Advanced Algorithmic Techniques (COMP523)

Recursion and Divide and Conquer Techniques #4

Recap and plan

Recap and plan

- **Last lecture:**
  - Finding the closest pair of points.
  - Integer Multiplication.

Recap and plan

- **Last lecture:**
  - Finding the closest pair of points.
  - Integer Multiplication.
- **This lecture:**
  - The Selection problem.
    - i.e., finding the i^{th}-order statistic in an array.

The selection problem

- **Definition:** The i^{th}-order statistic of a set of n (distinct) elements is the i^{th} smallest element.
  - i.e., the element which is larger than exactly *i-1* other elements.

The Selection Problem: Selection(**A**[*1,…,n*],*i*)

Input: A set of n (distinct) numbers (in an array **A**) and a number *i*, with $1 \leq i \leq n$.
Output: The i^{th}-order statistic of the set.

An easy solution

An easy solution

- Sort the numbers in $O(n \log n)$ time using **MergeSort**.

An easy solution

- Sort the numbers in O(n log n) time using **MergeSort**.
- Return the *i*-th element of the sorted array.

An easy solution

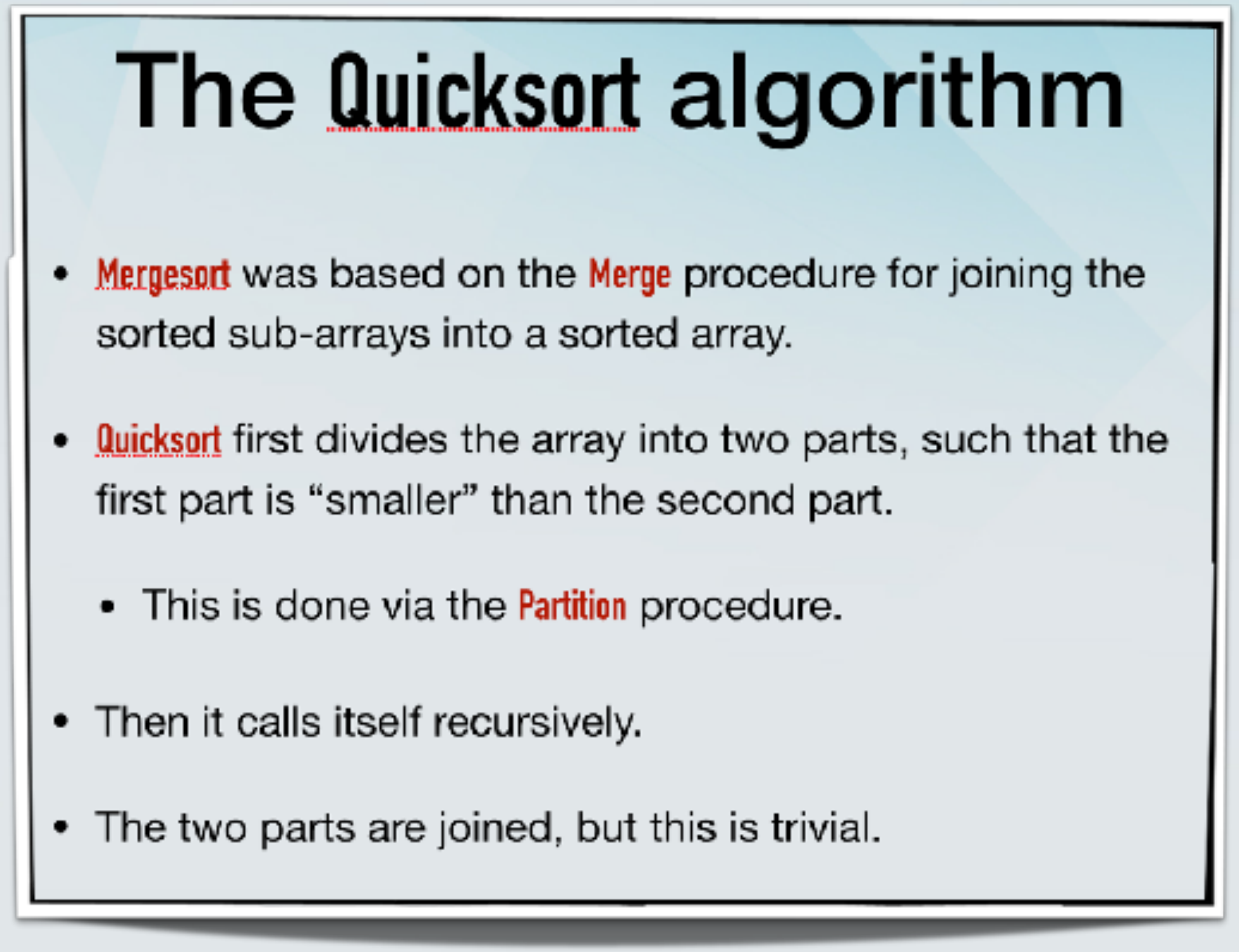
- Sort the numbers in $O(n \log n)$ time using **MergeSort**.
- Return the *i*-th element of the sorted array.
- Is sorting an overkill?

