Chapter 9
Medians and Order Statistics

Algorithm Analysis

School of CSEE

Order Statistics

- The *ith order statistic* in a set of n elements is the *i*th smallest element.
- The *minimum* is thus the 1st order statistic.
- The *maximum* is the nth order statistic.
- The *median* is
  - the $(n+1)/2$ order statistic, if n is odd.
  - lower median $i=n/2$, upper median $i=(n/2)+1$, if n is even.
- *How can we calculate order statistics?*
- *What is the running time?*

Finding Min and Max Simultaneously

- Input : *n* numbers
- Output : min and max
- *n*-1 comparisons for min, *n*-1 comparisons for max : total 2*n*-2 comparisons
- Can we do better?
- Don't compare each element to min and max separately, but process elements in pairs - compare the elements of a pair to each other.
- Then compare the larger element to the current max so far, and compare the smaller element to the current min so far.

Finding Min and Max Simultaneously

- Only 3 comparisons for every 2 elements
- For initial min and max :
  - If n is even, compare the first two elements and set the larger to max and the smaller to min. Then process the rest in pairs.

    : 1 initial comparison and $3(n-2)/2$ more comparisons $= 3n/2 - 2$
  - If n is odd, set both min and max to the first element. Then process the rest in pairs.

    : If n is odd, $3(n-1)/2$ comparisons

Finding Order Statistics: The Selection Problem

- A more interesting problem is *selection*: finding the *i*th smallest element of a set
- Input: A set *A* of *n* (distinct) elements 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*

Naive algorithm: Sort and index *i* th element.

Worst-case running time $= \Theta(n \lg n) + \Theta(1)$

$= \Theta(n \lg n)$,

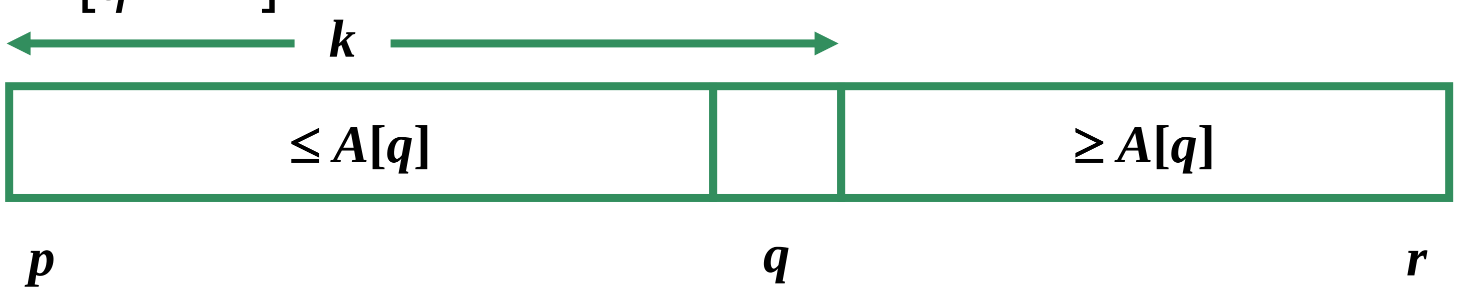
Using merge sort.

Finding Order Statistics: The Selection Problem

- We will study two algorithms:
  - A practical randomized algorithm with $\Theta(n)$ expected running time
  - A cool algorithm of theoretical interest only with $\Theta(n)$ worst-case running time

Randomized Selection

- Key idea: use `partition()` from quicksort
  - But, only need to examine one subarray
  - This saving shows up in running time: $\Theta(n)$
- Partition the array $A[p..r]$ into two (possibly empty) subarrays $A[p..q\text{-}1]$ and $A[q\text{+}1..r]$
- If $i\text{=}k$, return $A[q]$.
- If $i<k$, find ith in the subarray $A[p..q\text{-}1]$.
- If $i>k$, find $(i\text{-}k)$th smallest element in the subarray $A[q\text{+}1..r]$.



