

LECTURE 14

QUICKSORT

MCS 275 Spring 2023

Emily Dumas

LECTURE 14: QUICKSORT

Reminders and announcements:

- Project 2 due 6pm Fri 24 Feb.
- Project 2 autograder opens by Mon 20 Feb.
 - Having at least partial work ready to submit at that time is a good goal.

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 extra memory proportional to the size of the list. (It needs a second list to store the output.)

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

Scan through the list, moving things smaller than the pivot to the beginning.

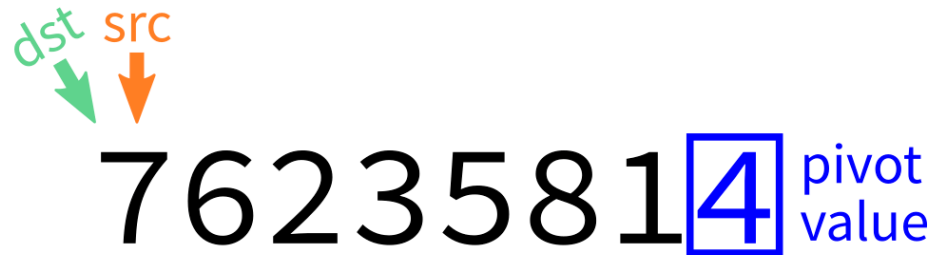
PARTITION ALGORITHM VISUALIZATION

76235814

PARTITION ALGORITHM VISUALIZATION

76235814 pivot
value

PARTITION ALGORITHM VISUALIZATION

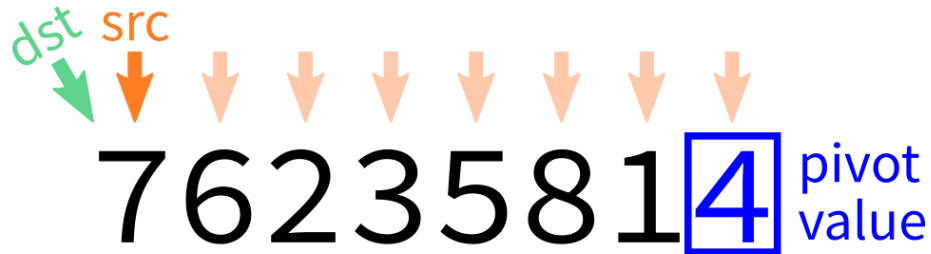


A diagram illustrating the partitioning step of a sorting algorithm. It shows an array of numbers: 7, 6, 2, 3, 5, 8, 1, and 4. The number 4 is highlighted with a blue square and labeled 'pivot value' in blue text. Above the array, a green arrow labeled 'dst' points to the position before the first element (7), and an orange arrow labeled 'src' points to the pivot element (4).