Divide and conquer

- Split the input into smaller inputs.
- Solve the problem for the smaller inputs recursively.
- Combine the solutions into a solution for the original problem.

A glance back at the QuickSort algorithm

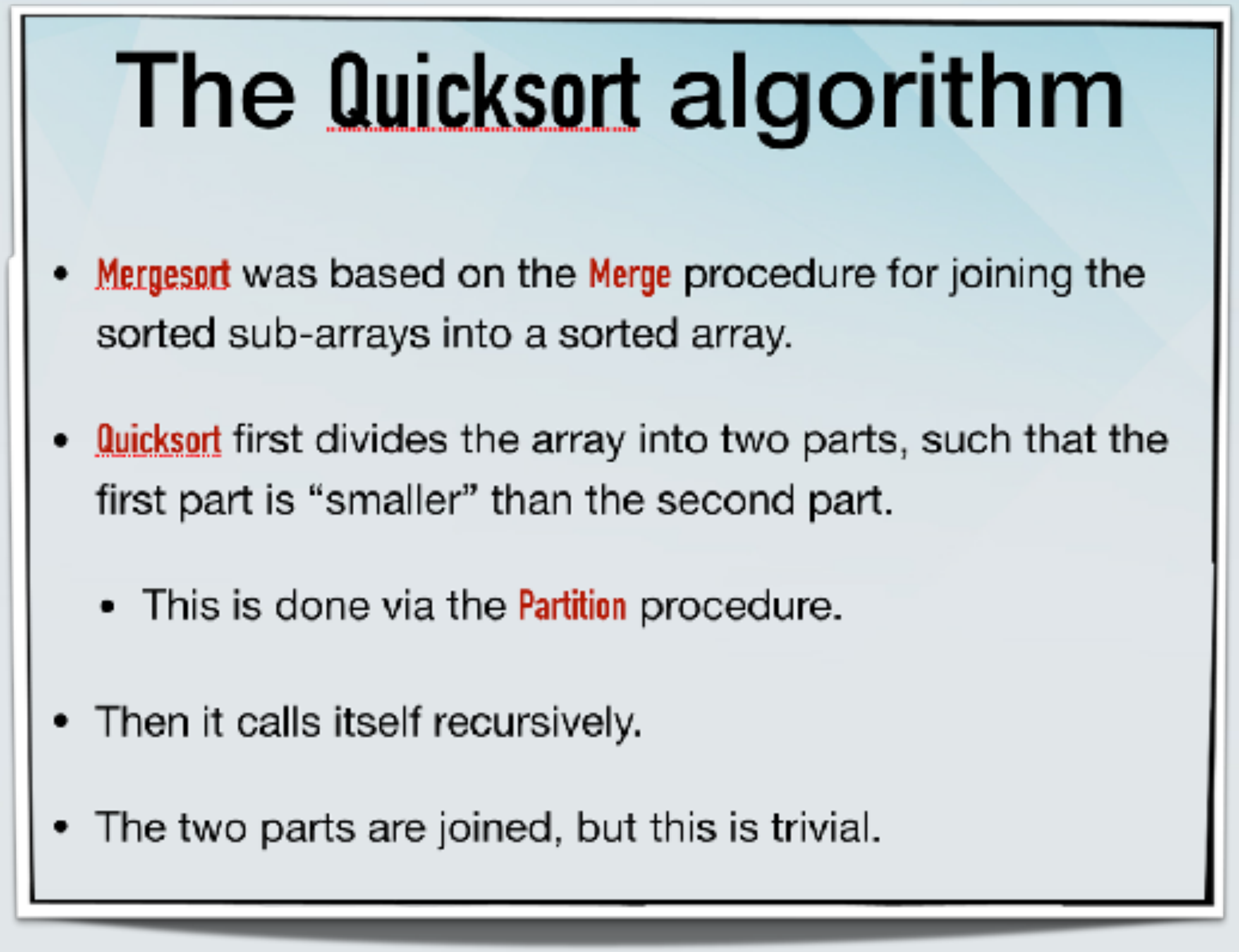
A glance back at the QuickSort algorithm



The Quicksort algorithm

- Mergesort was based on the Merge procedure for joining the sorted sub-arrays into a sorted array.
- Quicksort first divides the array into two parts, such that the first part is "smaller" than the second part.
  - This is done via the Partition procedure.
- Then it calls itself recursively.
- The two parts are joined, but this is trivial.

