ساختمان داده و الگوریتم ها (CE203)

جلسه ششم:
مرتب سازی سریع

سجاد شیرعلی شهرضا

پاییز 1401

شنبه، 16 مهر 1401

Based on Stanford CS161 Slides from Summer 2020 by Karey Shi

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

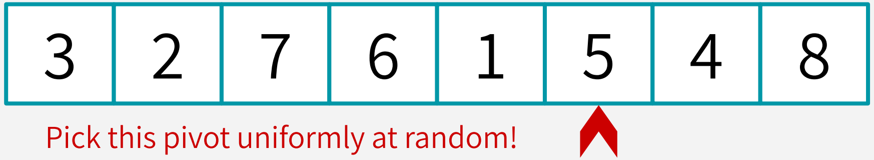
Select a pivot

3	2	7	6	1	5	4	8

Pick this pivot uniformly at random!

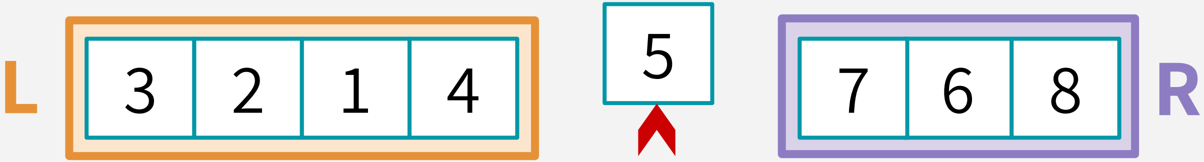
QUICKSORT: THE IDEA

Select a pivot



Pick this pivot uniformly at random!

Partition around it



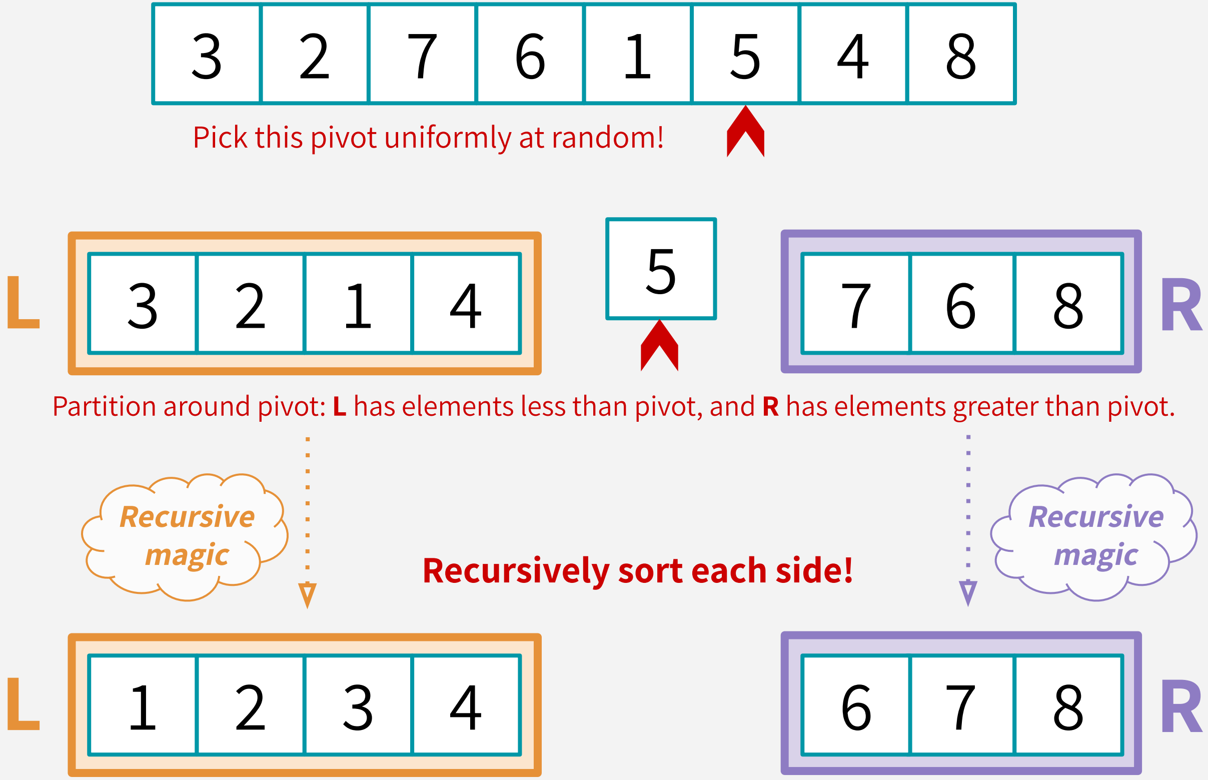
Partition around pivot: **L** has elements less than pivot, and **R** has elements greater than pivot.

QUICKSORT: THE IDEA

Select a pivot

Partition around it

Recurse!

