Selection

Selection

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

- if $k = 1$, minimum
- if $k = \lceil |S|/2 \rceil$, median

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

Randomized divide-and-conquer algorithm

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

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

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

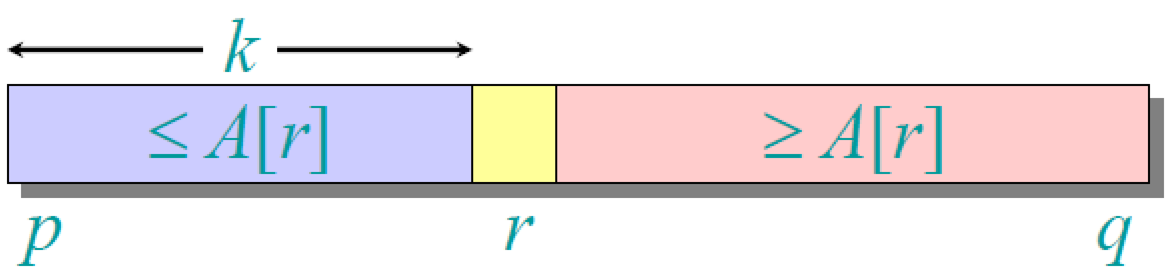
$k \leftarrow r - p + 1$ $\triangleright$ $k = \text{rank}(A[r])$

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 analysis

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

arithmetic series

Worse than sorting!

Best-case

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

- What if 9/10 : 1/10 split?
  - $T(n) = T(9n/10) + O(n)$

    $= O(n)$

Average-case

- Let's say that a pivot v is *good* if it lies within the 25th to 75th percentile of the array.
- It reduces the size of the subarray to at most 3/4 of the size of the array.
- A randomly chosen pivot has a 50% chance of being good.
- After two split operations on average, the array will shrink to at most ¾ of its size.
- Time taken on an array of size n $\leq$

(time taken on an array of size $3n/4$)+ (time to reduce array size to $\leq$ $3n/4$)

- Let $T(n)$ be the expected running time on an array of size n,

$$T(n) \leq T(3n/4) + O(n)$$

Analysis of expected time

- The analysis follows that of randomized quicksort, but it's a little different.
- Let T(*n*) = the random variable for the running time of RAND-SELECT on an input of size *n*, assuming random numbers are independent.
- For *k*= 0, 1, …, *n*–1, define the indicator random variable

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

Analysis

- To obtain an upper bound, assume that the *ith element always falls in the larger side of the partition:*

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

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

Analysis

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

Take expectations of both sides.

Analysis

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

Linearity of expectation.

Analysis

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

Independence of X_k from other random choices.

Analysis

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

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

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

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

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

Analysis

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

Upper terms appear twice.

Analysis

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

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: $$\sum_{k=\lfloor n/2 \rfloor}^{n-1} k \le \frac{3}{8} n^2$$

Substitution method

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

Substitute inductive hypothesis.

Substitution method

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

$$\le \frac{2c}{n}\left(\frac{3}{8}n^2\right) + \Theta(n)$$

Use fact.

Substitution method

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

Express as ***desired* – *residual***.

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*.