Example

Select the $i = 7^{th}$ smallest:

6	*10*	*13*	*5*	*8*	*3*	*2*	*11*	$i = 7$

pivot

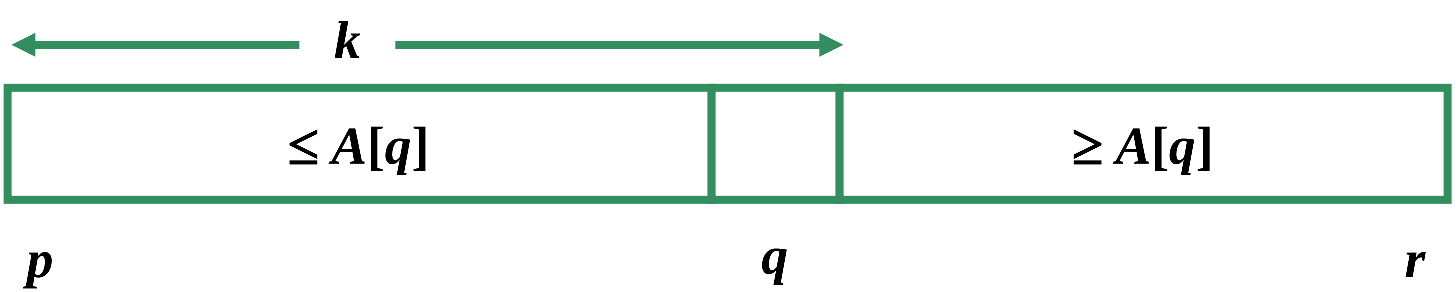
Partition:

2	*5*	*3*	*6*	*8*	*13*	*10*	*11*	$k = 4$

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

Randomized Selection

```
RandomizedSelect(A, p, r, i)
    if (p == r) then return A[p];
    q = RandomizedPartition(A, p, r)
    k = q - p + 1;
    if (i == k) then return A[q];
    if (i < k) then
        return RandomizedSelect(A, p, q-1, i);
    else
        return RandomizedSelect(A, q+1, r, i-k);
```



Intuition for analysis

- (All our analyses today assume that all elements are distinct.)
- Lucky:

$$T(n) = T(9n/10) + \Theta(n) \qquad n^{\log_{10/9} 1} = n^0 = 1$$

$$= \Theta(n) \qquad \text{CASE 3}$$

- Unlucky:

$$T(n) = T(n-1) + \Theta(n) \qquad \text{arithmetic series}$$

$$= \Theta(n^2)$$

Worse than sorting!

Randomized Selection

- Average case
  - For upper bound, assume *i*th element always falls in larger side of partition:

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

$$\leq \frac{2}{n}\sum_{k=n/2}^{n-1} T(k) + \Theta(n)$$

What happened here?

  - Let's show that T(*n*) = O(*n*) by substitution

Randomized Selection

- Assume T(n) $\le cn$ for sufficiently large c:

$$T(n) \le \frac{2}{n}\sum_{k=n/2}^{n-1} T(k) + \Theta(n)$$ ***The recurrence we started with***

$$\le \frac{2}{n}\sum_{k=n/2}^{n-1} ck + \Theta(n)$$ ***Substitute T(n) ≤ cn for T(k)***

$$= \frac{2c}{n}\left(\sum_{k=1}^{n-1} k - \sum_{k=1}^{n/2-1} k\right) + \Theta(n)$$ ***"Split" the recurrence***

$$= \frac{2c}{n}\left(\frac{1}{2}(n-1)n - \frac{1}{2}\left(\frac{n}{2}-1\right)\frac{n}{2}\right) + \Theta(n)$$ ***Expand arithmetic series***

$$= c(n-1) - \frac{c}{2}\left(\frac{n}{2}-1\right) + \Theta(n)$$ ***Multiply it out***

Randomized Selection

$$T(n) \leq c(n-1) - \frac{c}{2}\left(\frac{n}{2} - 1\right) + \Theta(n)$$ *The recurrence so far*

$$= cn - c - \frac{cn}{4} + \frac{c}{2} + \Theta(n)$$ *Multiply it out*

$$= cn - \frac{cn}{4} - \frac{c}{2} + \Theta(n)$$ *Subtract c/2*

$$= cn - \left(\frac{cn}{4} + \frac{c}{2} - \Theta(n)\right)$$ *Rearrange the arithmetic*

$$\leq cn \quad \text{(if c is big enough)}$$ *What we set out to prove*

Thus, T(n) = O(n). And T(n) = Ω(n) since it takes at least n-1 comparisons at first **`RandomizedPartition()`**. Therefore, T(n) = **Θ**(n).

