LECTURE 17

QUICKSORT

MCS 275 Spring 2021

Emily Dumas

LECTURE 17: QUICKSORT

Course bulletins:

- Starting with Quiz 6, you will have 48 hours for quizzes (Noon Sunday to Noon Tuesday).
- Project 2 description updated with sample data and modules policy.
- Project 2 due 6pm CST Friday, February 26.
- Check out the recursion sample code.
- Worksheet 7 will explore recursive maze solver / generator in more depth.

PLAN

- Discuss mergesort a bit more
- Introduce quicksort
- Implement quicksort

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.

MERGESORT

Last time we discussed and implemented mergesort.

History: Developed by von Neumann (1945) and Goldstine (1947).

But is it a good way 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

Another comparison sort typically implemented using recursion. Developed by Hoare, 1959.

Unlike mergesort, it uses very little temporary storage, and only ever **swaps** pairs of elements.

QUICKSORT SUMMARY

Starting with an unsorted list:

- If the list has 0 or 1 elements, return immediately.
- Partition: Choose an element (the **pivot**). Rearrange so elements smaller than the pivot come before it, elements larger than the pivot come after it.
- 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

2 3 1 4 5 8 7 6

QUICKSORT VISUALIZATION

23145876

QUICKSORT VISUALIZATION

13245876

QUICKSORT VISUALIZATION

13245876

QUICKSORT VISUALIZATION

13245876

QUICKSORT VISUALIZATION

13245876

QUICKSORT VISUALIZATION

1 2 3 4 5 8 7 6

QUICKSORT VISUALIZATION

12345876

QUICKSORT VISUALIZATION

1 2 3 4 5 8 7 6

QUICKSORT VISUALIZATION

1 2 3 4 5 8 7 6

QUICKSORT VISUALIZATION

1 2 3 4 5 8 7 6

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

ASSUME WE HAVE PARTITION

For the moment, we'll take the partition step as a "black box", assuming we already have:

```
def partition(L,start,end):
    """Look at L[start:end].  Take the last element as a pivot
    Move elements around so that any value less than the pivot
    appears before it, and any element greater than or equal t
    the pivot appears after it.  L is modified in place.  The
    final location of the pivot is returned."""
    # TODO: Add code here
```

Note this function uses the last element as a pivot. Later we'll discuss other options.

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.

PARTITION

How to write `partition(L,start,end)`?

Recall we plan to make a version that uses the last element of `L[start:end]` as the pivot.