A glance back at the QuickSort algorithm



The Quicksort algorithm

- Mergesort was based on the Merge procedure for joining the sorted sub-arrays into a sorted array.
- Quicksort first divides the array into two parts, such that the first part is "smaller" than the second part.
    - This is done via the Partition procedure.
- Then it calls itself recursively.
- The two parts are joined, but this is trivial.

Recall the Partition procedure

Revisiting the Partition procedure

```
Procedure Partition(A[$i,\ldots,j$])

    Choose a pivot element x of A

    $k = i-1$

    For $h = i$ to $j-1$ do

        If A[$h$] $\leq$ x
            $k = k + 1$
            Swap A[$k$] with A[$h$]

    Swap A[$k+1$] with A[$j$]

Return $k+1$
```

Running time **O(n)**

Revisiting the Partition procedure

```
Procedure Partition($A[i,\dots,j]$),$x$)

    ~~Choose a pivot element x of A~~

    $k = i-1$

    For $h = i$ to $j-1$ do

        If $A[h] \leq x$
            $k = k + 1$
            Swap $A[k]$ with $A[h]$

        Swap $A[k+1]$ with $A[j]$

Return $k+1$
```

Running time **O(n)**

What does Partition do?

What does Partition do?

- Using the element **x**, it divides the array **A** into three parts: **A**[*1,…x-1*], **A**[**x**] and **A**[*x+1, … , n*].

What does Partition do?

- Using the element **x**, it divides the array **A** into three parts: **A**[$1,\dots x-1$], **A**[**x**] and **A**[$x+1, \dots , n$].
- Then, we can reduce the search for the i-th element to one of the three subarrays.

What does Partition do?

- Using the element **x**, it divides the array **A** into three parts: **A**[*1,…x-1*], **A**[**x**] and **A**[*x+1, … , n*].
- Then, we can reduce the search for the *i*-th element to one of the three subarrays.
- How can we choose the element **x** *appropriately*, such that the subarrays **A**[*1,…x-1*] and **A**[*x+1, … , n*] are of (approximately) equal size?

What does Partition do?

What does Partition do?

- How can we choose the element **x** *appropriately*, such that the subarrays **A**[*1,…x-1*] and **A**[*x+1, … , n*] are of (approximately) equal size?

What does Partition do?

- How can we choose the element **x** *appropriately*, such that the subarrays **A**[*1,…x-1*] and **A**[*x+1, … , n*] are of (approximately) equal size?

- We could find the median of the array and use that as the value **x**.

What does Partition do?

- How can we choose the element **x** *appropriately*, such that the subarrays **A**[*1,…x-1*] and **A**[*x+1, … , n*] are of (approximately) equal size?

- We could find the median of the array and use that as the value **x**.

    - The median is the number that is larger than exactly ($n+1)/2 - 1$ numbers.

What does Partition do?

- How can we choose the element **x** *appropriately*, such that the subarrays **A**[$1,\dots x-1$] and **A**[$x+1, \dots , n$] are of (approximately) equal size?

- We could find the median of the array and use that as the value **x**.
    - The median is the number that is larger than exactly $(n+1)/2 - 1$ numbers.
    - The median is the [$(n+1)/2$]th-*order statistic.*

What does Partition do?

- How can we choose the element **x** *appropriately*, such that the subarrays **A**[*1,…x-1*] and **A**[*x+1, … , n*] are of (approximately) equal size?

- We could find the median of the array and use that as the value **x**.

    - The median is the number that is larger than exactly $(n+1)/2 - 1$ numbers.

    - The median is the $[(n+1)/2]^{th}$*-order statistic.*

- What is an algorithm for finding the median?

What does Partition do?

- How can we choose the element **x** *appropriately*, such that the subarrays **A**[*1,…x-1*] and **A**[*x+1, … , n*] are of (approximately) equal size?

- We could find the median of the array and use that as the value **x**.

    - The median is the number that is larger than exactly $(n+1)/2 - 1$ numbers.

    - The median is the $[(n+1)/2]^{th}$*-order statistic.*

- What is an algorithm for finding the median?

