CSI - 3105 Design & Analysis of Algorithms
Course 4

Jean-Lou De Carufel

Fall 2020

Selection Problem

Input : Sequence S of n numbers and an integer k with $1 \leq k \leq n$.

Output : The k-th smallest element in S.

Selection Problem

Input : Sequence S of n numbers and an integer k with $1 \leq k \leq n$.

Output : The k-th smallest element in S.

- $k = 1 \quad \rightarrow$

Selection Problem

Input : Sequence S of n numbers and an integer k with $1 \leq k \leq n$.

Output : The k-th smallest element in S.

- $k = 1 \quad \rightarrow \quad$ smallest element in S

Selection Problem

Input : Sequence S of n numbers and an integer k with $1 \leq k \leq n$.

Output : The k-th smallest element in S.

- $k = 1 \quad \rightarrow \quad$ smallest element in S
- $k = n \quad \rightarrow$

Selection Problem

Input : Sequence S of n numbers and an integer k with $1 \leq k \leq n$.

Output : The k-th smallest element in S.

- $k = 1 \quad \rightarrow \quad$ smallest element in S
- $k = n \quad \rightarrow \quad$ largest element in S

Selection Problem

Input : Sequence S of n numbers and an integer k with $1 \leq k \leq n$.

Output : The k-th smallest element in S.

- $k = 1 \quad \rightarrow \quad$ smallest element in S
- $k = n \quad \rightarrow \quad$ largest element in S
- $k = n/2 \quad \rightarrow$

Selection Problem

Input : Sequence S of n numbers and an integer k with $1 \leq k \leq n$.

Output : The k-th smallest element in S.

- $k = 1 \quad \rightarrow \quad$ smallest element in S
- $k = n \quad \rightarrow \quad$ largest element in S
- $k = n/2 \quad \rightarrow \quad$ median of S

Algorithm *EasySelect*(S, k)

Input: Sequence S of n numbers and an integer k with $1 \leq k \leq n$.

Output: The k-th smallest element of S.

1: Sort S

2: **return** the element at position k

Algorithm *EasySelect*(S, k)

Input: Sequence S of n numbers and an integer k with $1 \leq k \leq n$.

Output: The k-th smallest element of S.

1: Sort S

2: **return** the element at position k

What is the running time?

Algorithm *EasySelect*(S, k)

Input: Sequence S of n numbers and an integer k with $1 \leq k \leq n$.

Output: The k-th smallest element of S.

1: Sort S

2: **return** the element at position k

What is the running time?

- Sorting using Merge Sort takes $O(n \log(n))$ time.

Algorithm *EasySelect*(S, k)

Input: Sequence S of n numbers and an integer k with $1 \leq k \leq n$.

Output: The k-th smallest element of S.

1: Sort S

2: **return** the element at position k

What is the running time?

- Sorting using Merge Sort takes $O(n \log(n))$ time.
- The return step takes $O(1)$ time.

Algorithm *EasySelect*(S, k)

Input: Sequence S of n numbers and an integer k with $1 \leq k \leq n$.

Output: The k-th smallest element of S.

1: Sort S

2: **return** the element at position k

What is the running time?

- Sorting using Merge Sort takes $O(n \log(n))$ time.
- The return step takes $O(1)$ time.
- TOTAL: $O(n \log(n)) + O(1) = O(n \log(n))$ time.

Algorithm *EasySelect*(S, k)

Input: Sequence S of n numbers and an integer k with $1 \leq k \leq n$.

Output: The k-th smallest element of S.

1: Sort S

2: **return** the element at position k

What is the running time?

- Sorting using Merge Sort takes $O(n \log(n))$ time.
- The return step takes $O(1)$ time.
- TOTAL: $O(n \log(n)) + O(1) = O(n \log(n))$ time.

What is the bottleneck?

Algorithm *EasySelect*(S, k)

Input: Sequence S of n numbers and an integer k with $1 \leq k \leq n$.

Output: The k-th smallest element of S.

