

LECTURE 17

QUICKSORT

MCS 275 Spring 2022

Emily Dumas

LECTURE 17: QUICKSORT

Course bulletins:

- Project 2 due 6pm Fri 25 Feb.
- Project 2 autograder opens Mon 21 Feb.
- Homework 6 posted. (Shorter than usual.)

TRANSFORMATION VS MUTATION

Last time we wrote a mergesort function that acts as a *transformation*: A list is given as input, a new sorted list is returned.

Another approach we could consider is sorting as a *mutation*: A list is provided, the function reorders its items and returns nothing.

IN PLACE

A sorting transformation always uses an amount of memory that is at least as large as the list. (It needs a second list to store the output, after all.)

A sort that operates as a mutation has the possibility of using only a fixed amount of memory to do its work.

Doing so is called an **in place** sorting method.

QUICKSORT

A recursive in place sorting method that, like mergesort, is reasonably efficient and widely used.

PARTITION

Let's first study something weaker than sorting.

Given a list \mathcal{L} , let p be the last element of \mathcal{L} .

We want to rearrange \mathcal{L} so that it looks like:

$$[\textit{items} < p, \mathbf{p}, \textit{items} \geq p]$$

We say \mathcal{L} has been **partitioned** at p , and we call p the **pivot**.

PARTITION ALGORITHM

Idea: Move all small things to the front.

76235814

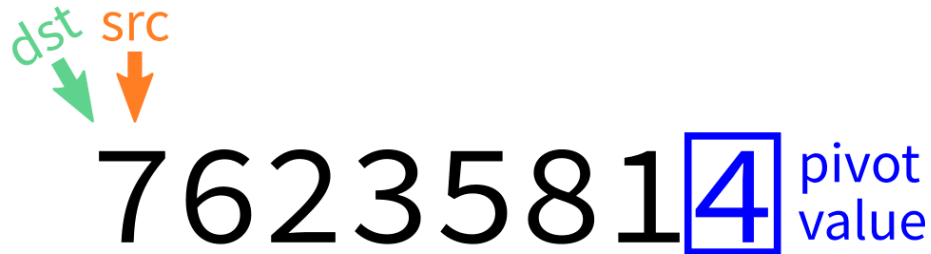
PARTITION ALGORITHM

Idea: Move all small things to the front.

76235814 pivot
value

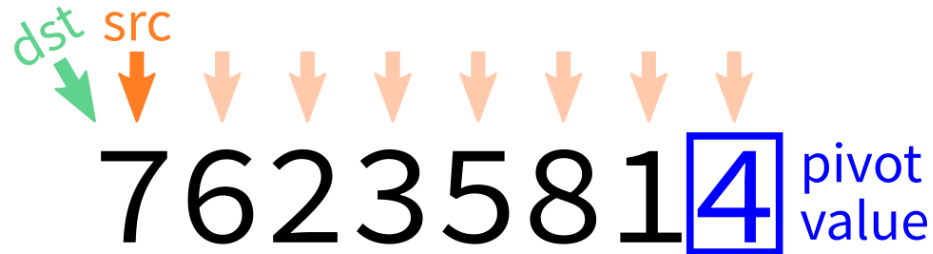
PARTITION ALGORITHM

Idea: Move all small things to the front.



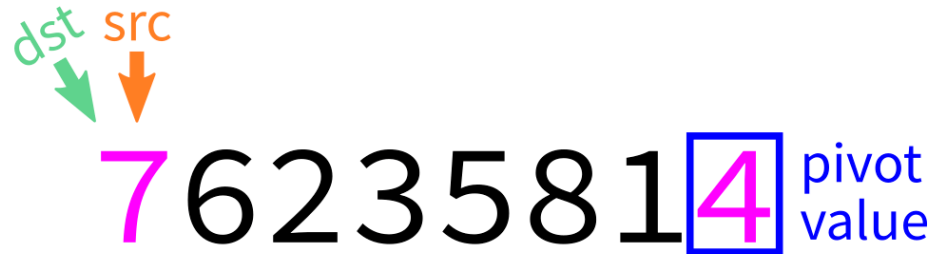
PARTITION ALGORITHM

Idea: Move all small things to the front.



