Advanced Algorithms
COMP4121

Aleks Ignjatovic

School of Computer Science and Engineering
University of New South Wales

Order Statistics

Let us get to work: Order Statistics algorithms

- **Problem:** Given n [illegible]s, select the i^{th} smallest element;

Let us get to work: Order Statistics algorithms

- **Problem:** Given n [illegible]s, select the i^{th} smallest element;
  - for $i = 1$ we get [illegible]**mum**;

- **Problem:** Given n [illegible]s, select the i^{th} smallest element;
  - for $i = 1$ we get [illegible]**mum**;
  - for $i = n$ we get the **maximum**;

Let us get to work: Order Statistics algorithms

- **Problem:** Given n [illegible]s, select the i^{th} smallest element;
  - for $i = 1$ we get [illegible]**mum**;
  - for $i = n$ we get the **maximum**;
  - for $i = \lfloor \frac{n+1}{2} \rfloor$ we get the **median**.

Let us get to work: Order Statistics algorithms

- **Problem:** Given n [illegible]s, select the i^{th} smallest element;
    - for $i = 1$ we get [illegible]**mum**;
    - for $i = n$ we get the **maximum**;
    - for $i = \lfloor \frac{n+1}{2} \rfloor$ we get the **median**.

- We can find both the minimum and the maximum in $O(n)$ many steps (linear time).

- **Problem:** Given n [illegible]s, select the i^{th} smallest element;
  - for $i = 1$ we get [illegible]**mum**;
  - for $i = n$ we get the **maximum**;
  - for $i = \lfloor \frac{n+1}{2} \rfloor$ we get the **median**.

- We can find both the minimum and the maximum in $O(n)$ many steps (linear time).

- Can we find the median also in linear time?

- **Problem:** Given n [illegible]s, select the i^{th} smallest element;
  - for $i = 1$ we get the **minimum**;
  - for $i = n$ we get the **maximum**;
  - for $i = \lfloor \frac{n+1}{2} \rfloor$ we get the **median**.
- We can find both the minimum and the maximum in $O(n)$ many steps (linear time).
- Can we find the median also in linear time?
- Clearly, we can do it in time $n \log n$, just MergeSort the array and find the middle element(s) of the sorted array.

Let us get to work: Order Statistics algorithms

- **Problem:** Given n elements, select the i^{th} smallest element;
  - for $i = 1$ we get the **minimum**;
  - for $i = n$ we get the **maximum**;
  - for $i = \lfloor \frac{n+1}{2} \rfloor$ we get the **median**.

- We can find both the minimum and the maximum in $O(n)$ many steps (linear time).

- Can we find the median also in linear time?

- Clearly, we can do it in time $n \log n$, just MergeSort the array and find the middle element(s) of the sorted array.

- Can we do it faster???

Fast algorithms for finding the median

- We will show that … e done in linear time, by both a deterministic and b … omised algorithm.

Fast algorithms for finding the median

- We will show that [illegible] be done in linear time, by both a deterministic and [illegible]mised algorithm.
- Why bother with a randomised algorithm if it can be done in linear time with a deterministic algorithm?

Fast algorithms for finding the median

- We will show that [illegible] be done in linear time, by both a deterministic and [illegible]omised algorithm.
- Why bother with a randomised algorithm if it can be done in linear time with a deterministic algorithm?
- Because in practice the randomised algorithm runs much faster, having much smaller constant c in the bound for the run time $T(n) \leq c \cdot n$.

Fast algorithms for finding the median

- We will show that it can be done in linear time, by both a deterministic and by a randomised algorithm.
- Why bother with a randomised algorithm if it can be done in linear time with a deterministic algorithm?
- Because in practice the randomised algorithm runs much faster, having much smaller constant c in the bound for the run time $T(n) \leq c \cdot n$.
- It turns out that it is easier to solve (both deterministically and with randomisation) the more general problem of finding the i^{th} smallest element for an arbitrary i than to find just the median.

Fast randomised algorithm for finding the median

- **Problem:** Given n elements, select the i^{th} smallest element.

Fast randomised algorithm for finding the median

- **Problem:** Given n elements, select the i^{th} smallest element.
- **Idea:** Divide-and-conquer; by doing "one half" of the randomised QuickSort, operati[illegible] side of the partition only.

Fast randomised algorithm for finding the median

- **Problem:** Given n elements, select the i^{th} smallest element.
- **Idea:** Divide-and-conquer; by doing "one half" of the randomised QuickSort, operati side of the partition only.

RAND-SELECT$(A, p, r$ hoose the i^{th} smallest elt of $A[p..r]$*

Fast randomised algorithm for finding the median

- **Problem:** Given n elements, select the i^{th} smallest element.
- **Idea:** Divide-and-conquer; by doing "one half" of the randomised QuickSort, operati side of the partition only.

