Algorithms and Data Structures

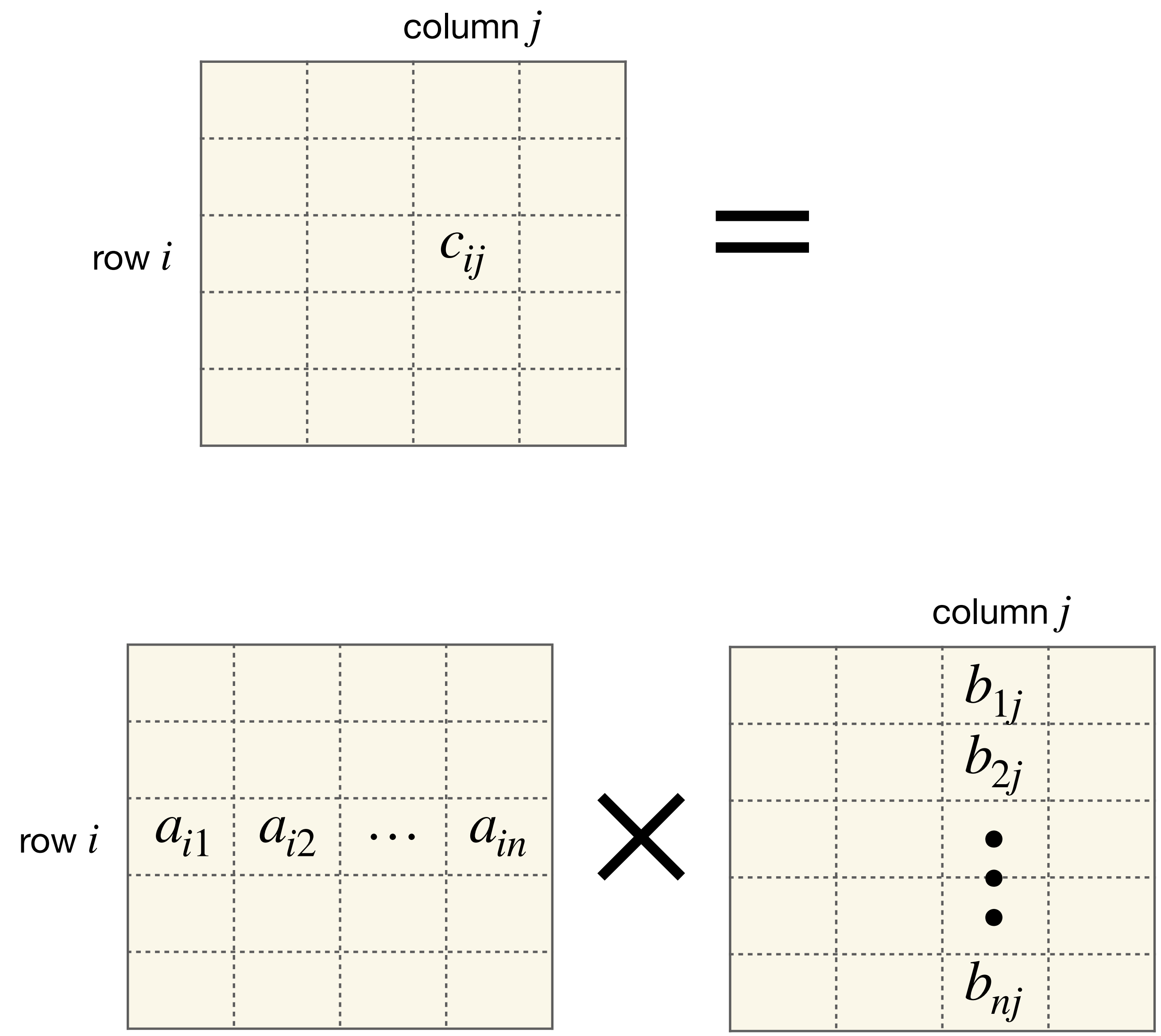
Upper and Lower Bounds for Sorting, Matrix Multiplication

Matrix Multiplication

Assume that we have two square $(n \times n)$-matrices $A = (a_{ij})_{1 \le i,j \le n}$ and $B = (b_{ij})_{1 \le i,j \le n}$

The product of A and B is the $(n \times n)$-matrix $C = (c_{ij})_{1 \le i,j \le n}$ with entries

$$c_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj}$$



Matrix Multiplication

Straightforward approach (3 nested loops): $O(n^3)$.