1: Sort S

2: **return** the element at position k

What is the running time?

- Sorting using Merge Sort takes $O(n \log(n))$ time.
- The return step takes $O(1)$ time.
- TOTAL: $O(n \log(n)) + O(1) = O(n \log(n))$ time.

What is the bottleneck?

Can we solve this problem without sorting?

First Attempt for a Faster Algorithm

Algorithm *Select*(S, k)

Input: Sequence S of n numbers and an integer k with $1 \leq k \leq n$.

Output: The k-th smallest element of S.

if $|S| = 1$ **then**
    return the only element in S
else
    Choose an element p in S (called the *pivot*)
    Split S into $S_<$, $S_=$ and $S_>$
    if $k \leq |S_<|$ **then**
        Run *Select*$(S_<, k)$
    else if $k > |S_<| + |S_=|$ **then**
        Run *Select*$(S_>, k - |S_<| - |S_=|)$
    else
        return p
    end if
end if

The running time of $Select(S, k)$ depends on the pivot p.
In the worst case,

- S is sorted
- $k = 1$
- in each recursive call, p is chosen as the largest element.

This gives $O(n^2)$ time.

The running time of $Select(S, k)$ depends on the pivot p.
In the worst case,

- S is sorted
- $k = 1$
- in each recursive call, p is chosen as the largest element.

This gives $O(n^2)$ time.

A good case would be: in each recursive call, p is chosen as the median.
In this case, the running time satisfies

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

The running time of $Select(S, k)$ depends on the pivot p.
In the worst case,

- S is sorted
- $k = 1$
- in each recursive call, p is chosen as the largest element.

This gives $O(n^2)$ time.

A good case would be: in each recursive call, p is chosen as the median.
In this case, the running time satisfies

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

Using the Master Theorem with $a = 1$, $b = 2$ and $d = 1$, we find $T(n) = O(n)$.

How to get close to a "good case"?

How to get close to a "good case"?

In each recursive call, choose p randomly. Intuitively, on average, p will be close to the median. *Study randomized algorithms for further details.*

How to get close to a "good case"?

In each recursive call, choose p randomly. Intuitively, on average, p will be close to the median. *Study randomized algorithms for further details.*

How *close* to a good case do we need to be to get a linear-time algorithm?

General Approach

Assume that all numbers are different. (The purpose of this assumption is only to simplify the discussion.)

General Approach

Assume that all numbers are different. (The purpose of this assumption is only to simplify the discussion.)

Assume that there is a constant $0 < \alpha < 1$ such that in $Select(S, p)$, it is always possible to choose a pivot p satisfying $|S_<| \leq \alpha\, n$ and $|S_>| \leq \alpha\, n$

General Approach

Assume that all numbers are different. (The purpose of this assumption is only to simplify the discussion.)

Assume that there is a constant $0 < \alpha < 1$ such that in $\mathit{Select}(S, p)$, it is always possible to choose a pivot p satisfying $|S_<| \le \alpha\, n$ and $|S_>| \le \alpha\, n$

Then the running time satisfies

$$\begin{aligned} T(n) &= T(\alpha\, n) + n \\ &= T(\alpha^2\, n) + \alpha\, n + n \\ &= T(\alpha^3\, n) + \alpha^2\, n + \alpha\, n + n \\ &\vdots \\ &= O(n) \end{aligned}$$

So how do we find such a pivot?

So how do we find such a pivot?

Blum, Floyd, Pratt, Rivest and Tarjan (1973) discovered the following technique.

The Algorithm

Step 1 : Divide the input sequence into $\frac{n}{5}$ groups, each of size 5.

Step 2 : For $i = 1, 2, ..., n/5$, compute the median of the i-th group, call this median m_i.

Step 3 : Compute the median p of $m_1, m_2, ..., m_{n/5}$.

Step 4 : Run $Select(S, k)$ using the p of Step 3 as the pivot.

The Algorithm

Step 1 : Divide the input sequence into $\frac{n}{5}$ groups, each of size 5.