Rand-Select$(A, p, r$ hoose the i^{th} smallest elt of $A[p..r]$*

1. **if** $p = r$ & $i = 1$ **then return** $A[p]$;

Fast randomised algorithm for finding the median

- **Problem:** Given n elements, select the i^{th} smallest element.
- **Idea:** Divide-and-conquer; by doing "one half" of the randomised QuickSort, operating on one side of the partition only.

RAND-SELECT(A, p, r, i) to choose the i^{th} smallest elt of $A[p..r]$*

1. **if** $p = r$ & $i = 1$ **then return** $A[p]$; **else**
2. choose a random pivot pv from $A[p..r]$;

Fast randomised algorithm for finding the median

- **Problem:** Given n elements, select the i^{th} smallest element.
- **Idea:** Divide-and-conquer; by doing "one half" of the randomised QuickSort, operating on one side of the partition only.

RAND-SELECT(A, p, r, i) — choose the i^{th} smallest elt of $A[p..r]$*

1. **if** $p = r$ & $i = 1$ **then return** $A[p]$; **else**
2. choose a random pivot pv from $A[p..r]$;
3. reorder the array $A[p..r]$ such that $A[p..(q-1)] \leq A[q]$ and $A[(q+1)..r] > A[q]$ where $A[q]$ is the randomly chosen pivot pv;

Fast randomised algorithm for finding the median

- **Problem:** Given n elements, select the i^{th} smallest element.
- **Idea:** Divide-and-conquer; by doing "one half" of the randomised QuickSort, operating on one side of the partition only.

RAND-SELECT(A, p, r, i) *choose the i^{th} smallest elt of $A[p..r]$*

1. **if** $p = r$ & $i = 1$ **then return** $A[p]$; **else**
2. choose a random pivot pv from $A[p..r]$;
3. reorder the array $A[p..r]$ such that $A[p..(q-1)] \leq A[q]$ and $A[(q+1)..r] > A[q]$ where $A[q]$ is the randomly chosen pivot pv; (fill the details how we do this in place, just as it is done in an implementation of the QuickSort)

Fast randomised algorithm for finding the median

- **Problem:** Given n elements, select the i^{th} smallest element.
- **Idea:** Divide-and-conquer; by doing "one half" of the randomised QuickSort, operating on one side of the partition only.

Rand-Select(A, p, r, i) (choose the i^{th} smallest elt of $A[p..r]$)*

1. **if** $p = r$ & $i = 1$ **then return** $A[p]$; **else**
2. choose a random pivot pv from $A[p..r]$;
3. reorder the array $A[p..r]$ such that $A[p..(q-1)] \leq A[q]$ and $A[(q+1)..r] > A[q]$ where $A[q]$ is the randomly chosen pivot pv; (fill the details how we do this in place, just as it is done in an implementation of the QuickSort)
4. $k \leftarrow q - p + 1$ (k is the number of elements $\leq$ pv $= A[q]$)

Fast randomised algorithm for finding the median

- **Problem:** Given n elements, select the i^{th} smallest element.
- **Idea:** Divide-and-conquer; by doing "one half" of the randomised QuickSort, operati[illegible] side of the partition only.

RAND-SELECT$(A, p, r$[illegible]hoose the i^{th} smallest elt of $A[p..r]$*

1. **if** $p = r$ & $i = 1$ **then return** $A[p]$; **else**
2. choose a random pivot pv from $A[p..r]$;
3. reorder the array $A[p..r]$ such that $A[p..(q-1)] \leq A[q]$ and $A[(q+1)..r] > A[q]$ where $A[q]$ is the randomly chosen pivot pv; (fill the details how we do this in place, just as it is done in an implementation of the QuickSort)
4. $k \leftarrow q - p + 1$ (k is the number of elements $\leq$ pv $= A[q]$)
5. **if** $k = i$ **then return** $A[q]$;

Fast randomised algorithm for finding the median

- **Problem:** Given n elements, select the i^{th} smallest element.
- **Idea:** Divide-and-conquer; by doing "one half" of the randomised QuickSort, operating on one side of the partition only.

$\textsc{Rand-Select}(A, p, r, i)$: choose the i^{th} smallest elt of $A[p..r]$*

1. **if** $p = r$ & $i = 1$ **then return** $A[p]$; **else**
2. choose a random pivot pv from $A[p..r]$;
3. reorder the array $A[p..r]$ such that $A[p..(q-1)] \leq A[q]$ and $A[(q+1)..r] > A[q]$ where $A[q]$ is the randomly chosen pivot pv; (fill the details how we do this in place, just as it is done in an implementation of the QuickSort)
4. $k \leftarrow q - p + 1$ $\quad$ (k is the number of elements $\leq$ pv $= A[q]$)
5. **if** $k = i$ **then return** $A[q]$; **else**
6. **if** $i < k$ **then return** $\textsc{Rand-Select}(A, p, q-1, i)$;

Fast randomised algorithm for finding the median

- **Problem:** Given n elements, select the i^{th} smallest element.
- **Idea:** Divide-and-conquer; by doing "one half" of the randomised QuickSort, operating on one side of the partition only.