Naive Divide & Conquer approach: $O(n^3)$

Can we do better than that?

How to calculate $x^2 - y^2$

Straightforward approach: Two multiplications and one subtraction (addition).

We could also use the identity $x^2 - y^2 = (x + y) \cdot (x - y)$: One multiplication and two additions.

For scalars like x and y, multiplications and additions cost the same.

For (possibly large matrices), multiplications are more expensive!

Divide and Conquer…

Suppose we divide our matrices A and B as follows:

$$A = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} \qquad B = \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix}$$

We can write C as:

$$\begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{pmatrix} = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} \times \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix}$$

$$= \begin{pmatrix} A_{11} \cdot B_{11} + A_{12} \cdot B_{21} & A_{11} \cdot B_{12} + A_{12} \cdot B_{22} \\ A_{21} \cdot B_{11} + A_{22} \cdot B_{21} & A_{21} \cdot B_{12} + A_{22} \cdot B_{22} \end{pmatrix}$$

…using fewer multiplications.

$$P_1 = (A_{11} + A_{22}) \cdot (B_{11} + B_{22})$$

$$P_2 = (A_{21} + A_{22}) \cdot B_{11}$$

$$P_3 = A_{11} \cdot (B_{12} - B_{22})$$

$$P_4 = A_{22} \cdot (-B_{11} + B_{21})$$

$$P_5 = (A_{11} + A_{22}) \cdot B_{22}$$

$$P_6 = (-A_{11} + A_{21}) \cdot (B_{11} + B_{12})$$

$$P_7 = (A_{12} - A_{22}) \cdot (B_{21} + B_{22})$$

$$C_{11} = P_1 + P_4 - P_5 + P_7 \qquad C_{12} = P_3 + P_5$$

$$C_{21} = P_2 + P_4 \qquad C_{22} = P_1 + P_3 - P_2 + P_6$$

Try it at home: Check C_{12}, C_{21}, and C_{22}.

Let's calculate C_{11}

$P_1 = (A_{11} + A_{22}) \cdot (B_{11} + B_{22})$

$P_2 = (A_{21} + A_{22}) \cdot B_{11}$

$P_3 = A_{11} \cdot (B_{12} - B_{22})$

$P_4 = A_{22} \cdot (-B_{11} + B_{21})$

$P_5 = (A_{11} + A_{22}) \cdot B_{22}$

$P_6 = (-A_{11} + A_{21}) \cdot (B_{11} + B_{12})$

$P_7 = (A_{12} - A_{22}) \cdot (B_{21} + B_{22})$

$$C_{11} = P_1 + P_4 - P_5 + P_7$$

$$\begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{pmatrix} = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} \times \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix}$$

$$= \begin{pmatrix} A_{11} \cdot B_{11} + A_{12} \cdot B_{21} & A_{11} \cdot B_{12} + A_{12} \cdot B_{22} \\ A_{21} \cdot B_{11} + A_{22} \cdot B_{21} & A_{21} \cdot B_{12} + A_{22} \cdot B_{22} \end{pmatrix}$$

$$P_1 = (A_{11} + A_{22}) \cdot (B_{11} + B_{22}) = A_{11} \cdot B_{11} + A_{11} \cdot B_{22} + A_{22} \cdot B_{11} + A_{22} \cdot B_{22}$$

$$P_4 = A_{22} \cdot (-B_{11} + B_{21}) = -A_{22} \cdot B_{11} + A_{22} \cdot B_{21}$$

$$P_5 = (A_{11} + A_{12}) \cdot B_{22} = A_{11} \cdot B_{22} + A_{12} \cdot B_{22}$$

$$P_7 = (A_{12} - A_{22}) \cdot (B_{21} + B_{22}) = A_{12} \cdot B_{21} + A_{12} \cdot B_{22} - A_{22} \cdot B_{21} - A_{22} \cdot B_{22}$$

$$P_1 + P_4 = A_{11} \cdot B_{11} + A_{11} \cdot B_{22} + A_{22} \cdot B_{22} + A_{22} \cdot B_{21}$$

$$P_1 + P_4 - P_5 = A_{11} \cdot B_{11} + A_{22} \cdot B_{22} + A_{22} \cdot B_{21} - A_{12} \cdot B_{22}$$

$$P_1 + P_4 - P_5 + P_7 = A_{11} \cdot B_{11} + A_{12} \cdot B_{21}$$

How many multiplications do we need?