Worst-case linear-time selection algorithm

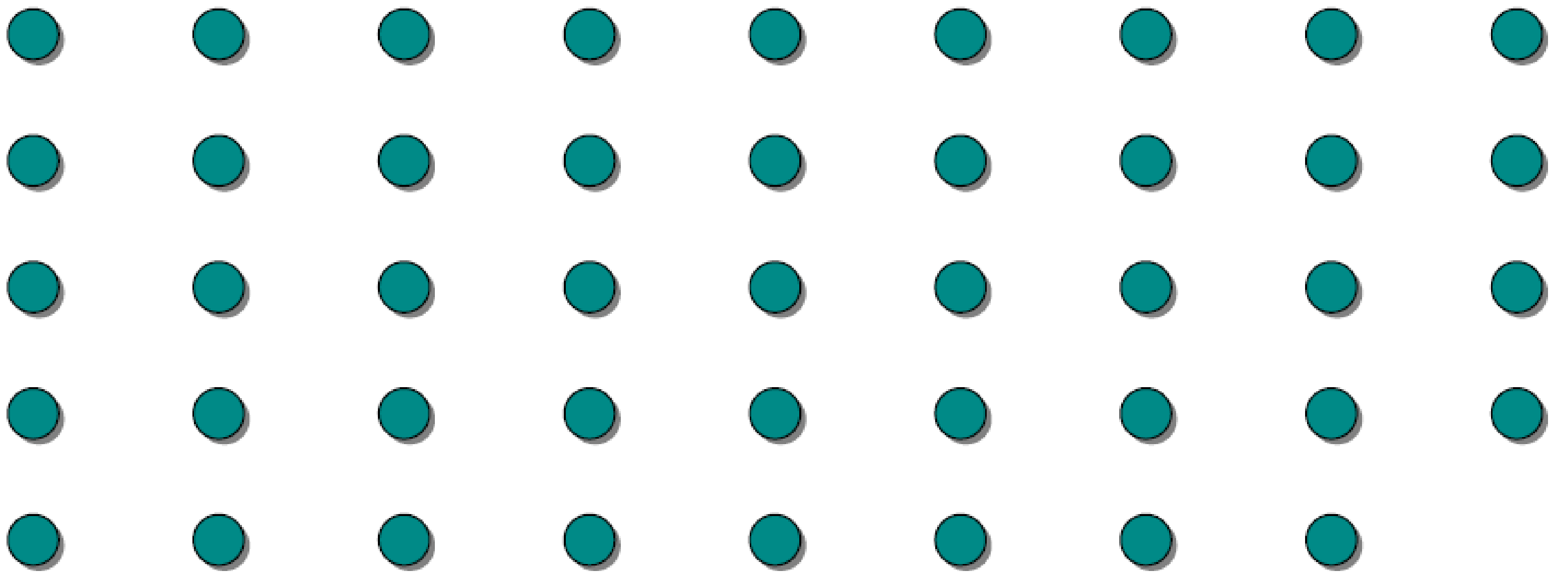
```
SELECT(A[1..n], k):
  if n ≤ 25
      use brute force
  else
      m ← ⌈n/5⌉
      for i ← 1 to m
          B[i] ← SELECT(A[5i − 4..5i], 3)        ⟨⟨Brute force!⟩⟩
      mom ← SELECT(B[1..m], ⌊m/2⌋)               ⟨⟨Recursion!⟩⟩

      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
```