RAND-SELECT(A, p, r, i) chooses the i^{th} smallest elt of $A[p..r]$*

1. **if** $p = r$ & $i = 1$ **then return** $A[p]$; **else**
2. choose a random pivot pv from $A[p..r]$;
3. reorder the array $A[p..r]$ such that $A[p..(q-1)] \leq A[q]$ and $A[(q+1)..r] > A[q]$ where $A[q]$ is the randomly chosen pivot pv; (fill the details how we do this in place, just as it is done in an implementation of the QuickSort)
4. $k \leftarrow q - p + 1$ $\quad$ (k is the number of elements $\leq$ pv $= A[q]$)
5. **if** $k = i$ **then return** $A[q]$; **else**
6. **if** $i < k$ **then return** RAND-SELECT$(A, p, q-1, i)$;
7. **else return** RAND-SELECT$(A, q+1, r, i-k)$.

Analysis of RAND-SELECT(A, p, r, i)

- Clearly, the worst c[illegible]time is $\Theta(n^2)$.

- Clearly, the worst case running time is $\Theta(n^2)$.
- This happens, for example, in a very unlikely event that you always pick either the smallest or the largest element of the array.

- Clearly, the worst c[illegible]time is $\Theta(n^2)$.
- This happens, for e[illegible]n a very unlikely event that you always pick either t[illegible]est or the largest element of the array.
- In such a case during each call of RAND-SELECT the size of the array drops only by 1.

- Clearly, the worst c[illegible]time is $\Theta(n^2)$.
- This happens, for e[illegible]n a very unlikely event that you always pick either t[illegible]est or the largest element of the array.
- In such a case during each call of RAND-SELECT the size of the array drops only by 1.
- Due to reshuffling of elements around the pivot, each iteration of RAND-SELECT costs the length of the array, and you get

$$T(n) = c(n + (n-1) + (n-2) + \ldots + 1) = \Theta(n^2)$$

Analysis of RAND-SELECT(A, p, r, i)

- Clearly, the worst case running time is $\Theta(n^2)$.
- This happens, for example, in a very unlikely event that you always pick either the smallest or the largest element of the array.
- In such a case during each call of RAND-SELECT the size of the array drops only by 1.

- Due to reshuffling of elements around the pivot, each iteration of RAND-SELECT costs the length of the array, and you get

$$T(n) = c(n + (n-1) + (n-2) + \ldots + 1) = \Theta(n^2)$$

- This is very unlikely to happen; in fact, as we will now see, most of the time the partitions will be reasonably well balanced.

Analysis of RAND-SELECT(A, p, r, i)

- Let us first assume that all the elements in the array are **distinct**.

Analysis of RAND-SELECT(A, p, r, i)

- Let us first assume that all the elements in the array are **distinct**.
- Let us call a partit[illegible]*nced partition* if the ratio between the number of element[illegible]maller piece and the number of elements in the lar[illegible]is not worse than 1 to 9 (9 is kind of arbitrary here, any small number > 2 would do).

- Let us first assume that all the elements in the array are **distinct**.
- Let us call a partit[illegible]*nced partition* if the ratio between the number of element[illegible]maller piece and the number of elements in the lar[illegible]is not worse than 1 to 9 (9 is kind of arbitrary here, any small number > 2 would do).
- What is the probability that we get a balanced partition after choosing the pivot?

- Let us first assume that all the elements in the array are **distinct**.
- Let us call a partition a *balanced partition* if the ratio between the number of elements in the smaller piece and the number of elements in the larger piece is not worse than 1 to 9 (9 is kind of arbitrary here, any small number > 2 would do).
- What is the probability that we get a balanced partition after choosing the pivot?
- Clearly, this happens if we chose an element which is neither among the smallest $1/10$ nor among the largest $1/10$ of all elements.

Analysis of RAND-SELECT(A, p, r, i)

- Let us first assume that all the elements in the array are **distinct**.
- Let us call a partition a *balanced partition* if the ratio between the number of elements in the smaller piece and the number of elements in the larger piece is not worse than 1 to 9 (9 is kind of arbitrary here, any small number > 2 would do).
- What is the probability that we get a balanced partition after choosing the pivot?
- Clearly, this happens if we chose an element which is neither among the smallest $1/10$ nor among the largest $1/10$ of all elements.
- Thus, the probability to end up with a balanced partition is $1 - 2/10 = 8/10$.

