ساختمان داده و الگوریتم ها (CE203)

جلسه نهم: مرتب سازی سریع

سجاد شیرعلی شهرضا

پاییز 1400

شنبه، 1 آبان 1400

اطلاع رسانی

- بخش مرتبط کتاب برای این جلسه: 5

مرتب سازی سریع

یک نمونه واقعی و کاربردی از الگوریتم های تصادفی

QUICKSORT OVERVIEW

EXPECTED RUNNING TIME

$O(n \log n)$

WORST-CASE RUNNING TIME

$O(n^2)$

QUICKSORT OVERVIEW

EXPECTED RUNNING TIME

$O(n \log n)$

WORST-CASE RUNNING TIME

$O(n^2)$

In practice, it works great! It’s competitive with MergeSort (& often better in some contexts!), and it runs *in place* (no need for lots of additional memory)

QUICKSORT: THE IDEA

Let's use DIVIDE-and-CONQUER again!

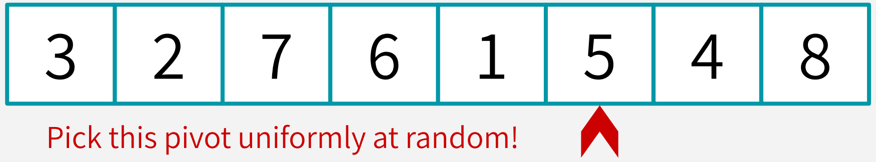
Select a pivot *at random*

Partition around it

Recursively sort L and R!

QUICKSORT: THE IDEA

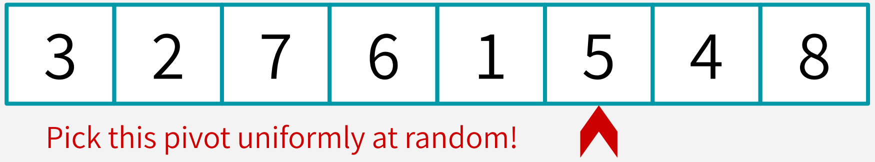
Select a pivot



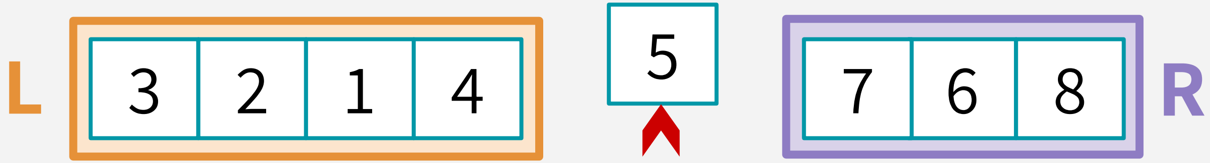
QUICKSORT: THE IDEA

Select a pivot

Partition around it



Pick this pivot uniformly at random!



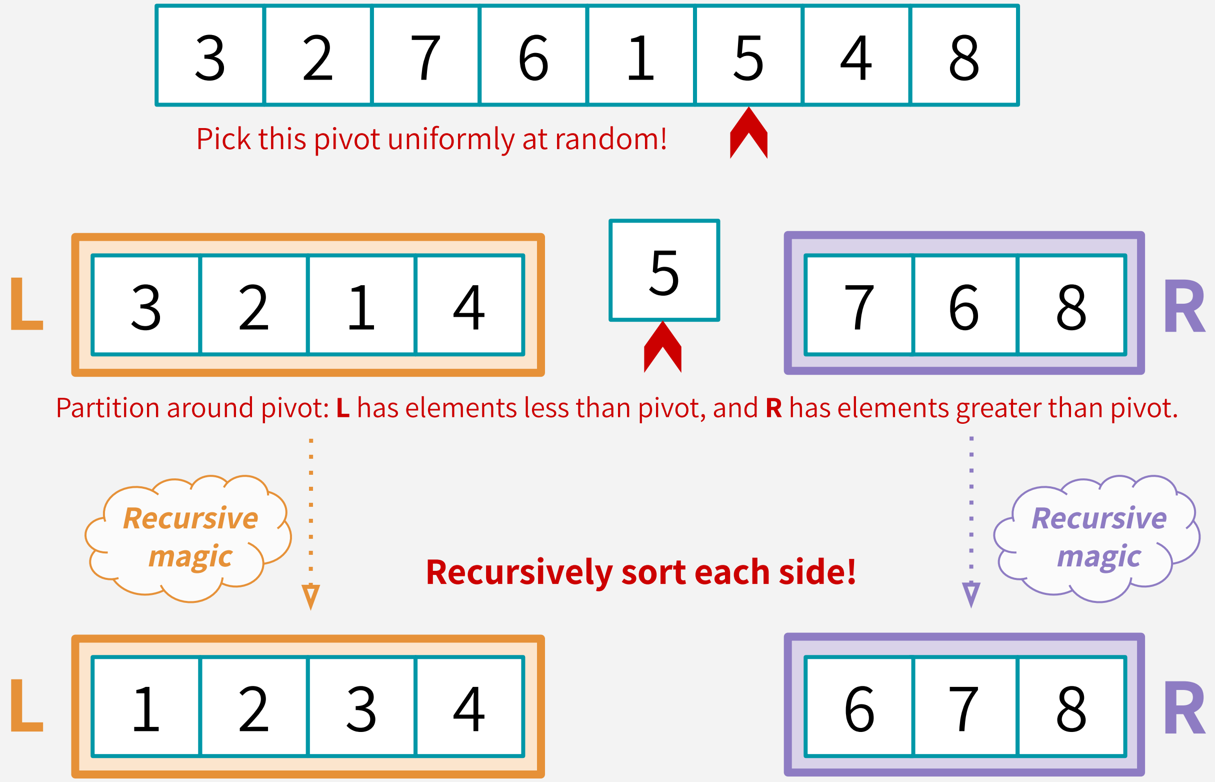
Partition around pivot: **L** has elements less than pivot, and **R** has elements greater than pivot.

QUICKSORT: THE IDEA

Select a pivot

Partition around it

Recurse!



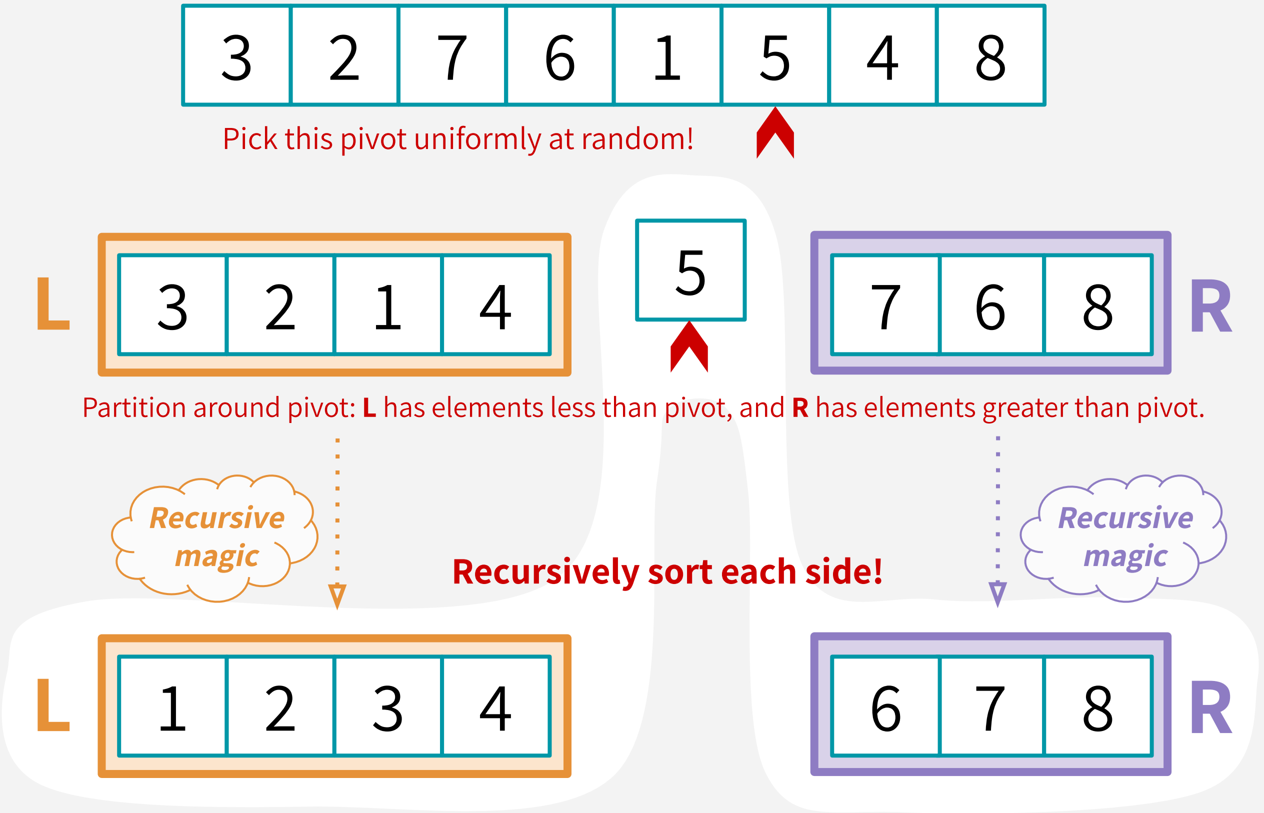
3 2 7 6 1 5 4 8
Pick this pivot uniformly at random!
L
3 2 1 4
5
7 6 8
R
Partition around pivot: L has elements less than pivot, and R has elements greater than pivot.
Recursive magic
Recursively sort each side!
Recursive magic
L
1 2 3 4
6 7 8
R

QUICKSORT: THE IDEA

Select a pivot

Partition around it

Recurse!