Worst-Case Linear-Time Selection

- Randomized algorithm works well in practice.

  But in worst case its time complexity is $O(n^2)$.
- What follows is a worst-case linear time algorithm, really of theoretical interest only.
- Basic idea:
  - Generate a good partitioning element
  - Call this element x

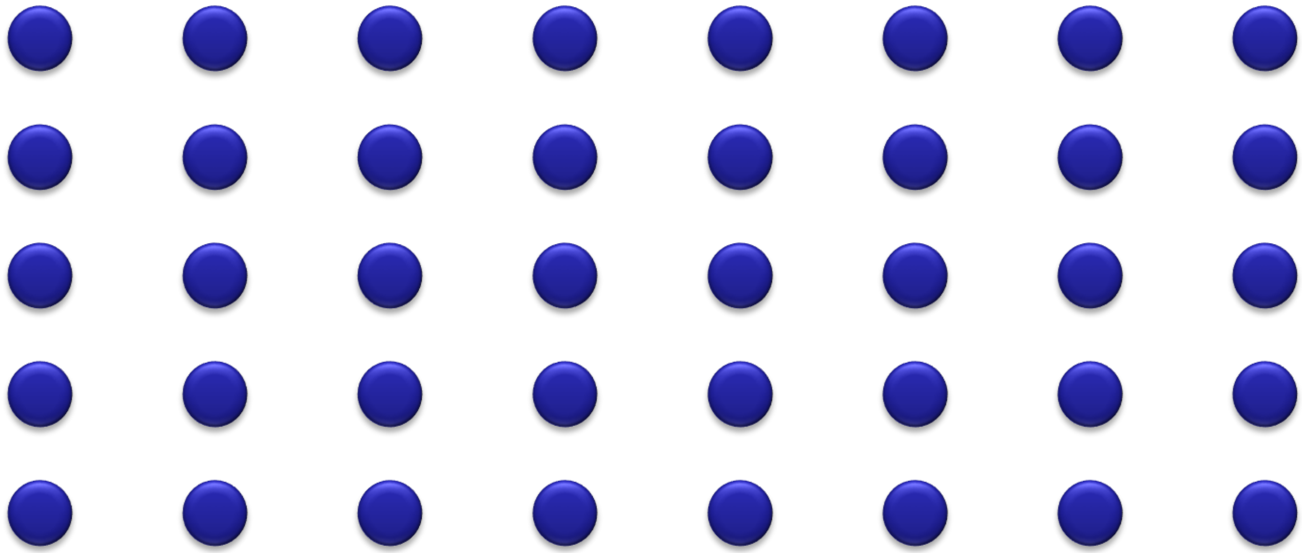
Worst-Case Linear-Time Selection

- Select (i, n)
  1. Divide n elements into groups of 5
  2. Find median of each group (*How? How long?*)
  3. Use Select() recursively to find median x of the $\lceil n/5 \rceil$ medians
  4. Partition the n elements around x. Let k = rank(x)
  5. if (i == k) then return x

