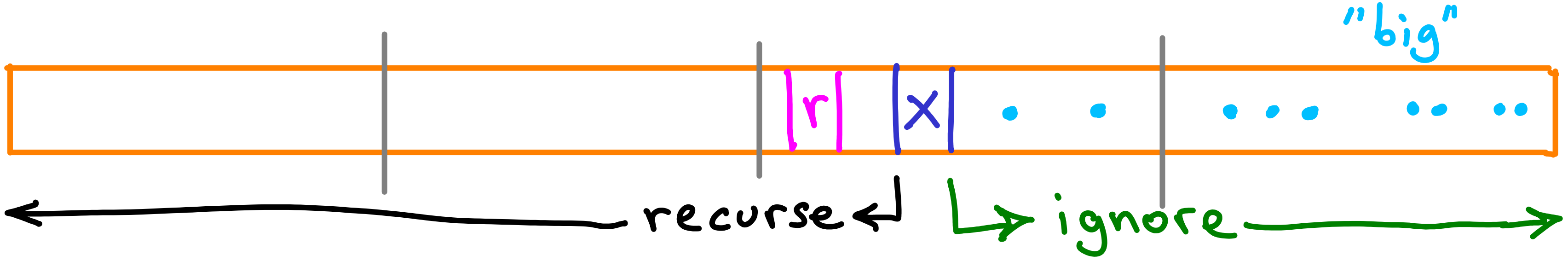


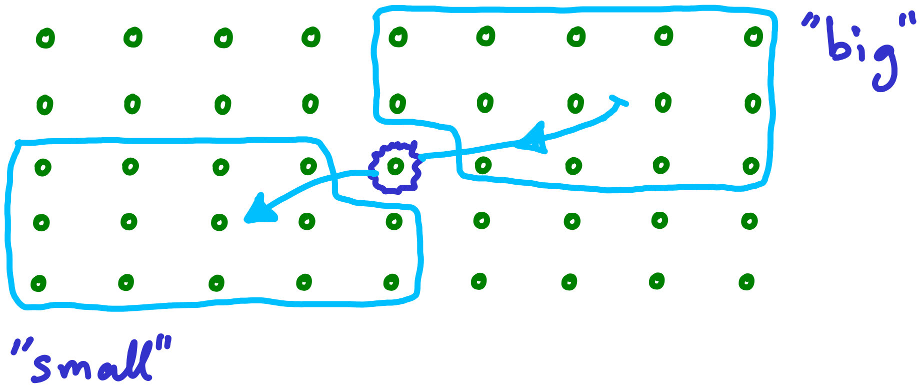
$$\#\text{"big"} = \#\text{"small"} \geqslant 3\cdot\left\lfloor\frac{\lfloor n/5\rfloor}{2}\right\rfloor \geqslant 3\cdot\left\lfloor\frac{n}{10}\right\rfloor$$

$$\geqslant \frac{1}{4}n \quad \text{for } n \geqslant 50$$

if x is not at the target rank/index, and we need to search lower (i.e., rank(x) > target), then recurse on all elements except "big"

[symmetrically, if searching for target > rank(x), recurse on all except "small"]





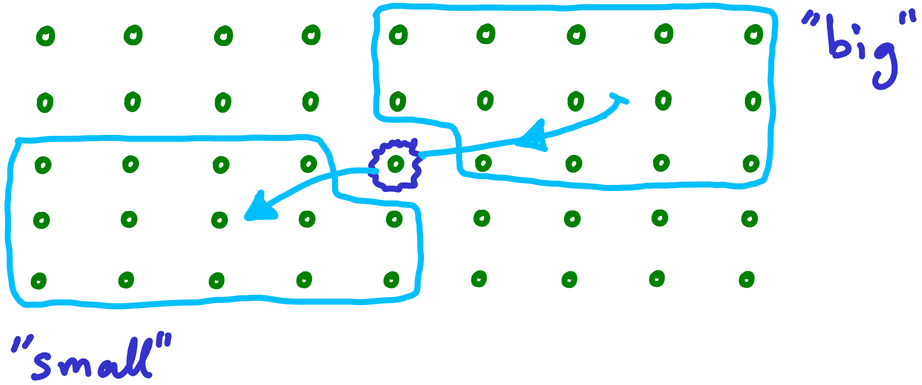
$$\#\text{"big"} = \#\text{"small"} \geq 3 \cdot \left\lfloor \frac{\lfloor n/5 \rfloor}{2} \right\rfloor \geq 3 \cdot \left\lfloor \frac{n}{10} \right\rfloor$$

$$\geq \frac{1}{4} n \quad \text{for } n \geq 50$$

if x is not at the target rank/index, and we need to search lower (i.e., rank(x) > target), then recurse on all elements except "big"