Step 2 : For $i = 1, 2, ..., n/5$, compute the median of the i-th group, call this median m_i.

Step 3 : Compute the median p of $m_1, m_2, ..., m_{n/5}$.

Step 4 : Run $Select(S, k)$ using the p of Step 3 as the pivot.

Is p a good pivot? Why?

The Algorithm

Step 1 : Divide the input sequence into $\frac{n}{5}$ groups, each of size 5.

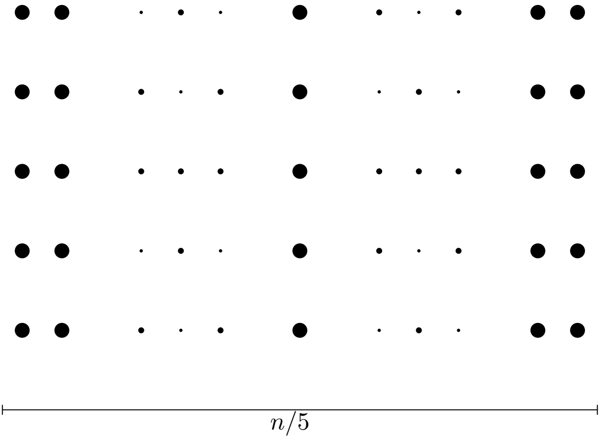
Step 2 : For $i = 1, 2, ..., n/5$, compute the median of the i-th group, call this median m_i.

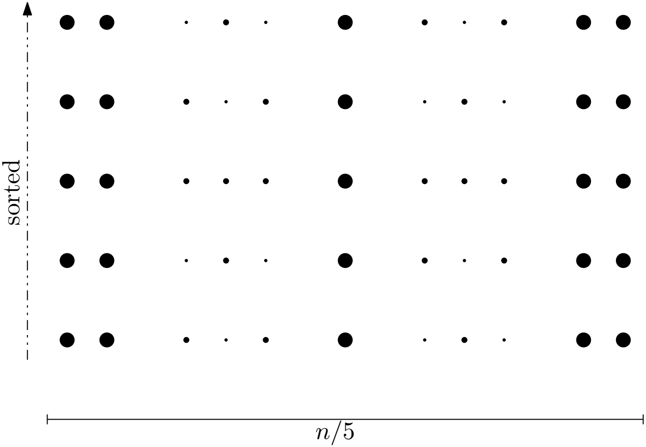
Step 3 : Compute the median p of $m_1, m_2, ..., m_{n/5}$.

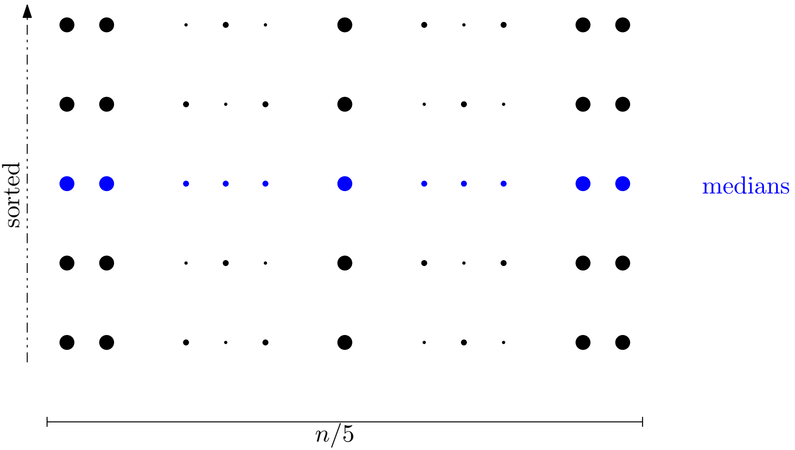
Step 4 : Run $Select(S, k)$ using the p of Step 3 as the pivot.