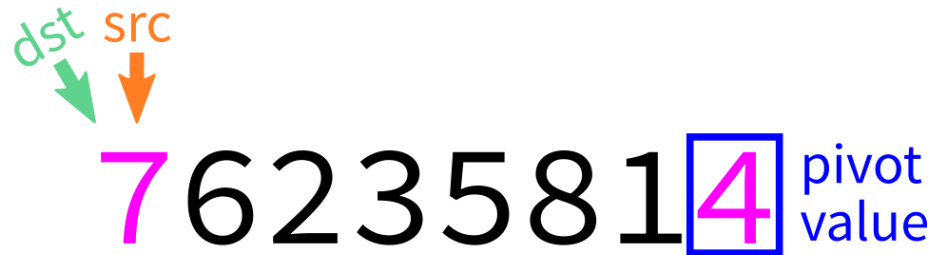
dst src

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

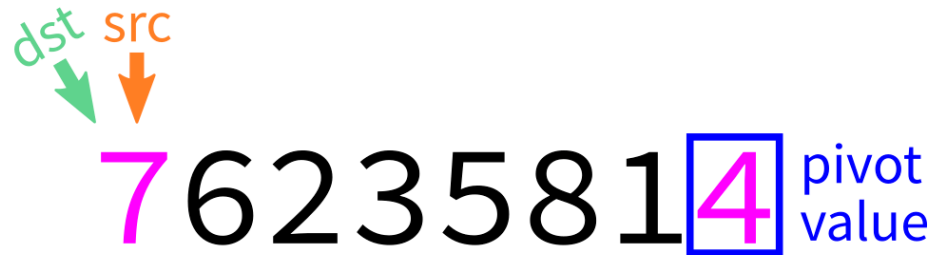
PARTITION ALGORITHM VISUALIZATION



PARTITION ALGORITHM VISUALIZATION



PARTITION ALGORITHM VISUALIZATION



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

PARTITION ALGORITHM VISUALIZATION

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

PARTITION ALGORITHM VISUALIZATION

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

PARTITION ALGORITHM VISUALIZATION

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

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

PARTITION ALGORITHM VISUALIZATION

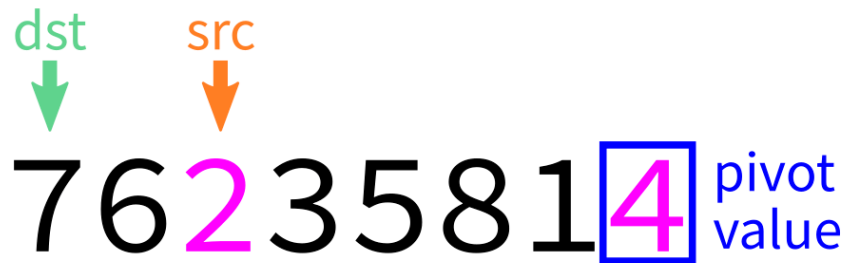
dst src
↓ ↓
7 6 2 3 5 8 1 4 pivot
value

PARTITION ALGORITHM VISUALIZATION

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

PARTITION ALGORITHM VISUALIZATION

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



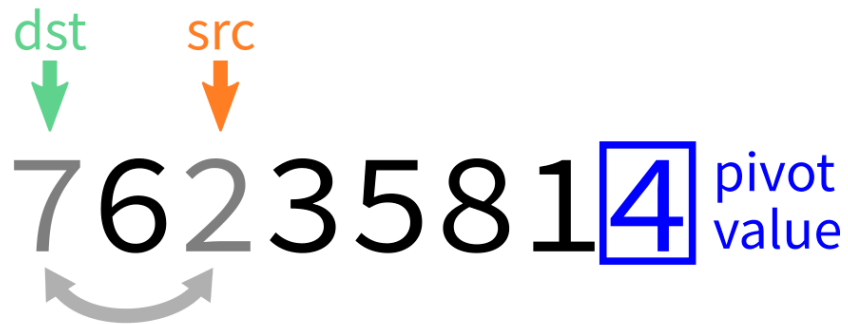
The diagram illustrates the partitioning step of a sorting algorithm. It shows an array of numbers: 7, 6, 2, 3, 5, 8, 1, 4. The number 4 is designated as the 'pivot value' and is enclosed in a blue box. A green arrow labeled 'dst' points to the first element (7), and an orange arrow labeled 'src' points to the third element (2). The pivot value 4 is also labeled 'pivot value' in blue text.

PARTITION ALGORITHM VISUALIZATION

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

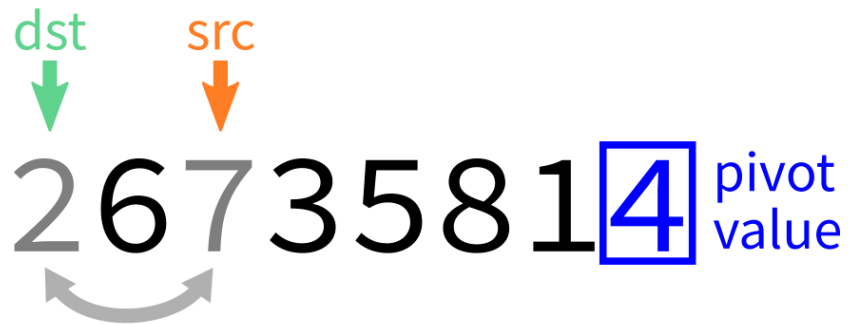
$L[src] < \text{pivot}$: **swap** $L[src], L[dst]$