$$P_1 = (A_{11} + A_{22}) \cdot (B_{11} + B_{22})$$
$$P_2 = (A_{21} + A_{22}) \cdot B_{11}$$
$$P_3 = A_{11} \cdot (B_{12} - B_{22})$$
$$P_4 = A_{22} \cdot (-B_{11} + B_{21})$$
$$P_5 = (A_{11} + A_{22}) \cdot B_{22}$$
$$P_6 = (-A_{11} + A_{21}) \cdot (B_{11} + B_{12})$$
$$P_7 = (A_{12} - A_{22}) \cdot (B_{21} + B_{22})$$

We have 7 multiplications of $n/2 \times n/2$ matrices.

$$C_{11} = P_1 + P_4 - P_5 + P_7 \qquad C_{12} = P_3 + P_5$$
$$C_{21} = P_2 + P_4 \qquad C_{22} = P_1 + P_3 - P_2 + P_6$$

Recurrence Relation

We have 7 multiplications of $n/2 \times n/2$ matrices.

As before, the additions will take $O(n^2)$ time.

$$T(n) = 7T(n/2) + O(n^2)$$
$$T(n) = O(n^{\log_2 7}) = O(n^{2.81})$$

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

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

Matrix Multiplication Upper and Lower Bounds

Strassen's method achieves a running time of $O(n^{\log_2 7}) = O(n^{2.81})$

Is this the best we can do? Lower bounds?

Obvious lower bound: $\Omega(n^2)$

Better upper/lower bounds?

Timeline of matrix multiplication exponent

Year	Bound on omega	Authors
1969	2.8074	Strassen[1]
1978	2.796	Pan[10]
1979	2.780	Bini, Capovani [it], Romani[11]
1981	2.522	Schönhage[12]
1981	2.517	Romani[13]
1981	2.496	Coppersmith, Winograd[14]
1986	2.479	Strassen[15]
1990	2.3755	Coppersmith, Winograd[16]
2010	2.3737	Stothers[17]
2012	2.3729	Williams[18][19]
2014	2.3728639	Le Gall[20]
2020	2.3728596	Alman, Williams[21][22]
2022	2.371866	Duan, Wu, Zhou[23]
2024	2.371552	Williams, Xu, Xu, and Zhou[2]

Selection

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**[$1,\ldots,n$], 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

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.

The Partition procedure

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

~~Choose a **pivot element** x of **A**~~

$k = i$

For $h = i$ to j do

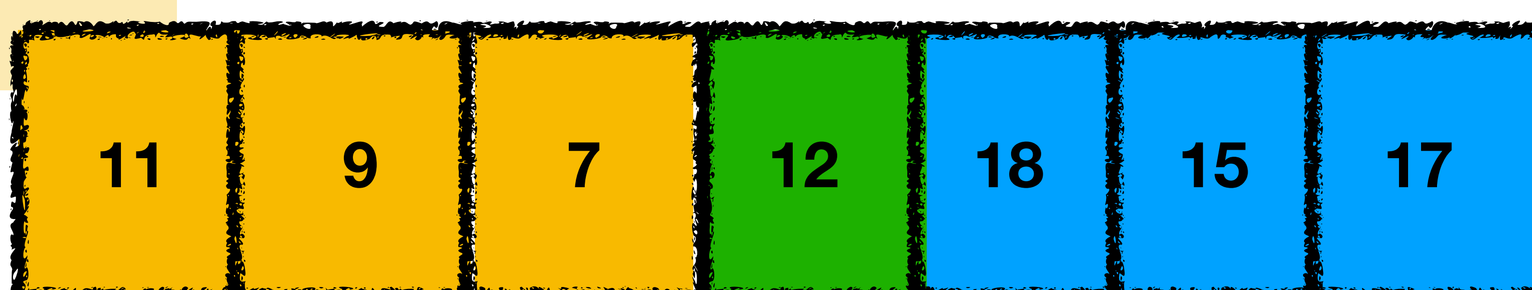
If **A**[h] < x

Swap **A**[k] with **A**[h]
$k = k + 1$

Swap **A**[k] with **A**[h]