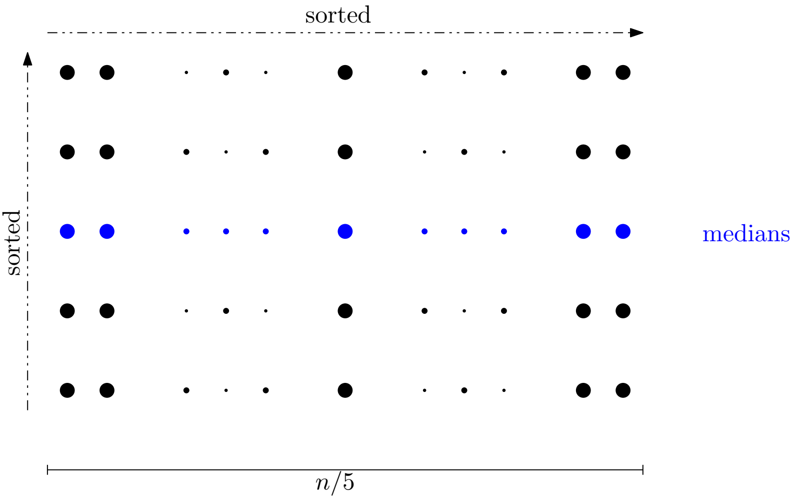
Is p a good pivot? Why?

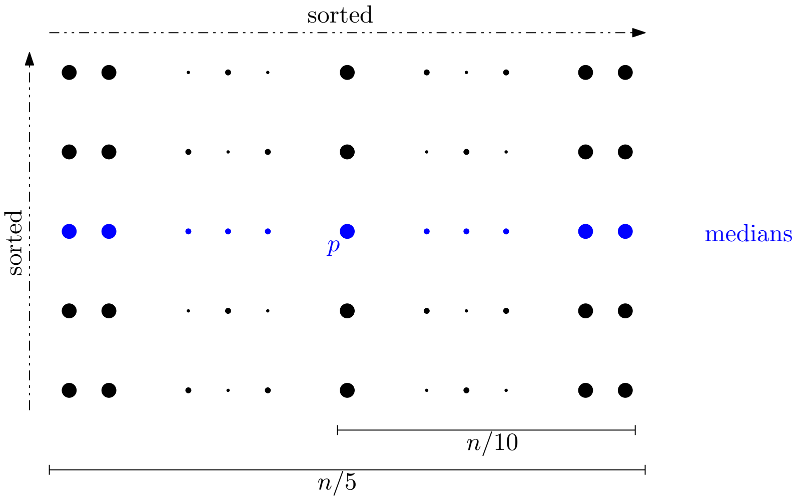
We have to figure out how many numbers in S are larger than p.