3 2 7 6 1 5 4 8
Pick this pivot uniformly at random!
L
3 2 1 4
5
7 6 8
R
Partition around pivot: L has elements less than pivot, and R has elements greater than pivot.
Recursive magic
Recursively sort each side!
Recursive magic
L
1 2 3 4
6 7 8
R

QUICKSORT: PSEUDO-PSEUDOCODE

```
QUICKSORT(A):
    if len(A) <= 1:
        return
    pivot = random.choice(A)
    PARTITION A into:
        L (less than pivot) and
        R (greater than pivot)
    Replace A with [L, pivot, R]
    QUICKSORT(L)
    QUICKSORT(R)
```

RECURRENCE RELATION

```
QUICKSORT(A):
    if len(A) <= 1:
        return
    pivot = random.choice(A)
    PARTITION A into:
        L (less than pivot) and
        R (greater than pivot)
    Replace A with [L, pivot, R]
    QUICKSORT(L)
    QUICKSORT(R)
```

Recurrence Relation for QUICKSORT

$$T(n) = T(|L|) + T(|R|) + O(n)$$
$$T(0) = T(1) = O(1)$$

IDEAL RUNTIME?

```
QUICKSORT(A):
    if len(A) <= 1:
        return
    pivot = random.choice(A)
    PARTITION A into:
        L (less than pivot) and
        R (greater than pivot)
    Replace A with [L, pivot, R]
    QUICKSORT(L)
    QUICKSORT(R)
```

Recurrence Relation for QUICKSORT

$$T(n) = T(|L|) + T(|R|) + O(n)$$
$$T(0) = T(1) = O(1)$$

In an ideal world, the pivot would split the array exactly in half, and we'd get:

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

IDEAL RUNTIME?

```
QUICKSORT(A):
    if len(A) <= 1:
        return
    pivot = random
    PARTITION A ir
        L (less th
        R (greater
    Replace A with [L, pivot, R]
    QUICKSORT(L)
    QUICKSORT(R)
```

Recurrence Relation for QUICKSORT

$) + T(|R|) + O(n)$

$T(1) = O(1)$

In an ideal world:

$$T(n) = 2 \cdot T(n/2) + O(n)$$

$$T(n) = \mathbf{O(n \log n)}$$

, the pivot would split the array exactly in half, and we'd get:

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

WORST-CASE RUNTIME

```
QUICKSORT(A):
    if len(A) <= 1:
        return
    pivot = random.choice(A)
    PARTITION A into:
        L (less than pivot) and
        R (greater than pivot)
    Replace A with [L, pivot, R]
    QUICKSORT(L)
    QUICKSORT(R)
```

Recurrence Relation for QUICKSORT

$$T(n) = T(|L|) + T(|R|) + O(n)$$
$$T(0) = T(1) = O(1)$$

WORST-CASE RUNTIME

```
QUICKSORT(A):
    if len(A) <= 1:
        return
    pivot = random.choice(A)
    PARTITION A into:
        L (less than pivot) and
        R (greater than pivot)
    Replace A with [L, pivot, R]
    QUICKSORT(L)
    QUICKSORT(R)
```

Recurrence Relation for QUICKSORT

$$T(n) = T(|L|) + T(|R|) + O(n)$$
$$T(0) = T(1) = O(1)$$

With the unluckiest randomness, the pivot would be either min(A) or max(A):

$$T(n) = T(0) + T(n-1) + O(n)$$

WORST-CASE RUNTIME

```
QUICKSORT(A):
    if len(A) <= 1:
        return
    pivot = ra
    PARTITION 
        L (less
        R (grea
    Replace A w
    QUICKSORT(L)
    QUICKSORT(R)
```

Recurrence Relation for QUICKSORT

T(|R|) + O(n)

) = O(1)

With the worst "randomness"

$$T(n) = T(n\text{-}1) + O(n)$$

$$T(n) = \mathbf{O(n^2)}$$

(recursion tree/table or substitution method!)

domness, the pivot
in(A) or max(A):

$$T(n) = T(0) + T(n-1) + O(n)$$

سوال؟

EXPECTED RUNTIME = O(n log n)

EXPECTED RUNTIME = O(n log n)

AN **INCORRECT** PROOF:

EXPECTED RUNTIME = O(n log n)

AN **INCORRECT** PROOF:

- $E[\,|L|\,] = E[\,|R|\,] = (n - 1)/2$

AN ASIDE: why is E[|L|] = (n-1)/2?

$$E[\,|L|\,] = E[\,|R|\,]$$

(by symmetry)

$$E[\,|L| + |R|\,] = n - 1$$

(because L and R make up everything except the pivot)

$$E[\,|L|\,] + E[\,|R|\,] = n - 1$$

(by linearity of expectation)

$$2 \cdot E[\,|L|\,] = n - 1$$

(plugging the first line)

$$E[\,|L|\,] = (n - 1)/2$$

(Solving for E[|L|])

EXPECTED RUNTIME = O(n log n)

AN **INCORRECT** PROOF:

- E[|**L**|] = E[|**R**|] = (n - 1)/2
- If this occurs, then T(n) = T(|**L**|) + T(|**R**|) + O(n) could be written as T(n) = 2T(n/2) + O(n).

EXPECTED RUNTIME = O(n log n)

AN **INCORRECT** PROOF:

- $E[|L|] = E[|R|] = (n-1)/2$
- If this occurs, then $T(n) = T(|L|) + T(|R|) + O(n)$ could be written as $T(n) = 2T(n/2) + O(n)$.
- Therefore, the expected running time is $O(n \log n)$!

EXPECTED RUNTIME = O(n log n)

AN **INCORRECT** PROOF:

- E[|**L**|] = E[|**R**|] = (n - 1)/2
- If this occurs, then T(n) = T(|**L**|) + T(|**R**|) + O(n) could be written as T(n) = 2T(n/2) + O(n).
- Therefore, the expected running time is O(n log n)!

Why is this wrong?
Well, for starters, we can use the exact same argument to prove something false...

SLOWSORT

```
SLOWSORT(A):
    if len(A) <= 1:
        return                randomly choose either!
    pivot = either max(A) OR min(A)
    PARTITION A into:
        L (less than pivot) and
        R (greater than pivot)
    Replace A with [L, pivot, R]
    SLOWSORT(L)
    SLOWSORT(R)
```

SLOWSORT

```
SLOW SORT(A):
    if len(A) <= 1:
        return
    pivot = either max(A) or min(A)
    PARTITION A into:
        L (less than pivot) and
        R (greater than pivot)
    Replace A with [L, pivot, R]
    SLOW SORT(L)
    SLOW SORT(R)
```

randomly choose either!

Recurrence Relation for SLOW SORT

$$T(n) = T(|L|) + T(|R|) + O(n)$$
$$T(0) = T(1) = O(1)$$

SLOWSORT

```
SLOW SORT(A):
    if len(A) <= 1:
        return                randomly choose either!
    pivot = either max(A) or min(A)
    PARTITION A into:
        L (less than pivot) and
        R (greater than pivot)
    Replace A with [L, pivot, R]
    SLOW SORT(L)
    SLOW SORT(R)
```

Recurrence Relation for SLOW SORT

$$T(n) = T(|L|) + T(|R|) + O(n)$$
$$T(0) = T(1) = O(1)$$

Same recurrence relation!

We also still have:

$E[|L|] = E[|R|] = (n-1)/2$

But now, one of **|L|** or **|R|** is *always* n-1
& the runtime is **$\Theta(n^2)$**, with probability 1

SLOWSORT

```
SLOWSORT(A):
    if len(A)
        returr
    pivot = e
    PARTITION
        L (les
        R (gre
    Replace A
    SLOWSORT(
    SLOWSORT(R)
```

Recurrence Relation for

SORT

T(|R|) + O(n)

) = O(1)

ce relation!

ll have:

] = (n-1)/2

| or |**R**| is *always* n-1

& the runtime is **Θ(n²)**, with probability 1

RED FLAG:

We could use the exact same (incorrect) proof to prove that **SLOWSort** has expected runtime **O(n log n)**, when it actually has expected runtime of **$\Theta(n^2)$**…

EXPECTED RUNTIME = O(n log n)

AN **INCORRECT** PROOF:

- $E[\,|L|\,] = E[\,|R|\,] = (n - 1)/2$
- If this occurs, then $T(n) = T(\,|L|\,) + T(\,|R|\,) + O(n)$ could be written as $T(n) = 2T(n/2) + O(n)$.
- Therefore, the expected running time is $O(n \log n)$!

Why is this wrong?

EXPECTED RUNTIME = O(n log n)

AN

Basically:

E[f(x)**] is *not necessarily* the same as** f(**E[**x**]**)

e.g. $E[X^2]$ is not the same as $(E[X])^2$

We were reasoning about T(**E[**x**]**) instead of **E[**T(x)**]**

Why is this wrong?

EXPECTED RUNTIME = O(n log n)

Instead, to prove that the expected runtime of QuickSort is O(n log n), we're going to count the **number of comparisons** that this algorithm performs, and take the expectation of that!

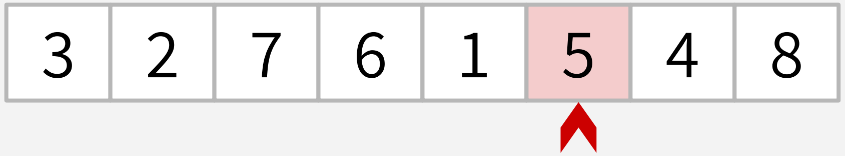
How many times are any two items compared?

سوال؟

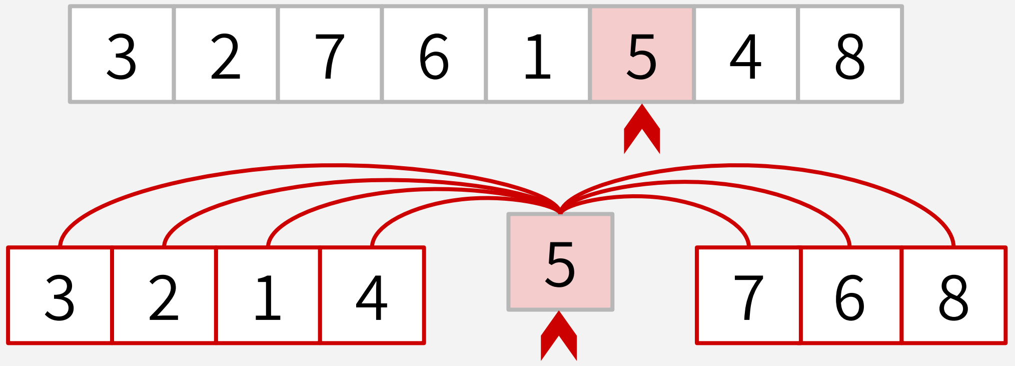
امید ریاضی زمان اجرای مرتب سازی سریع

راه حل درست: تعداد مورد انتظار مقایسه دو عنصر با همدیگر چند بار است؟

HOW MANY COMPARISONS?