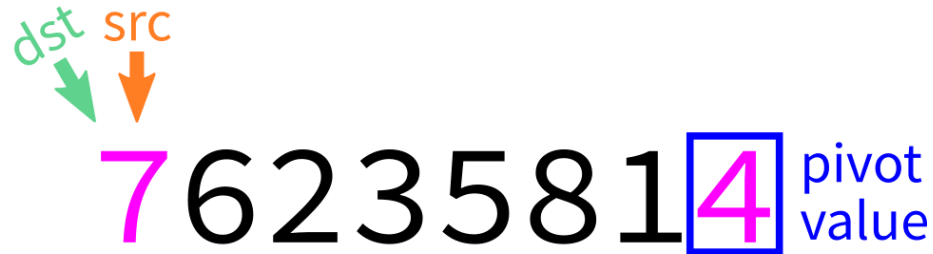
PARTITION ALGORITHM

Idea: Move all small things to the front.



PARTITION ALGORITHM

Idea: Move all small things to the front.



$L[src] \geq \text{pivot}$: do nothing

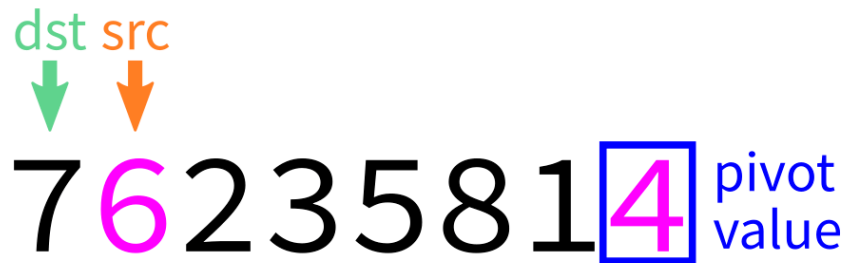
PARTITION ALGORITHM

Idea: Move all small things to the front.

dst src
↓ ↓
7 6 2 3 5 8 1 **4** pivot
value

PARTITION ALGORITHM

Idea: Move all small things to the front.



PARTITION ALGORITHM

Idea: Move all small things to the front.

dst src
↓ ↓
7 6 2 3 5 8 1 4 pivot
value

$L[\text{src}] \geq \text{pivot}$: do nothing

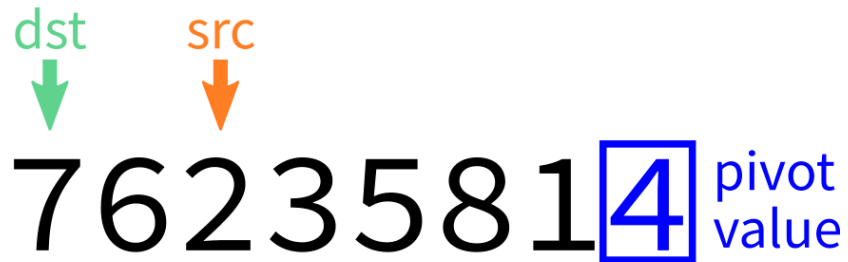
PARTITION ALGORITHM

Idea: Move all small things to the front.

dst src
↓ ↓
7 6 2 3 5 8 1 **4** pivot
value

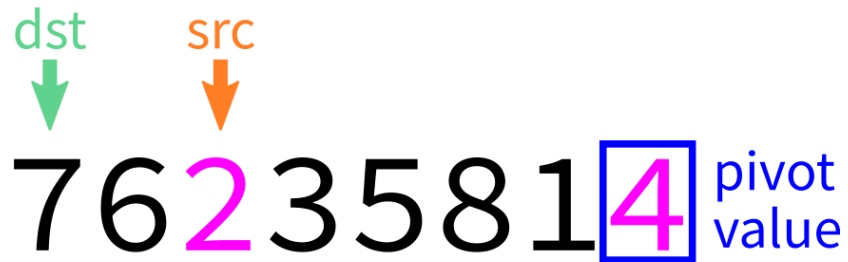
PARTITION ALGORITHM

Idea: Move all small things to the front.