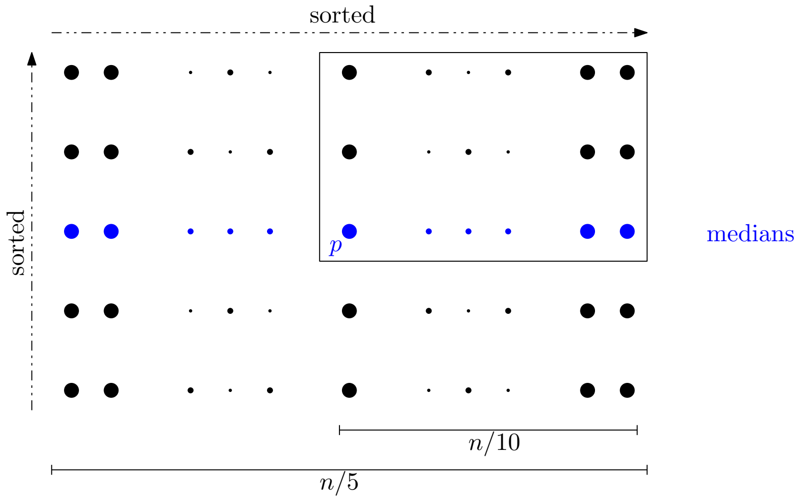

$n/5$

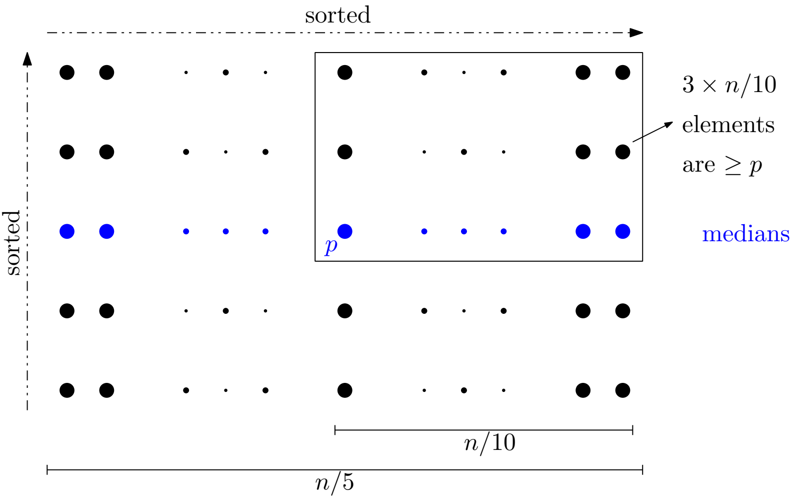

sorted
$n/5$

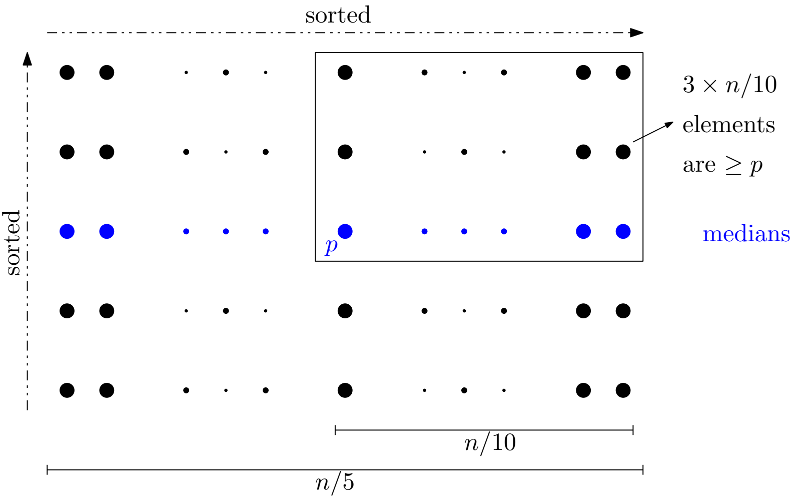

sorted
medians
$n/5$


sorted
sorted
medians
$n/5$

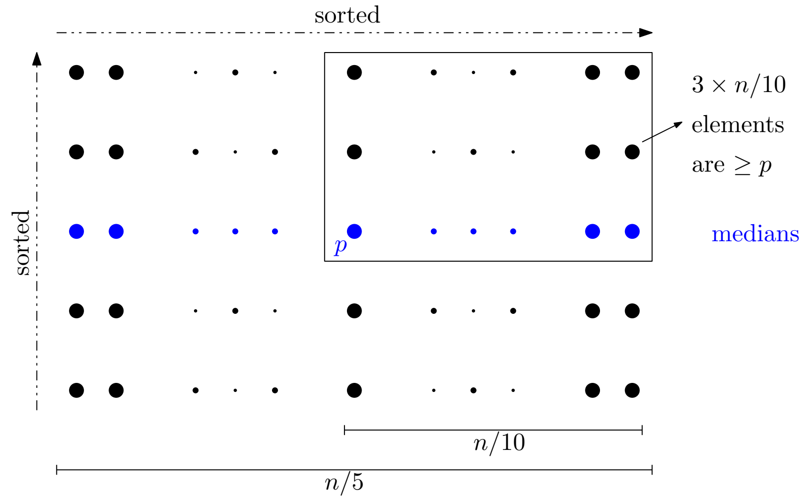

sorted
sorted
p
medians
$n/10$
$n/5$


sorted
sorted
medians
p
$n/10$
$n/5$


sorted
sorted
p
$3 \times n/10$
elements
are $\geq p$
medians
$n/10$
$n/5$

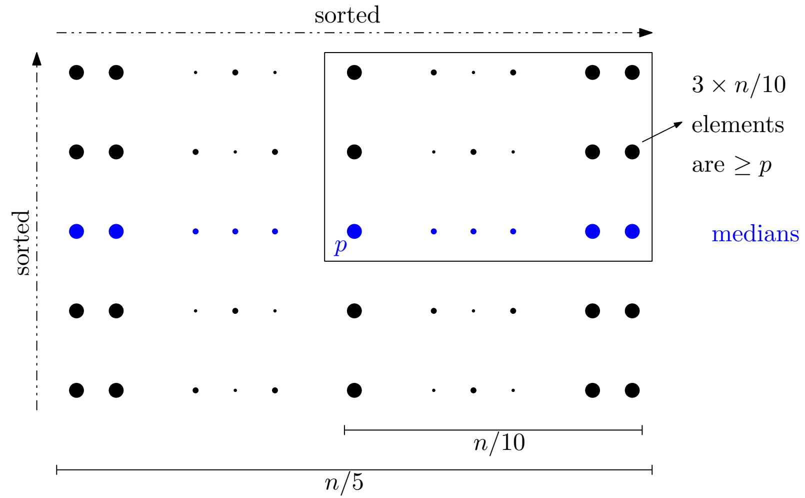


Hence, at least $\frac{3}{10}n$ elements are $\geq p$.



Hence, at least $\frac{3}{10}n$ elements are $\geq p$.

Thus, at most $\frac{7}{10}n$ elements are $\leq p$.



Hence, at least $\frac{3}{10}n$ elements are $\geq p$.

Thus, at most $\frac{7}{10}n$ elements are $\leq p$.

In other words, $|S_<| \leq \frac{7}{10}n$.

Using a symmetric argument, we can show that with this choice of pivot, $|S_>| \leq \frac{7}{10}n$.

Using a symmetric argument, we can show that with this choice of pivot, $|S_>| \leq \frac{7}{10}n$.

Hence, with this choice of pivot, we have $\alpha = \frac{7}{10}$.

Step 1 : Divide the input sequence into $\frac{n}{5}$ groups, each of size 5.