- Let us first assume that all the elements in the array are **distinct**.
- Let us call a partit[illegible]*nced partition* if the ratio between the number of elements[illegible]maller piece and the number of elements in the lar[illegible]is not worse than 1 to 9 (9 is kind of arbitrary here, any small number > 2 would do).
- What is the probability that we get a balanced partition after choosing the pivot?
- Clearly, this happens if we chose an element which is neither among the smallest $1/10$ nor among the largest $1/10$ of all elements.
- Thus, the probability to end up with a balanced partition is $1 - 2/10 = 8/10$.
- Let us find the expected number of partitions between two consecutive balanced partitions.

Analysis of RAND-SELECT(A, p, r, i)

- The probability to get another balanced partition immediately after a balanced partition is $\frac{8}{10}$;

- The probability to get another balanced partition immediately after a balanced partition is $\frac{8}{10}$;
- The probability to [illegible] partitions to get another balanced partition is $\frac{2}{10}\frac{8}{10}$;

Analysis of RAND-SELECT(A, p, r, i)

- The probability to get another balanced partition immediately after a balanced partition is $\frac{8}{10}$;
- The probability to [illegible] partitions to get another balanced partition is $\frac{2}{10}\frac{8}{10}$;
- In general, the prob[illegible]at you will need k partitions to end up with another balanced partition is $\left(\frac{2}{10}\right)^{k-1} \cdot \frac{8}{10}$.

Analysis of RAND-SELECT(A, p, r, i)

- The probability to get another balanced partition immediately after a balanced partition is $\frac{8}{10}$;
- The probability to [illegible] partitions to get another balanced partition is $\frac{2}{10}\frac{8}{10}$;
- In general, the prob[illegible]hat you will need k partitions to end up with another balanced partition is $\left(\frac{2}{10}\right)^{k-1} \cdot \frac{8}{10}$.
- Thus, the expected number of partitions between two balanced partitions is

$$E = 1 \cdot \frac{8}{10} + 2 \cdot \frac{2}{10}\frac{8}{10} + 3 \cdot \left(\frac{2}{10}\right)^2 \cdot \frac{8}{10} + \ldots$$

$$= \frac{8}{10} \cdot \sum_{k=0}^{\infty} (k+1)\left(\frac{2}{10}\right)^k = \frac{8}{10} S$$

where

$$S = 1 + 2 \cdot \frac{2}{10} + 3 \cdot \left(\frac{2}{10}\right)^2 + 4 \cdot \left(\frac{2}{10}\right)^3 + 5 \cdot \left(\frac{2}{10}\right)^4 + \ldots$$

Evaluating $S = \sum_{k=0}^{\infty}(k+1)\left(\frac{2}{10}\right)^k$

- How do we evaluate such a sum S??

- How do we evaluate such a sum S??
- Trick # 1:

$$
\begin{aligned}
S &= 1 + 2\cdot\frac{2}{10} + 3\cdot\left(\frac{2}{10}\right)^2 + 4\cdot\left(\frac{2}{10}\right)^3 + 5\cdot\left(\frac{2}{10}\right)^4 + \dots \\
&= 1 + \frac{2}{10} + \left(\frac{2}{10}\right)^2 + \left(\frac{2}{10}\right)^3 + \left(\frac{2}{10}\right)^4 + \dots \\
&\qquad + \frac{2}{10} + \left(\frac{2}{10}\right)^2 + \left(\frac{2}{10}\right)^3 + \left(\frac{2}{10}\right)^4 + \dots \\
&\qquad\qquad + \left(\frac{2}{10}\right)^2 + \left(\frac{2}{10}\right)^3 + \left(\frac{2}{10}\right)^4 + \dots \\
&\qquad\qquad\qquad + \left(\frac{2}{10}\right)^3 + \left(\frac{2}{10}\right)^4 + \dots
\end{aligned}
$$

Evaluating $S = \sum_{k=0}^{\infty}(k+1)\left(\frac{2}{10}\right)^k$

- Summing each row separately we obtain

$$
\begin{aligned}
1 + \frac{2}{10} + \left(\frac{2}{10}\right)^2 + \left(\frac{2}{10}\right)^3 + \left(\frac{2}{10}\right)^4 + \cdots &= \frac{1}{1-\frac{2}{10}} = \frac{10}{8} \\
+ \frac{2}{10} + \left(\frac{2}{10}\right)^2 + \left(\frac{2}{10}\right)^3 + \left(\frac{2}{10}\right)^4 + \cdots &= \frac{2}{10}\,\frac{10}{8} \\
+ \left(\frac{2}{10}\right)^2 + \left(\frac{2}{10}\right)^3 + \left(\frac{2}{10}\right)^4 + \cdots &= \left(\frac{2}{10}\right)^2 \frac{10}{8} \\
+ \left(\frac{2}{10}\right)^3 + \left(\frac{2}{10}\right)^4 + \cdots &= \left(\frac{2}{10}\right)^3 \frac{10}{8} \\
\cdots &
\end{aligned}
$$