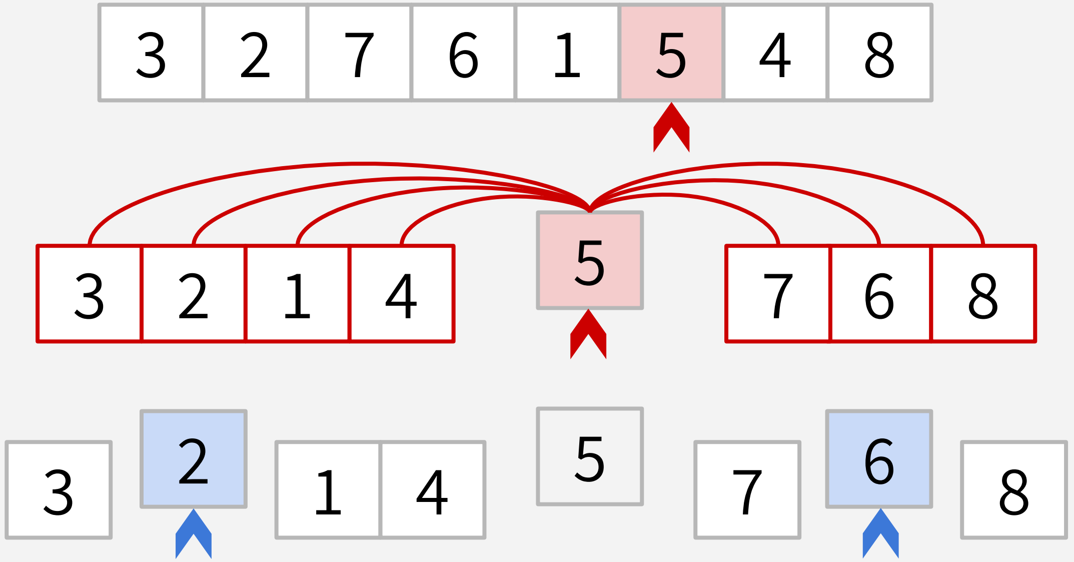


HOW MANY COMPARISONS?



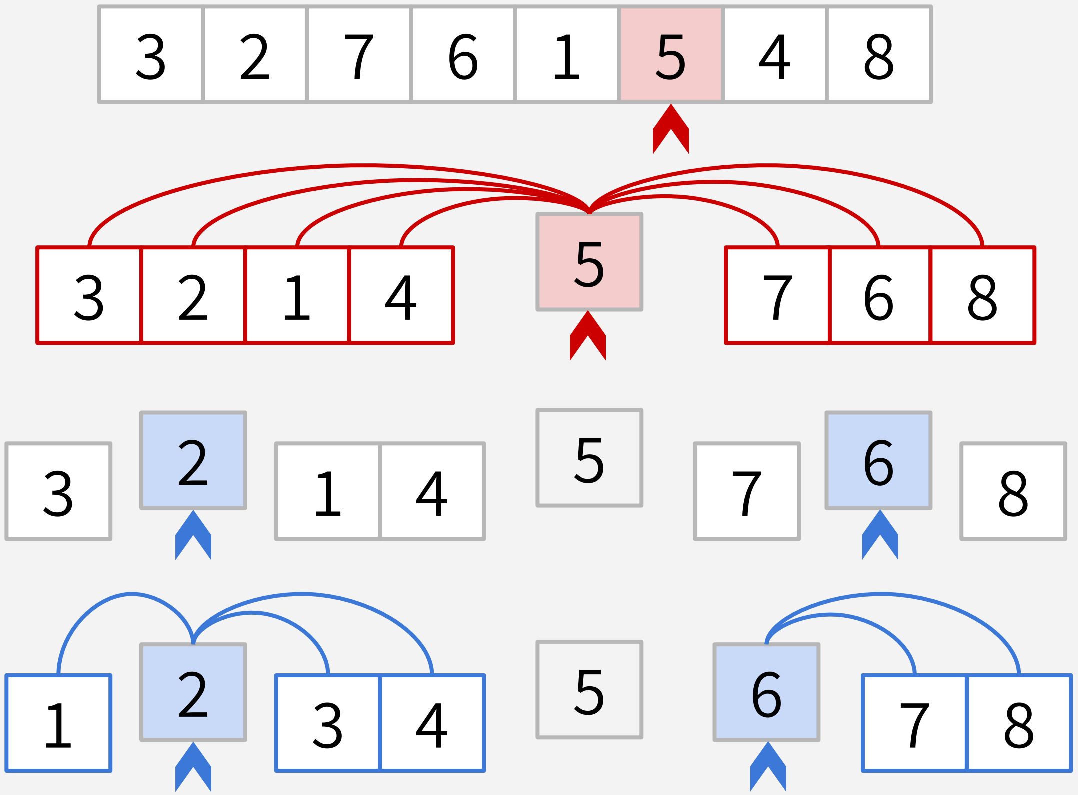
Everything is compared to 5 once in this first step… and then never again with **5**.

HOW MANY COMPARISONS?



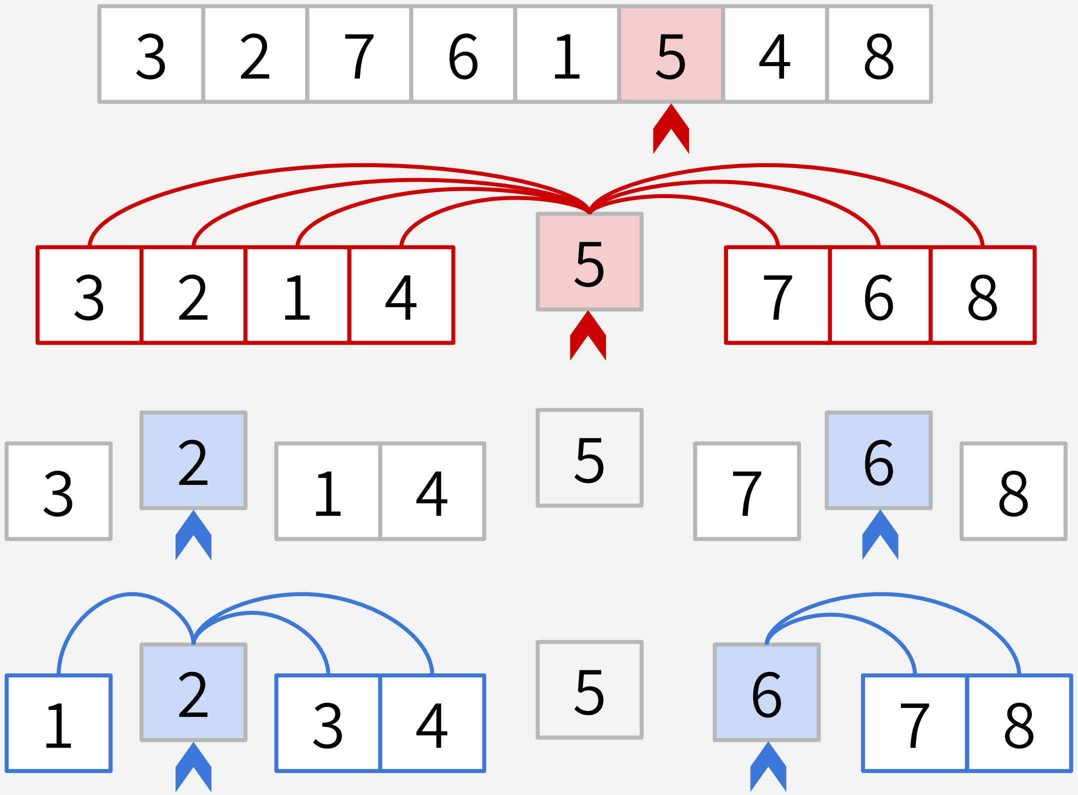
Everything is compared to 5 once in this first step… and then never again with **5**.

HOW MANY COMPARISONS?



Everything is compared to 5 once in this first step… and then never again with **5**.

HOW MANY COMPARISONS?



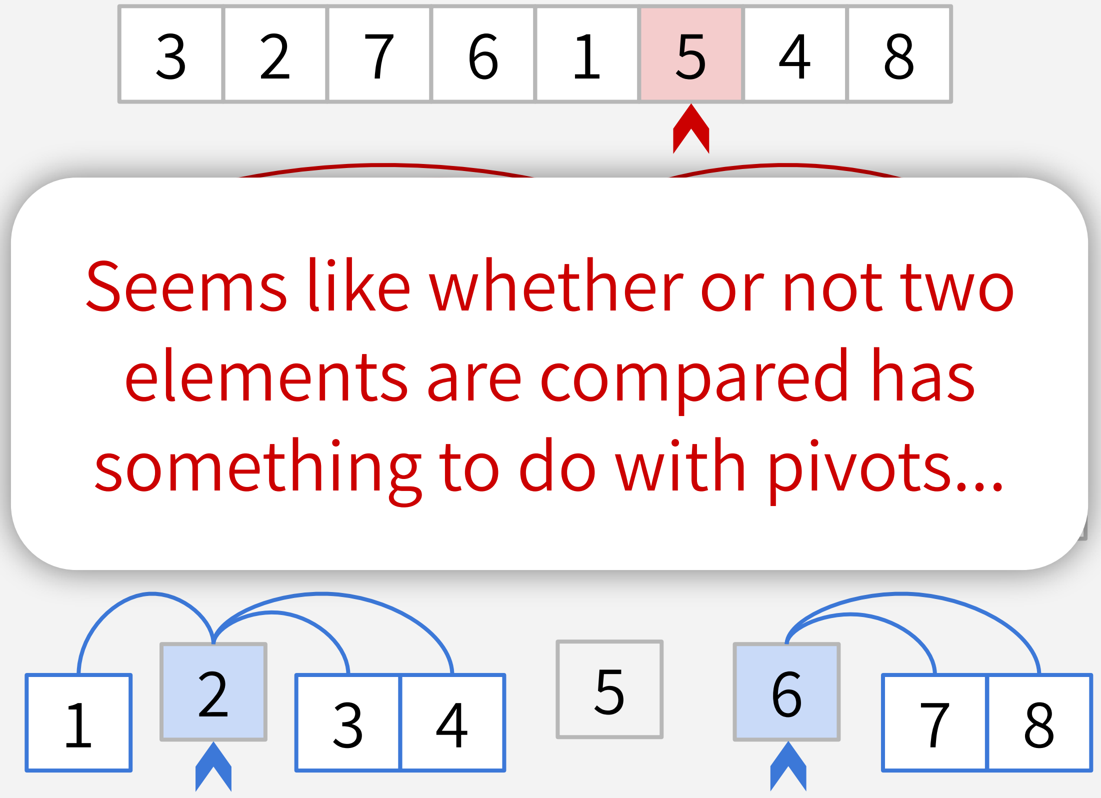
Everything is compared to 5 once in this first step… and then never again with **5**.

Only 1, 3, & 4 are compared to **2**.

And only 7 & 8 are compared with **6**.

No comparisons ever happen between two numbers on opposite sides of 5.

HOW MANY COMPARISONS?



Seems like whether or not two elements are compared has something to do with pivots...