    - Selection(**A**[*1,…,n*],$(n+1)/2$)

Let’s try to do that…

```
Algorithm Selection(A[$1,\ldots,n$],$i$)

    x = Selection(A[$1,\ldots,n$],$(n+1)/2$)
    y = Partition(A[$1,\ldots,n$],x)
```

Let's try to do that...

```
Algorithm Selection(A[$1,\ldots,n$],$i$)

    x = Selection(A[$1,\ldots,n$],$(n+1)/2$)
    y = Partition(A[$1,\ldots,n$],x)
```

Do you see a problem?

Let's try to do that…

```
Algorithm Selection(A[1,…,n],i)

    x = Selection(A[1,…,n],(n+1)/2)
    y = Partition(A[1,…,n],x)
```

Do you see a problem?

Before you conquer, you need to divide!

Let’s try to do that…

```
Algorithm Selection(A[$1,\ldots,n$],$i$)

    x = Selection(A[$1,\ldots,n$],$(n+1)/2$)
    y = Partition(A[$1,\ldots,n$],x)
```

Do you see a problem?

Before you conquer, you need to divide!

Are we stuck?

- We need to partition the array into two using a good pivot element (the median).
    - Or otherwise the running time of the recursion will be bad!
- But to find the median, we need an algorithm for selection!

Are we stuck?

- We need to partition the array into two using a good pivot element (something “close” to the median).
  - Or otherwise the running time of the recursion will be bad!
- But to find the median, we need an algorithm for selection!

A good pivot element

- Split the array **A** into sub-arrays with **5** elements each.
  - The last one might have fewer elements.

A good pivot element

A good pivot element

□□□□□□□□□□□□□□□□□□

□□□□□

A good pivot element

□□□□□□□□□□□□□

□□□□□ □□□□□

A good pivot element

[illegible]

[illegible] [illegible] [illegible]

A good pivot element

[illegible]

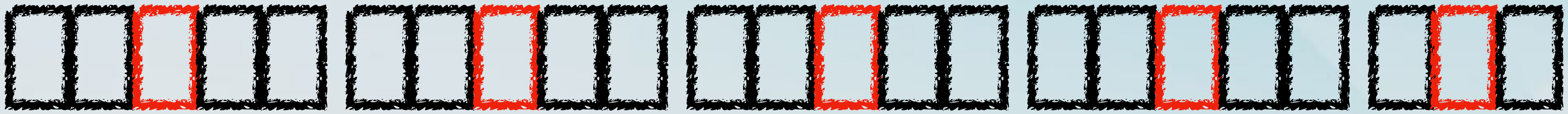
[illegible]

A good pivot element

A good pivot element

- Split the array **A** into sub-arrays with **5** elements each.
    - The last one might have fewer elements.
- For each one of those, find the median.

A good pivot element



A good pivot element

- Split the array **A** into sub-arrays with **5** elements each.
    - The last one might have fewer elements.
- For each one of those, find the median.
    - How do we do that?

A good pivot element

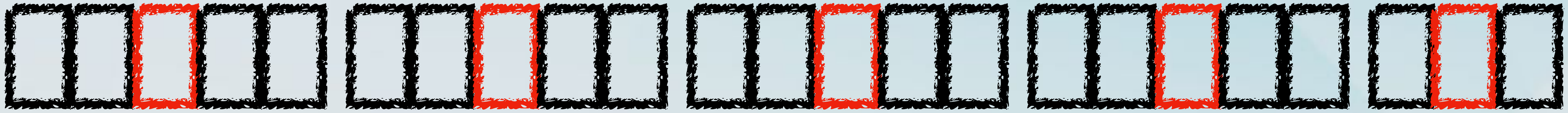
- Split the array **A** into sub-arrays with **5** elements each.
    - The last one might have fewer elements.
- For each one of those, find the median.
    - How do we do that?