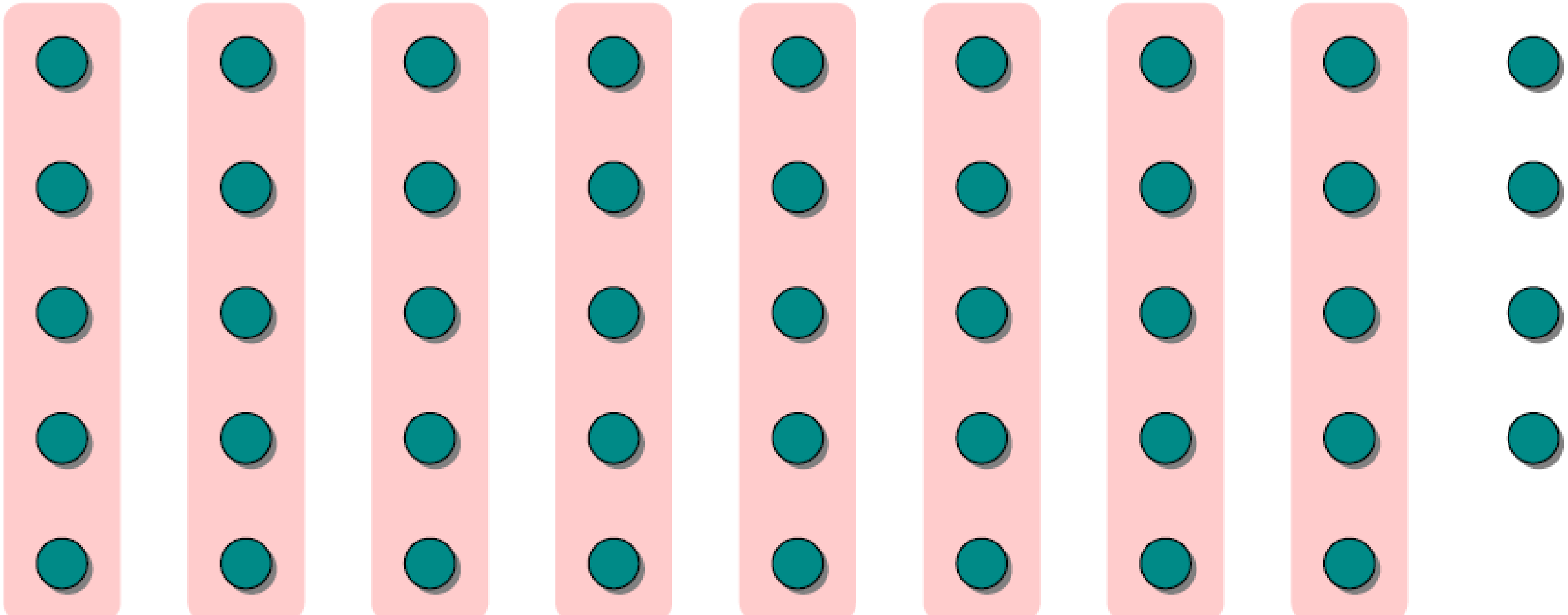
Algorithm

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

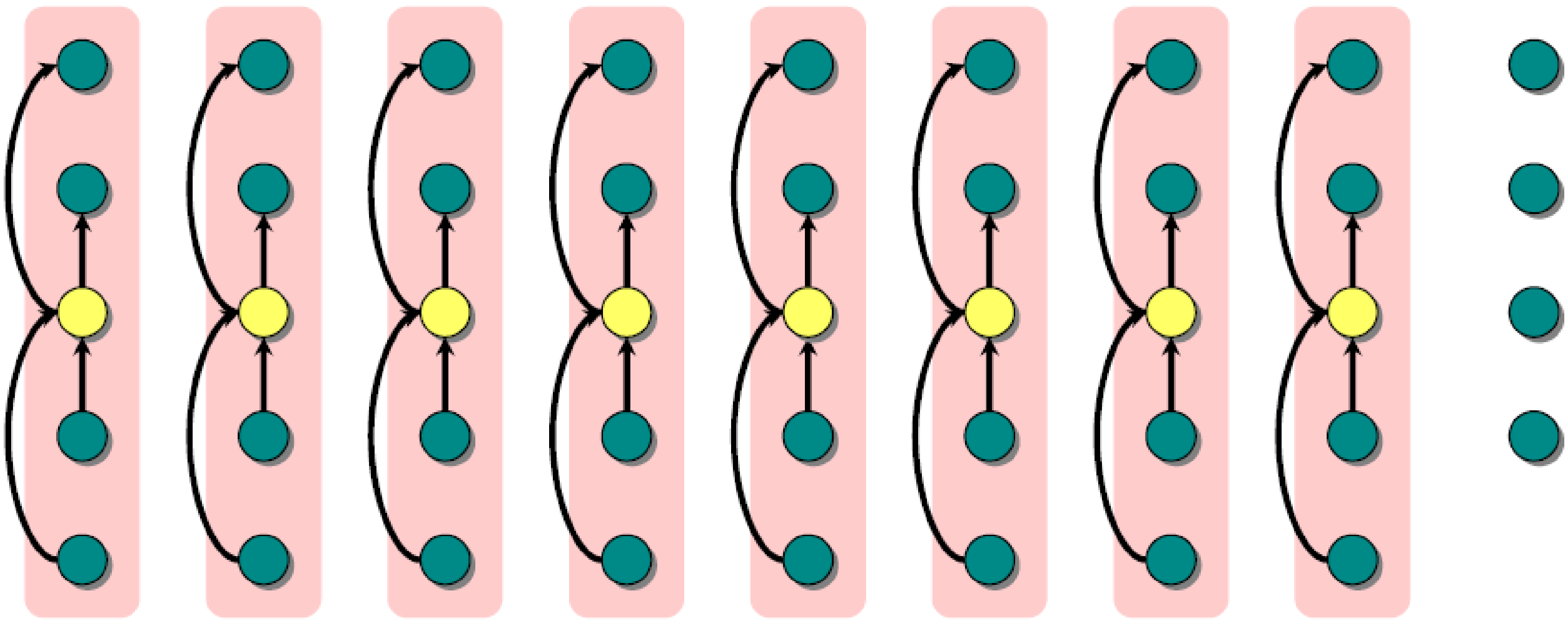
   else if $k > r$ then recursively SELECT the(k–r)th smallest element in the upper part