[symmetrically, if searching for target > rank(x), recurse on all except "small"]

$$T(n) = \ldots\ldots\ldots + \underbrace{\Theta(n)}_{\text{steps 1\&2: split into groups \& find medians of 5}}$$



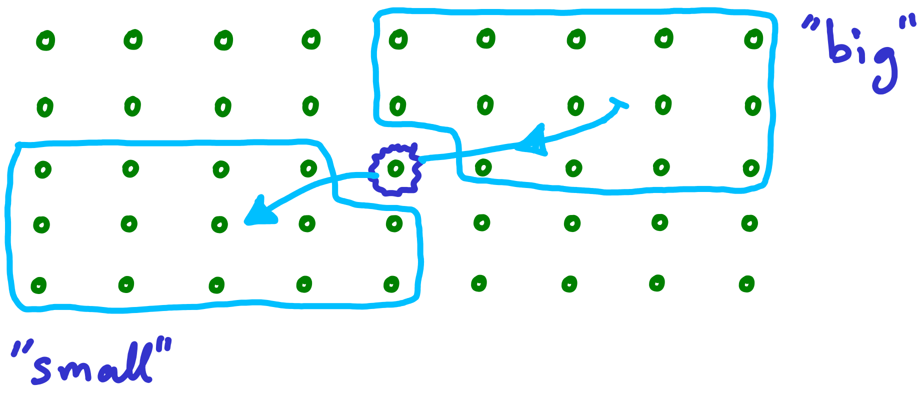
$$\#\text{"big"} = \#\text{"small"} \geq 3\cdot\left\lfloor\frac{\lfloor n/5\rfloor}{2}\right\rfloor \geq 3\cdot\left\lfloor\frac{n}{10}\right\rfloor$$

$$\geq \frac{1}{4}n \quad \text{For } n \geq 50$$

if x is not at the target rank/index, and we need to search lower (i.e., rank(x) > target), then recurse on all elements except "big"

[symmetrically, if searching for target > rank(x), recurse on all except "small"]

$$T(n) = \underbrace{T\left(\frac{n}{5}\right)}_{\text{find } x} + \underbrace{\Theta(n)}_{\text{steps 1\&2: split into groups \& find medians of 5 \& partition}}$$



$$\#\text{"big"} = \#\text{"small"} \geq 3 \cdot \frac{\lceil n/5 \rceil}{\lfloor 2 \rfloor} \geq 3 \cdot \frac{n}{\lfloor 10 \rfloor}$$

$$\geq \frac{1}{4} n \quad \text{For } n \geq 50$$

if x is not at the target rank/index, and we need to search lower (i.e., rank(x) > target), then recurse on all elements except "big"

[symmetrically, if searching for target > rank(x), recurse on all except "small"]

$$T(n) \leq \underbrace{T\left(\frac{n}{5}\right)}_{\text{find } x} + \underbrace{T\left(\frac{3n}{4}\right)}_{\text{recurse if rank}(x) \neq \text{target}} + \underbrace{\Theta(n)}_{\text{steps 1\&2: split into groups \& find medians of 5 \& partition}}$$

$$T(n) \leq T\left(\frac{n}{5}\right) + T\left(\frac{3n}{4}\right) + \Theta(n)$$

Claim $T(n) \leq c \cdot n$

$$T(n) \leq T\left(\frac{n}{5}\right) + T\left(\frac{3n}{4}\right) + \Theta(n)$$

Claim $T(n) \leq c \cdot n$

$$\leq c \cdot \frac{n}{5} + c \cdot \frac{3n}{4} + dn$$

$$T(n) \leq T\left(\frac{n}{5}\right) + T\left(\frac{3n}{4}\right) + \Theta(n)$$

Claim $T(n) \leq c \cdot n$

$$\leq c \cdot \frac{n}{5} + c \cdot \frac{3n}{4} + dn$$

$$= \frac{19}{20} cn + dn$$

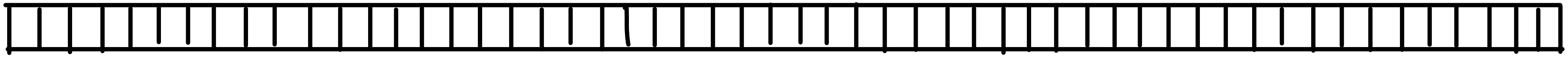
$$T(n) \le T\left(\frac{n}{5}\right) + T\left(\frac{3n}{4}\right) + \Theta(n)$$