Run InsertionSort

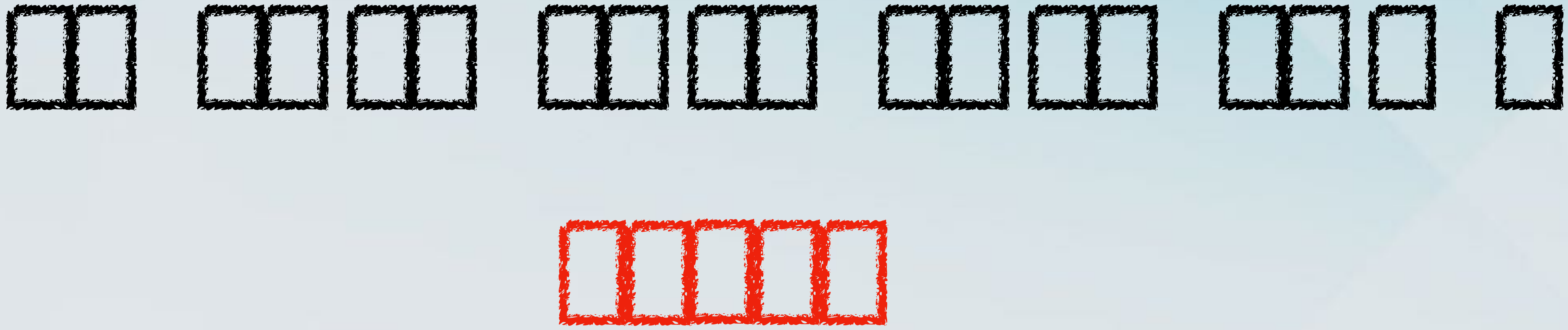
A good pivot element

- Split the array **A** into sub-arrays with **5** elements each.
    - The last one might have fewer elements.
- For each one of those, find the median.
- Find the **median-of-medians**.

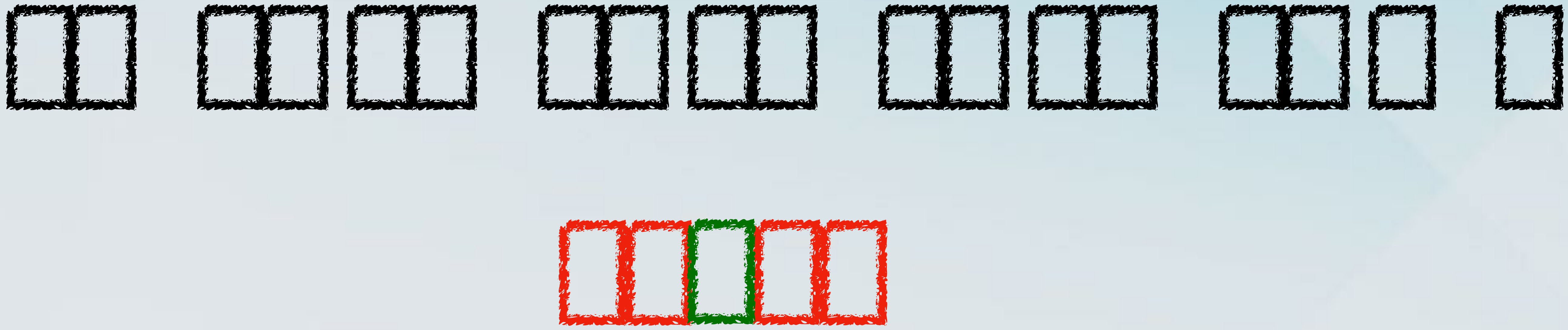
Median of medians



Median of medians



Median of medians



A good pivot element

- Split the array **A** into sub-arrays with **5** elements each.
    - The last one might have fewer elements.
- For each one of those, find the median.
- Find the **median-of-medians**.
    - How do we do that?

A good pivot element

- Split the array **A** into sub-arrays with **5** elements each.
  - The last one might have fewer elements.
- For each one of those, find the median.
- Find the **median-of-medians**.
  - How do we do that?

    Run Selection

This failed...

```
Algorithm Selection(A[$1,\ldots,n$],$i$)

    x = Selection(A[$1,\ldots,n$],$(n+1)/2$)
    y = Partition(A[$1,\ldots,n$],x)
```

...but this won't.

```
Algorithm Selection(A[$1,\ldots,n$],$i$)

    Split the array A into n/5 arrays of size 5
    For each subarray $A_i$, find the median
    Let $m_1, m_2, \ldots, m_{n/5}$ be those medians

    x = Selection(A[$1,\ldots,n$],$(n+1)/2$)   /*Find the median of medians */
    y = Partition(A[$1,\ldots,n$],x)   /*Partition the array using x as the pivot */
```

The Selection algorithm (not exactly pseudocode)

```
Algorithm Selection(A[$1,\ldots,n$],$i$)

    Split the array A into n/5 arrays of size 5
    For each subarray $A_i$, find the median
    Let $m_1, m_2, \ldots, m_{n/5}$ be those medians

     x = Selection(A[$1,\ldots,n$],$(n+1)/2$)  /*Find the median of medians */
     k = Partition(A[$1,\ldots,n$],x)  /*Partition the array using x as the pivot */

     k-1 is the number of elements in the lower subarray.

     If $i$ = k, return x
     If $i$ < k, return Selection(A[$1,\ldots,k-1$],$i$)
     If $i$ > k, return Selection(A[$k+1,\ldots,n$],$i-k$)
```