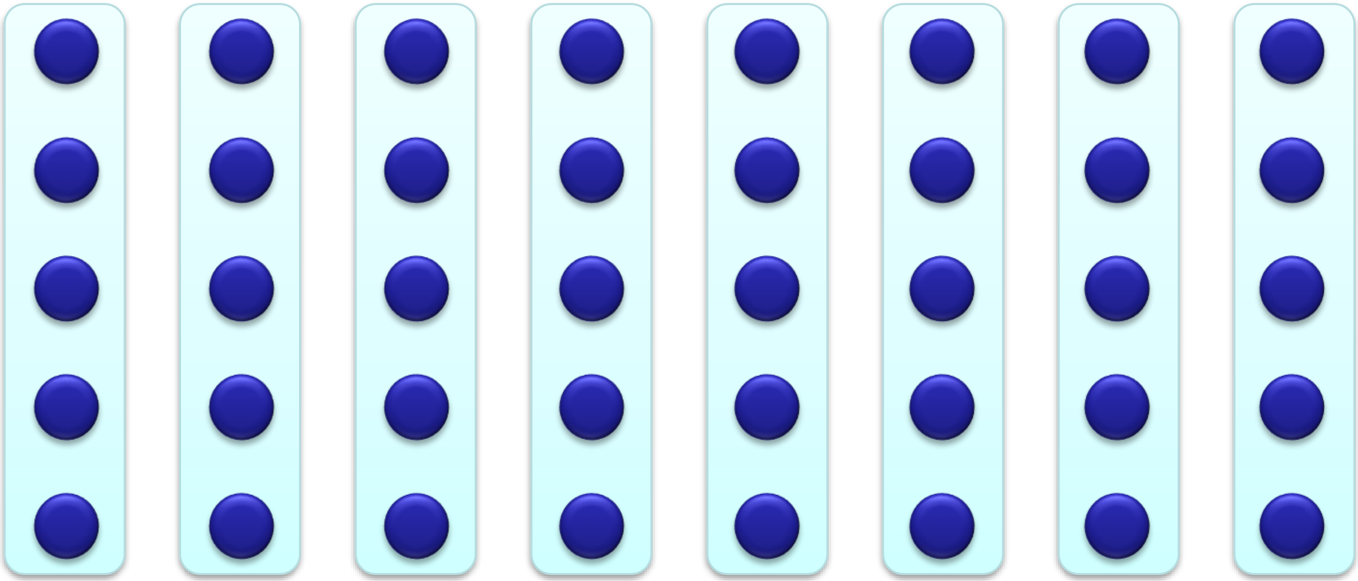
     if (i < k) then use Select() recursively to find ith smallest element in first partition

     else (i > k) use Select() recursively to find (i-k)th smallest element in last partition