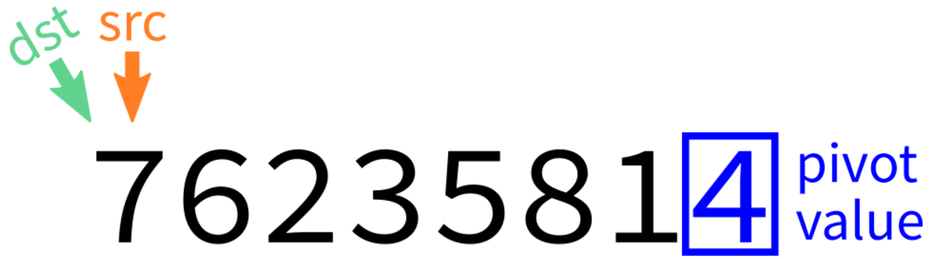
PARTITION VISUALIZATION

76235814

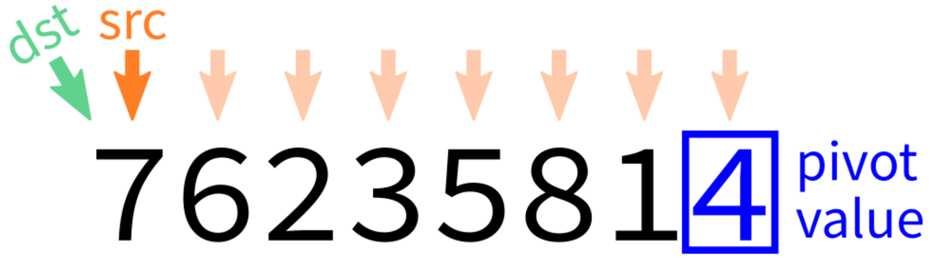
PARTITION VISUALIZATION

7 6 2 3 5 8 1 $\boxed{4}$ pivot value

PARTITION VISUALIZATION



PARTITION VISUALIZATION



PARTITION VISUALIZATION

dst src

7623581$\boxed{4}$ pivot value

PARTITION VISUALIZATION

dst src

7 6 2 3 5 8 1 $\boxed{4}$ pivot value

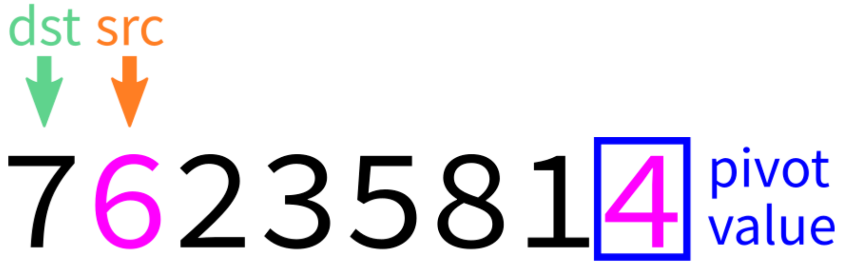
L[src] ≥ pivot: do nothing

PARTITION VISUALIZATION

dst src

7623581$\boxed{4}$ pivot value

PARTITION VISUALIZATION



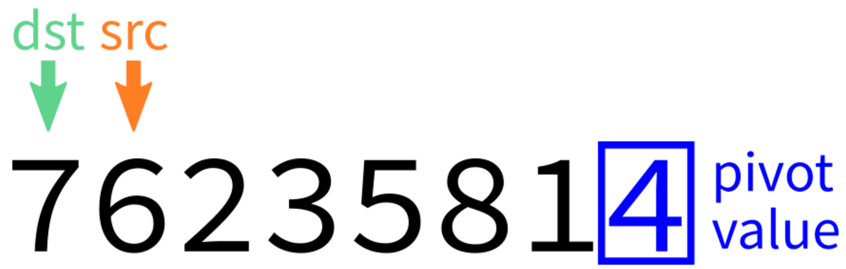
PARTITION VISUALIZATION

dst src

7 6 2 3 5 8 1 $\boxed{4}$ pivot value

L[src] ≥ pivot: do nothing

PARTITION VISUALIZATION

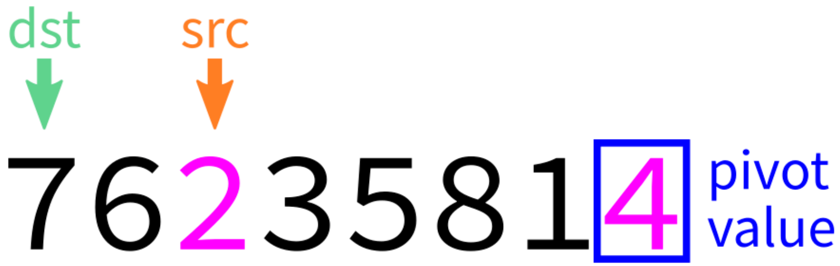


PARTITION VISUALIZATION

dst src