Step 2 : For $i = 1, 2, ..., n/5$, compute the median of the i-th group, call this median m_i.

Step 3 : Compute the median p of $m_1, m_2, ..., m_{n/5}$.

Step 4 : Run $Select(S, k)$ using the p of Step 3 as the pivot.

Step 1 : Divide the input sequence into $\frac{n}{5}$ groups, each of size 5.

Step 2 : For $i = 1, 2, ..., n/5$, compute the median of the i-th group, call this median m_i.

Step 3 : Compute the median p of $m_1, m_2, ..., m_{n/5}$.

Step 4 : Run $Select(S, k)$ using the p of Step 3 as the pivot.

Let $T(n)$ be the running time of this algorithm on an input of length n.

Step 1 : Divide the input sequence into $\frac{n}{5}$ groups, each of size 5.

Step 2 : For $i = 1, 2, ..., n/5$, compute the median of the i-th group, call this median m_i.

Step 3 : Compute the median p of $m_1, m_2, ..., m_{n/5}$.

Step 4 : Run *Select*(S, k) using the p of Step 3 as the pivot.

Let $T(n)$ be the running time of this algorithm on an input of length n.

Step 1 :

Step 2 :

Step 3 :

Step 4 :

Step 1 : Divide the input sequence into $\frac{n}{5}$ groups, each of size 5.

Step 2 : For $i = 1, 2, ..., n/5$, compute the median of the i-th group, call this median m_i.