   else return *mom*

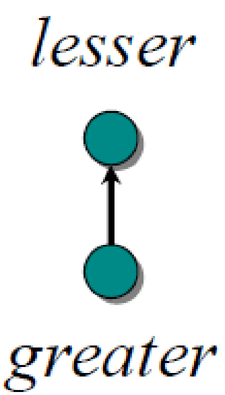


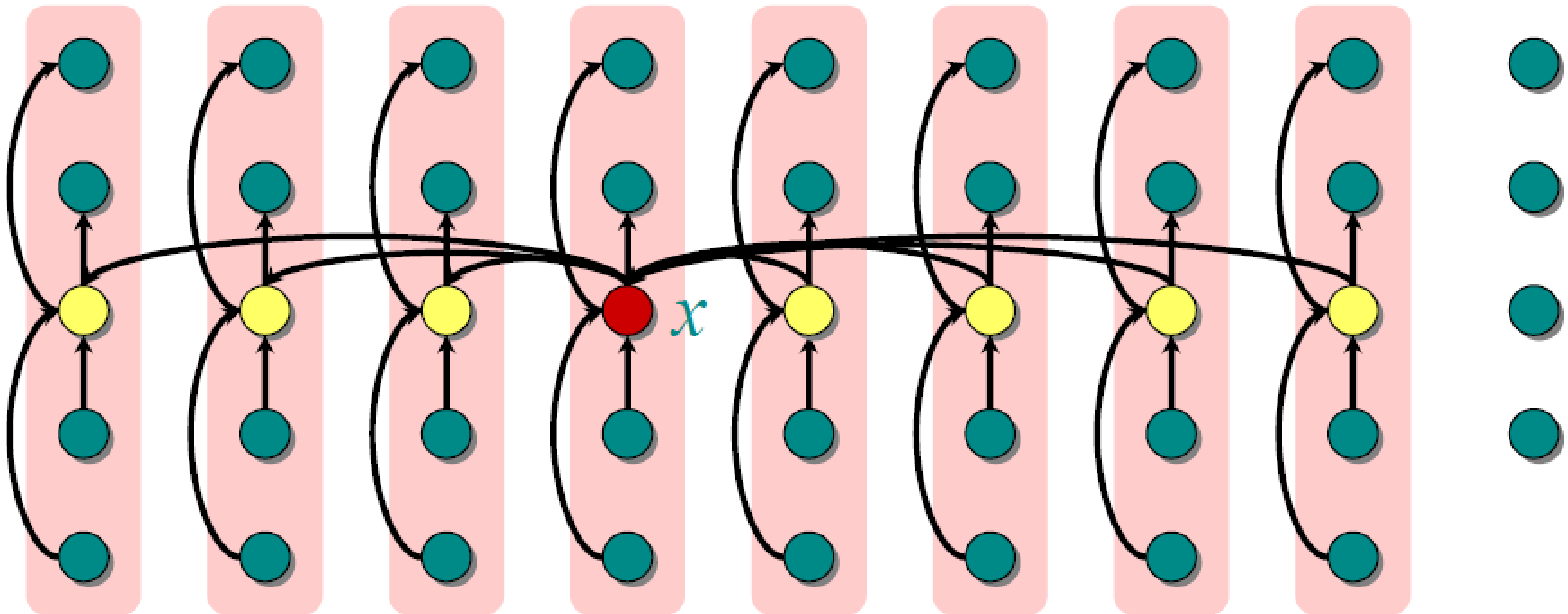


1. Divide the n elements into groups of 5.