- Summing each row separately we obtain

$$
\begin{aligned}
1 + \frac{2}{10} + \left(\frac{2}{10}\right)^2 + \left(\frac{2}{10}\right)^3 + \left(\frac{2}{10}\right)^4 + \ldots &= \frac{1}{1-\frac{2}{10}} = \frac{10}{8} \\
+ \frac{2}{10} + \left(\frac{2}{10}\right)^2 + \left(\frac{2}{10}\right)^3 + \left(\frac{2}{10}\right)^4 + \ldots &= \frac{2}{10}\,\frac{10}{8} \\
+ \left(\frac{2}{10}\right)^2 + \left(\frac{2}{10}\right)^3 + \left(\frac{2}{10}\right)^4 + \ldots &= \left(\frac{2}{10}\right)^2 \frac{10}{8} \\
+ \left(\frac{2}{10}\right)^3 + \left(\frac{2}{10}\right)^4 + \ldots &= \left(\frac{2}{10}\right)^3 \frac{10}{8} \\
\ldots
\end{aligned}
$$

- We can now sum the right hand side column:

$$S = \frac{10}{8}\left(1 + \frac{2}{10} + \left(\frac{2}{10}\right)^2 + \left(\frac{2}{10}\right)^3 + \left(\frac{2}{10}\right)^4 + \ldots\right)$$

$$= \frac{10}{8}\frac{1}{1-\frac{2}{10}} = \left(\frac{10}{8}\right)^2$$

$$S = \frac{10}{8}\left(1 + \frac{2}{10} + \left(\frac{2}{10}\right)^2 + \left(\frac{2}{10}\right)^3 + \left(\frac{2}{10}\right)^4 + \ldots\right)$$

$$= \frac{10}{8}\frac{1}{1-\frac{2}{10}} = \left(\frac{10}{8}\right)^2$$

Thus, we obtain

$$E = \frac{8}{10}S = \frac{8}{10}\left(\frac{10}{8}\right)^2 = \frac{10}{8} = \frac{5}{4} < 2$$

A useful digression (Trick #2): $S = \sum_{k=0}^{\infty}(k+1)\left(\frac{2}{10}\right)^k$ evaluated another way

Note that

$$\sum_{k=0}^{\infty} q^k = \frac{1}{1-q}.$$

By differentiating both sides with respect to q we get

$$\sum_{k=1}^{\infty} k\, q^{k-1} = \frac{1}{(1-q)^2}.$$

Substituting $q = 2/10$ we get that $S = (10/8)^2$.

Performance of RAND-SELECT:

- So, on average, there are only $5/4$ partitions between two balanced partitions.

Performance of Rand-Select:

- So, on average, there are only 5/4 partitions between two balanced partitions.
- Note that before the first balanced partition the size of the array can be bounded by n, after the first balanced partition the size of the array is $\leq 9/10\, n$, after the second balanced partition the size of the array is $\leq (9/10)^2\, n$ and so on...

Performance of RAND-SELECT:

- So, on average, there are only 5/4 partitions between two balanced partitions.
- Note that before the first balanced partition the size of the array can be bounded by n, after the first balanced partition the size of the array is $\leq 9/10\, n$, after the second balanced partition the size of the array is $\leq (9/10)^2\, n$ and so on...
- Consequently the total **average** (expected) run time satisfies

Performance of Rand-Select:

- So, on average, there are only $5/4$ partitions between two balanced partitions.
- Note that before the first balanced partition the size of the array can be bounded by n, after the first balanced partition the size of the array is $\leq 9/10\, n$, after the second balanced partition the size of the array is $\leq (9/10)^2\, n$ and so on...

- Consequently the total **average** (expected) run time satisfies

$$T(n) < 5/4\, n + 5/4\, \frac{9}{10}\, n + 5/4 \left(\frac{9}{10}\right)^2 n + 5/4 \left(\frac{9}{10}\right)^3 n + \ldots$$

$$= \frac{5/4\, n}{1 - \frac{9}{10}} = \frac{50}{4}\, n = 12.5\, n$$

- Where did we tacitly assume that all elements are distinct?

Performance of Rand-Select:

- So, on average, there are only 5/4 partitions between two balanced partitions.
- Note that before the first balanced partition the size of the array can be bounded by n, after the first balanced partition the size of the array is $\leq 9/10\,n$, after the second balanced partition the size of the array is $\leq (9/10)^2\,n$ and so on...