Initially…

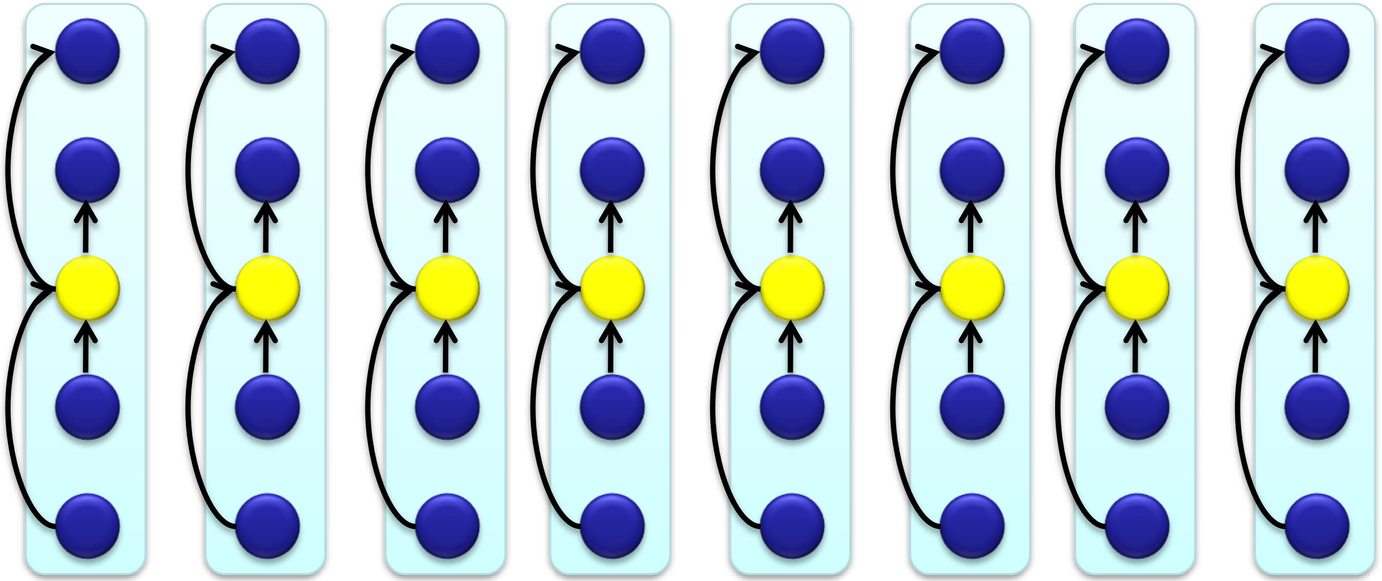


Initially…

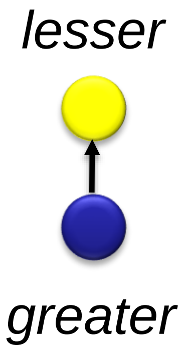


Divide the n elements into groups of 5 : O(*n*)

Step 2



Find median of each group : **Θ**(n) ?



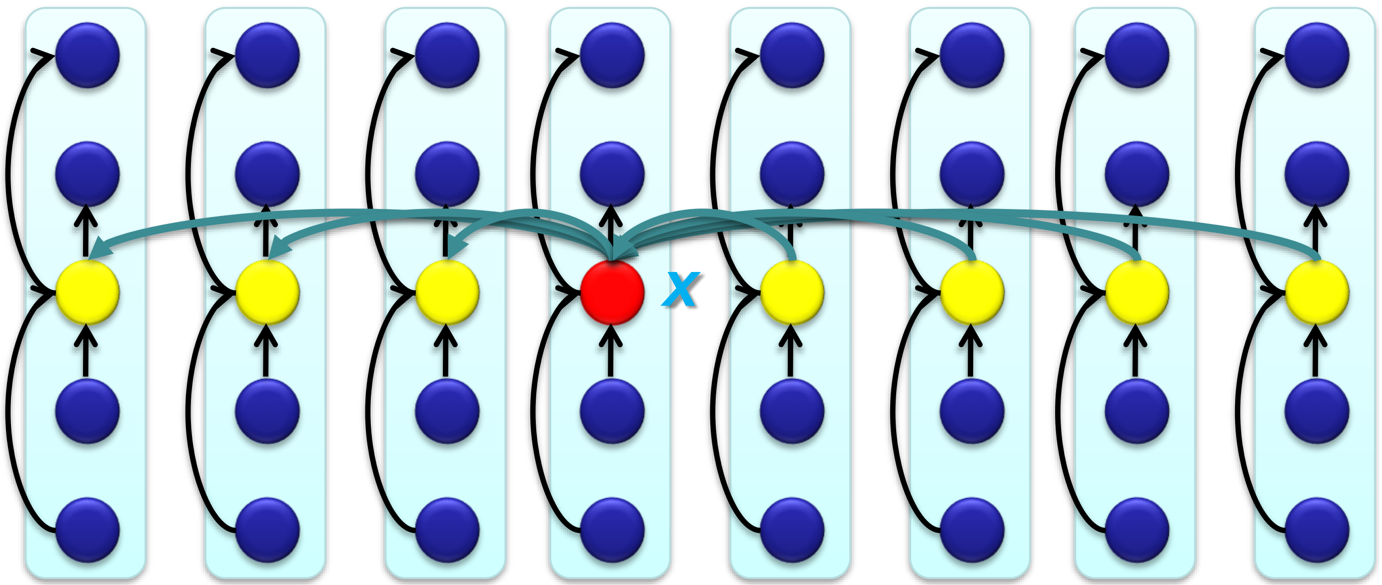
Step 2

Finding median of each group

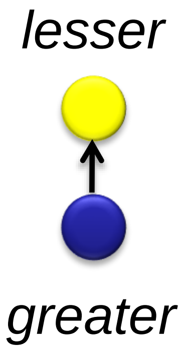
- It takes 6 comparisons to find median of 5 elements.