Zooming in

```
If $i$ = k, return **x**
If $i$ < k, return Selection(A[$1,\ldots,k-1$],$i$)
If $i$ > k, return Selection(A[$k+1,\ldots,n$],$i-k$)
```

- We are looking for the i^{th}-order statistic.
- If i=k, then x is the answer - it is larger than k-1 elements.
- If $i \leq$ k, the answer cannot be in the second part, as then i would be larger than at least k-1 elements.
- For the same reason, if $i \geq$ k, the answer cannot be in the first part.

Running time

Algorithm Selection(**A**[$1,\ldots,n$],i)

```
    Split the array A into n/5 arrays of size 5
    For each subarray A_i, find the median
    Let $m_1, m_2, \ldots, m_{n/5}$ be those medians

     x = Selection(A[$1,\ldots,n$],$(n+1)/2$)   /*Find the median of medians */
     k = Partition(A[$1,\ldots,n$],x)   /*Partition the array using x as the pivot */

     k-1 is the number of elements in the lower subarray.

     If $i$ = k, return x
     If $i$ < k, return Selection(A[$1,\ldots,k-1$],$i$)
     If $i$ > k, return Selection(A[$k+1,\ldots,n$],$i-k$)
```

Running time

```
Algorithm Selection(A[1,…,n],i)

O(n)   Split the array A into n/5 arrays of size 5
       For each subarray A_i, find the median
       Let $m_1, m_2, \dots, m_{n/5}$ be those medians

         x = Selection(A[1,…,n],(n+1)/2)   /*Find the median of medians */
         k = Partition(A[1,…,n],x)   /*Partition the array using x as the pivot */

         k-1 is the number of elements in the lower subarray.

         If $i = k$, return x
         If $i < k$, return Selection(A[1,…,k-1],i)
         If $i > k$, return Selection(A[k+1,…,n],i-k)
```

Running time

Algorithm **Selection**(**A**[$1,\ldots,n$],i)

O(n) Split the array **A** into n/5 arrays of size 5
O(n) For each subarray $\mathbf{A_i}$, find the median
Let $m_1, m_2, \ldots, m_{n/5}$ be those medians

```
x = Selection(A[$1,\ldots,n$],$(n+1)/2$)   /*Find the median of medians */
k = Partition(A[$1,\ldots,n$],x)   /*Partition the array using x as the pivot */

k-1 is the number of elements in the lower subarray.

If $i$ = k, return x
If $i$ < k, return Selection(A[$1,\ldots,k-1$],$i$)
If $i$ > k, return Selection(A[$k+1,\ldots,n$],$i-k$)
```

Running time

```
Algorithm Selection(A[$1,\dots,n$],$i$)

O(n)    Split the array A into n/5 arrays of size 5
        For each subarray $A_i$, find the median
O(n)    Let $m_1, m_2, \dots, m_{n/5}$ be those medians

T(n/5)  x = Selection(A[$1,\dots,n$],$(n+1)/2$)   /*Find the median of medians */
        k = Partition(A[$1,\dots,n$],x)   /*Partition the array using x as the pivot */

        k-1 is the number of elements in the lower subarray.

        If $i$ = k, return x
        If $i$ < k, return Selection(A[$1,\dots,k-1$],$i$)
        If $i$ > k, return Selection(A[$k+1,\dots,n$],$i-k$)
```

Running time

Algorithm **Selection**(**A**[$1,\dots,n$],i)

```
O(n)     Split the array A into n/5 arrays of size 5
O(n)     For each subarray A_i, find the median
         Let $m_1, m_2, \dots, m_{n/5}$ be those medians

T(n/5)   x = Selection(A[$1,\dots,n$],$(n+1)/2$)   /*Find the median of medians */
O(n)     k = Partition(A[$1,\dots,n$],x)   /*Partition the array using x as the pivot */

         k-1 is the number of elements in the lower subarray.

         If $i$ = k, return x
         If $i$ < k, return Selection(A[$1,\dots,k-1$],$i$)
         If $i$ > k, return Selection(A[$k+1,\dots,n$],$i-k$)
```