PARTITION ALGORITHM VISUALIZATION



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

PARTITION ALGORITHM VISUALIZATION



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

PARTITION ALGORITHM VISUALIZATION

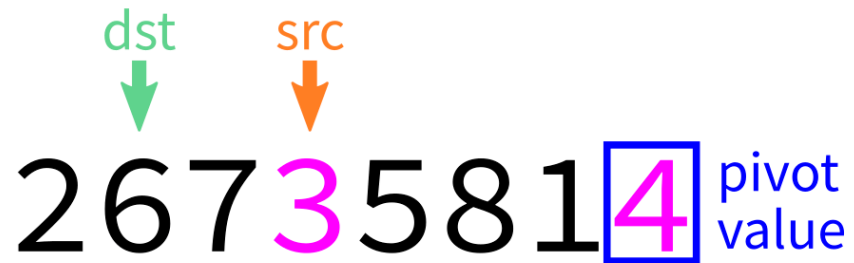
dst src
↓ ↓
2 6 7 3 5 8 1 **4** pivot
value

PARTITION ALGORITHM VISUALIZATION

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

PARTITION ALGORITHM VISUALIZATION

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



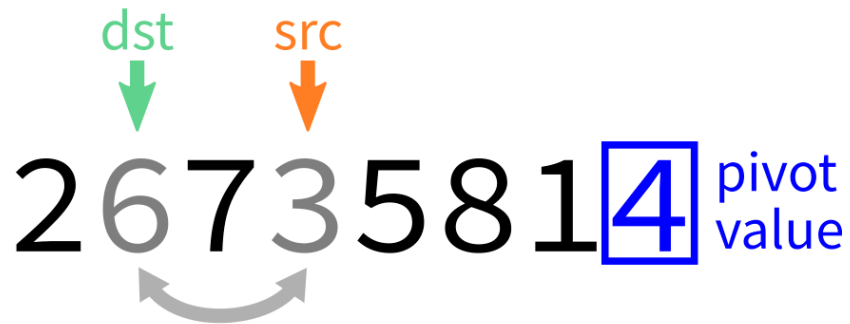
The diagram illustrates a step in the partitioning process of an array [2, 6, 7, 3, 5, 8, 1, 4]. A green arrow labeled 'dst' points to the element '2' at index 0. An orange arrow labeled 'src' points to the element '3' at index 3. The element '4' at index 7 is enclosed in a blue box and labeled 'pivot value' in blue text. The element '3' is highlighted in pink.

PARTITION ALGORITHM VISUALIZATION

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

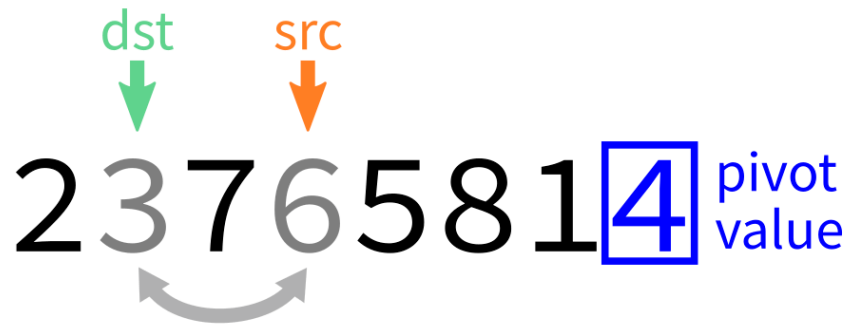
$L[src] < \text{pivot}$: **swap** $L[src], L[dst]$