7623581$\boxed{4}$ pivot value

PARTITION VISUALIZATION



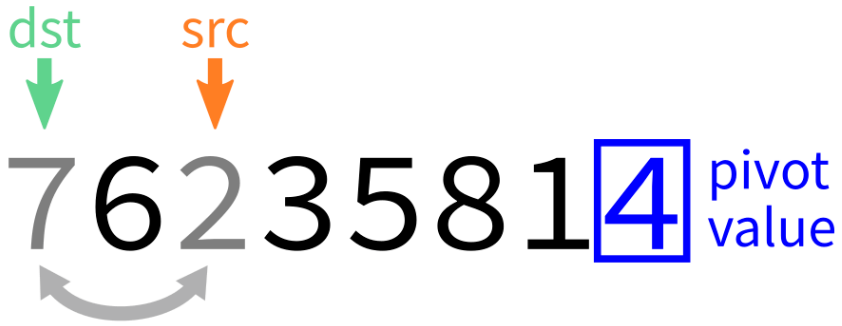
PARTITION VISUALIZATION

dst src

7 6 2 3 5 8 1 $\boxed{4}$ pivot value

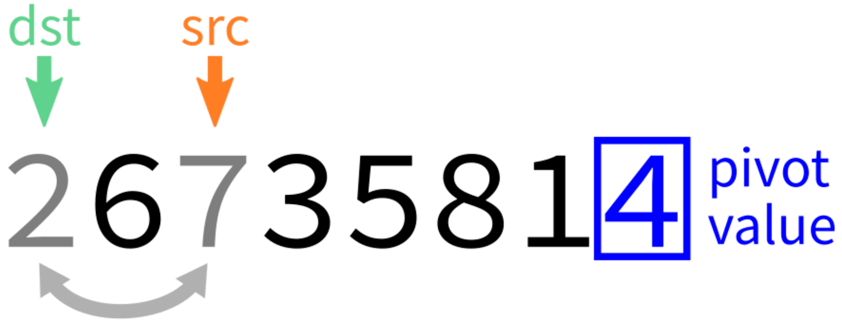
L[src] < pivot: **swap** L[src],L[dst]

PARTITION VISUALIZATION



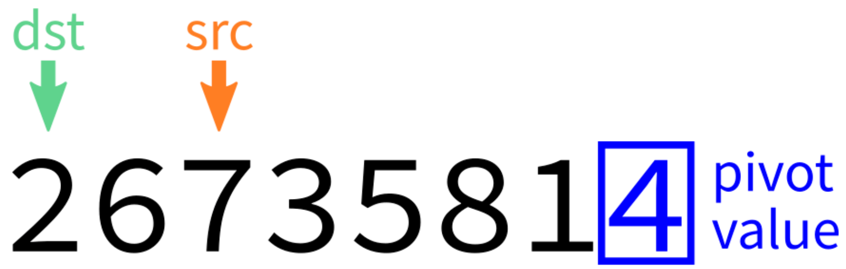
L[src] < pivot: **swap** L[src],L[dst]

PARTITION VISUALIZATION



L[src] < pivot: **swap** L[src],L[dst]

PARTITION VISUALIZATION



PARTITION VISUALIZATION

dst src

2673581$\boxed{4}$ pivot value

PARTITION VISUALIZATION

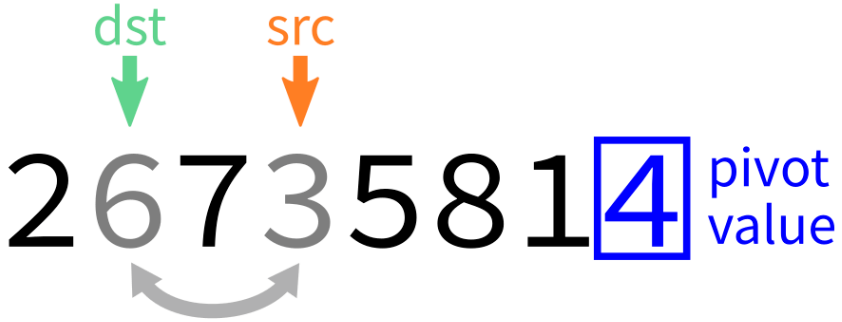
PARTITION VISUALIZATION

dst src

2 6 7 3 5 8 1 $\boxed{4}$ pivot value

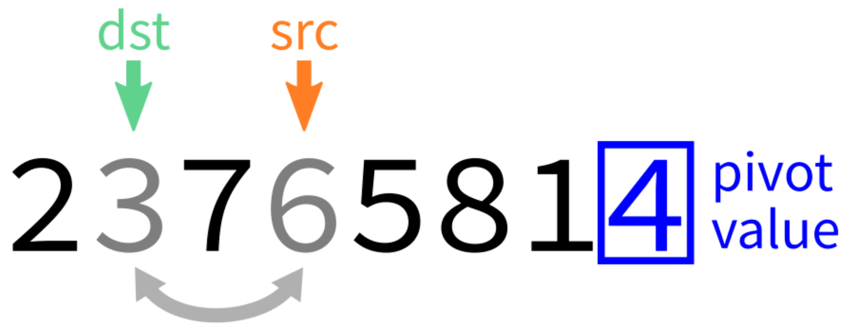
L[src] < pivot: **swap** L[src],L[dst]

PARTITION VISUALIZATION



L[src] < pivot: **swap** L[src],L[dst]