Everything is compared to 5 once in this first step… and then never again with **5**.

Only 1, 3, & 4 are compared to **2**.

And only 7 & 8 are compared with **6**.

No comparisons ever happen between two numbers on opposite sides of 5.

HOW MANY COMPARISONS?

Each pair of elements is compared either **0** or **1** times.

Let $\mathbf{X_{a,b}}$ be a Bernoulli/indicator random variable such that:

$\mathbf{X_{a,b} = 1}$	if **a** and **b** are compared
$\mathbf{X_{a,b} = 0}$	otherwise

HOW MANY COMPARISONS?

Each pair of elements is compared either **0** or **1** times.

Let $\mathbf{X_{a,b}}$ be a Bernoulli/indicator random variable such that:

$\mathbf{X_{a,b} = 1}$	if **a** and **b** are compared
$\mathbf{X_{a,b} = 0}$	otherwise

In our example, $\mathbf{X_{2,5}}$ took on the value **1** since **2** and **5** were compared.
On the other hand, $\mathbf{X_{3,7}}$ took on the value **0** since **3** and **7** are *not* compared.

HOW MANY COMPARISONS?

Each pair of elements is compared either **0** or **1** times.

Let $\mathbf{X_{a,b}}$ be a Bernoulli/indicator random variable such that:

$\mathbf{X_{a,b} = 1}$	if **a** and **b** are compared
$\mathbf{X_{a,b} = 0}$	otherwise

In our example, $\mathbf{X_{2,5}}$ took on the value **1** since **2** and **5** were compared.
On the other hand, $\mathbf{X_{3,7}}$ took on the value **0** since **3** and **7** are *not* compared.

Total number of comparisons =

$$\mathbb{E}\left[\sum_{a=0}^{n-2}\sum_{b=a+1}^{n-1} X_{a,b}\right] = \sum_{a=0}^{n-2}\sum_{b=a+1}^{n-1} \mathbb{E}[X_{a,b}]$$

HOW MANY COMPARISONS?

Each pair of elements is compared either **0** or **1** times.

Let $\mathbf{X_{a,b}}$ be a Bernoulli/indicator random variable such that:

$\mathbf{X_{a,b} = 1}$ if **a** and **b** are compared

$\mathbf{X_{a,b} = 0}$ otherwise

In our example, $\mathbf{X_{2,5}}$ took on the value **1** since **2** and **5** were compared.
On the other hand, $\mathbf{X_{3,7}}$ took on the value **0** since **3** and **7** are *not* compared.

Total number of comparisons =

$$\mathbb{E}\left[\sum_{a=0}^{n-2}\sum_{b=a+1}^{n-1} X_{a,b}\right] \underset{\text{by linearity of expectation!}}{=} \sum_{a=0}^{n-2}\sum_{b=a+1}^{n-1} \mathbb{E}\left[X_{a,b}\right]$$

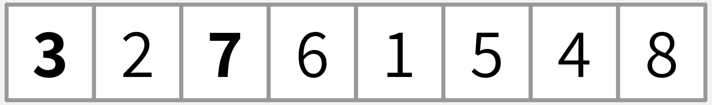
We need to figure out this value!

HOW MANY COMPARISONS?

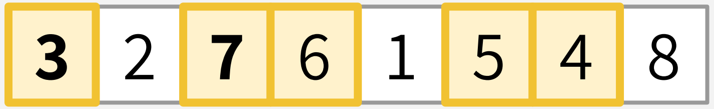
So, what's $\mathbf{E[X_{a,b}]}$?

$$E[X_{a,b}] = 1 \cdot P(X_{a,b} = 1) + 0 \cdot P(X_{a,b} = 0) = P(X_{a,b} = 1)$$

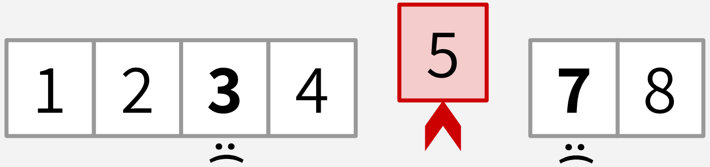
So, what's $\mathbf{P(X_{a,b} = 1)}$? It's the probability that **a** and **b** are compared. Consider this example:



$\mathbf{P(X_{3,7} = 1)}$ is the probability that **3** and **7** are compared.



This is exactly the probability that either 3 or 7 is first picked to be a pivot out of the highlighted entries.



If 4, 5, or 6 get picked as a pivot first, then 3 and 7 would be separated and never see each other again.

HOW MANY COMPARISONS?

So, what's $\mathbf{E[X_{a,b}]}$?

$P(X_{a,b} = 1)$ *aka probability that **a** & **b** are compared*

=

probability that either **a** or **b** are selected as a pivot before elements between **a** and **b**.

=

$$\frac{2}{\text{(\# elements from } \mathbf{a} \text{ to } \mathbf{b}\text{, inclusive)}}$$

So … nple:

3 … ed.

3 … first … es.

1	2	**3**	4

5

7	8

If 4, 5, or 6 get picked as a pivot first, then 3 and 7 would be separated and never see each other again.

HOW MANY COMPARISONS?

So, what's **E[X$_{a,b}$]**?

P(X$_{a,b}$ = 1) *aka probability that **a** & **b** are compared*

=

probability that either **a** or **b** are selected as a pivot before elements between **a** and **b**.

=

$$\frac{2}{\mathbf{b} - \mathbf{a} + 1}$$

So

nple:

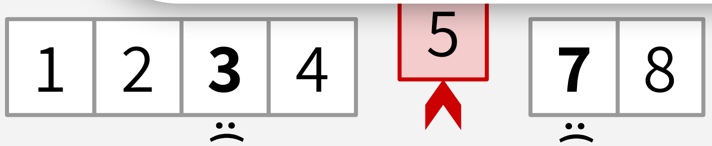
3

ed.

3

first

es.



If 4, 5, or 6 get picked as a pivot first, then 3 and 7 would be separated and never see each other again.

QUICKSORT EXPECTED RUNTIME

Total number of comparisons = $$\sum_{a=0}^{n-2} \sum_{b=a+1}^{n-1} \mathbb{E}[X_{a,b}]$$

QUICKSORT EXPECTED RUNTIME

Total number of comparisons =

$$\sum_{a=0}^{n-2} \sum_{b=a+1}^{n-1} \mathbb{E}[X_{a,b}] = \sum_{a=0}^{n-2} \sum_{b=a+1}^{n-1} \frac{2}{b-a+1}$$

We just computed $E[X_{a,b}] = P(X_{a,b,} = 1)$

QUICKSORT EXPECTED RUNTIME

Total number of comparisons =

$$\sum_{a=0}^{n-2}\sum_{b=a+1}^{n-1}\mathbb{E}[X_{a,b}] = \sum_{a=0}^{n-2}\sum_{b=a+1}^{n-1}\frac{2}{b-a+1}$$

$$= \sum_{a=0}^{n-2}\sum_{c=1}^{n-a-1}\frac{2}{c+1}$$

We just computed $E[X_{a,b}] = P(X_{a,b,} = 1)$

Introduce c = b – a to make notation nicer

QUICKSORT EXPECTED RUNTIME

Total number of comparisons =

$$\sum_{a=0}^{n-2}\sum_{b=a+1}^{n-1}\mathbb{E}[X_{a,b}] = \sum_{a=0}^{n-2}\sum_{b=a+1}^{n-1}\frac{2}{b-a+1}$$

$$= \sum_{a=0}^{n-2}\sum_{c=1}^{n-a-1}\frac{2}{c+1}$$

$$\leq \sum_{a=0}^{n-1}\sum_{c=1}^{n-1}\frac{2}{c+1}$$

We just computed $E[X_{a,b}] = P(X_{a,b,} = 1)$

Introduce c = b – a to make notation nicer

Increase summation limits to make them nicer (hence the ≤)

QUICKSORT EXPECTED RUNTIME

Total number of comparisons =

$$\sum_{a=0}^{n-2}\sum_{b=a+1}^{n-1}\mathbb{E}\left[X_{a,b}\right] = \sum_{a=0}^{n-2}\sum_{b=a+1}^{n-1}\frac{2}{b-a+1}$$

We just computed E[$X_{a,b}$] = P($X_{a,b,}$ = 1)

$$= \sum_{a=0}^{n-2}\sum_{c=1}^{n-a-1}\frac{2}{c+1}$$

Introduce c = b – a to make notation nicer

$$\leq \sum_{a=0}^{n-1}\sum_{c=1}^{n-1}\frac{2}{c+1}$$

Increase summation limits to make them nicer (hence the ≤)

$$= 2n\sum_{c=1}^{n-1}\frac{1}{c+1}$$

Nothing in the summation depends on a, so pull 2 out

QUICKSORT EXPECTED RUNTIME

Total number of comparisons =

$$\sum_{a=0}^{n-2}\sum_{b=a+1}^{n-1}\mathbb{E}[X_{a,b}] = \sum_{a=0}^{n-2}\sum_{b=a+1}^{n-1}\frac{2}{b-a+1}$$

We just computed $E[X_{a,b}] = P(X_{a,b,} = 1)$

$$= \sum_{a=0}^{n-2}\sum_{c=1}^{n-a-1}\frac{2}{c+1}$$

Introduce c = b – a to make notation nicer

$$\leq \sum_{a=0}^{n-1}\sum_{c=1}^{n-1}\frac{2}{c+1}$$

Increase summation limits to make them nicer (hence the ≤)

$$= 2n\sum_{c=1}^{n-1}\frac{1}{c+1}$$

Nothing in the summation depends on a, so pull 2 out

$$\leq 2n\sum_{c=1}^{n-1}\frac{1}{c}$$

decrease each denominator ⟶ we get the harmonic series!

QUICKSORT EXPECTED RUNTIME

Total number of comparisons =

$$\sum_{a=0}^{n-2}\sum_{b=a+1}^{n-1}\mathbb{E}\left[X_{a,b}\right] = \sum_{a=0}^{n-2}\sum_{b=a+1}^{n-1}\frac{2}{b-a+1}$$

$$= \sum_{a=0}^{n-2}\sum_{c=1}^{n-a-1}\frac{2}{c+1}$$

$$\leq \sum_{a=0}^{n-1}\sum_{c=1}^{n-1}\frac{2}{c+1}$$

$$= 2n\sum_{c=1}^{n-1}\frac{1}{c+1}$$

$$\leq 2n\sum_{c=1}^{n-1}\frac{1}{c}$$

$$= \boxed{O(n\log n)}$$

We just computed $E[X_{a,b}] = P(X_{a,b,} = 1)$

Introduce c = b – a to make notation nicer

Increase summation limits to make them nicer (hence the ≤)

Nothing in the summation depends on a, so pull 2 out

decrease each denominator ⟶ we get the harmonic series!