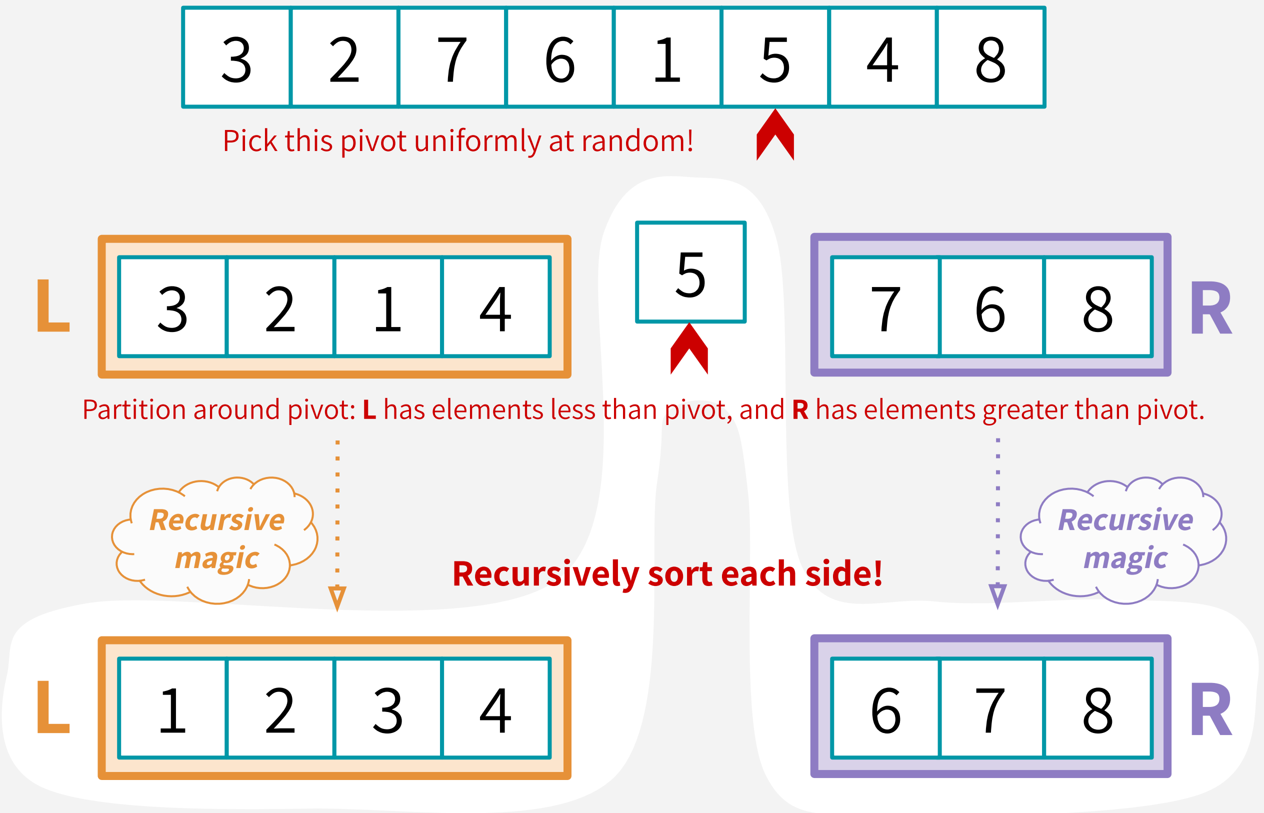


QUICKSORT: THE IDEA

Select a pivot

Partition around it

Recurse!



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

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

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

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

$T(1) = O(1)$

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

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

In an ideal world:

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

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

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

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

$) = O(1)$

domness, the pivot

nin(A) or max(A):

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

With the worst "randomness"

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

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

(recursion tree/table or substitution method!)

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?

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
    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)
```

randomly choose either!

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)
```

Recurrence Relation for SLOW SORT

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

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)
```

Recurrence Relation for SLOWSORT

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

Same recurrence relation!

We also still have:

$$\mathbf{E[|L|] = E[|R|] = (n\text{-}1)/2}$$

SLOWSORT

```
SLOWSORT(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]
    SLOWSORT(L)
    SLOWSORT(R)
```

randomly choose either!

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:

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

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)
        retur
    pivot = e
    PARTITION
        L (les
        R (gre
    Replace A
    SLOWSORT(
    SLOWSORT(R)
```

Recurrence Relation for
SORT

T(|R|) + O(n)
) = O(1)

nce relation!
ll have:
] = (n-1)/2

or **|R|** is *always* n-1
& the runtime is $\Theta(\mathbf{n^2})$, 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 $\Theta(\mathbf{n^2})$...

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?

EXPECTED RUNTIME = O(n log n)

Inste ort is

O(n **sons**

that t that!

Will see it in

Algorithm Design Course!

How red?

سوال؟

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:

$O(n \log n)$

BETTER WORST CASE RUNTIME

- Select a better pivot
  - Ideally, split the array into two equal parts
  - Select the median as pivot

BETTER WORST CASE RUNTIME

- Select a better pivot
  - Ideally, split the array into two equal parts
  - Select the median as pivot
- If the pivot is median, then we will have:
  - $T(n) = 2T(n/2) + O(n) = O(n \log n)$

BETTER WORST CASE RUNTIME

- Select a better pivot
  - Ideally, split the array into two equal parts
  - Select the median as pivot
- If the pivot is median, then we will have:
  - $T(n) = 2T(n/2) + O(n) = O(n \log n)$
- How to select the median in $O(n)$?

BETTER WORST CASE RUNTIME

- Select a better pivot
  - Ideally, split the array into two equal parts
  - Select the median as pivot
- If the pivot is median, then we will have:
  - $T(n) = 2T(n/2) + O(n) = O(n \log n)$
- How to select the median in $O(n)$?
  - Will also see it in Algorithm Design course!

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 (Lomuto) 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)