- Consequently the total **average** (expected) run time satisfies

$$T(n) < 5/4\,n + 5/4\,\frac{9}{10}\,n + 5/4\left(\frac{9}{10}\right)^2 n + 5/4\left(\frac{9}{10}\right)^3 n + \ldots$$

$$= \frac{5/4\,n}{1 - \frac{9}{10}} = \frac{50}{4}\,n = 12.5\,n$$

- Where did we tacitly assume that all elements are distinct?
- How did we estimate the probability of choosing the pivot which results in a balanced partition?

Performance of Rand-Select:

- Note that if all elements are the same Rand-Select would run in quadratic time no matter which elements are chosen as pivots - they are all equal.
- **Homework:** Modify Rand-Select so that it runs in linear time even when there are many repetitions.

Performance of Rand-Select:

- Note that if all elements are the same Rand-Select would run in quadratic time no matter which elements are chosen as pivots - they are all equal.
- **Homework:** Modify Rand-Select so that it runs in linear time even when there are many repetitions.
- You might want try to modify slightly the way how the array is reordered.

Performance of Rand-Select:

- Note that if all elements are the same Rand-Select would run in quadratic time no matter which elements are chosen as pivots - they are all equal.
- **Homework:** Modify Rand-Select so that it runs in linear time even when there are many repetitions.
- You might want try to modify slightly the way how the array is reordered.
- In practice Rand-Select runs really fast, for the same reason the Randomised QuickSort runs fast; it can be implemented as a very tight loop.

Performance of RAND-SELECT:

- Note that if all elements are the same RAND-SELECT would run in quadratic time no matter which elements are chosen as pivots - they are all equal.
- **Homework:** Modify RAND-SELECT so that it runs in linear time even when there are many repetitions.
- You might want try to modify slightly the way how the array is reordered.

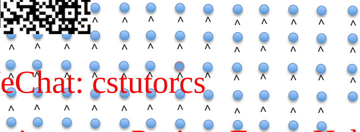
- In practice RAND-SELECT runs really fast, for the same reason the Randomised QuickSort runs fast: it can be implemented as a very tight loop.
- In 1972 Blum, Floyd, Pratt, Rivest and Tarjan designed a deterministic Order Statistic Selection which runs in linear time in the worst case.

Performance of RAND-SELECT:

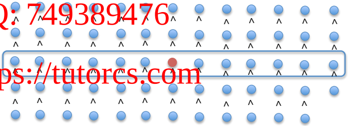
- Note that if all elements are the same RAND-SELECT would run in quadratic time no matter which elements are chosen as pivots - they are all equal.
- **Homework:** Modify RAND-SELECT so that it runs in linear time even when there are many repetitions.
- You might want try to modify slightly the way how the array is reordered.
- In practice RAND-SELECT runs really fast, for the same reason the Randomised QuickSort runs fast: it can be implemented as a very tight loop.
- In 1972 Blum, Floyd, Pratt, Rivest and Tarjan designed a deterministic Order Statistic Selection which runs in linear time in the worst case.
- **Main idea:** Use a recursive call of the very same algorithm to choose a good pivot!

- **Algorithm Select(n, i):**

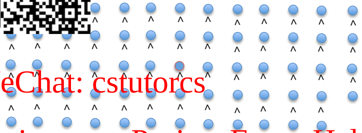
  - Split the numbers in groups of five (the last group might contain less than 5 elements).
  - Order each group by brute force in an increasing order.



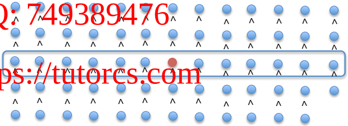
  - Take the collection of all $\lfloor \frac{n}{5} \rfloor$ middle elements of each group (i.e., the medians of each group of five).



- **Algorithm Select**(n, i) :
  - Split the numbers in groups of five (the last group might contain less than 5 elem
  - Order each grou e force in an increasing order.



  - Take the collection of all $\lfloor \frac{n}{5} \rfloor$ middle elements of each group (i.e., the medians of each group of five).



  - Apply recursively SELECT algorithm to find the median p of this collection.

- **Algorithm Select(n, i) continued:**

- **Algorithm Select(n, i) continued:**
  - partition all ele[...]g p as a pivot;

- **Algorithm Select(n, i) continued:**
  - partition all ele[illegible]g p as a pivot;
  - Let k be the nu[illegible]ements in the subset of all elements smaller than the[illegible]

Deterministic Linear Time Algorithm for Order Statistics

- **Algorithm Select(n, i) continued:**
  - partition all ele g p as a pivot;
  - Let k be the nu ements in the subset of all elements smaller than the
  - **if** $i = k$ **then return** p

- **Algorithm Select(n, i) continued:**
  - partition all ele[illegible]g p as a pivot;
  - Let k be the nu[illegible]ements in the subset of all elements smaller than the[illegible]
  - **if** $i = k$ **then return** p
  - **else if** $i < k$ **then** recursively SELECT the i^{th} smallest element of the set of elements smaller than the pivot.