QUICKSORT EXPECTED RUNTIME

Total number of comparisons =

$$\sum_{a=0}^{n-2}\sum_{b=a+1}^{n-1}\mathbb{E}[X_{a,b}] = \sum_{a=0}^{n-2}\sum_{b=a+1}^{n-1}\frac{2}{b-a+1}$$

We just computed $E[X_{a,b}] = P(X_{a,b,} = 1)$

$$= \sum_{a=0}^{n-2}\sum_{c=1}^{n-a-1}\frac{2}{c+1}$$

Introduce c = b – a to make notation nicer

$$\leq \sum_{a=0}^{n-1}\sum_{c=1}^{n-1}\frac{2}{c+1}$$

Increase summation limits to make them nicer (hence the ≤)

$$= 2n\sum_{c=1}^{n-1}\frac{1}{c+1}$$

Nothing in the summation depends on a, so pull 2 out

$$\leq 2n\sum_{c=1}^{n-1}\frac{1}{c}$$

decrease each denominator ⟶ we get the harmonic series!

$$= O(n\log n)$$

If E[# comparisons] = O(n log n), does this mean E[running time] is also O(n log n)?

YES! Intuitively, the runtime is dominated by comparisons.

QUICKSORT

```
QUICKSORT(A):
    if len(A) <= 1:
        return
    pivot = random.choice(A)
    PARTITION A into:
        L (less than pivot) and
        R (greater than pivot)
    Replace A with [L, pivot, R]
    QUICKSORT(L)
    QUICKSORT(R)
```

Worst case runtime:

$O(n^2)$

Expected runtime:

$O(n \log n)$

سوال؟

مرتب سازی سریع در عمل

چگونگی پیاده سازی (و آیا واقعا کسی از آن استفاده می کند؟)

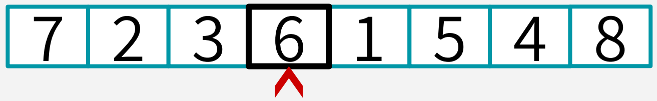
IMPLEMENTING QUICKSORT

In practice, a more clever approach is used to implement PARTITION, so that the entire QuickSort algorithm can be implemented “in-place”

(i.e. via swaps, rather than constructing separate L or R subarrays)

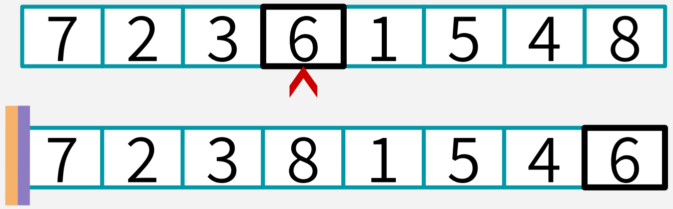
AN EXAMPLE IN-PLACE PARTITION

AN EXAMPLE IN-PLACE PARTITION



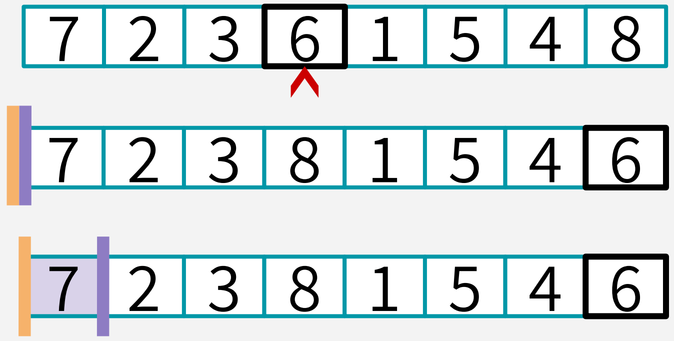
Choose pivot & swap with last element so pivot is at the end.

AN EXAMPLE IN-PLACE PARTITION



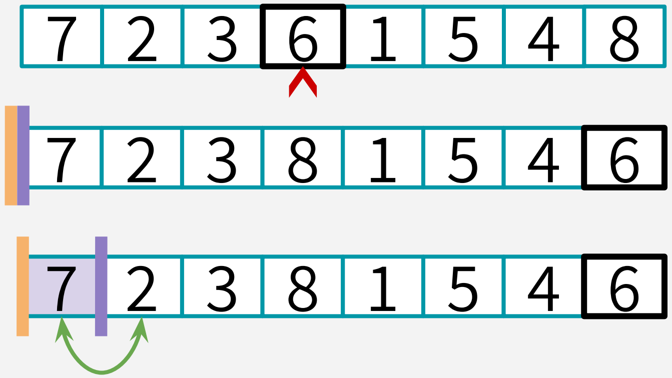
Choose pivot & swap with last element so pivot is at the end. ⇨ Initialize ▮ and ▮

AN EXAMPLE IN-PLACE PARTITION



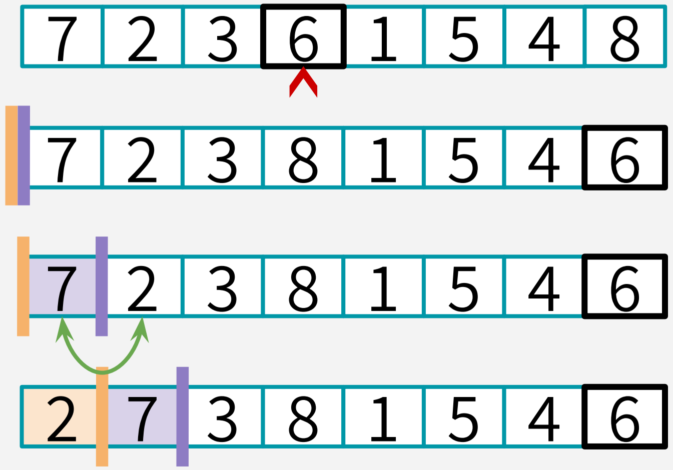
Choose pivot & swap with last element so pivot is at the end. ⇨ Initialize ▮ and ▮ ⇨ Increment ▮ until it sees something smaller than pivot, **swap** the things ahead of the bars & increment both bars

AN EXAMPLE IN-PLACE PARTITION



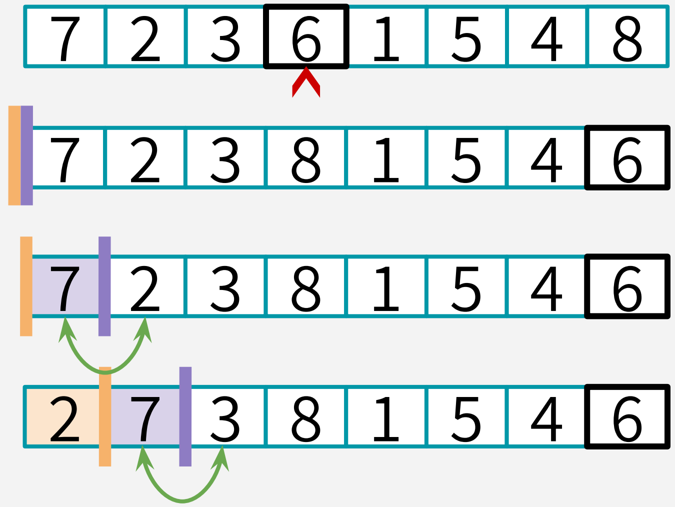
Choose pivot & swap with last element so pivot is at the end. ⇨ Initialize 🟧 and 🟪 ⇨ Increment 🟪 until it sees something smaller than pivot, **swap** the things ahead of the bars & increment both bars

AN EXAMPLE IN-PLACE PARTITION



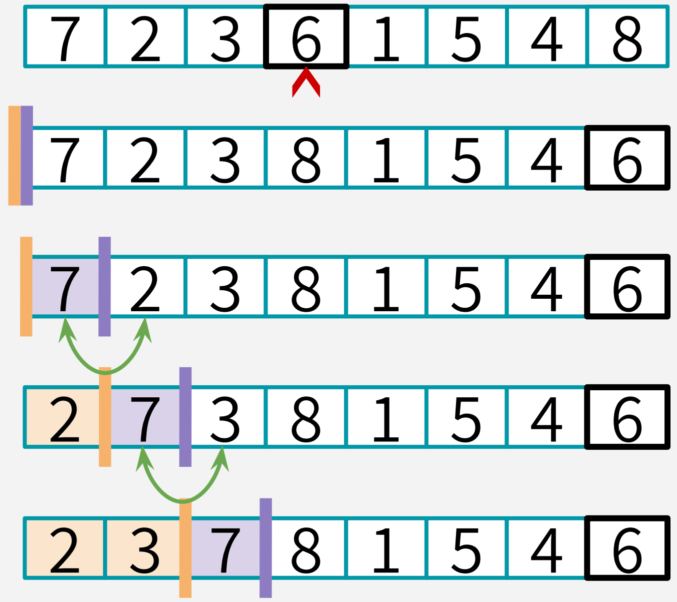
Choose pivot & swap with last element so pivot is at the end. ⇨ Initialize 🟧 and 🟪 ⇨ Increment 🟪 until it sees something smaller than pivot, **swap** the things ahead of the bars & increment both bars