PARTITION ALGORITHM VISUALIZATION



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

PARTITION ALGORITHM VISUALIZATION



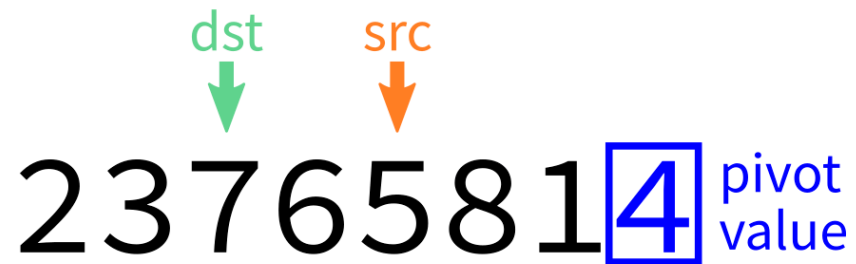
$L[src] < \text{pivot}$: **swap** $L[src], L[dst]$

PARTITION ALGORITHM VISUALIZATION

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

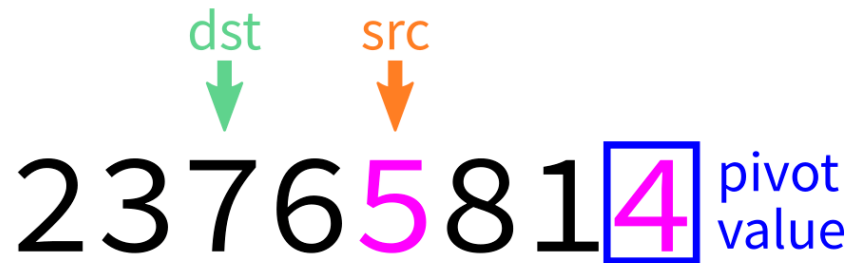
PARTITION ALGORITHM VISUALIZATION

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



The diagram illustrates the partitioning step of a sorting algorithm. It shows an array of numbers: 2, 3, 7, 6, 5, 8, 1, and 4. The number 4 is highlighted with a blue square and labeled 'pivot value'. Above the array, two arrows point downwards: a green arrow labeled 'dst' pointing to the element 7, and an orange arrow labeled 'src' pointing to the element 6. This represents the state where elements are being compared and potentially swapped relative to the pivot.

PARTITION ALGORITHM VISUALIZATION



A diagram illustrating the partitioning step of a sorting algorithm. It shows a sequence of numbers: 2, 3, 7, 6, 5, 8, 1, and 4. Above the number 7 is a green arrow pointing down, labeled 'dst'. Above the number 5 is an orange arrow pointing down, labeled 'src'. The number 4 is enclosed in a blue square box, and to its right, the text 'pivot value' is written in blue. The number 5 is highlighted in magenta.