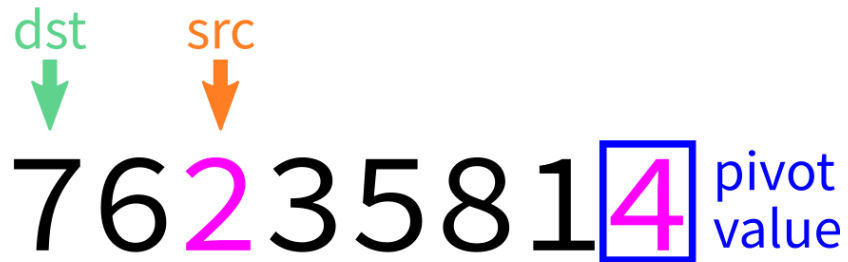
PARTITION ALGORITHM

Idea: Move all small things to the front.



PARTITION ALGORITHM

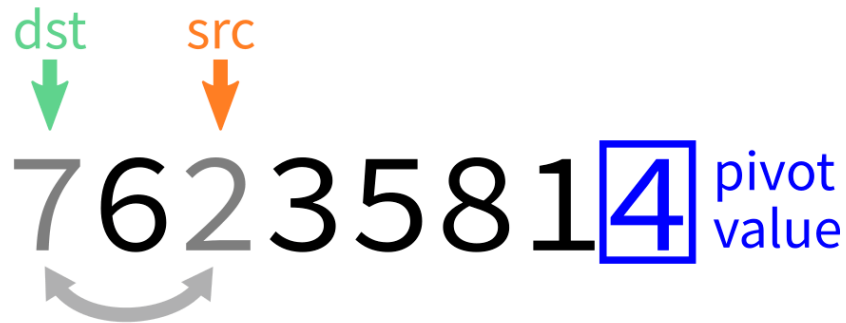
Idea: Move all small things to the front.



$L[src] < \text{pivot}$: **swap** $L[src], L[dst]$

PARTITION ALGORITHM

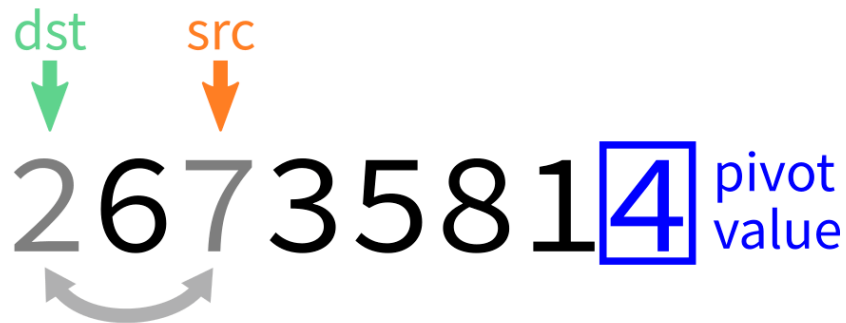
Idea: Move all small things to the front.



$L[src] < \text{pivot}$: **swap** $L[src], L[dst]$

PARTITION ALGORITHM

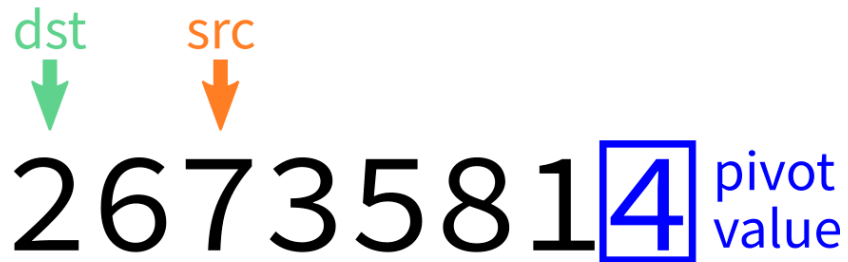
Idea: Move all small things to the front.



$L[src] < \text{pivot}$: **swap** $L[src], L[dst]$

PARTITION ALGORITHM

Idea: Move all small things to the front.