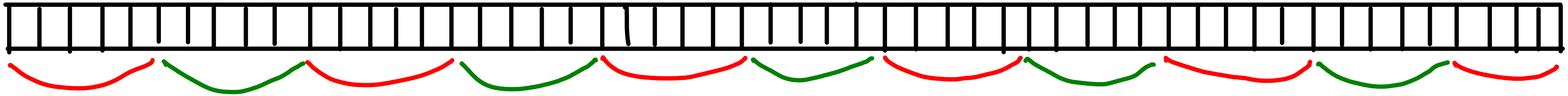
Claim $T(n) \le c \cdot n$

$$\le c \cdot \frac{n}{5} + c \cdot \frac{3n}{4} + dn$$

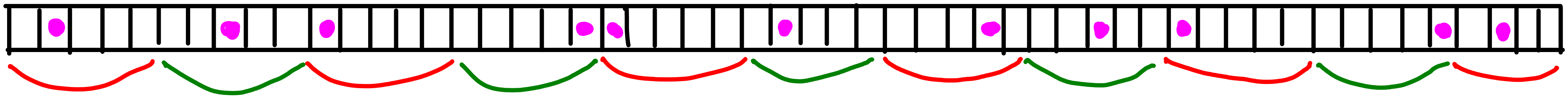
$$= \frac{19}{20} cn + dn = cn - \left(\frac{1}{20} cn - dn\right) \le cn \text{ if } c > 20d$$