AN EXAMPLE IN-PLACE PARTITION



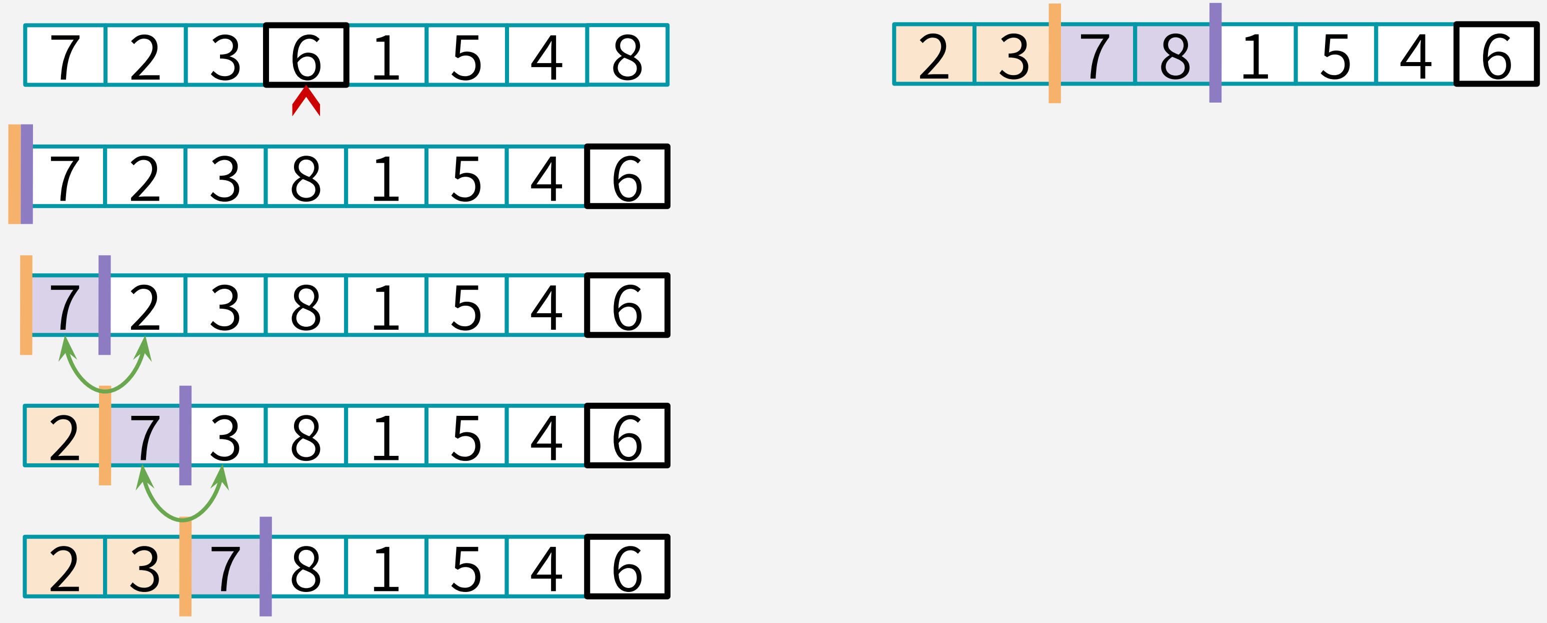
Choose pivot & swap with last element so pivot is at the end. ⇨ Initialize 🟧 and 🟪 ⇨ Increment 🟪 until it sees something smaller than pivot, **swap** the things ahead of the bars & increment both bars ⇨ Repeat until the 🟪 bar reaches the end, then swap the pivot into the right place.

AN EXAMPLE IN-PLACE PARTITION



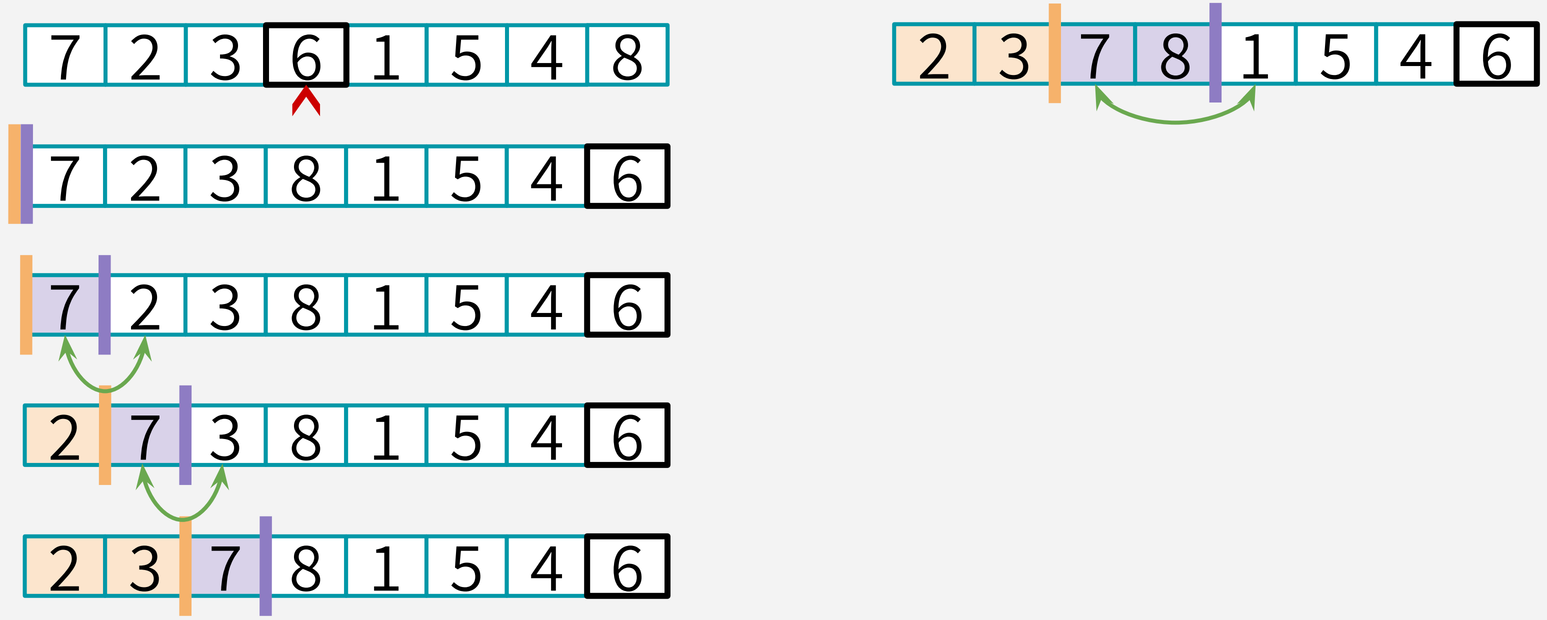
Choose pivot & swap with last element so pivot is at the end. ⇨ Initialize ▮ and ▮ ⇨ Increment ▮ until it sees something smaller than pivot, **swap** the things ahead of the bars & increment both bars ⇨ Repeat until the ▮ bar reaches the end, then swap the pivot into the right place.

AN EXAMPLE IN-PLACE PARTITION



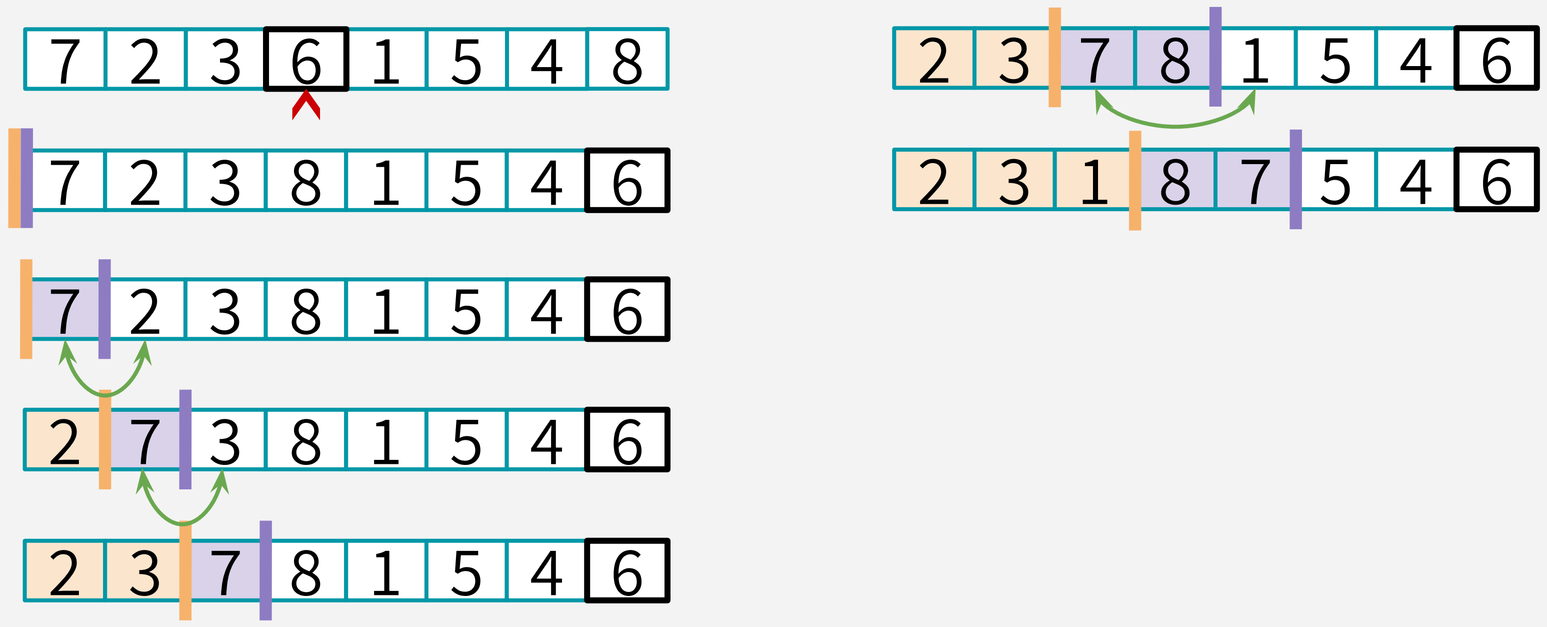
Choose pivot & swap with last element so pivot is at the end. ⇨ Initialize ▮ and ▮ ⇨ Increment ▮ until it sees something smaller than pivot, **swap** the things ahead of the bars & increment both bars ⇨ Repeat until the ▮ bar reaches the end, then swap the pivot into the right place.