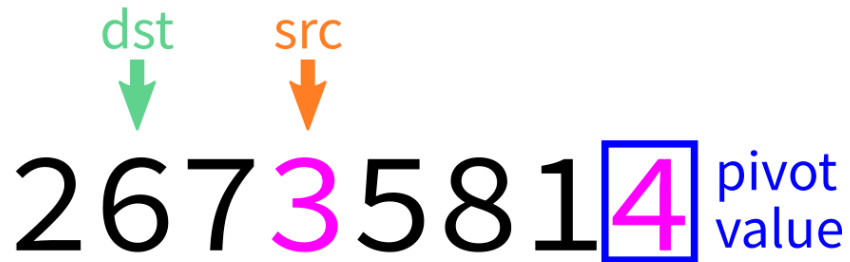
PARTITION ALGORITHM

Idea: Move all small things to the front.



PARTITION ALGORITHM

Idea: Move all small things to the front.



PARTITION ALGORITHM

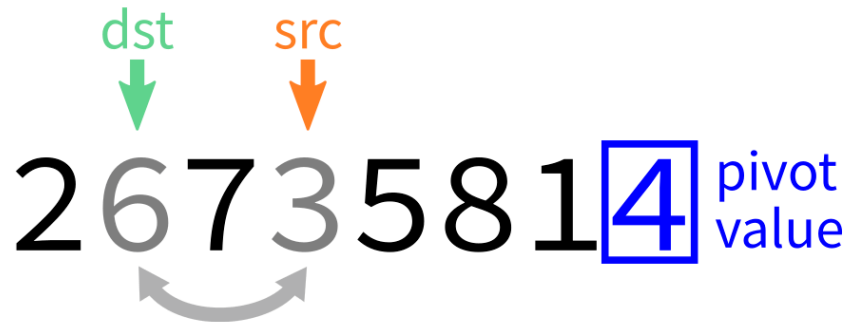
Idea: Move all small things to the front.

dst src
↓ ↓
2 6 7 3 5 8 1 4 pivot
value

$L[src] < \text{pivot}$: **swap** $L[src], L[dst]$

PARTITION ALGORITHM

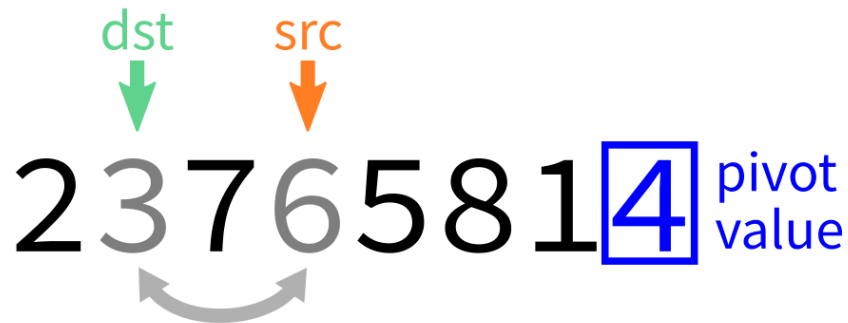
Idea: Move all small things to the front.



$L[src] < \text{pivot}$: **swap** $L[src], L[dst]$

PARTITION ALGORITHM

Idea: Move all small things to the front.



$L[src] < \text{pivot}$: **swap** $L[src], L[dst]$

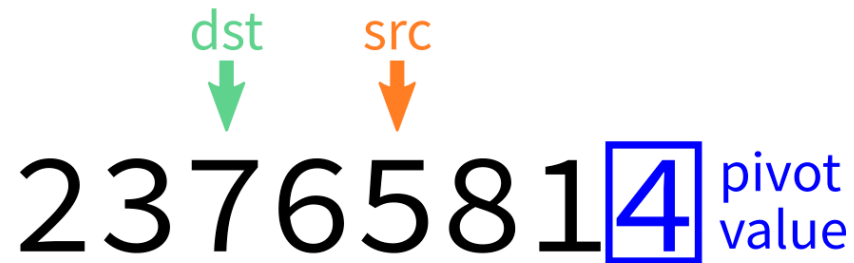
PARTITION ALGORITHM

Idea: Move all small things to the front.