  - Can you prove it?
- We have $\lceil n/5 \rceil$ groups
- Thus, it takes $6(n/5) = \boldsymbol{\Theta}(n)$

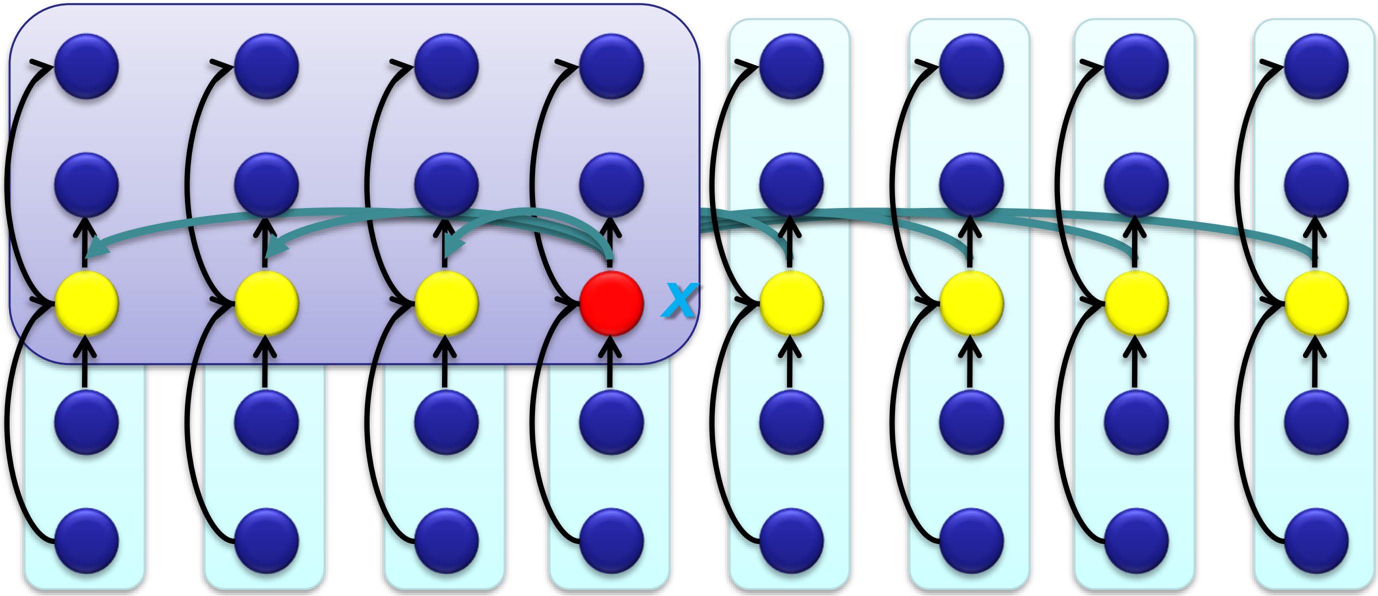
Step 3

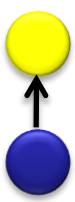


Use Select() recursively to find median x of the $\lceil n/5 \rceil$ medians : T($\lceil n/5 \rceil$)