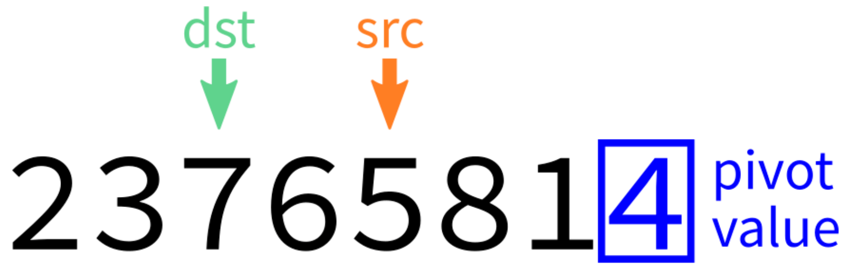
PARTITION VISUALIZATION



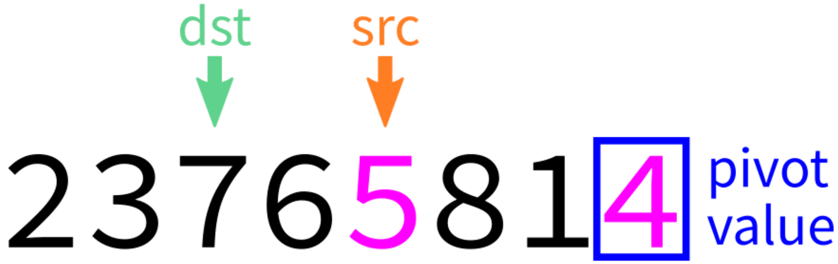
L[src] < pivot: **swap** L[src],L[dst]