2 3 7 6 5 8 1 4 pivot value

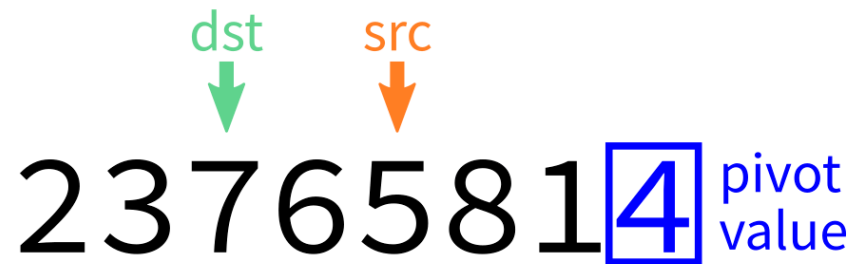
PARTITION ALGORITHM VISUALIZATION

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

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

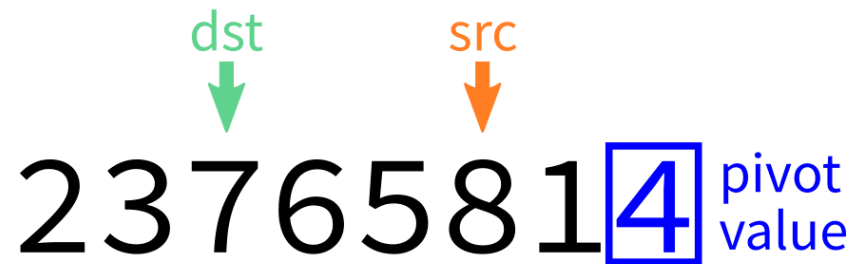
PARTITION ALGORITHM VISUALIZATION

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

A diagram illustrating the partitioning step of a sorting algorithm. It shows an array of numbers: 2, 3, 7, 6, 5, 8, 1, and 4. Above the number 7 is a green arrow pointing down, labeled 'dst'. Above the number 6 is an orange arrow pointing down, labeled 'src'. The number 4 is enclosed in a blue square box, and to its right, the text 'pivot value' is written in blue. The entire diagram is set against a white background within a dark blue frame.

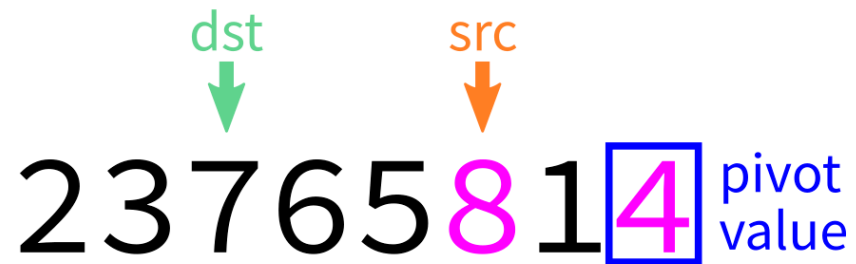
PARTITION ALGORITHM VISUALIZATION

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

A diagram illustrating the partitioning step of a sorting algorithm. It shows an array of numbers: 2, 3, 7, 6, 5, 8, 1, and 4. Above the number 7 is a green arrow pointing down, labeled 'dst'. Above the number 4 is an orange arrow pointing down, labeled 'src'. The number 4 is enclosed in a blue square box, and to its right, the text 'pivot value' is written in blue. The entire diagram is set against a white rectangular background.

PARTITION ALGORITHM VISUALIZATION

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



The diagram illustrates a step in the partitioning process of an array [2, 3, 7, 6, 5, 8, 1, 4]. A green arrow labeled 'dst' points to the element 7, and an orange arrow labeled 'src' points to the element 8. The element 4 is enclosed in a blue box and labeled 'pivot value'.

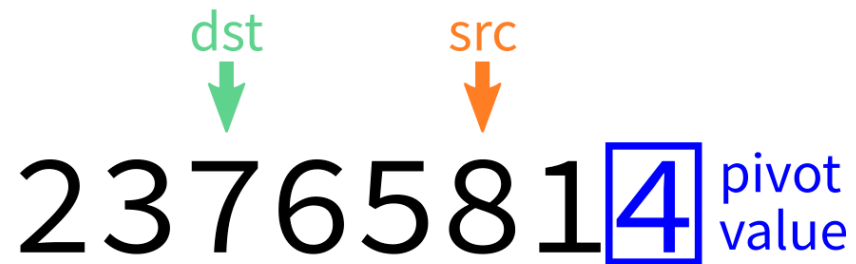
PARTITION ALGORITHM VISUALIZATION

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

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

PARTITION ALGORITHM VISUALIZATION

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

A diagram illustrating the partitioning step of a sorting algorithm. It shows an array of numbers: 2, 3, 7, 6, 5, 8, 1, and 4. Above the number 7 is a green arrow pointing down, labeled 'dst'. Above the number 4 is an orange arrow pointing down, labeled 'src'. The number 4 is enclosed in a blue square, and to its right, the text 'pivot value' is written in blue. The entire diagram is set against a white background within a dark blue frame.