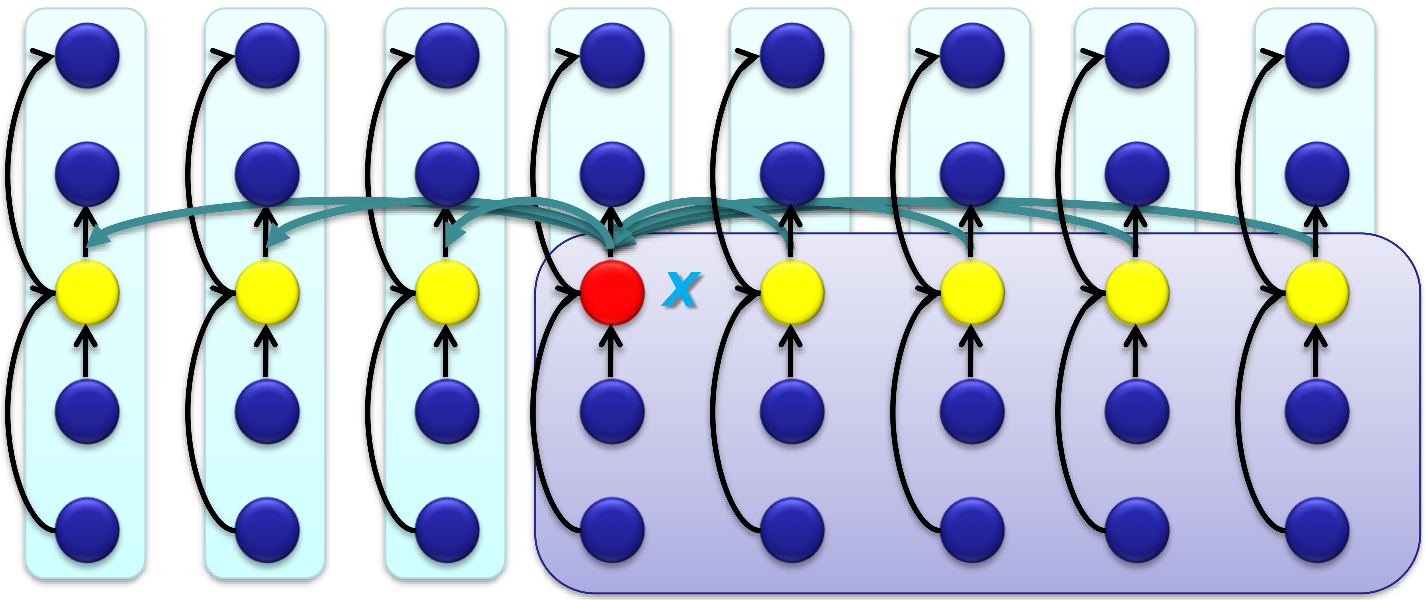
Around the pivot



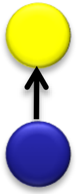


At least half of the medians are smaller than x, which is at least $\lceil n/5 \rceil / 2 = \lceil n/10 \rceil$ group medians. Therefore at least $3\lceil n/10 \rceil$ elements are smaller than x.

Around the pivot



Similarly, at least $3\lceil n/10 \rceil$ elements are greater than x.



Worst-Case Linear-Time Selection

- At least $3\lceil n/10 \rceil$ elements are smaller than x and at least $3\lceil n/10 \rceil$ elements are greater than x.

- In step4, compare the pivot with the remaining $\Theta(4n/10)$ elements : $\Theta(n)$

- Step 5 takes at most $T(7n/10)$, which is the worst case.

Worst-Case Linear-Time Selection

- Select (i, n)

$\Theta(n)$ 1. Divide n elements into groups of 5

$\Theta(n)$ 2. Find median of each group

$T(n/5)$ 3. Use Select() recursively to find median x of the $\lceil n/5 \rceil$ medians

$\Theta(n)$ 4. Partition the n elements around x. Let k = rank(x)

5. if (i == k) then return x

$T(7n/10)$ if (i < k) then use Select() recursively to find ith smallest element in first partition

else (i > k) use Select() recursively to find (i-k)th smallest element in last partition

Worst-Case Linear-Time Selection

- The recurrence is therefore:

  Assuming $T(n) = \Theta(1)$ for small enough n. Use $n < 140$

$$T(n) \le \begin{cases} \Theta(1) & \text{if } n<140 \\ T(n/5) + T(7n/10 + 6) + \Theta(n) & \text{if } n\ge 140. \end{cases}$$

- Solve this recurrence by substitution.

  (or $\Theta(n)$ term is approximately $1.6n$. Thus we can solve this equation using recursion tree method.)

$$T(n) = \Theta(n)$$