Deterministic Linear Time Algorithm for Order Statistics

- **Algorithm Select(n, i) continued:**
  - partition all elements using p as a pivot;
  - Let k be the number of elements in the subset of all elements smaller than the pivot;
  - **if** $i = k$ **then return** p
  - **else if** $i < k$ **then** recursively SELECT the i^{th} smallest element of the set of elements smaller than the pivot.
  - **else** recursively SELECT the $(i-k)^{th}$ smallest element of the set of elements larger than the pivot.

- **Algorithm Select(n, i) continued:**
  - partition all elements using p as a pivot;
  - Let k be the number of elements in the subset of all elements smaller than the [illegible]
  - **if** $i = k$ **then return** p
  - **else if** $i < k$ **then** recursively SELECT the i^{th} smallest element of the set of elements smaller than the pivot.
  - **else** recursively SELECT the $(i-k)^{th}$ smallest element of the set of elements larger than the pivot.

- **Note:** This algorithm is the same as RAND-SELECT except for the way how we chose the pivot.

Deterministic Linear Time Algorithm for Order Statistics

- **Algorithm Select(n, i) continued:**
  - partition all ele[illegible]g p as a pivot;
  - Let k be the nu[illegible]ements in the subset of all elements smaller than the[illegible]
  - **if** $i = k$ **then return** p
  - **else if** $i < k$ **then** recursively SELECT the i^{th} smallest element of the set of elements smaller than the pivot.
  - **else** recursively SELECT the $(i - k)^{th}$ smallest element of the set of elements larger than the pivot.

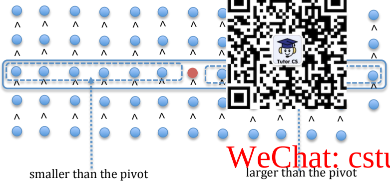
- **Note:** This algorithm is the same as RAND-SELECT except for the way how we chose the pivot.
  - instead of choosing pivot randomly we called recursively the very same algorithm to pick the pivot as the median of the middle elements of the groups of five elements.

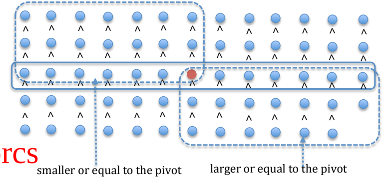
Deterministic Linear Time Algorithm for Order Statistics

- What have we accomplished by such a choice of the pivot?

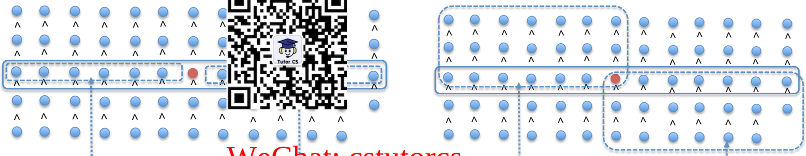
Deterministic Linear Time Algorithm for Order Statistics

- What have we accomplished by such a choice of the pivot?



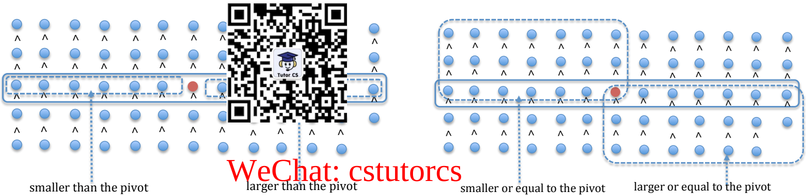


- What have we accomplished by such a choice of the pivot?



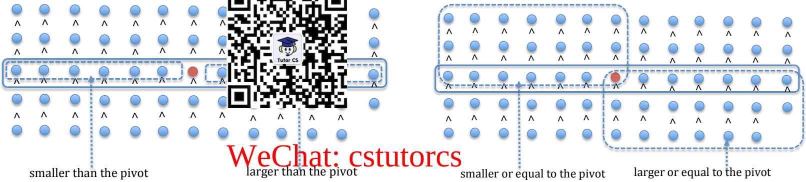
- Note that at least $\lfloor (n/5)/2 \rfloor = \lfloor n/10 \rfloor$ group medians are smaller or equal to the pivot; and at least that many larger than the pivot.

- What have we accomplished by such a choice of the pivot?



- Note that at least $\lfloor (n/5)/2 \rfloor = \lfloor n/10 \rfloor$ group medians are smaller or equal to the pivot; and at least that many larger than the pivot.
- But this implies that at least $\lfloor 3n/10 \rfloor$ of the total number of elements are smaller than the pivot, and that many elements larger than the pivot.

- What have we accomplished by such a choice of the pivot?