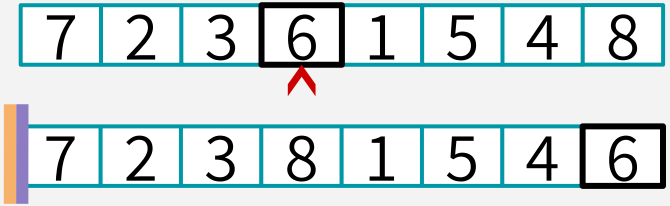
AN EXAMPLE IN-PLACE PARTITION

AN EXAMPLE IN-PLACE PARTITION

7	2	3	**6**	1	5	4	8

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

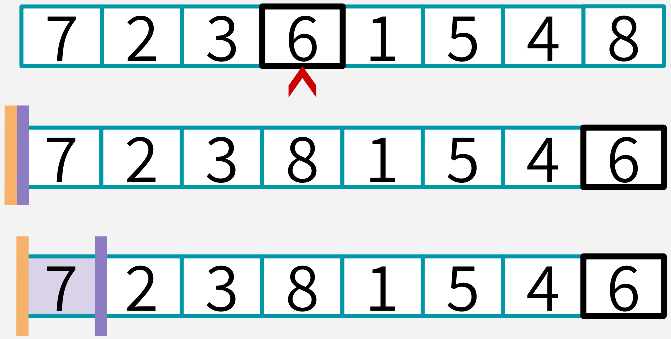
AN EXAMPLE IN-PLACE PARTITION



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

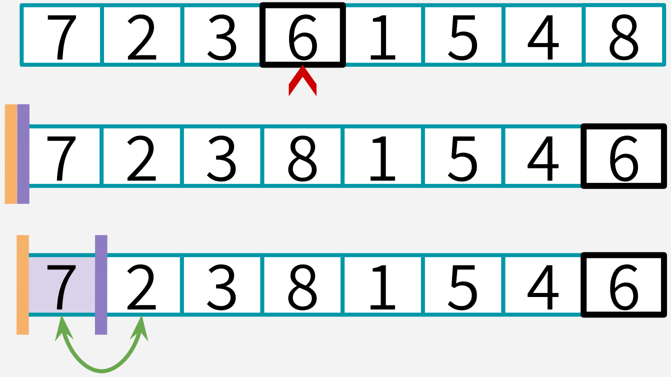
Initialize and

AN EXAMPLE IN-PLACE PARTITION



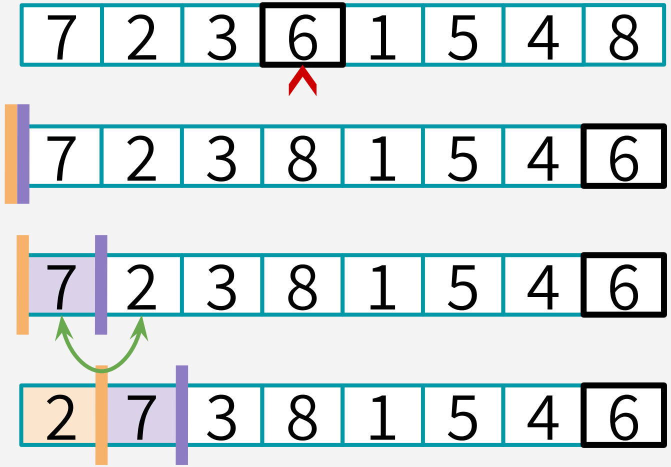
Choose pivot & swap with last element so pivot is at the end. ⇨ Initialize and ⇨ Increment until it sees something smaller than pivot, **swap** the things ahead of the bars & increment both bars

AN EXAMPLE IN-PLACE PARTITION



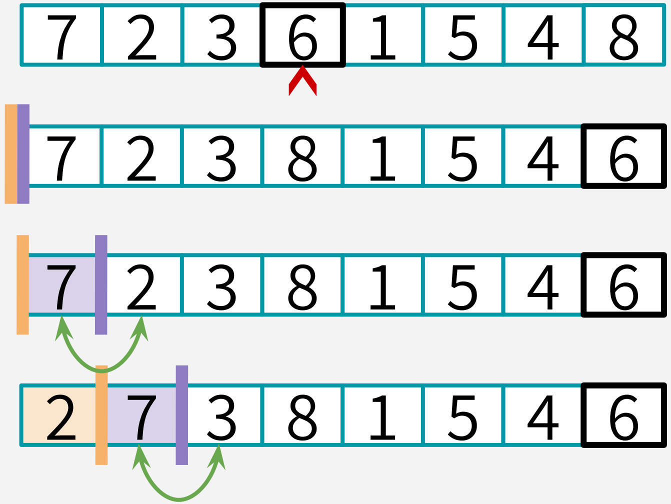
Choose pivot & swap with last element so pivot is at the end. ⇨ Initialize and ⇨ Increment until it sees something smaller than pivot, **swap** the things ahead of the bars & increment both bars

AN EXAMPLE IN-PLACE PARTITION



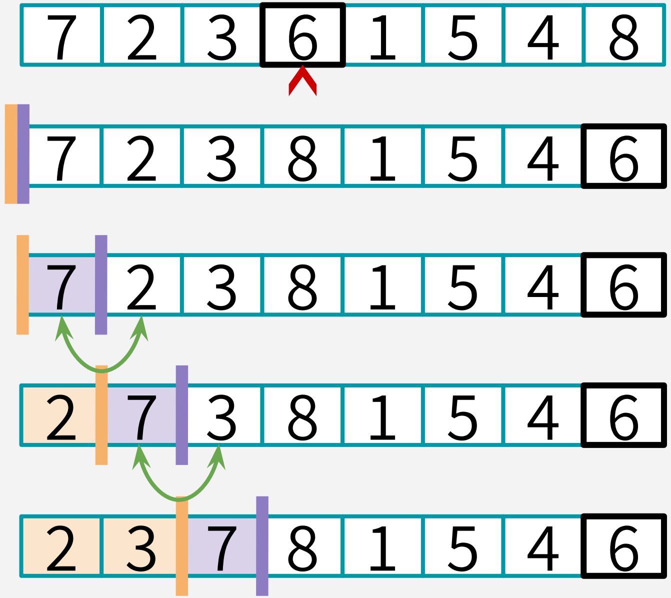
Choose pivot & swap with last element so pivot is at the end. ⇨ Initialize and ⇨ Increment until it sees something smaller than pivot, **swap** the things ahead of the bars & increment both bars