QED

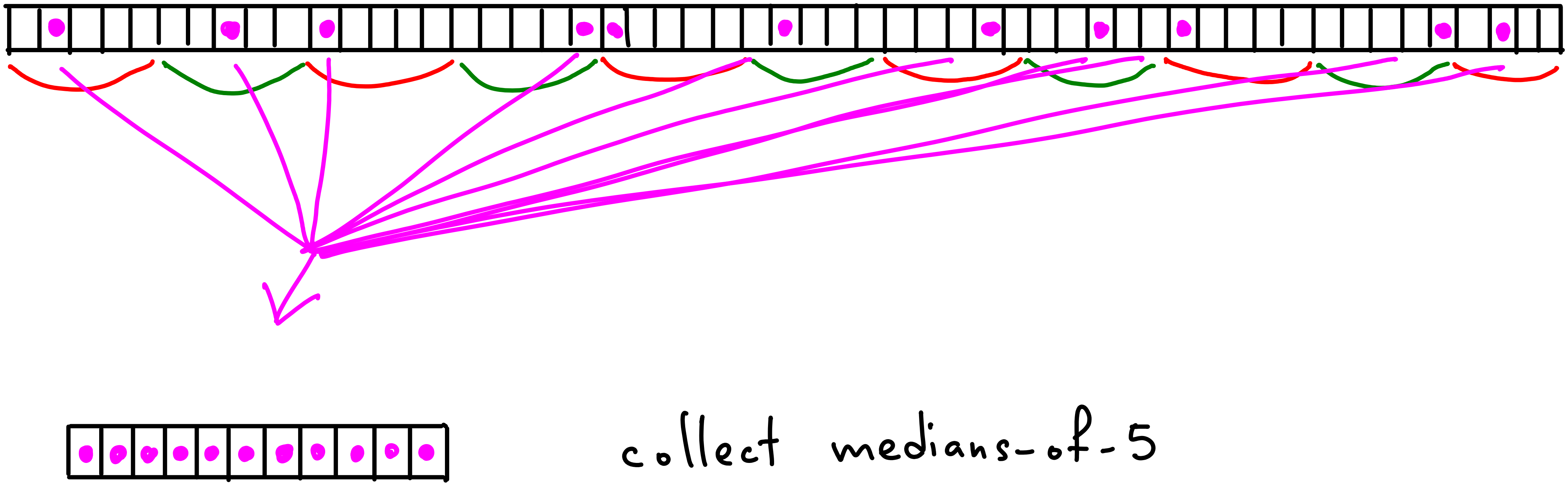


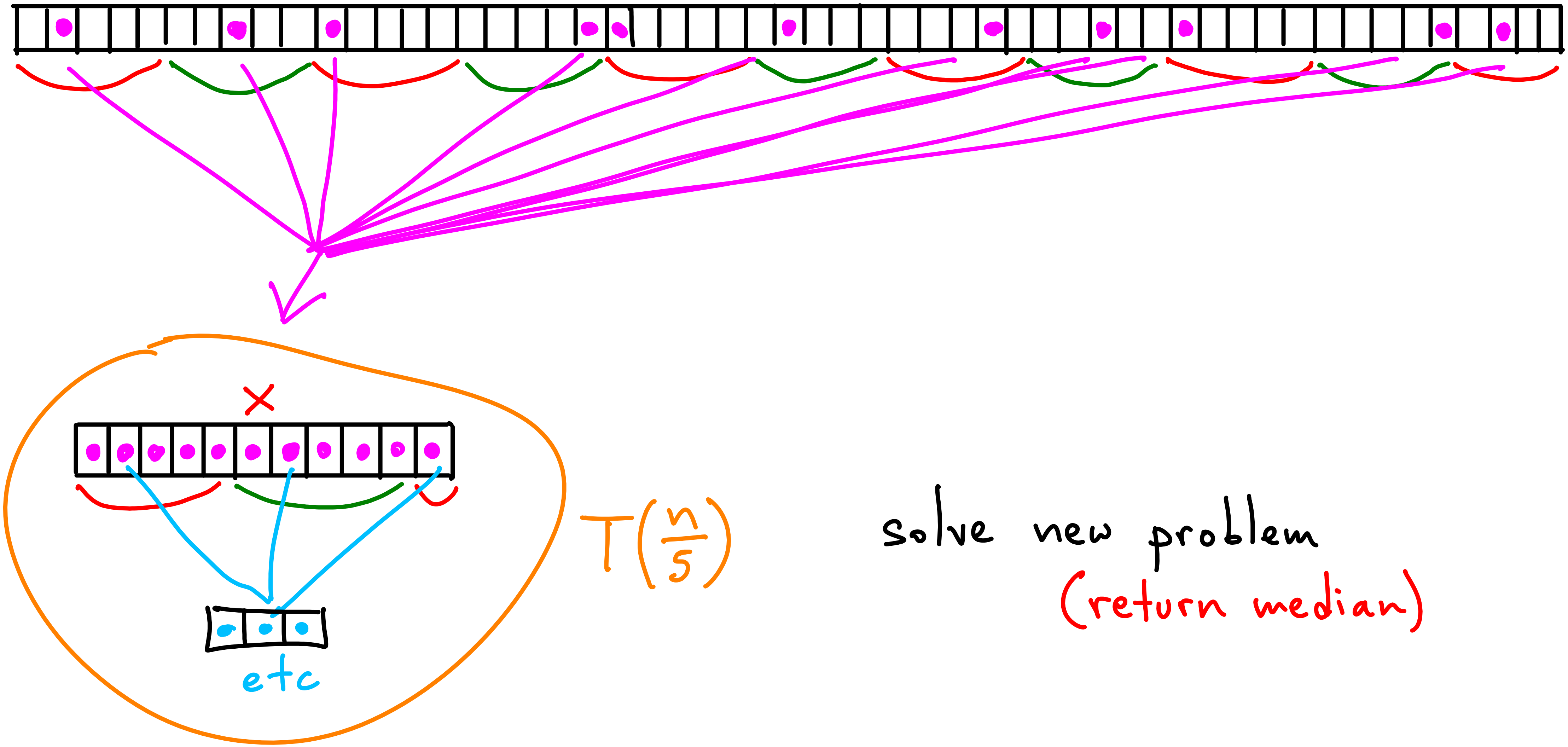


groups of 5

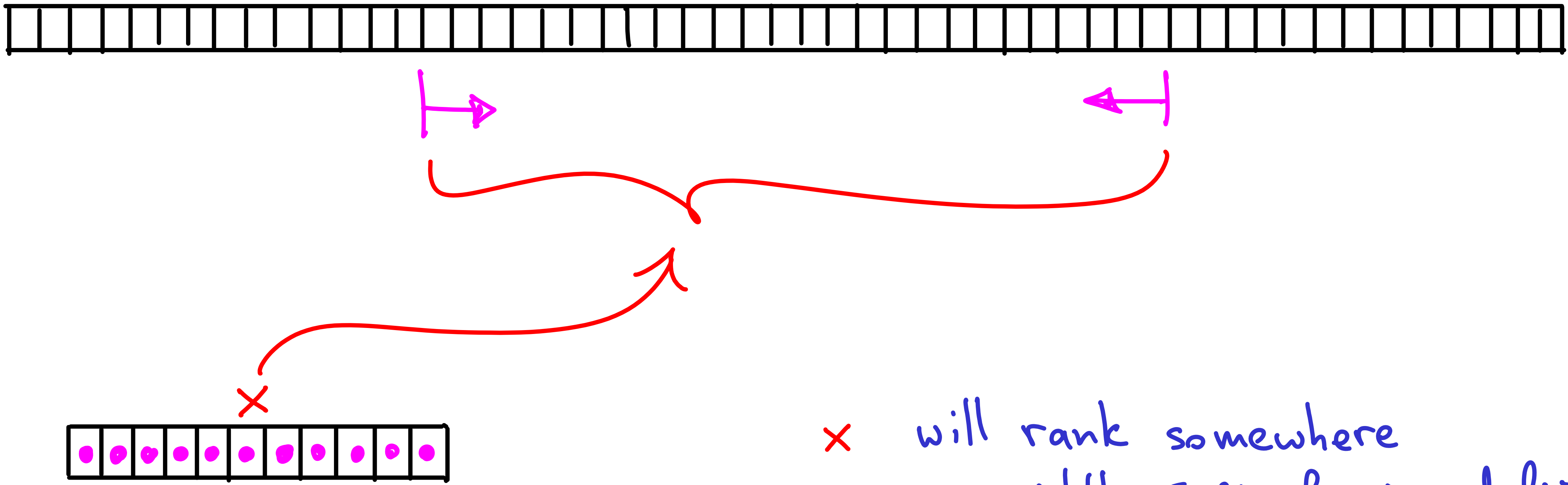


median within each group

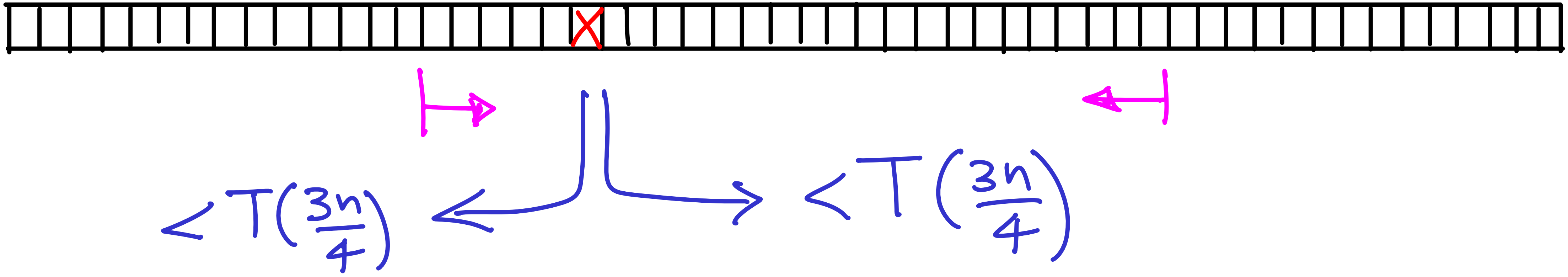

collect medians-of-5

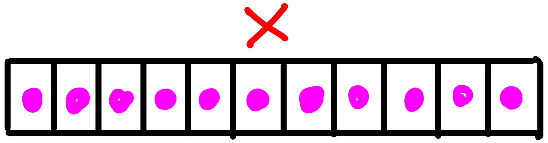


solve new problem
(return median)



× will rank somewhere
in middle 50% of original list





Find rank(x)

if $x \neq$ target, recurse on $< \frac{3}{4}$ of list

What were they thinking?

What were they thinking?

- Goal: $\Theta(n)$ [$\Omega(n)$ lower bound; $O(n\log n)$ is trivial]

What were they thinking?

- Goal: $\Theta(n)$ [$\Omega(n)$ lower bound; $O(n\log n)$ is trivial]

- Exploit ~geometric series: $T(n) = T(\frac{n}{b}) + O(n)$

What were they thinking?

- Goal: $\Theta(n)$ [$\Omega(n)$ lower bound; $O(n \log n)$ is trivial]
- Exploit ~geometric series: $T(n) = T\left(\frac{n}{b}\right) + O(n)$

OR $T(n) = T(xn) + T(yn) + O(n)$

... where $x+y<1$

What were they thinking? (my guess)

- Goal: $\Theta(n)$ [$\Omega(n)$ lower bound; $O(n \log n)$ is trivial]
- Exploit ~geometric series: $T(n) = T(\frac{n}{b}) + O(n)$

  OR $T(n) = T(xn) + T(yn) + O(n)$

  ... where $x + y < 1$

→ spend $T(xn) + O(n)$ time

to make sure that only yn candidates remain