Return k

Running time **O(n)**



The Partition procedure (with the pivot element as input)

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

~~Choose a **pivot element** x of **A**~~

$k = i$

For $h = i$ to j do

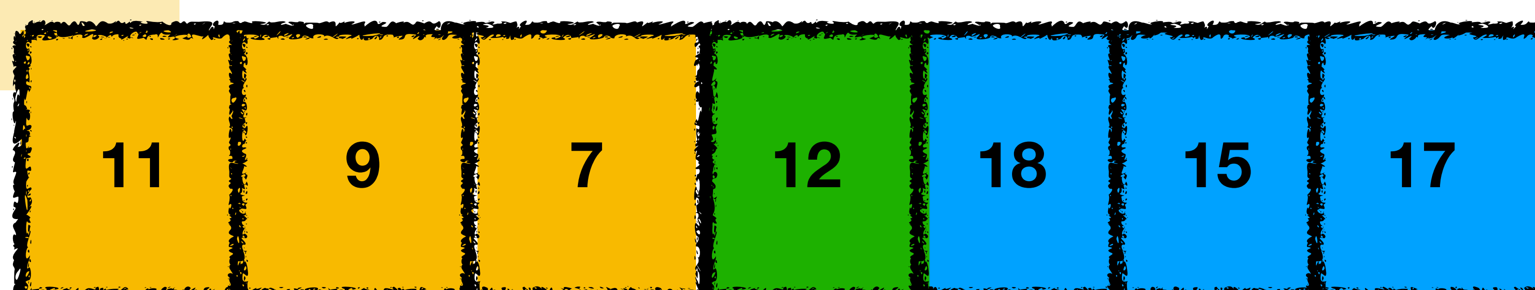
If **A**[h] < x

Swap **A**[k] with **A**[h]
$k = k + 1$

Swap **A**[k] with **A**[h]

Return k

Running time **O(n)**



What does Partition do?

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

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**$[1,\ldots,x-1]$ and **A**$[x+1,\ldots,n]$ are of (approximately) equal size?

What does Partition do?

How can we choose the element x *appropriately*, such that the subarrays **A**[$1,\ldots,x-1$] and **A**[$x+1,\ldots,n$] 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 $\frac{n+1}{2}-1$ numbers.

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

What is an algorithm for finding the median?

Selection(A[$1,\ldots,n$],$(n+1)/2$)

Let's try to do that…

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

Do you see a problem?

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

k = **Partition**(**A**[$1,\ldots,n$], x)

Let's try to do that…

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

Do you see a problem?

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

k = **Partition**(**A**$[1,\ldots,n]$, x)

Before we conquer, we 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

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?

Run **InsertionSort**

A good pivot element

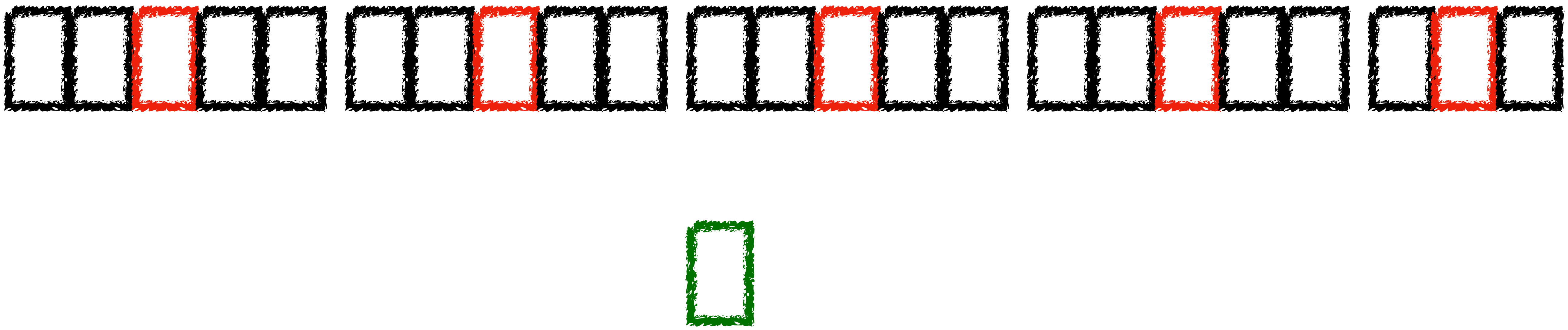
Split the array **A** into sub-arrays with ***5** elements* each.