AN EXAMPLE IN-PLACE PARTITION



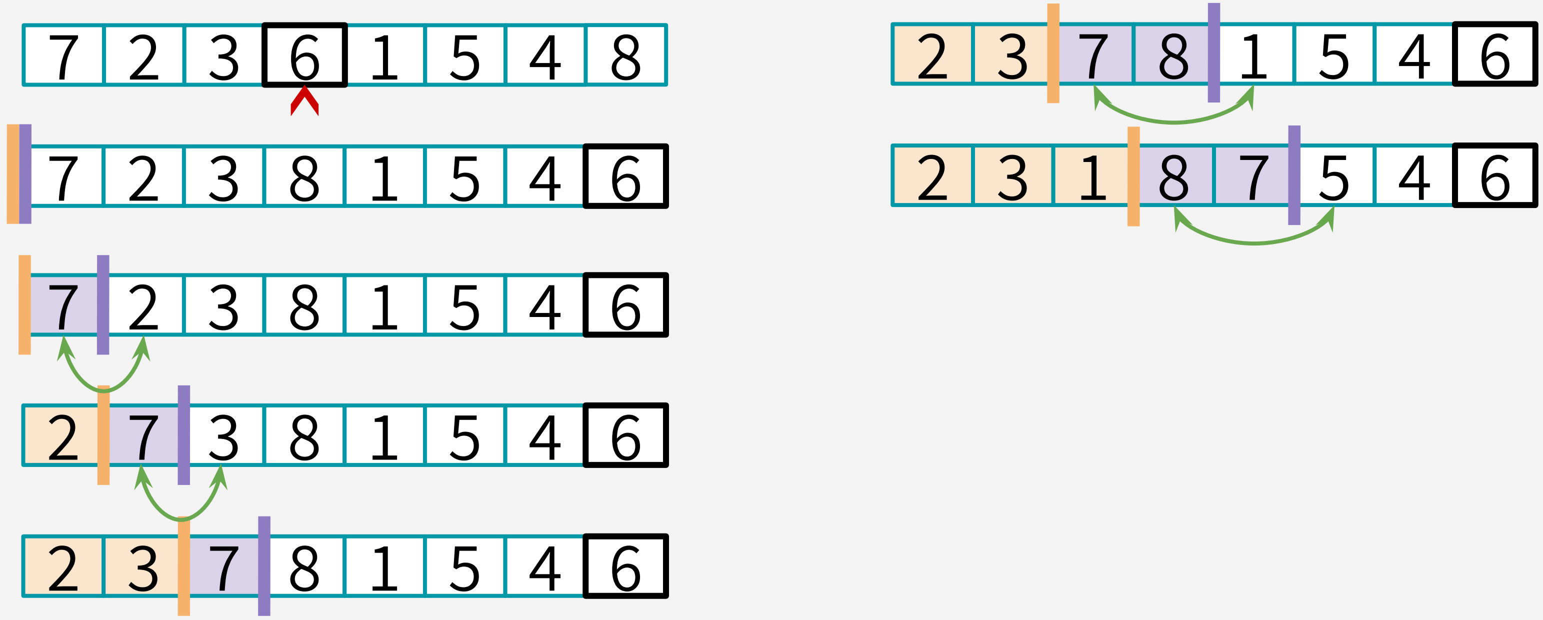
Choose pivot & swap with last element so pivot is at the end. ⇨ Initialize ▮ and ▮ ⇨ Increment ▮ until it sees something smaller than pivot, **swap** the things ahead of the bars & increment both bars ⇨ Repeat until the ▮ bar reaches the end, then swap the pivot into the right place.

AN EXAMPLE IN-PLACE PARTITION



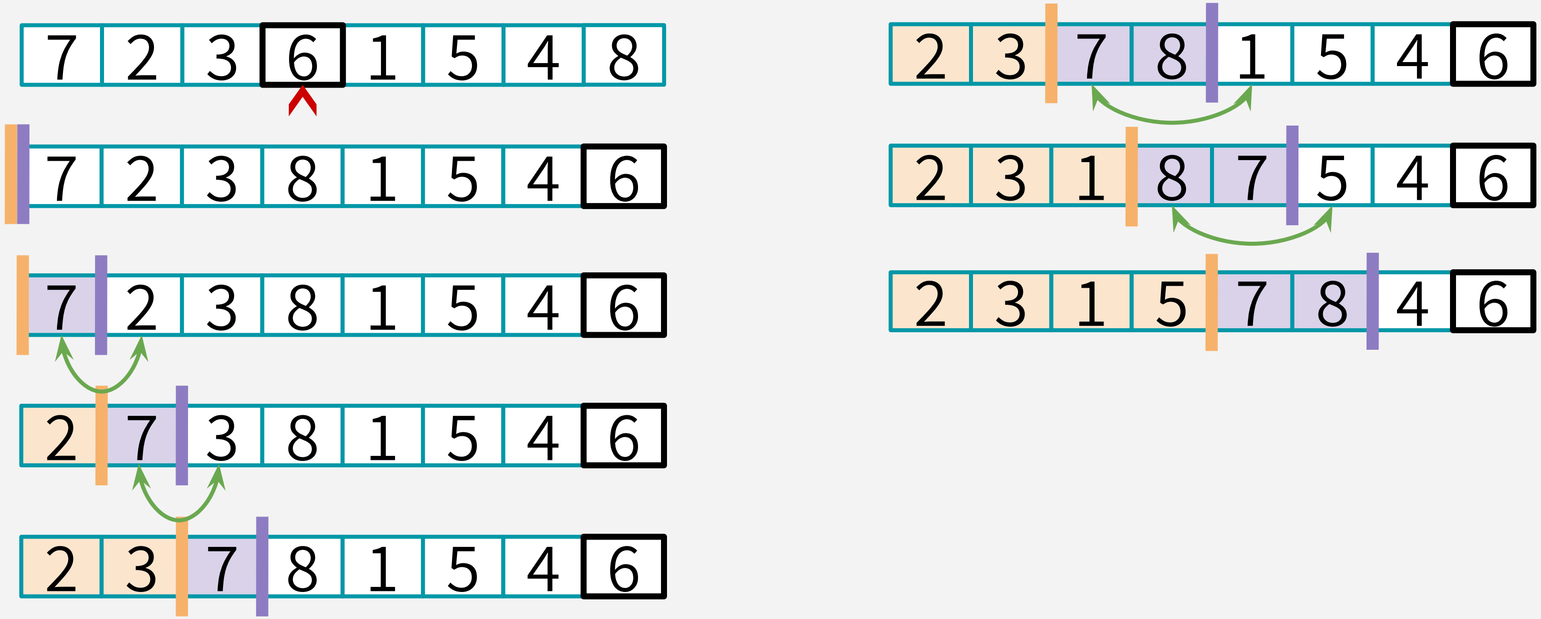
Choose pivot & swap with last element so pivot is at the end. ⇨ Initialize ▮ and ▮ ⇨ Increment ▮ until it sees something smaller than pivot, **swap** the things ahead of the bars & increment both bars ⇨ Repeat until the ▮ bar reaches the end, then swap the pivot into the right place.

AN EXAMPLE IN-PLACE PARTITION



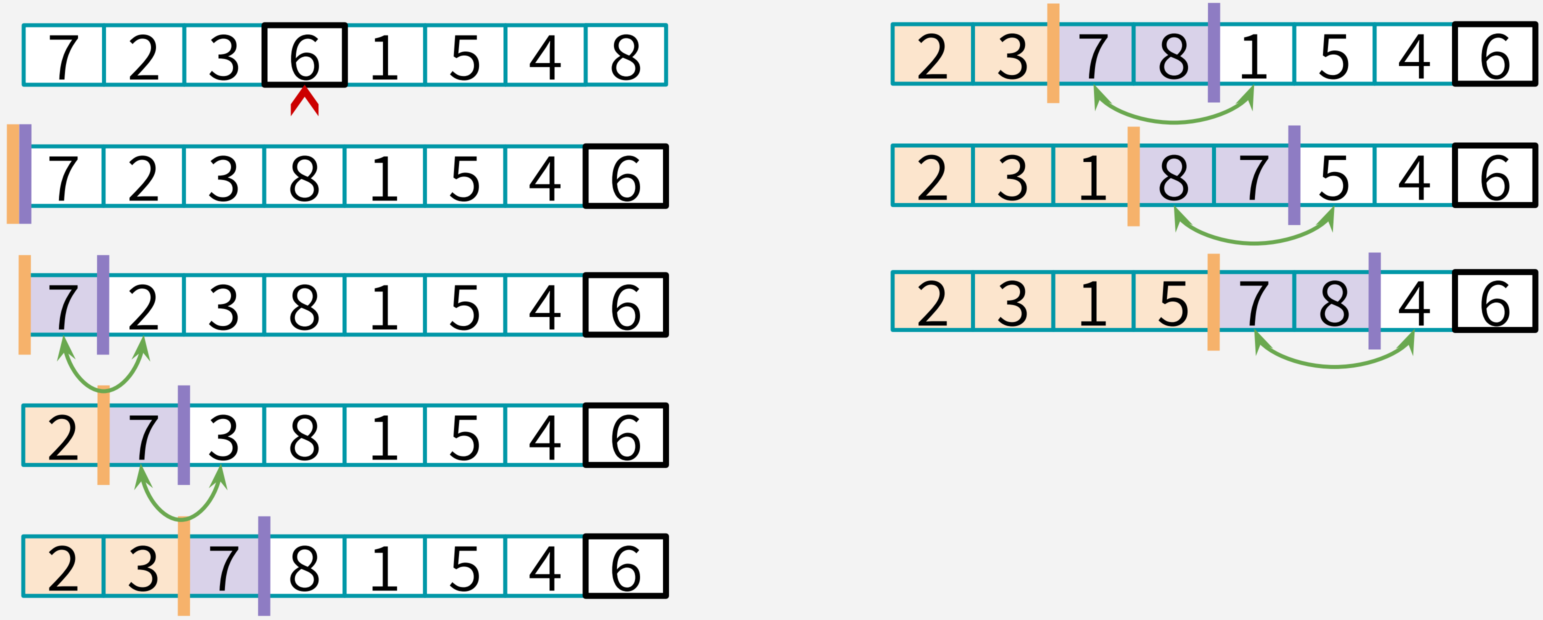
Choose pivot & swap with last element so pivot is at the end. ⇨ Initialize 🟧 and 🟪 ⇨ Increment 🟪 until it sees something smaller than pivot, **swap** the things ahead of the bars & increment both bars ⇨ Repeat until the 🟪 bar reaches the end, then swap the pivot into the right place.