Step 3 : Compute the median p of $m_1, m_2, ..., m_{n/5}$.

Step 4 : Run $Select(S, k)$ using the p of Step 3 as the pivot.

Let $T(n)$ be the running time of this algorithm on an input of length n.

Step 1 : $O(n)$ time

Step 2 :

Step 3 :

Step 4 :

Step 1 : Divide the input sequence into $\frac{n}{5}$ groups, each of size 5.

Step 2 : For $i = 1, 2, ..., n/5$, compute the median of the i-th group, call this median m_i.

Step 3 : Compute the median p of $m_1, m_2, ..., m_{n/5}$.

Step 4 : Run $Select(S, k)$ using the p of Step 3 as the pivot.

Let $T(n)$ be the running time of this algorithm on an input of length n.

Step 1 : $O(n)$ time

Step 2 : $O(n)$ time

Step 3 :

Step 4 :

Step 1 : Divide the input sequence into $\frac{n}{5}$ groups, each of size 5.

Step 2 : For $i = 1, 2, ..., n/5$, compute the median of the i-th group, call this median m_i.

Step 3 : Compute the median p of $m_1, m_2, ..., m_{n/5}$.

Step 4 : Run $Select(S, k)$ using the p of Step 3 as the pivot.

Let $T(n)$ be the running time of this algorithm on an input of length n.

Step 1 : $O(n)$ time

Step 2 : $O(n)$ time

Step 3 : ?

Step 4 :

Step 1 : Divide the input sequence into $\frac{n}{5}$ groups, each of size 5.

Step 2 : For $i = 1, 2, ..., n/5$, compute the median of the i-th group, call this median m_i.

Step 3 : Compute the median p of $m_1, m_2, ..., m_{n/5}$.

Step 4 : Run $Select(S, k)$ using the p of Step 3 as the pivot.

Let $T(n)$ be the running time of this algorithm on an input of length n.

Step 1 : $O(n)$ time

Step 2 : $O(n)$ time

Step 3 : ?

Step 4 : $T\left(\frac{7}{10}n\right) + O(n)$ time

Step 1 : Divide the input sequence into $\frac{n}{5}$ groups, each of size 5.

Step 2 : For $i = 1, 2, ..., n/5$, compute the median of the i-th group, call this median m_i.

Step 3 : Compute the median p of $m_1, m_2, ..., m_{n/5}$.

Step 4 : Run $Select(S, k)$ using the p of Step 3 as the pivot.

Let $T(n)$ be the running time of this algorithm on an input of length n.

Step 1 : $O(n)$ time

Step 2 : $O(n)$ time

Step 3 : ?

Step 4 : $T\left(\frac{7}{10}n\right) + O(n)$ time

To do Step 3, recursively compute the $\frac{n}{10}$-th smallest element of the sequence $m_1, m_2, ..., m_{n/5}$.

Step 1 : Divide the input sequence into $\frac{n}{5}$ groups, each of size 5.

Step 2 : For $i = 1, 2, ..., n/5$, compute the median of the i-th group, call this median m_i.

Step 3 : Compute the median p of $m_1, m_2, ..., m_{n/5}$.

Step 4 : Run $Select(S, k)$ using the p of Step 3 as the pivot.

Let $T(n)$ be the running time of this algorithm on an input of length n.

Step 1 : $O(n)$ time

Step 2 : $O(n)$ time

Step 3 : ?

Step 4 : $T\left(\frac{7}{10}n\right) + O(n)$ time

To do Step 3, recursively compute the $\frac{n}{10}$-th smallest element of the sequence $m_1, m_2, ..., m_{n/5}$. This takes $T\left(\frac{n}{5}\right)$ time.

Step 1 : Divide the input sequence into $\frac{n}{5}$ groups, each of size 5.

Step 2 : For $i = 1, 2, ..., n/5$, compute the median of the i-th group, call this median m_i.

Step 3 : Compute the median p of $m_1, m_2, ..., m_{n/5}$.