AN EXAMPLE IN-PLACE PARTITION



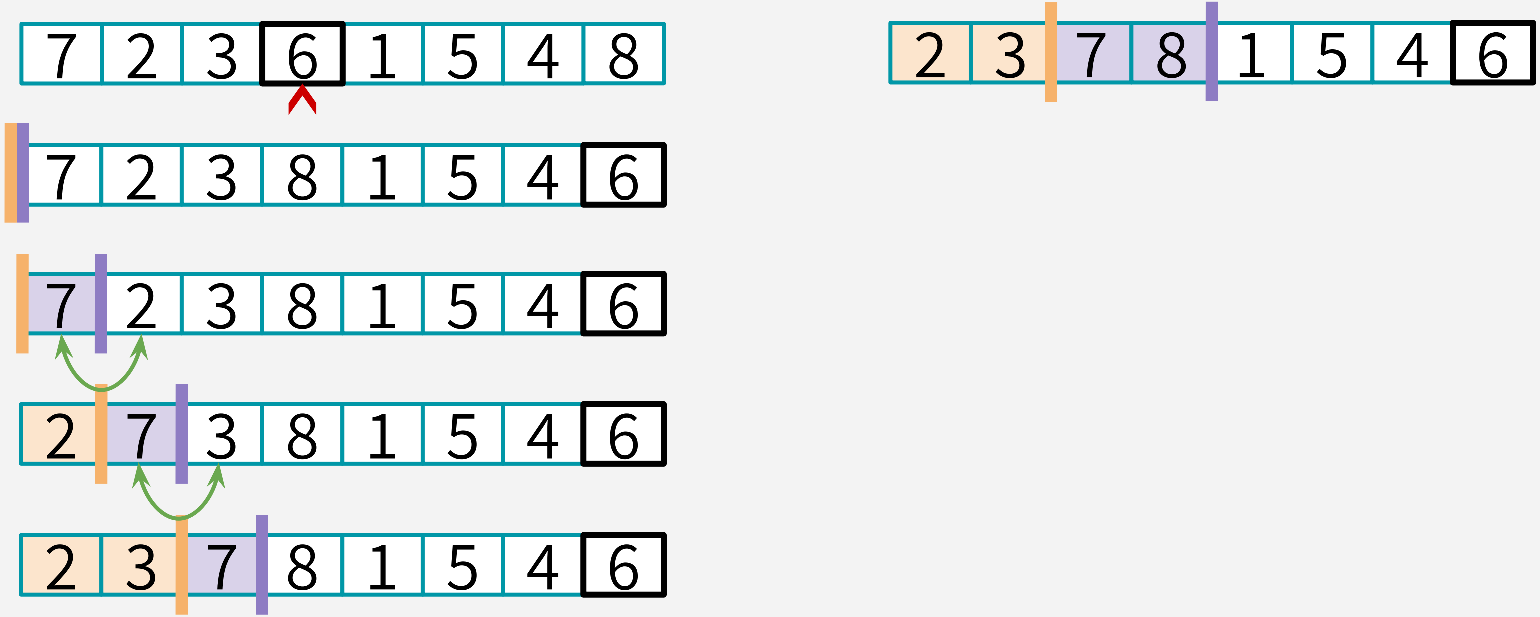
Choose pivot & swap with last element so pivot is at the end. ⇨ Initialize and ⇨ Increment until it sees something smaller than pivot, **swap** the things ahead of the bars & increment both bars ⇨ Repeat until the bar reaches the end, then swap the pivot into the right place.

AN EXAMPLE IN-PLACE PARTITION



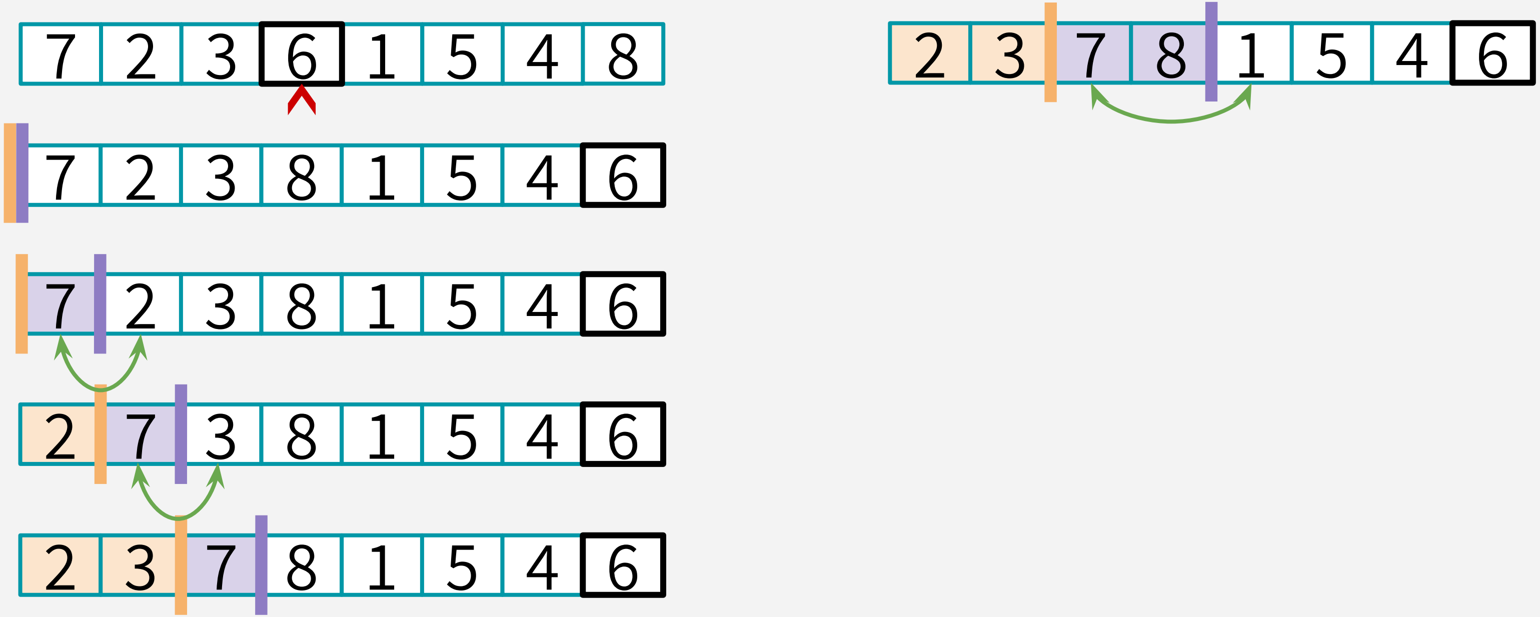
Choose pivot & swap with last element so pivot is at the end. ⇨ Initialize and ⇨ Increment until it sees something smaller than pivot, **swap** the things ahead of the bars & increment both bars ⇨ Repeat until the bar reaches the end, then swap the pivot into the right place.