The last one might have fewer elements.

For each one of those, find the *median*.

Find the **median-of-medians**.

Median of medians



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**$[1,\ldots,n]$,$(n+1)/2$)

k = **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**[$m_1, \ldots, m_{n/5}$],$(n/5+1)/2$)

/*Find the median of medians */

k = **Partition**(**A**[$1,\ldots,n$], x) /*Partition the array using x as the pivot */

The Selection algorithm

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**[$m_1, \ldots, m_{n/5}$],$(n/5 + 1)/2$)

/*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)

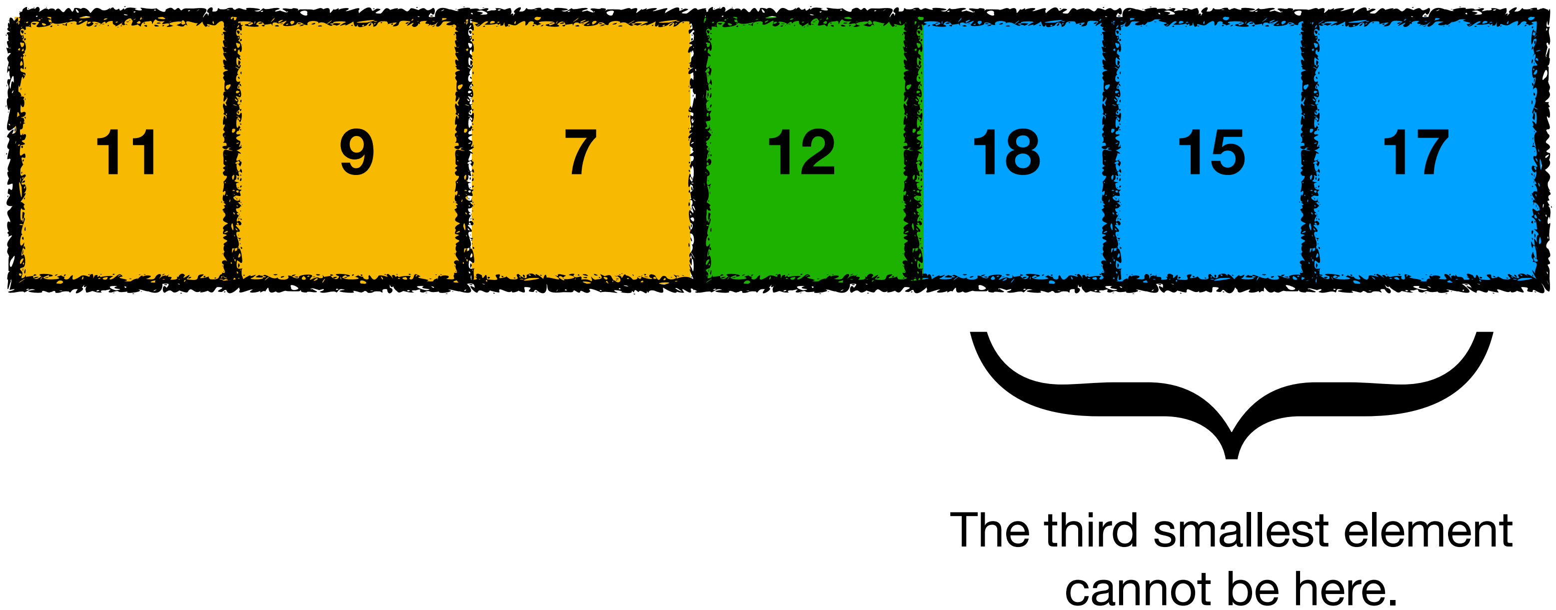
If i > k, return **Selection**(**A**[$k+1,\ldots,n$], $i-k$)

We are looking for the i^{th}-order statistic.