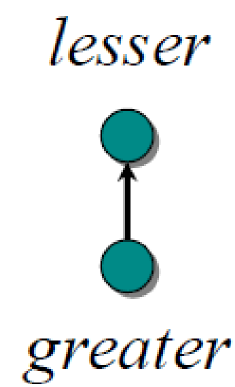


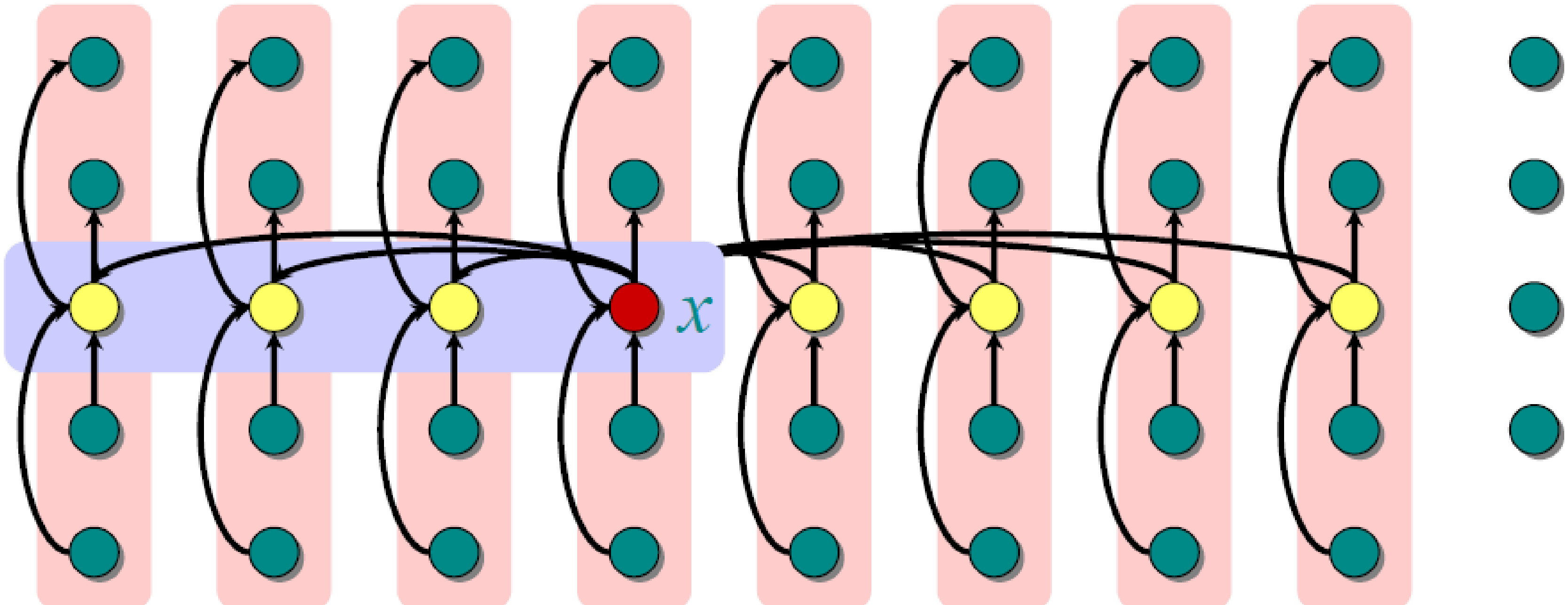
1. Divide the n elements into groups of 5. Find the median of each 5-element group by rote.