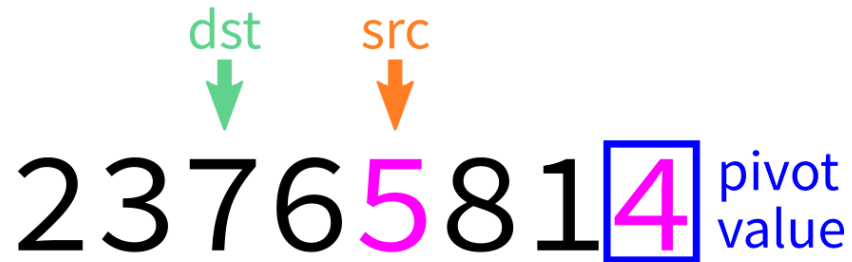
PARTITION ALGORITHM

Idea: Move all small things to the front.



PARTITION ALGORITHM

Idea: Move all small things to the front.



PARTITION ALGORITHM

Idea: Move all small things to the front.

dst src
↓ ↓
23765814 pivot
value

$L[src] \geq \text{pivot}$: do nothing

PARTITION ALGORITHM

Idea: Move all small things to the front.

dst src
↓ ↓
23765814 pivot
value

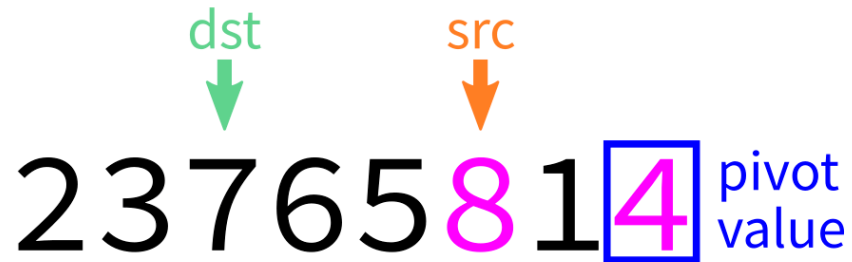
PARTITION ALGORITHM

Idea: Move all small things to the front.

dst src
↓ ↓
23765814 pivot
 value

PARTITION ALGORITHM

Idea: Move all small things to the front.



PARTITION ALGORITHM

Idea: Move all small things to the front.

dst src
↓ ↓
23765814 pivot
 value

$L[src] \geq \text{pivot}$: do nothing

PARTITION ALGORITHM

Idea: Move all small things to the front.

dst src
↓ ↓
23765814 pivot
 value

PARTITION ALGORITHM

Idea: Move all small things to the front.



PARTITION ALGORITHM

Idea: Move all small things to the front.



PARTITION ALGORITHM

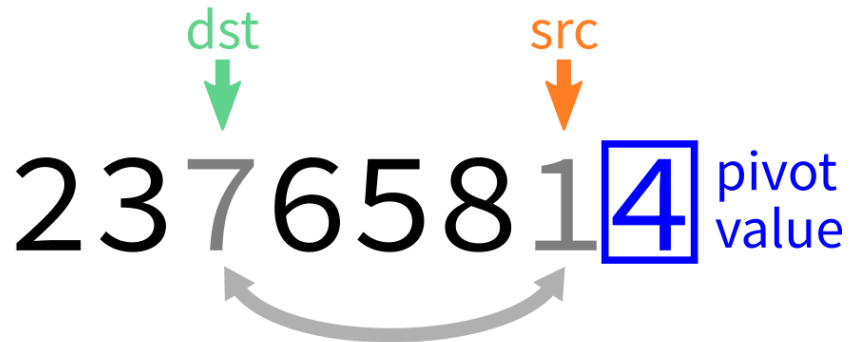
Idea: Move all small things to the front.