PARTITION ALGORITHM VISUALIZATION

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



The diagram illustrates the partitioning step of a sorting algorithm. It shows an array of numbers: 2, 3, 7, 6, 5, 8, 1, and 4. A green arrow labeled 'dst' points to the number 7, and an orange arrow labeled 'src' points to the number 1. The number 4 is highlighted with a blue square and labeled 'pivot value'.

PARTITION ALGORITHM VISUALIZATION

dst src
↓ ↓
23765814 pivot
value



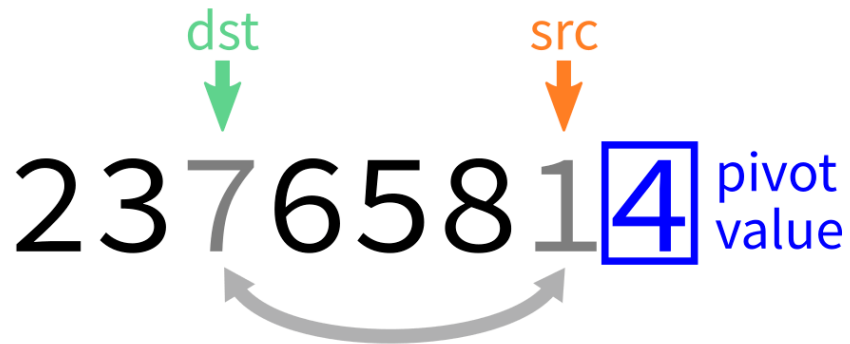
The diagram illustrates the partitioning step of a sorting algorithm. It shows a sequence of numbers: 2, 3, 7, 6, 5, 8, 1, and 4. Above the number 7 is a green arrow pointing down, labeled 'dst'. Above the number 1 is an orange arrow pointing down, labeled 'src'. The number 4 is enclosed in a blue square box, and to its right, the text 'pivot value' is written in blue. The number 1 is highlighted in pink.

PARTITION ALGORITHM VISUALIZATION

dst src
↓ ↓
23765814 pivot
value

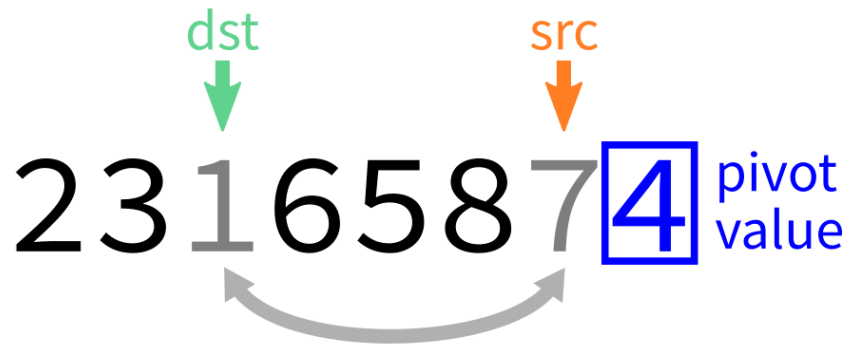
$L[src] < \text{pivot}$: **swap** $L[src], L[dst]$

PARTITION ALGORITHM VISUALIZATION



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

PARTITION ALGORITHM VISUALIZATION



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

PARTITION ALGORITHM VISUALIZATION

dst src
↓ ↓
23165874 pivot
value



The diagram illustrates the partitioning step of a sorting algorithm. It shows an array of numbers: 2, 3, 1, 6, 5, 8, 7, and 4. The number 4 is highlighted with a blue square and labeled 'pivot value' in blue text. Above the array, a green arrow labeled 'dst' points to the position before the first element (2), and an orange arrow labeled 'src' points to the element 7, which is immediately before the pivot 4.

PARTITION ALGORITHM VISUALIZATION

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

The diagram illustrates the partitioning step of a sorting algorithm. It shows an array of numbers: 2, 3, 1, 6, 5, 8, 7, and 4. The number 4 is highlighted with a blue square and labeled 'pivot value'. A green arrow labeled 'dst' points to the number 6, indicating its current position. An orange arrow labeled 'src' points to the number 4, indicating its current position.