PARTITION VISUALIZATION

PARTITION VISUALIZATION



PARTITION VISUALIZATION



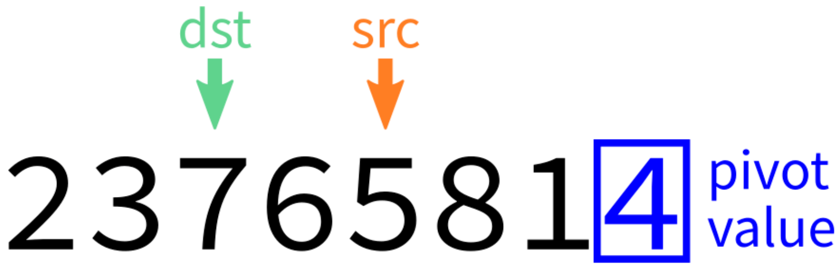
PARTITION VISUALIZATION

dst src

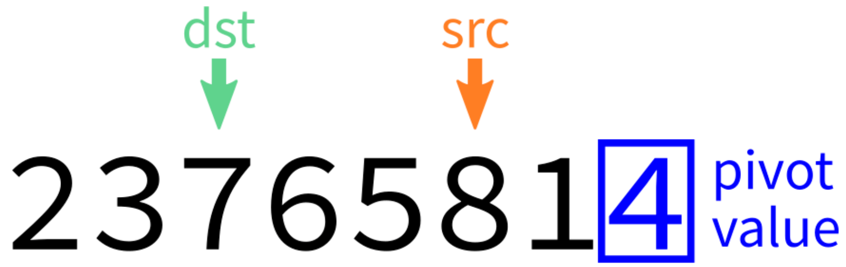
2 3 7 6 5 8 1 $\boxed{4}$ pivot value

L[src] ≥ pivot: do nothing

PARTITION VISUALIZATION



PARTITION VISUALIZATION

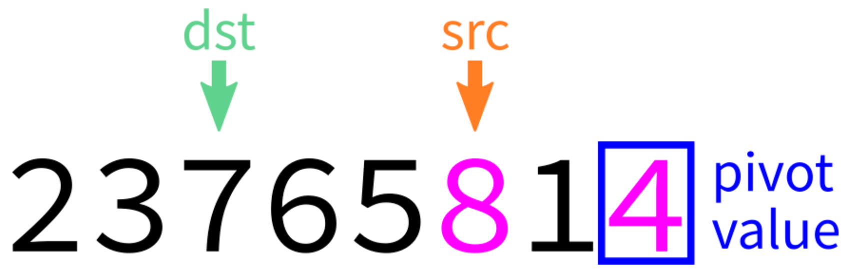


PARTITION VISUALIZATION

dst src

2 3 7 6 5 8 1 $\boxed{4}$ pivot value

PARTITION VISUALIZATION



L[src] ≥ pivot: do nothing

PARTITION VISUALIZATION

dst src

2376581$\boxed{4}$ pivot value

PARTITION VISUALIZATION

PARTITION VISUALIZATION

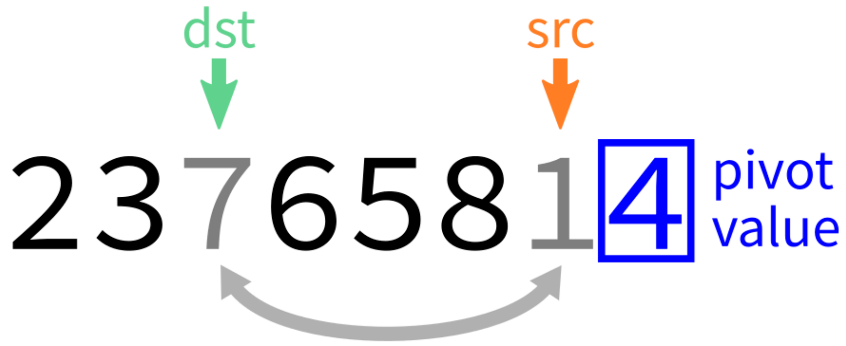
PARTITION VISUALIZATION

dst src

2 3 7 6 5 8 1 $\boxed{4}$ pivot value

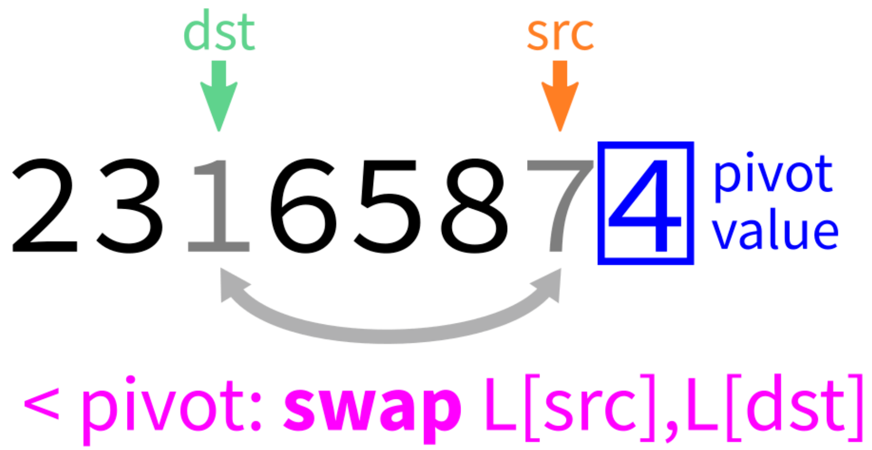
L[src] < pivot: **swap** L[src],L[dst]

PARTITION VISUALIZATION



L[src] < pivot: **swap** L[src],L[dst]

PARTITION VISUALIZATION



PARTITION VISUALIZATION

dst src

2 3 1 6 5 8 7 $\boxed{4}$ pivot value

PARTITION VISUALIZATION

PARTITION VISUALIZATION

PARTITION VISUALIZATION

L[src] ≥ pivot: do nothing

PARTITION VISUALIZATION

dst

2316587 $\boxed{4}$ pivot value

PARTITION VISUALIZATION

dst

2 3 1 6 5 8 7 $\boxed{4}$ pivot value

swap L[end-1] and L[dst]

PARTITION VISUALIZATION

dst

2 3 1 $\boxed{4}$ 5 8 7 6

swap L[end-1] and L[dst]

PARTITION VISUALIZATION

$2\ 3\ 1\ \boxed{4}\ 5\ 8\ 7\ 6$

PARTITION VISUALIZATION

$$\underbrace{2\ 3\ 1}\ \boxed{4}\ \underbrace{5\ 8\ 7\ 6}$$

PARTITION VISUALIZATION

$$\underbrace{2\,3\,1}_{<4}\,\boxed{4}\,\underbrace{5\,8\,7\,6}_{\geq 4}$$

REFERENCES

- You can refer to the general references about recursion that have appeared in several recent lectures. The rest of this list is specific to mergesort and quicksort.
- 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

- 2021-02-22 Moved unused slides to Lecture 18 and fix partition visualization
- 2021-02-17 Fix description of partition step in quicksort preview
- 2021-02-17 Initial publication