Step 4 : Run $Select(S, k)$ using the p of Step 3 as the pivot.

Let $T(n)$ be the running time of this algorithm on an input of length n.

Step 1 : $O(n)$ time

Step 2 : $O(n)$ time

Step 3 : $T\left(\frac{1}{5}n\right)$ time

Step 4 : $T\left(\frac{7}{10}n\right) + O(n)$ time

To do Step 3, recursively compute the $\frac{n}{10}$-th smallest element of the sequence $m_1, m_2, ..., m_{n/5}$. This takes $T\left(\frac{n}{5}\right)$ time.

Step 1 : Divide the input sequence into $\frac{n}{5}$ groups, each of size 5.

Step 2 : For $i = 1, 2, ..., n/5$, compute the median of the i-th group, call this median m_i.

Step 3 : Compute the median p of $m_1, m_2, ..., m_{n/5}$.

Step 4 : Run $Select(S, k)$ using the p of Step 3 as the pivot.

Let $T(n)$ be the running time of this algorithm on an input of length n.

Step 1 : $O(n)$ time

Step 2 : $O(n)$ time

Step 3 : $T\left(\frac{1}{5}n\right)$ time

Step 4 : $T\left(\frac{7}{10}n\right) + O(n)$ time

To do Step 3, recursively compute the $\frac{n}{10}$-th smallest element of the sequence $m_1, m_2, ..., m_{n/5}$. This takes $T\left(\frac{n}{5}\right)$ time.

Hence: $T(n) = T\left(\frac{1}{5}n\right) + T\left(\frac{7}{10}n\right) + O(n)$

How do we solve

$$T(n) = T\left(\frac{1}{5}n\right) + T\left(\frac{7}{10}n\right) + O(n)?$$

How do we solve

$$T(n) = T\left(\frac{1}{5}n\right) + T\left(\frac{7}{10}n\right) + O(n)?$$

Unfolding gets messy!

How do we solve

$$T(n) = T\left(\frac{1}{5}n\right) + T\left(\frac{7}{10}n\right) + O(n)?$$

Unfolding gets messy!

The Master theorem does not apply.

How do we solve

$$T(n) = T\left(\frac{1}{5}n\right) + T\left(\frac{7}{10}n\right) + O(n)?$$

Unfolding gets messy!

The Master theorem does not apply.

We use induction to show that $T(n) = O(n)$.

$$\begin{aligned} T(n) = \; & O(n) && \text{Step 1} \\ & + O(n) && \text{Step 2} \\ & + \;\; ? && \text{Step 3} \\ & + O(n) + T\left(\tfrac{7}{10}n\right) && \text{Step 4} \end{aligned}$$

How do we do Step 3: Recursively compute the $\frac{n}{10}$-th smallest element of the sequence $m_1, m_2, \ldots, m_{n/5}$. This takes $T(n/5)$ time.

We obtain the recurrence

$$T(n) = n + T\left(\frac{n}{5}\right) + T\left(\frac{7}{10}n\right)$$

→ Unfolding get messy

→ Master theorem does not apply!

Let us use induction to show that $T(n) = O(n)$.

<u>Claim:</u> $T(n) \le c \cdot n$ for some constant c

<u>Proof:</u> By choosing c sufficiently large, the claim is true for "small" n (this is the base case of the induction).

Let n be "large" and assume $T(m) \le c \cdot m$ for all $1 \le m < n$. Then

$$T(n) = n + T\left(\tfrac{n}{5}\right) + T\left(\tfrac{7}{10}n\right)$$

$$\le n + c \cdot \tfrac{n}{5} + c \cdot \tfrac{7}{10} n \quad \text{by the induction hypothesis}$$

$$= n + \tfrac{9}{10} c \cdot n$$

Is $n + \frac{9}{10} c \cdot n \le cn$?

Yes, provided that $c \geq 10$. $\square$

<u>Conclusion:</u> The k-th smallest element in a sequence of n numbers can be computed in $O(n)$ time.