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



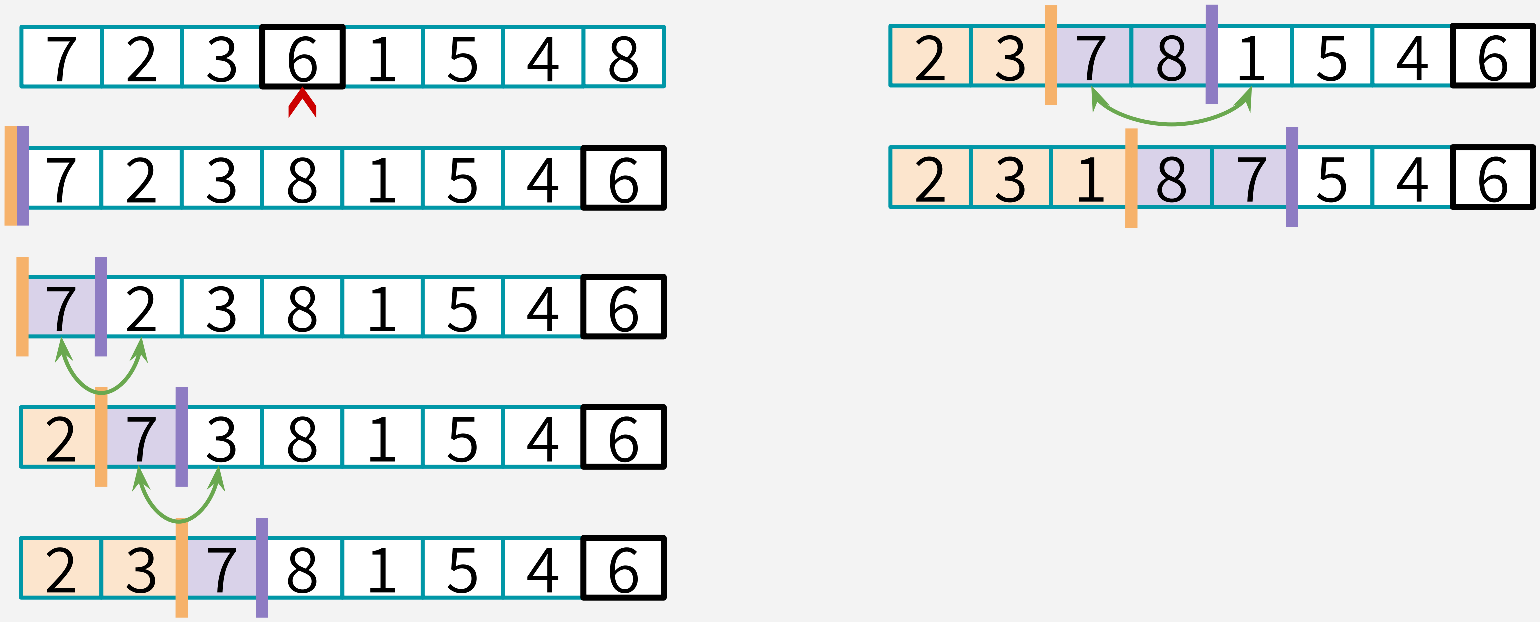
Choose pivot & swap with last element so pivot is at the end.

Initialize and

Increment until it sees something smaller than pivot, **swap** the things ahead of the bars & increment both bars

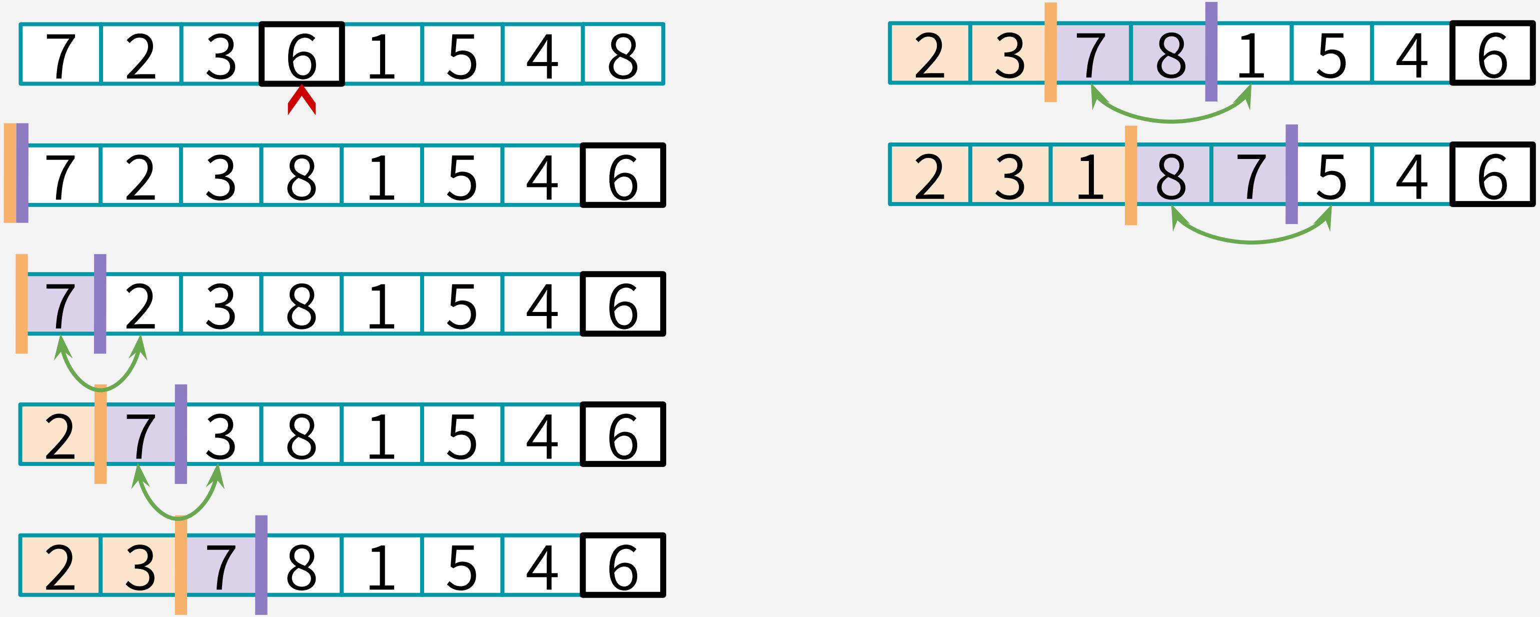
Repeat until the bar reaches the end, then swap the pivot into the right place.