Running time

Algorithm Selection(**A**[$1,\ldots,n$],i)

```
O(n)    Split the array A into n/5 arrays of size 5
        For each subarray A_i, find the median
O(n)    Let $m_1, m_2, \ldots, m_{n/5}$ be those medians

T(n/5)  x = Selection(A[$1,\ldots,n$],$(n+1)/2$)  /*Find the median of medians */
O(n)    k = Partition(A[$1,\ldots,n$],x)  /*Partition the array using x as the pivot */

        k-1 is the number of elements in the lower subarray.

O(1)    If $i$ = k, return x
        If $i$ < k, return Selection(A[$1,\ldots,k-1$],$i$)
        If $i$ > k, return Selection(A[$k+1,\ldots,n$],$i-k$)
```

Running time

```
Algorithm Selection(A[1,…,n],i)

O(n)     Split the array A into n/5 arrays of size 5
O(n)     For each subarray A_i, find the median
         Let $m_1, m_2, \dots, m_{n/5}$ be those medians

T(n/5)   x = Selection(A[1,…,n],(n+1)/2)  /*Find the median of medians */
O(n)     k = Partition(A[1,…,n],x)  /*Partition the array using x as the pivot */

         k-1 is the number of elements in the lower subarray.

O(1)     If $i = k$, return x
T(|S_max|)  If $i < k$, return Selection(A[1,…,k-1],i)
         If $i > k$, return Selection(A[k+1,…,n],i-k)
```

$|S_{max}| = \max(k-1, n-k)$

Running time

Running time

$T(n) \leq T(n/5) + T(|S_{max}|) + bn$

Running time

$T(n) \leq T(n/5) + T(|S_{max}|) + bn$

Before we proceed, we have to bound $|S_{max}|$.

Bounding the size of the subarrays

Bounding the size of the subarrays

- **x** is a median of medians.

Bounding the size of the subarrays

- x is a median of medians.
- At least (…) subarrays have “baby medians” ≥ x.

Bounding the size of the subarrays

- **x** is a median of medians.
- At least half of the subarrays have “baby medians” ≥ **x**.

Bounding the size of the subarrays

- **x** is a median of medians.
- At least half of the subarrays have “baby medians” ≥ **x**.
- Each one of these groups has at least (…) elements > **x.**

Bounding the size of the subarrays

- **x** is a median of medians.
- At least half of the subarrays have "baby medians" ≥ **x**.
- Each one of these groups has at least 3 elements > **x.**

Bounding the size of the subarrays

- **x** is a median of medians.
- At least half of the subarrays have “baby medians” ≥ **x**.
- Each one of these groups has at least 3 elements > **x.**
    - Because **x** ≤ their “baby median”.
    - Except possibly

Bounding the size of the subarrays

- **x** is a median of medians.
- At least half of the subarrays have “baby medians” ≥ **x**.
- Each one of these groups has at least 3 elements > **x.**
    - Because **x** ≤ their “baby median”.
    - Except possibly the group containing **x** and

Bounding the size of the subarrays

- **x** is a median of medians.
- At least half of the subarrays have “baby medians” ≥ **x**.
- Each one of these groups has at least 3 elements > **x.**
  - Because **x** ≤ their “baby median”.
  - Except possibly the group containing **x** and the group that has fewer than 5 elements.

Bounding the size of the subarrays

What is the total number of elements larger than **x**?

$$3\left(\left\lceil \frac{1}{2}\cdot\left\lceil\frac{n}{5}\right\rceil\right\rceil - 2\right) \geq \frac{3n}{10} - 6$$

# elements > x in each of those groups

# groups

groups who could be exceptions

# groups with “baby medians” > x

This means that the size of the lower subarray is at most **7n/10 + 6**

Bounding the size of the subarrays

Bounding the size of the subarrays

- The size of the lower subarray is at most **7n/10 + 6**

Bounding the size of the subarrays

- The size of the lower subarray is at most **7n/10 + 6**
- A symmetric argument shows that the size of the upper subarray is at most **7n/10 + 6**

Bounding the size of the subarrays

- The size of the lower subarray is at most **7n/10 + 6**
- A symmetric argument shows that the size of the upper subarray is at most **7n/10 + 6**
- Back to the recurrence:

  $T(n) \leq T(n/5) + T\,(|S_{max}|) + cn = T(n/5) + T\,(7n/10+6) + bn$

Solving the recurrence

Solving the recurrence

- Lets guess that $T(n) \le cn$, for some constant c.