If i = k, then x is the answer - it is larger than $k-1=i-1$ elements.

If i < k, the answer cannot be in the second part, as then i would be larger than at least $k-1=i-1$ elements.

Zooming in



We are looking for the third smallest element ($i = 3$)

And in our case the pivot is in the fourth position, $k = 4$

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 $i < k$, 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 $i > k$, the answer cannot be in the first part.

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**[$m_1, \ldots, m_{n/5}$],$(n/5+1)/2$)

/*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

$T(|S_{max}|)$ 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

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

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

Bounding the size of the subarrays

x is a median of medians.

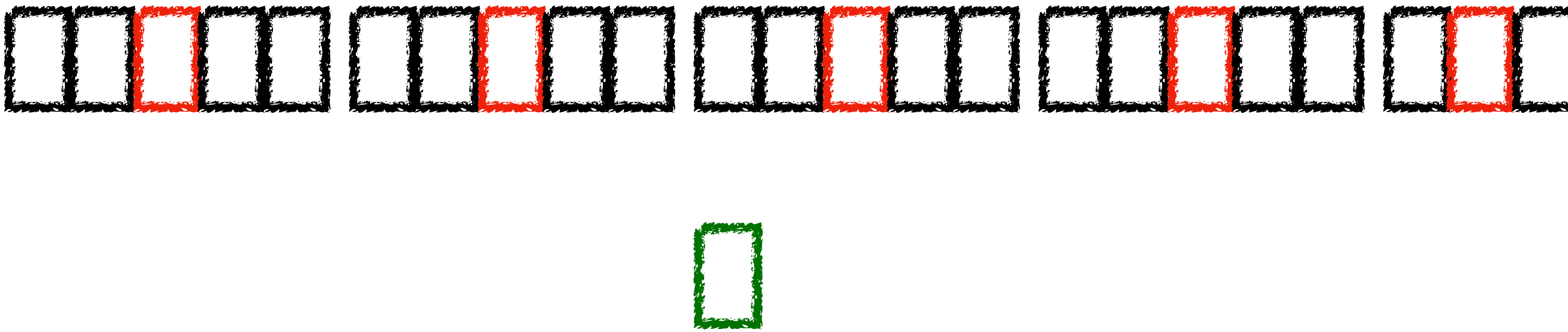
At least (…) subarrays have “baby medians” ≥ x.

Bounding the size of the subarrays

x is a median of medians.

At least *half of the* subarrays have “baby medians” $\geq x$.

Median of medians



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.

Median of medians

Bounding the size of the subarrays

x is a median of medians.

At least *half of the* subarrays have “baby medians” $\geq x$.

Each one of these groups has at least 3 elements $> x$.

Because $x \leq$ 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.

Median of medians

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 with “baby medians” > x

groups

groups who could be exceptions

This means that 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 $7n/10 + 6$

Back to the recurrence:

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

Solving the recurrence

Let's guess that $T(n) \leq cn$, for some constant c.

We get that

$T(n) \leq c(n/5) + c(7n/10 + 6) + bn$

$= 9cn/10 + 6c + bn$

$= cn + (-cn/10 + 6c + bn)$

This is at most cn whenever $-cn/10 + 6c + bn \leq 0$, or equivalently, when $c \geq 10bn/(n - 60)$.

If $n \geq 120$, then $n/(n - 60) \leq 2$ and then, it suffices to have $c \geq 20b$.

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 $a = \max\{ T(n) / n , n \leq 120\}$ and let $c = \max\{a, 20b\}$.

We will prove the statement by induction.

Base case: For every $n \leq 120$, $T(n) \leq \max\{ T(n) / n , n \leq 120\} n = an \leq \max\{a, 20b\}n = cn$

Inductive Step: Suppose that it holds for all n up to $k = 120$. Then for $n = k + 1$, we have $T(n) \leq cn + (-cn/10 + 6c + bn)$

This follows from the previous slide and the fact that $n > 120$ and $c \geq 20b$.