AN EXAMPLE IN-PLACE PARTITION



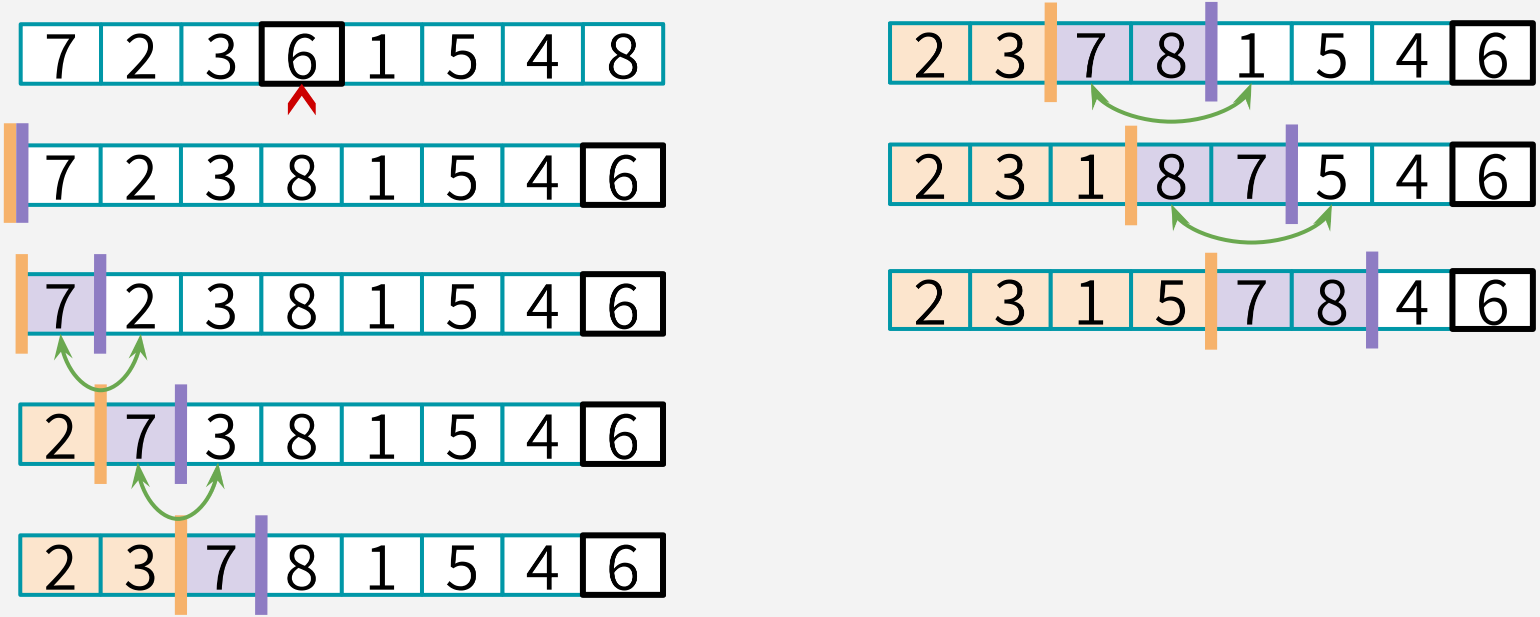
Choose pivot & swap with last element so pivot is at the end. ⇨ Initialize and ⇨ Increment until it sees something smaller than pivot, **swap** the things ahead of the bars & increment both bars ⇨ Repeat until the bar reaches the end, then swap the pivot into the right place.

AN EXAMPLE IN-PLACE PARTITION



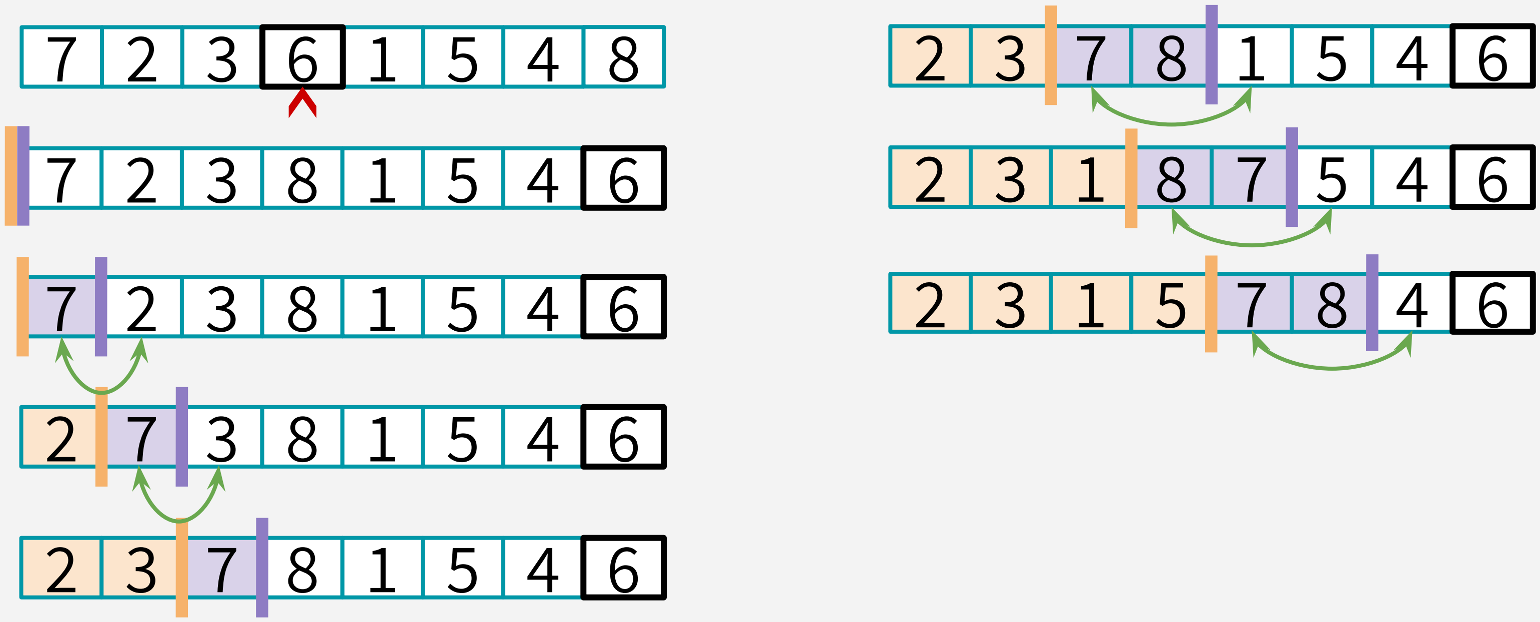
Choose pivot & swap with last element so pivot is at the end. ⇨ Initialize and ⇨ Increment until it sees something smaller than pivot, **swap** the things ahead of the bars & increment both bars ⇨ Repeat until the bar reaches the end, then swap the pivot into the right place.