Solving the recurrence

- Lets guess that $T(n) \le cn$, for some constant c.
- We get that

$$\begin{aligned} T(n) &\le c(n/5) + c(7n/10+6) + bn \\ &= 9cn/10 + 7c + bn \\ &= cn + (-cn/10 + 7c + bn) \end{aligned}$$

Solving the recurrence

- Lets guess that $T(n) \le cn$, for some constant c.
- We get that

  $$\begin{aligned} T(n) &\le c(n/5) + c(7n/10+6) + bn \\ &= 9cn/10 + 7c + bn \\ &= cn + (-cn/10 + 7c + bn) \end{aligned}$$

- This is at most cn whenever $-cn/10 + 7c + bn \le 0$, or equivalently, when $c \ge 10bn/(n-70)$.

Solving the recurrence

- Lets guess that $T(n) \le cn$, for some constant c.
- We get that

  $$\begin{aligned} T(n) &\le c(n/5) + c(7n/10+6) + bn \\ &= 9cn/10 + 7c + bn \\ &= cn + (-cn/10 + 7c + bn) \end{aligned}$$

- This is at most cn whenever $-cn/10 + 7c + bn \le 0$, or equivalently, when $c \ge 10bn/(n-70)$.
- If $n \ge 140$, then $n/(n-70) \le 2$ and then, it suffices to have $c \ge 20b$.

Solving the recurrence

Solving the recurrence

- We want to show that there is some constant $c > 0$, such that $T(n) \leq cn$ for all $n > 0$.

Solving the recurrence

- We want to show that there is some constant $c > 0$, such that $T(n) \leq cn$ for all $n > 0$.

- Let $a = \max\{ T(n) / n , n \leq 140\}$ and let $c = \max\{a, 20b\}$.

Solving the recurrence

- We want to show that there is some constant $c > 0$, such that $T(n) \leq cn$ for all $n > 0$.

- Let $a = \max\{ T(n) / n , n \leq 140\}$ and let $c = \max\{a, 20b\}$.

- We will prove the statement by induction.

Solving the recurrence

- We want to show that there is some constant $c > 0$, such that $T(n) \leq cn$ for all $n > 0$.
- Let $a = \max\{ T(n) / n , n \leq 140\}$ and let $c = \max\{a, 20b\}$.
- We will prove the statement by induction.
    - **Base case:** For every $n \leq 140$, $T(n) \leq cn$

Solving the recurrence

- We want to show that there is some constant $c > 0$, such that $T(n) \leq cn$ for all $n > 0$.
- Let $a = \max\{ T(n) / n , n \leq 140\}$ and let $c = \max\{a, 20b\}$.
- We will prove the statement by induction.
    - **Base case:** For every $n \leq 140$, $T(n) \leq cn$
    - **Inductive Step:** Suppose that it holds for all n up to k=140. Then for $n=k+1$, we have $T(n) \leq cn + (-cn/10 + 7c + bn)$

Solving the recurrence

- We want to show that there is some constant $c > 0$, such that $T(n) \leq cn$ for all $n > 0$.

- Let $a = \max\{ T(n) / n \, , n \leq 140\}$ and let $c = \max\{a, 20b\}$.

- We will prove the statement by induction.

  - **Base case:** For every $n \leq 140$, $T(n) \leq cn$

  - **Inductive Step:** Suppose that it holds for all n up to k=140. Then for $n=k+1$, we have $T(n) \leq cn + (-cn/10 + 7c + bn)$

    - This follows from the fact that $n > 140$ and $c \geq 20b$.

Bonus: The Master Theorem

Suppose $T(n) \leq \alpha T\left(\lceil n/b \rceil\right) + O(n^d)$

for some constants $\alpha > 0,\ b > 1$ and $d \geq 0$.

$$\text{Then,}\ \ T(n) = \begin{cases} O(n^d), & \text{if } d > \log_b \alpha \\ O(n^d \log_b n), & \text{if } d = \log_b \alpha \\ O(n^{\log_b \alpha}), & \text{if } d < \log_b \alpha \end{cases}$$

Bonus: The Master Theorem

Suppose $T(n) \leq \alpha T\left(\lceil n/b \rceil\right) + O(n^d)$
for some constants $\alpha > 0,\ b > 1$ and $d \geq 0$.

$$\text{Then,}\quad T(n) = \begin{cases} O(n^d), & \text{if } d > \log_b \alpha \\ O(n^d \log_b n), & \text{if } d = \log_b \alpha \\ O(n^{\log_b \alpha}), & \text{if } d < \log_b \alpha \end{cases}$$

Example: For **MergeSort**, α=b=2 and d=1, we get **O(n log n).**