PARTITION ALGORITHM VISUALIZATION

23165874

dst

src

4 pivot value



The diagram illustrates a step in the partitioning process of an array [2, 3, 1, 6, 5, 8, 7, 4]. A green arrow labeled 'dst' points to the element '6' at index 3. An orange arrow labeled 'src' points to the element '4' at index 7. The element '4' is highlighted with a blue square and labeled '4 pivot value' in blue text, indicating it is the current pivot.

PARTITION ALGORITHM VISUALIZATION

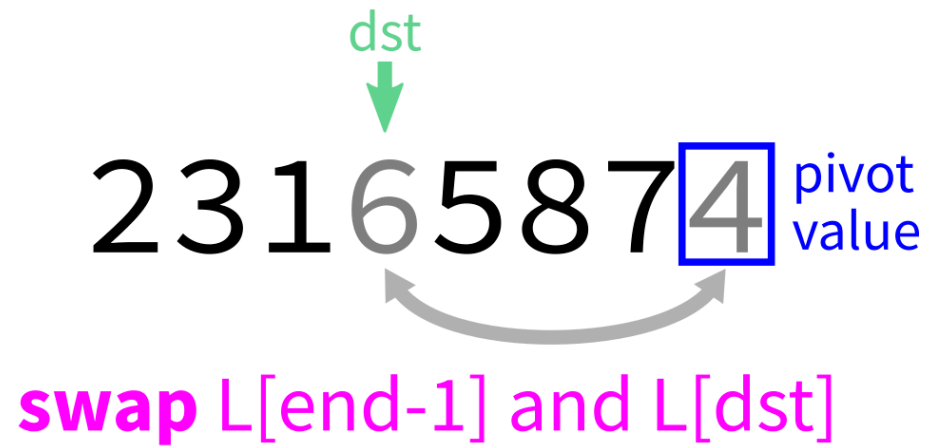
dst
↓
23165874 pivot
src
↓ value

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

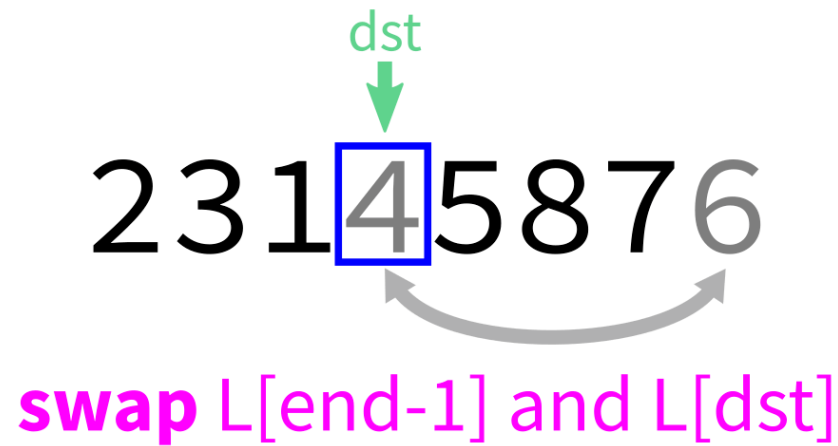
PARTITION ALGORITHM VISUALIZATION

dst
↓
23165874 pivot
value

PARTITION ALGORITHM VISUALIZATION



PARTITION ALGORITHM VISUALIZATION



PARTITION ALGORITHM VISUALIZATION

23145876

PARTITION ALGORITHM VISUALIZATION

23145876



The image shows the partitioning step of a sorting algorithm on the array [2, 3, 1, 4, 5, 8, 7, 6]. The pivot element '4' is highlighted with a blue square. An orange bracket is positioned under the elements [2, 3, 1], and a purple bracket is positioned under the elements [5, 8, 7, 6].