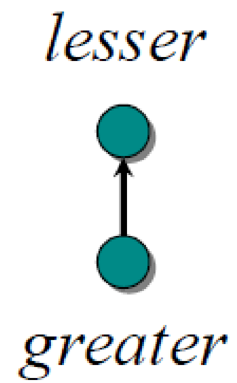


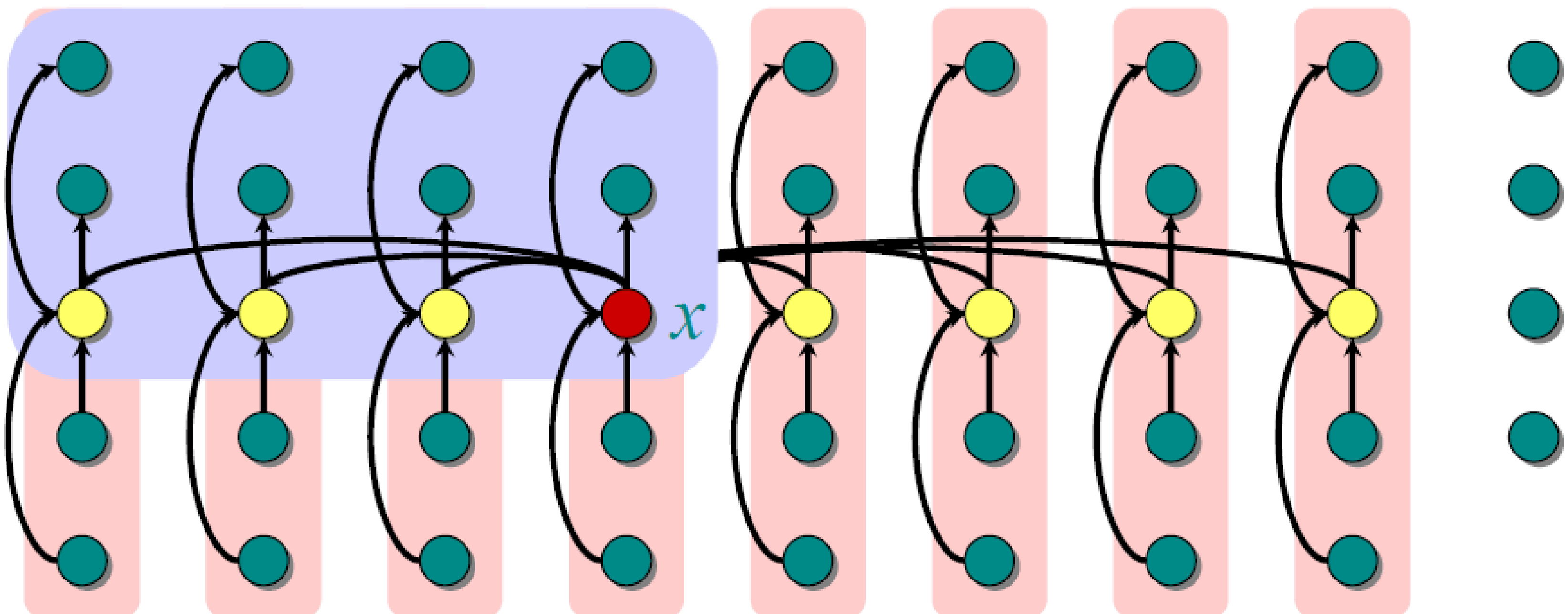
1. Divide the n elements into groups of 5. Find the median of each 5-element group by rote.
2. Recursively SELECT the median x of the $\lfloor n/5 \rfloor$ group medians to be the pivot.