dst
↓
23765814 pivot
src
↓
value

$L[src] < \text{pivot}$: **swap** $L[src], L[dst]$

PARTITION ALGORITHM

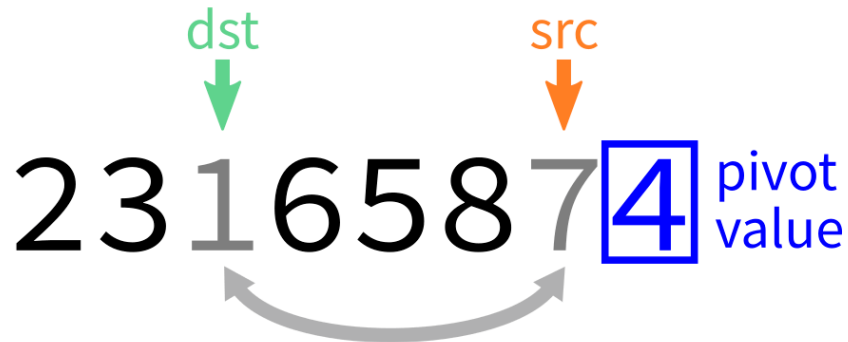
Idea: Move all small things to the front.



$L[src] < \text{pivot}$: **swap** $L[src], L[dst]$

PARTITION ALGORITHM

Idea: Move all small things to the front.



$L[src] < \text{pivot}$: **swap** $L[src], L[dst]$

PARTITION ALGORITHM

Idea: Move all small things to the front.



PARTITION ALGORITHM

Idea: Move all small things to the front.



PARTITION ALGORITHM

Idea: Move all small things to the front.



PARTITION ALGORITHM

Idea: Move all small things to the front.

dst
↓
23165874 pivot
src ↓ value

$L[\text{src}] \geq \text{pivot}$: do nothing

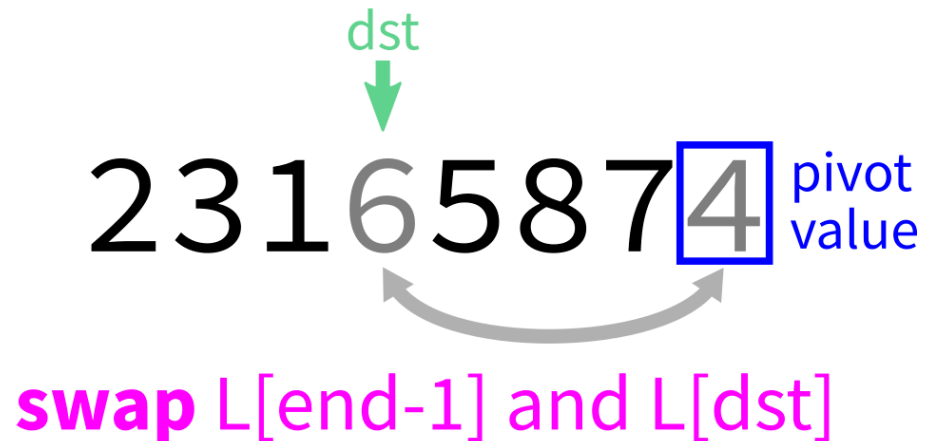
PARTITION ALGORITHM

Idea: Move all small things to the front.

dst
↓
2316587**4** pivot
value

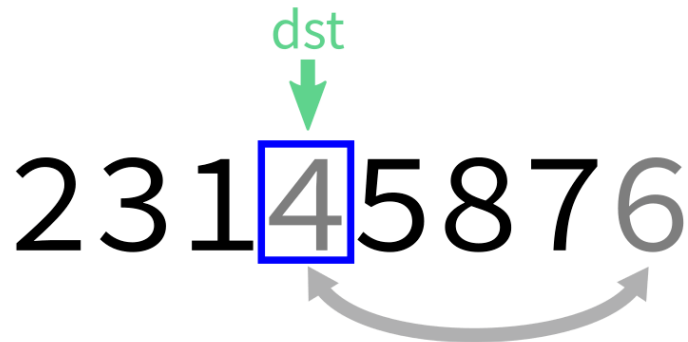
PARTITION ALGORITHM

Idea: Move all small things to the front.



PARTITION ALGORITHM

Idea: Move all small things to the front.



swap $L[\text{end}-1]$ and $L[\text{dst}]$

PARTITION ALGORITHM

Idea: Move all small things to the front.