AN EXAMPLE IN-PLACE PARTITION



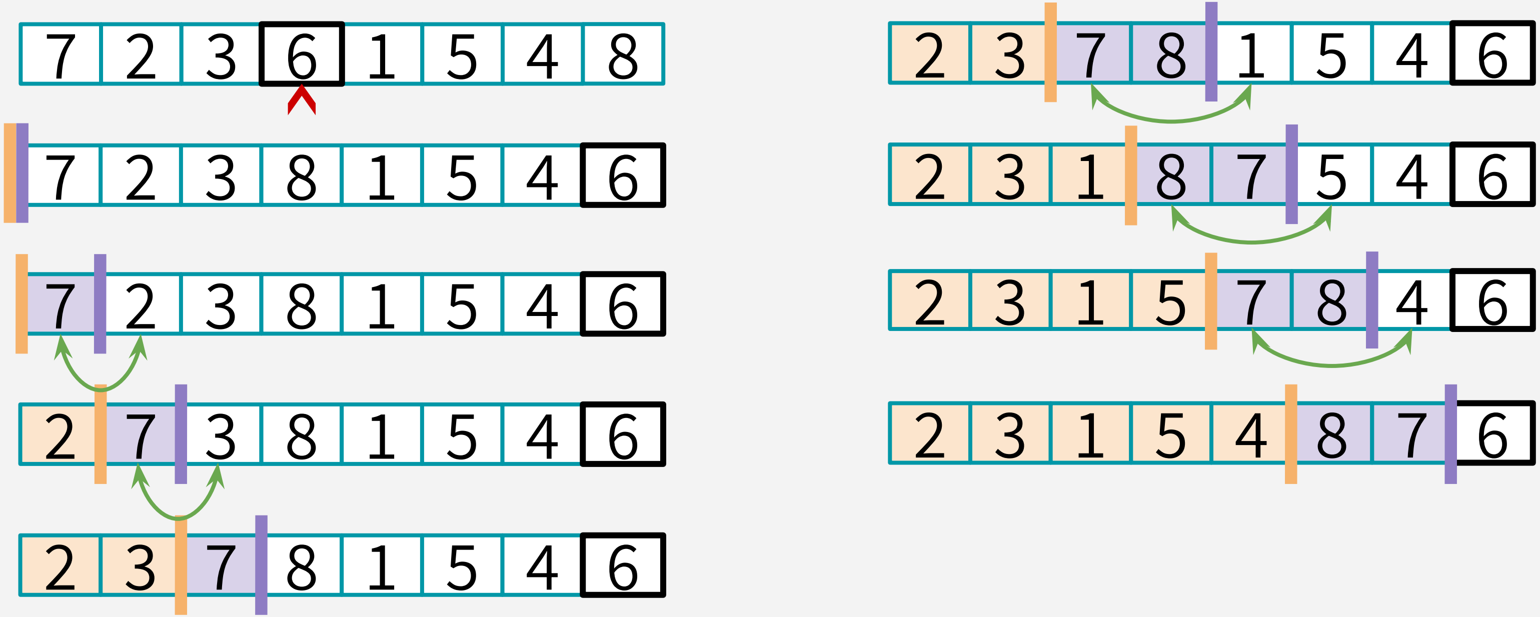
Choose pivot & swap with last element so pivot is at the end. ⇨ Initialize and ⇨ Increment until it sees something smaller than pivot, **swap** the things ahead of the bars & increment both bars ⇨ Repeat until the bar reaches the end, then swap the pivot into the right place.