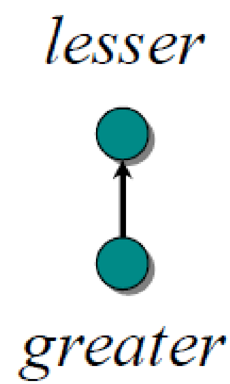
At least half the group medians are $\leq x$, which is at least $\lfloor \lfloor n/5 \rfloor /2 \rfloor = \lfloor n/10 \rfloor$ group medians.

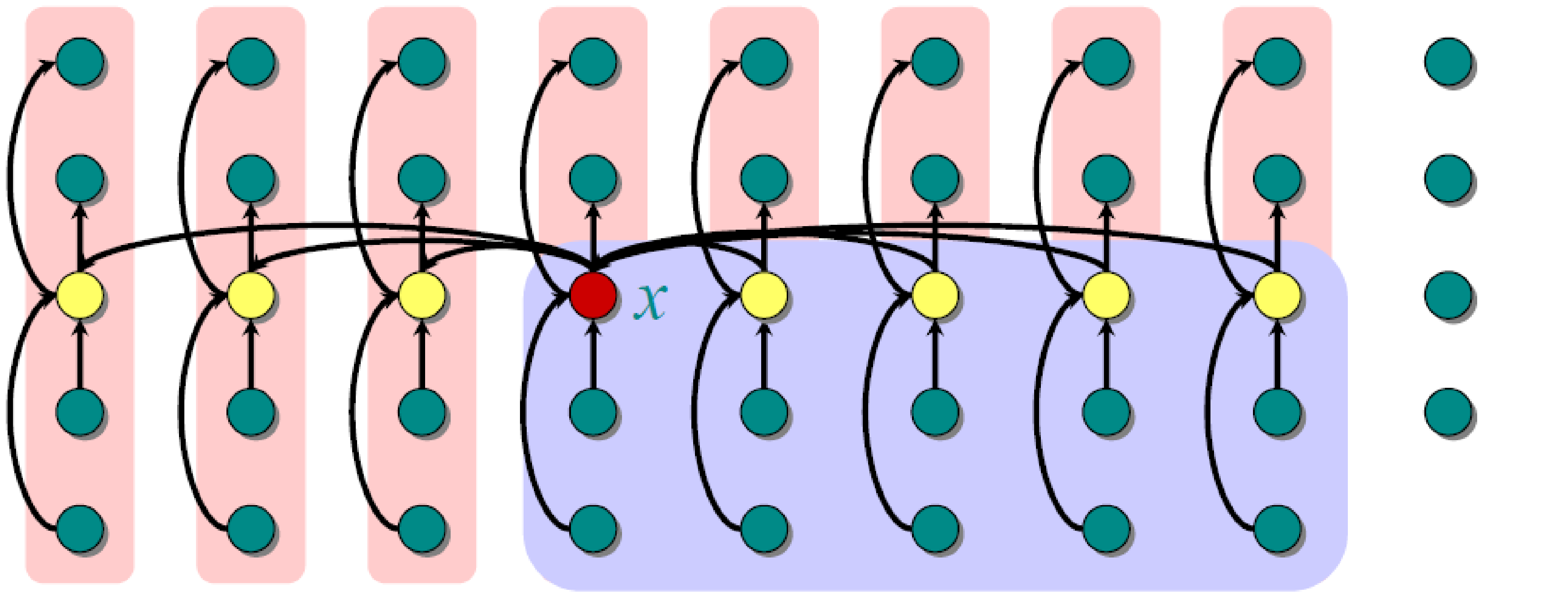




At least half the group medians are $\le x$, which is at least $\lfloor \lfloor n/5 \rfloor /2 \rfloor = \lfloor n/10 \rfloor$ group medians.

- 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 $\geq x$.

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 recursively 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!)