231**4**5876

PARTITION ALGORITHM

Idea: Move all small things to the front.



23145876

The diagram shows the array [2, 3, 1, 4, 5, 8, 7, 6] with the pivot element 4 highlighted by a blue square. An orange bracket is positioned under the elements 2, 3, and 1, which are all less than the pivot. A purple bracket is positioned under the elements 5, 8, 7, and 6, which are all greater than the pivot.

PARTITION ALGORITHM

Idea: Move all small things to the front.



AFTER PARTITION

The two chunks of the list on either side of the pivot may not be sorted.

But we could bring each of them closer to being sorted by partitioning them...

QUICKSORT SUMMARY

Starting with an unsorted list:

- If the list has 0 or 1 elements, return immediately.
- Otherwise, partition the list.
- Quicksort the part of the list before the pivot.
- Quicksort the part of the list after the pivot.

It's divide and conquer, but with no merge step. The hard work is instead in partitioning.

QUICKSORT VISUALIZATION

76235814

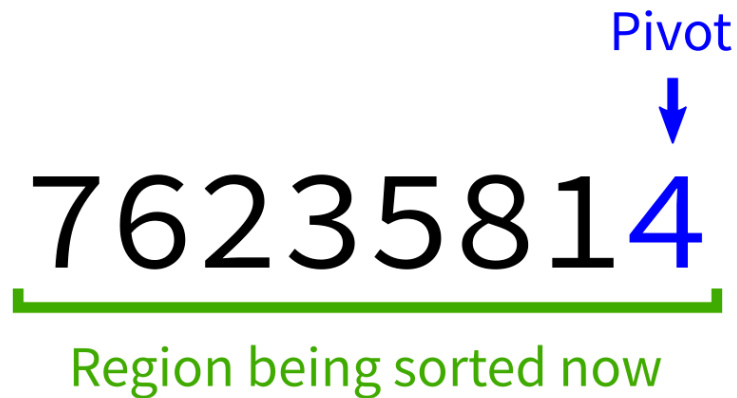
QUICKSORT VISUALIZATION

76235814

Region being sorted now

QUICKSORT VISUALIZATION

Pivot
↓
76235814
Region being sorted now

The image shows a sequence of numbers: 7, 6, 2, 3, 5, 8, 1, 4. The number 4 is highlighted in blue and labeled 'Pivot' with a blue arrow pointing down to it. A green horizontal line is drawn under the entire sequence of numbers, and the text 'Region being sorted now' is written in green below this line.