AN EXAMPLE IN-PLACE PARTITION



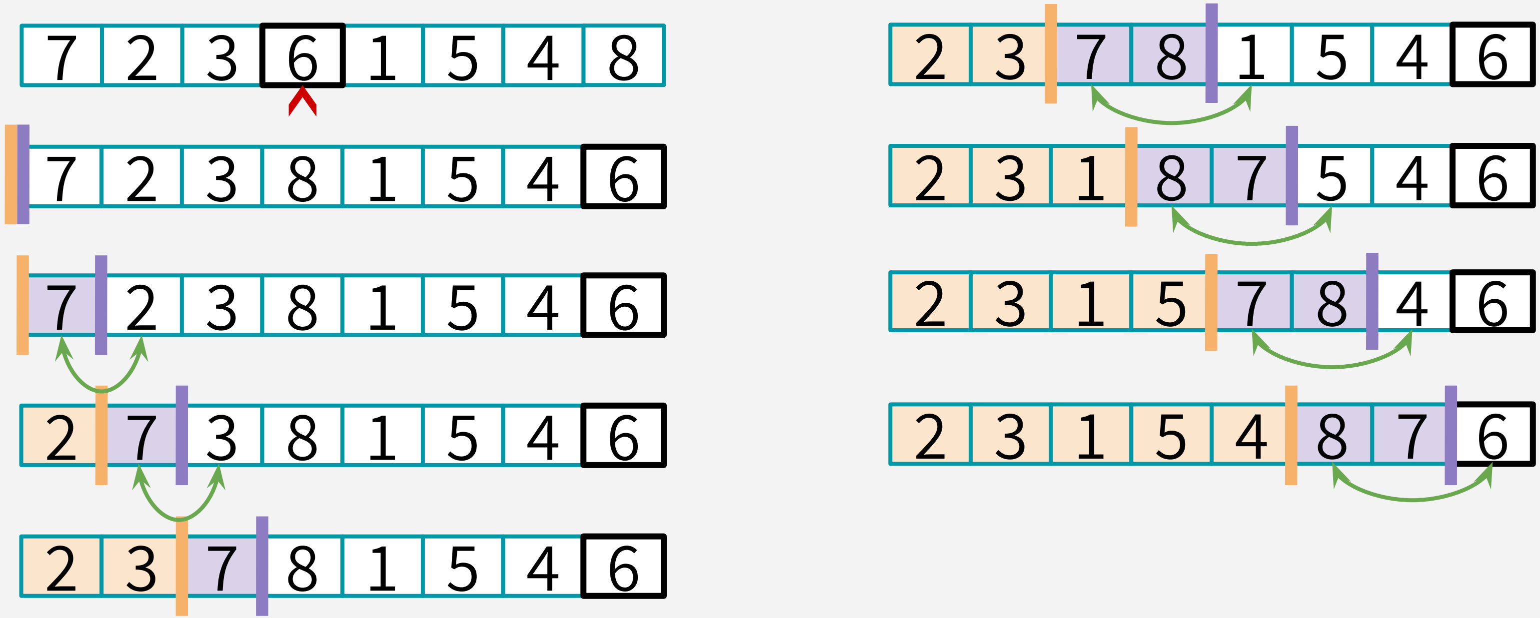
Choose pivot & swap with last element so pivot is at the end. ⇨ Initialize and ⇨ Increment until it sees something smaller than pivot, **swap** the things ahead of the bars & increment both bars ⇨ Repeat until the bar reaches the end, then swap the pivot into the right place.

AN EXAMPLE IN-PLACE PARTITION



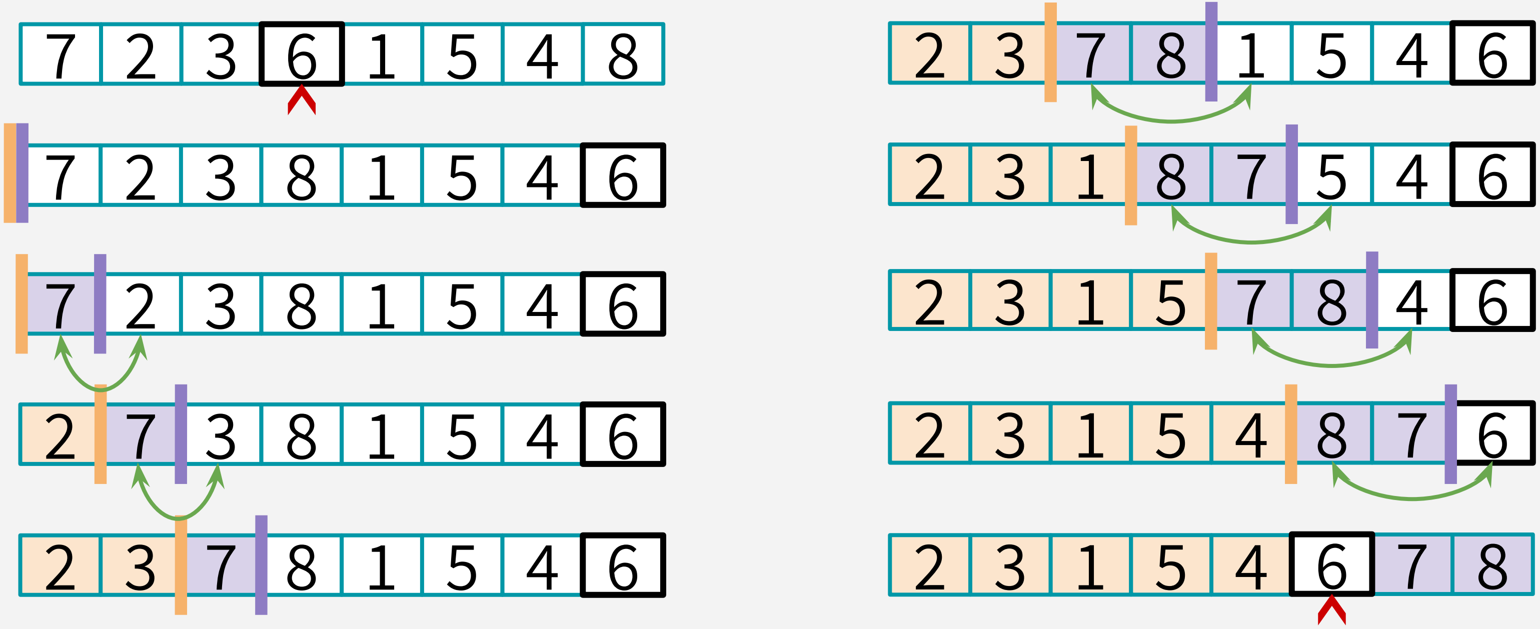
Choose pivot & swap with last element so pivot is at the end. ⇨ Initialize and ⇨ Increment until it sees something smaller than pivot, **swap** the things ahead of the bars & increment both bars ⇨ Repeat until the bar reaches the end, then swap the pivot into the right place.

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.

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)

سوال؟

QUICKSORT vs. MERGESORT

You do not need to understand any of this stuff right now

	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

سوال؟