- Note that at least $\lfloor (n/5)/2 \rfloor = \lfloor n/10 \rfloor$ group medians are smaller or equal to the pivot; and at least that many larger than the pivot.
- But this implies that at least $\lfloor 3n/10 \rfloor$ of the total number of elements are smaller than the pivot, and that many elements larger than the pivot.
- (the same caveat: we are assuming all elements are distinct; otherwise we have to slightly tweak the algorithm to split all elements equal to the pivot evenly between the two groups.)

- What is the run time of our algorithm?

Deterministic Linear Time Algorithm for Order Statistics

- What is the run time of our algorithm?

$$T(\quad /5) + T(7n/10) + Cn.$$

- What is the run time of our algorithm?

$$T(\text{[illegible]} /5) + T(7n/10) + Cn.$$

- Let us show that T[illegible]Cn for all n.

Deterministic Linear Time Algorithm for Order Statistics

- What is the run time of our algorithm?

$$T(n) \leq T(n/5) + T(7n/10) + Cn.$$

- Let us show that $T(n) \leq 10Cn$ for all n. Assume that this is true for all $k < n$ and let us prove it is true for n as well.

Deterministic Linear Time Algorithm for Order Statistics

- What is the run time of our algorithm?

$$T(n) \le T(n/5) + T(7n/10) + Cn.$$

- Let us show that $T(n) \le 11Cn$ for all n. Assume that this is true for all $k < n$ and let us prove it is true for n as well.
  - Note: this is a proof using the following form of induction: $\varphi(0)$ & $(\forall n)((\forall k < n)\varphi(k) \to \varphi(n)) \to (\forall n)\varphi(n)$.

- What is the run time of our algorithm?

$$T(n) = T(n/5) + T(7n/10) + Cn.$$

- Let us show that $T(n) < 11Cn$ for all n. Assume that this is true for all $k < n$ and let us prove it is true for n as well.
  - Note: this is a proof using the following form of induction: $\varphi(0)\ \&\ (\forall n)((\forall k < n)\varphi(k) \to \varphi(n)) \to (\forall n)\varphi(n)$.
- Thus, assume $T(n/5) < 11C \cdot n/5$ and $T(7n/10) < 11C \cdot 7n/10$;

Deterministic Linear Time Algorithm for Order Statistics

- What is the run time of our algorithm?

$$T(n) \leq T(n/5) + T(7n/10) + Cn.$$

- Let us show that $T(n) \leq 11Cn$ for all n. Assume that this is true for all $k < n$ and let us prove it is true for n as well.
  - Note: this is a proof using the following form of induction: $\varphi(0)\ \&\ (\forall n)((\forall k < n)\varphi(k) \rightarrow \varphi(n)) \rightarrow (\forall n)\varphi(n)$.
- Thus, assume $T(n/5) < 11C \cdot n/5$ and $T(7n/10) < 11C \cdot 7n/10$; then

$$T(n) \leq T(n/5) + T(7n/10) + Cn < 11C \cdot \frac{n}{5} + 11C \cdot \frac{7n}{10} + Cn$$

$$= 109\frac{Cn}{10} < 11C \cdot n$$

- What is the run time of our algorithm?

$$T(n) = T(n/5) + T(7n/10) + Cn.$$

- Let us show that $T(n) < 11Cn$ for all n. Assume that this is true for all $k < n$ and let us prove it is true for n as well.
  - Note: this is a proof using the following form of induction: $\varphi(0)\ \&\ (\forall n)((\forall k < n)\varphi(k) \rightarrow \varphi(n)) \rightarrow (\forall n)\varphi(n)$.

- Thus, assume $T(n/5) < 11C \cdot n/5$ and $T(7n/10) < 11C \cdot 7n/10$; then

$$T(n) \leq T(n/5) + T(7n/10) + Cn < 11C \cdot \frac{n}{5} + 11C \cdot \frac{7n}{10} + Cn$$
$$= 109\frac{Cn}{10} < 11C \cdot n$$

which proves out statement that $T(n) < 11C \cdot n$.

Deterministic Linear Time Algorithm for Order Statistics

- Note that this algo[illegible] a genuine recursion (rather than just an iteration) so its execution involves lots of traffic on the stack, which makes this algorithm slow in practice; the randomised version of it, RAND-SELECT, significantly outperforms it.

Deterministic Linear Time Algorithm for Order Statistics

- Note that this algorithm is a genuine recursion (rather than just an iteration) so its execution involves lots of traffic on the stack, which makes this algorithm slow in practice; the randomised version of it, RAND-SELECT, significantly outperforms it.

- Similarly RAND-QUICKSORT in practice outperforms MERGESORT, which, unlike RAND-QUICKSORT, is guaranteed to run in time $O(n \log n)$.