AN EXAMPLE IN-PLACE PARTITION



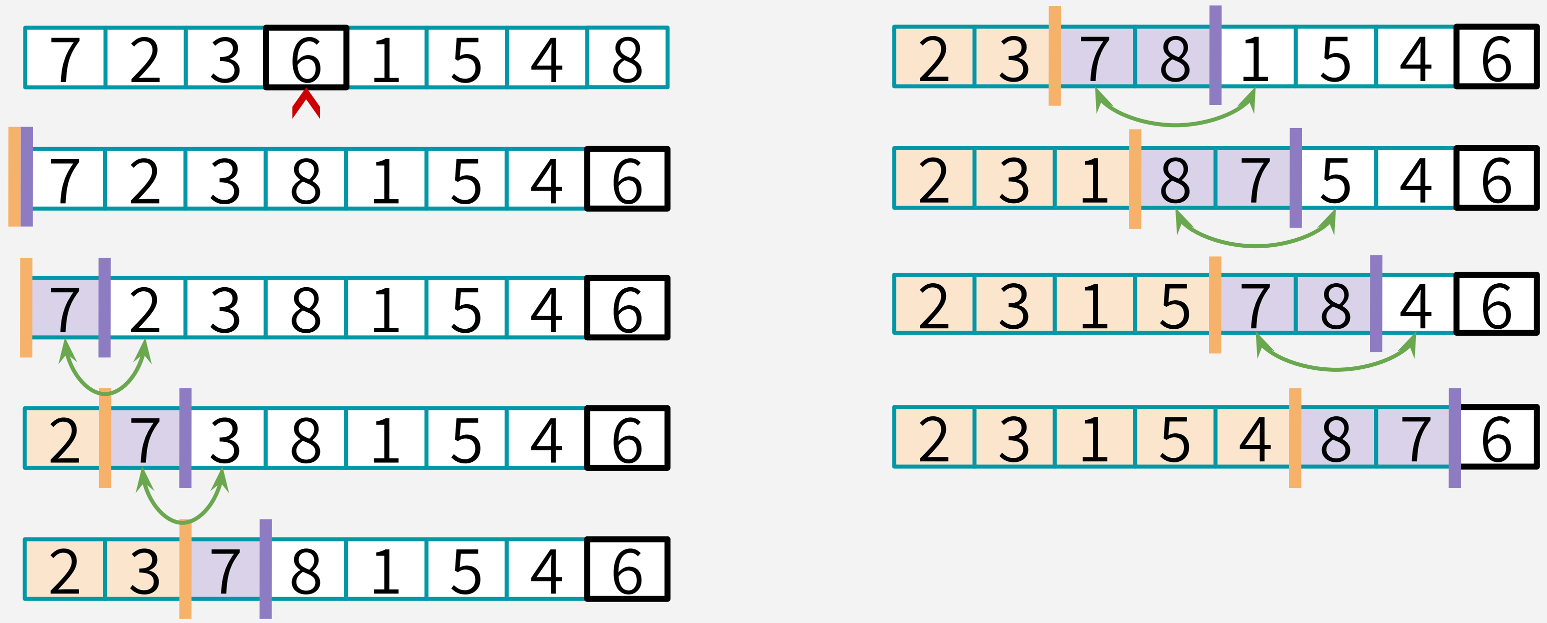
Choose pivot & swap with last element so pivot is at the end. ⇨ Initialize ▮ and ▮ ⇨ Increment ▮ until it sees something smaller than pivot, **swap** the things ahead of the bars & increment both bars ⇨ Repeat until the ▮ bar reaches the end, then swap the pivot into the right place.

AN EXAMPLE IN-PLACE PARTITION



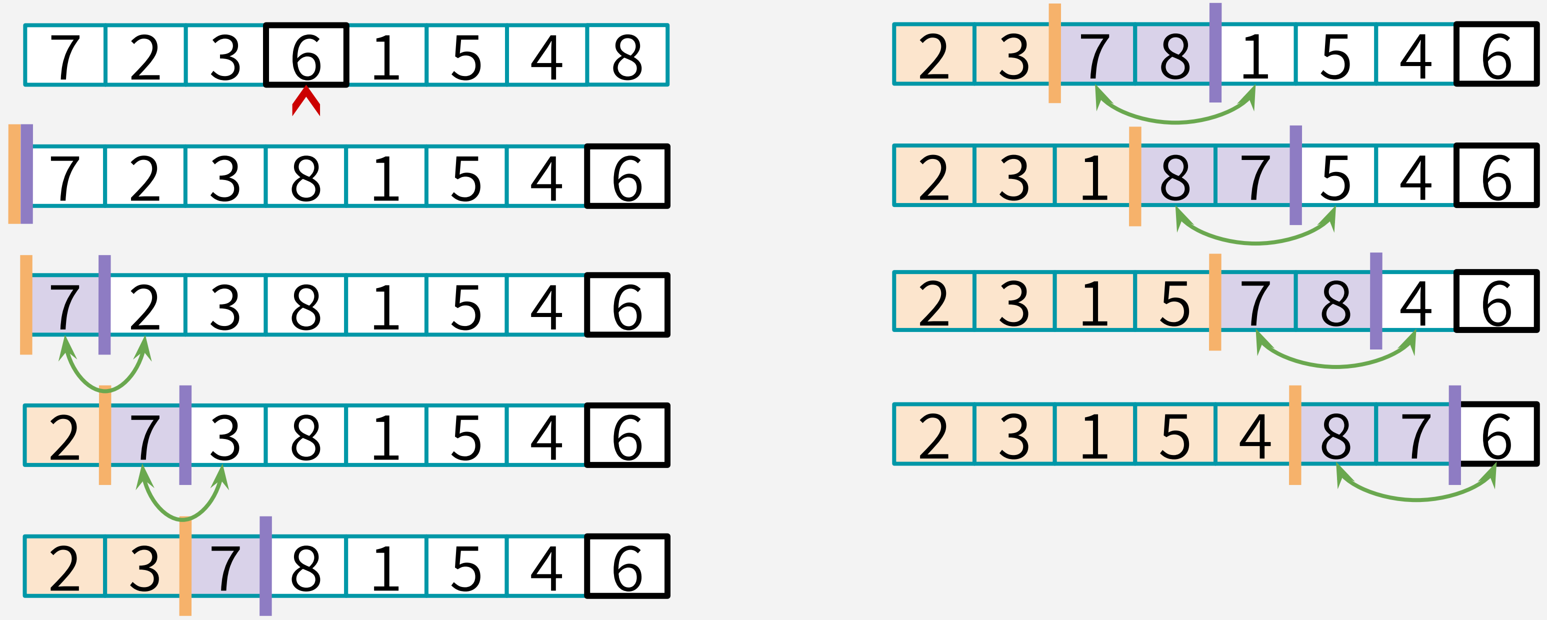
Choose pivot & swap with last element so pivot is at the end. ⇨ Initialize ▮ and ▮ ⇨ Increment ▮ until it sees something smaller than pivot, **swap** the things ahead of the bars & increment both bars ⇨ Repeat until the ▮ bar reaches the end, then swap the pivot into the right place.

AN EXAMPLE IN-PLACE PARTITION



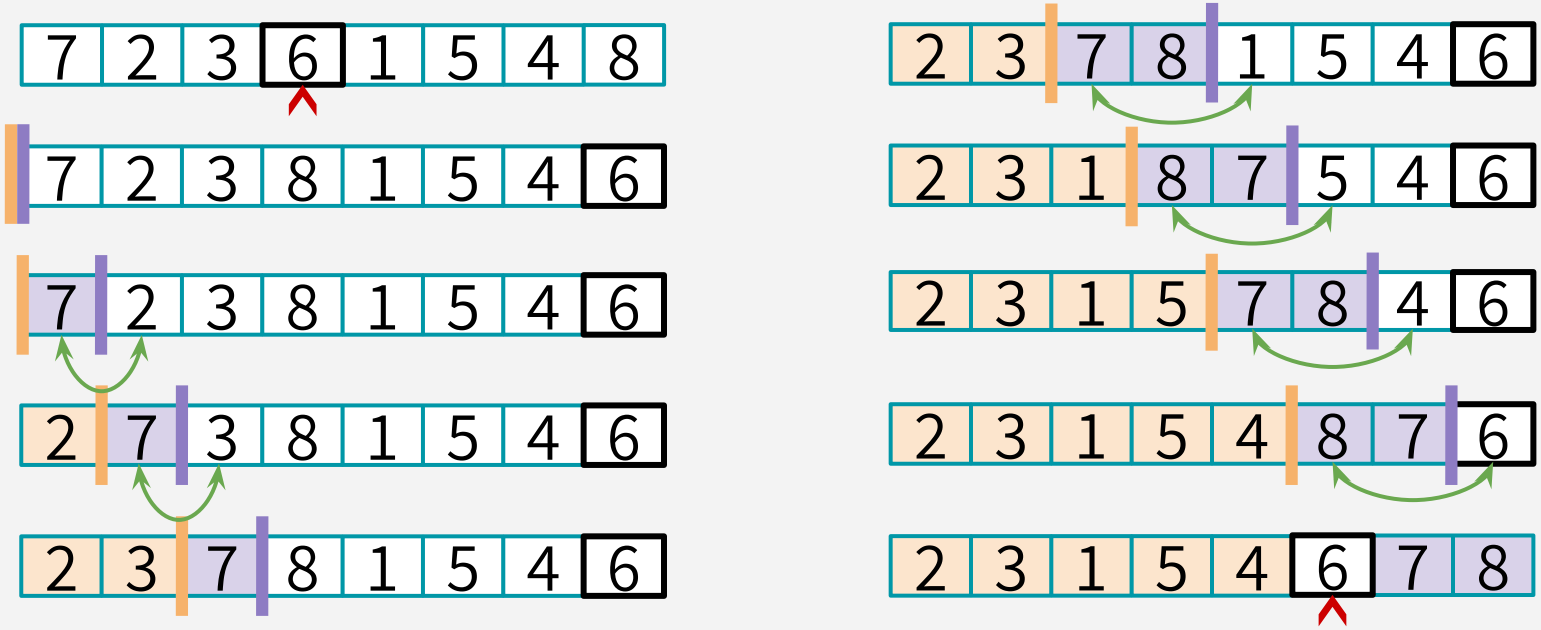
Choose pivot & swap with last element so pivot is at the end. ⇨ Initialize 🟧 and 🟪 ⇨ Increment 🟪 until it sees something smaller than pivot, **swap** the things ahead of the bars & increment both bars ⇨ Repeat until the 🟪 bar reaches the end, then swap the pivot into the right place.

AN EXAMPLE IN-PLACE PARTITION



Choose pivot & swap with last element so pivot is at the end. ⇨ Initialize ▮ and ▮ ⇨ Increment ▮ until it sees something smaller than pivot, **swap** the things ahead of the bars & increment both bars ⇨ Repeat until the ▮ bar reaches the end, then swap the pivot into the right place.

AN EXAMPLE IN-PLACE PARTITION



Choose pivot & swap with last element so pivot is at the end. ⇨ Initialize ▮ and ▮ ⇨ Increment ▮ until it sees something smaller than pivot, **swap** the things ahead of the bars & increment both bars ⇨ Repeat until the ▮ bar reaches the end, then swap the pivot into the right place.

سوال؟

IMPLEMENTING QUICKSORT

There's another in-place partition algorithm called Hoare Partition that's even more efficient as it performs less swaps.

(you're not responsible for knowing it in this class)

QUICKSORT vs. MERGESORT

You do not need to understand any of this stuff

	QuickSort (random pivot)	**MergeSort** (deterministic)
Runtime	**Worst-case: $O(n^2)$** **Expected: $O(n \log n)$**	**Worst-case: $O(n \log n)$**
Used by	Java (primitive types), C (qsort), Unix, gcc…	Java for objects, perl
In-place? **(i.e. with $O(\log n)$ extra memory)**	Yes, pretty easily!	Easy if you sacrifice runtime ($O(n\log n)$ MERGE runtime). Not so easy if you want to keep runtime & stability.
Stable?	No	Yes
Other Pros	Good cache locality if implemented for arrays	Merge step is really efficient with linked lists