PARTITION ALGORITHM VISUALIZATION

23145876



The diagram illustrates the partitioning of an array [2, 3, 1, 4, 5, 8, 7, 6] around a pivot element 4. The pivot is highlighted with a blue square. Elements less than the pivot (2, 3, 1) are grouped under an orange bracket labeled <4. Elements greater than or equal to the pivot (5, 8, 7, 6) are grouped under a purple bracket labeled ≥4.

<4 ≥ 4

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

QUICKSORT VISUALIZATION

Pivot

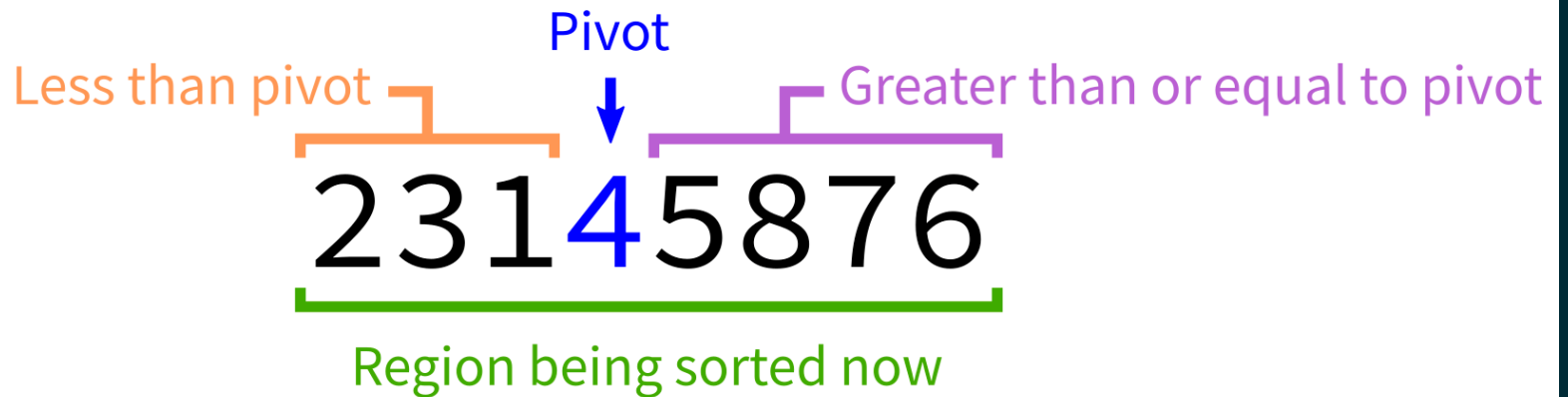


23145876



Region being sorted now

QUICKSORT VISUALIZATION



QUICKSORT VISUALIZATION

23145876

Region being sorted now

QUICKSORT VISUALIZATION

23145876

The image shows a sequence of numbers 23145876. The first three numbers, 2, 3, and 1, are grouped together by a green horizontal line underneath them. The number 4 is positioned between the green line and the rest of the sequence, indicating it is the pivot element. The numbers 5, 8, 7, and 6 follow 4.

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



The image shows a sequence of numbers: 1, 2, 3, 4, 5, 8, 7, 6. The numbers 1, 2, 3, and 4 are in a light gray color. The number 5 is in black and has a green horizontal line underneath it, indicating it is the pivot. The number 8 is in black. The number 7 is in black. The number 6 is in blue. This visualizes a step in the Quicksort algorithm where the pivot 5 is being compared with the elements 8 and 7.

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.

Algorithm partition:

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

Goal: Take $L[end-1]$ as a pivot, and reorder elements of L to partition $L[start:end]$ accordingly.

1. Let $pivot = L[end-1]$.
2. Initialize integer index $dst = start$.
3. For each integer src from $start$ to $end-1$:
 - If $L[src] < pivot$, swap $L[src]$ and $L[dst]$ and increment dst .
4. Swap $L[end-1]$ and $L[dst]$ to put the pivot in its proper place.
5. Return dst .

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

- Recursion references from [Lecture 10](#).
- Sorting visualizations:
 - [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 Last year's lecture on this topic finalized
- 2023-02-15 Updated for 2023