QUICKSORT VISUALIZATION

Pivot

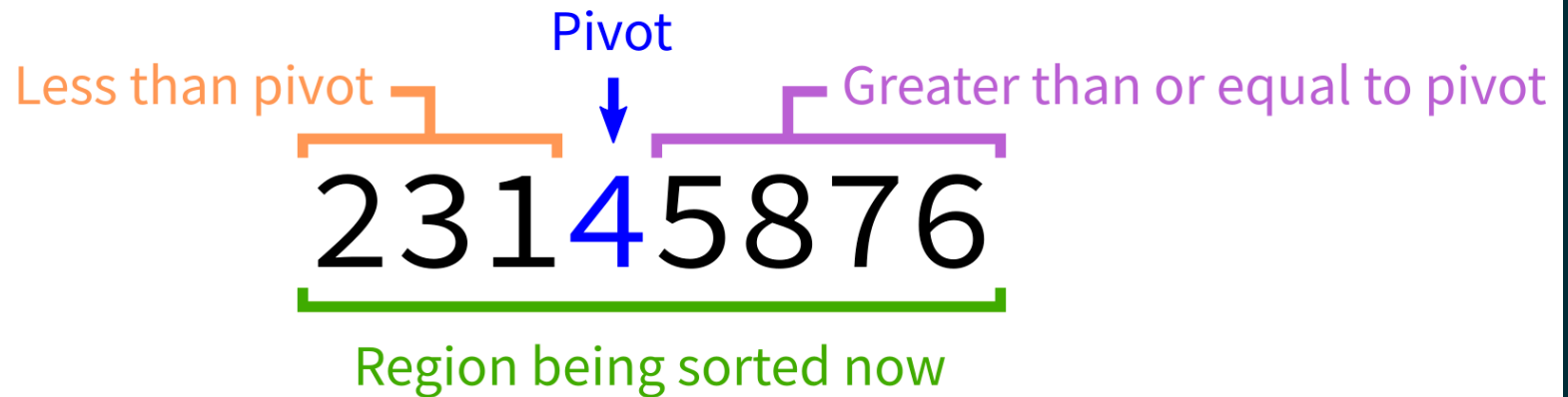


23145876



Region being sorted now

QUICKSORT VISUALIZATION



QUICKSORT VISUALIZATION

23145876

Region being sorted now

QUICKSORT VISUALIZATION

23145876



QUICKSORT VISUALIZATION

23145876

QUICKSORT VISUALIZATION

13245876

QUICKSORT VISUALIZATION

13245876



QUICKSORT VISUALIZATION

13245876

QUICKSORT VISUALIZATION

13245876

QUICKSORT VISUALIZATION

12345876

QUICKSORT VISUALIZATION

12345876



QUICKSORT VISUALIZATION

12345876



QUICKSORT VISUALIZATION

12345876

QUICKSORT VISUALIZATION

12345876



QUICKSORT VISUALIZATION

12345876

QUICKSORT VISUALIZATION

12345876



QUICKSORT VISUALIZATION

12345678

QUICKSORT VISUALIZATION

12345678



QUICKSORT VISUALIZATION

12345678



QUICKSORT VISUALIZATION

12345678



QUICKSORT VISUALIZATION

12345678

QUICKSORT VISUALIZATION

12345678

QUICKSORT VISUALIZATION

12345678

QUICKSORT VISUALIZATION

12345678



QUICKSORT VISUALIZATION

12345678



QUICKSORT VISUALIZATION

12345678

QUICKSORT VISUALIZATION

12345678

CODING TIME

Let's implement `quicksort` in Python.

Algorithm quicksort:

Input: list L and indices $start$ and end .

Goal: reorder elements of L so that $L[start:end]$ is sorted.

1. If $(end - start)$ is less than or equal to 1, return immediately.
2. Otherwise, call $partition(L)$ to partition the list, letting m be the final location of the pivot.
3. Call $quicksort(L, start, m)$ and $quicksort(L, m+1, end)$ to sort the parts of the list on either side of the pivot.

WHY DISCUSS ALGORITHMS?

Python lists have built-in `.sort()` method. Why talk about sorting?

1. Study cases of easy-to-explain problems solved in clever ways.
2. See patterns of thinking that work in other settings.

EVALUATING SORTS

Last time we discussed and implemented mergesort, developed by von Neumann (1945) and Goldstine (1947).

Today we discussed quicksort, first described by Hoare (1959) and the simpler partitioning scheme introduced by Lomuto.

But are these actually good ways to sort a list?

EFFICIENCY

Theorem: If you measure the time cost of mergesort in any of these terms

- Number of comparisons made
- Number of assignments (e.g. $L[i] = x$ counts as 1)
- Number of Python statements executed

then the cost to sort a list of length n is less than $Cn \log(n)$, for some constant C that only depends on which expense measure you chose.

ASYMPTOTICALLY OPTIMAL

$Cn \log(n)$ is pretty efficient for an operation that needs to look at all n elements. It's not linear in n , but it only grows a little faster than linear functions.

Furthermore, $Cn \log(n)$ is the best possible time for comparison sort of n elements (though different methods might have better C).

QUICKSORT

Is quicksort similarly efficient?

REFERENCES

- Making nice visualizations of sorting algorithms is a cottage industry in CS education. Some you might like to check out:
 - [2D visualization through color sorting](#) by Linus Lee
 - [Animated bar graph visualization of many sorting algorithms](#) by Alex Macy
 - Slanted line animated visualizations of [mergesort](#) and [quicksort](#) by Mike Bostock

REVISION HISTORY

- 2022-02-18 Initial publication

