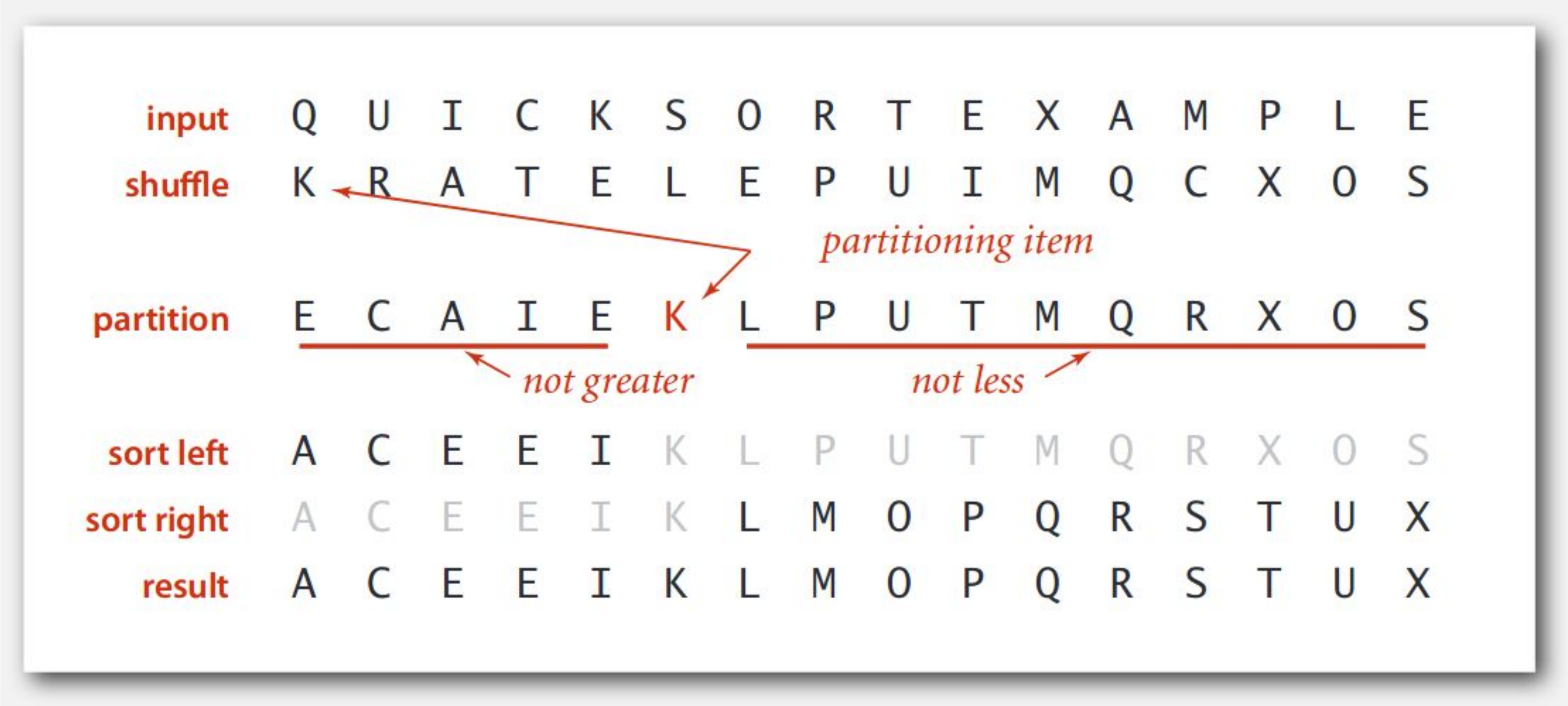
Quick Sort

Motivation. Merge sort uses auxiliary array.
Can we do O(n log(n)) sort in-place?

Quick sort. Idea

- Shuffle the array
- Partition so that, for some `j`:
    - Entry `a[j]` is in place
    - No larger entry to the left of `j`
    - No smaller entry to the right of `j`
- Sort each piece recursively

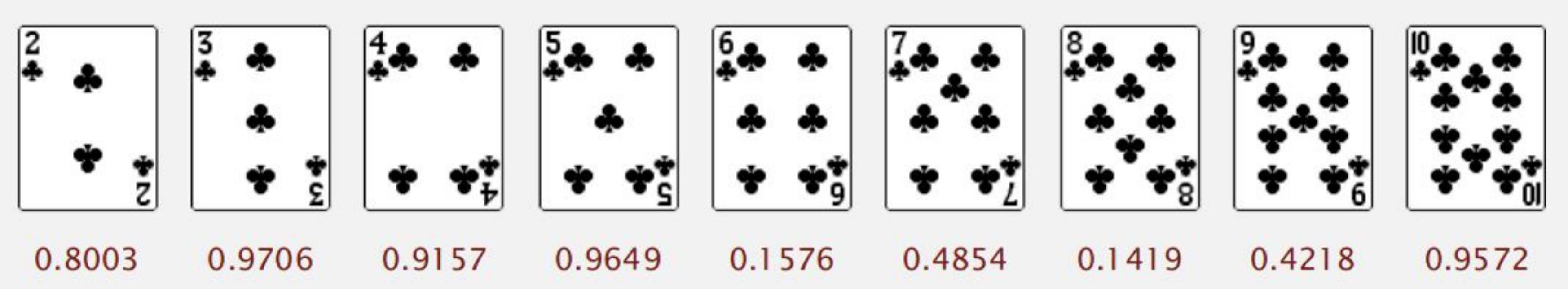
Quick sort. Idea



input	Q	U	I	C	K	S	O	R	T	E	X	A	M	P	L	E
shuffle	K	R	A	T	E	L	E	P	U	I	M	Q	C	X	O	S
partition	E	C	A	I	E	K	L	P	U	T	M	Q	R	X	O	S
sort left	A	C	E	E	I	K	L	P	U	T	M	Q	R	X	O	S
sort right	A	C	E	E	I	K	L	M	O	P	Q	R	S	T	U	X
result	A	C	E	E	I	K	L	M	O	P	Q	R	S	T	U	X

How to shuffle array?

- Shuffle sort
  - Generate a random real number for each array entry
  - Sort the array



How to shuffle array?

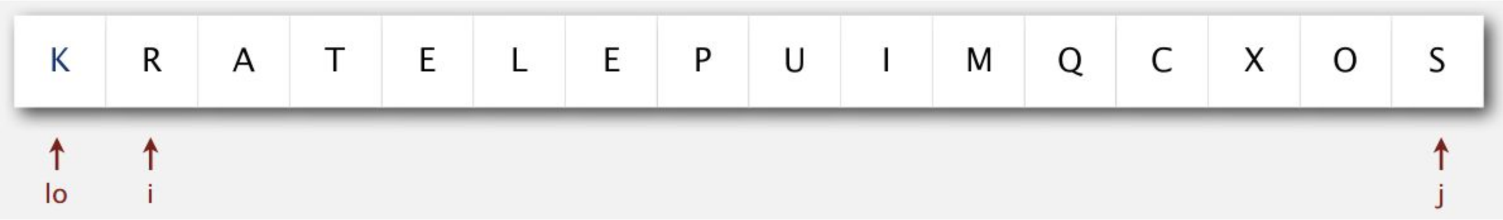
- Knuth shuffle:
  - In iteration `i`, pick integer `r` between `0` and `i` uniformly at random
  - Swap `a[i]` and `a[r]`

```
import random

def shuffle(arr):
   for i in range(len(arr)):
       r = random.randint(0, i)
       arr[r], arr[i] = arr[i], arr[r]
```

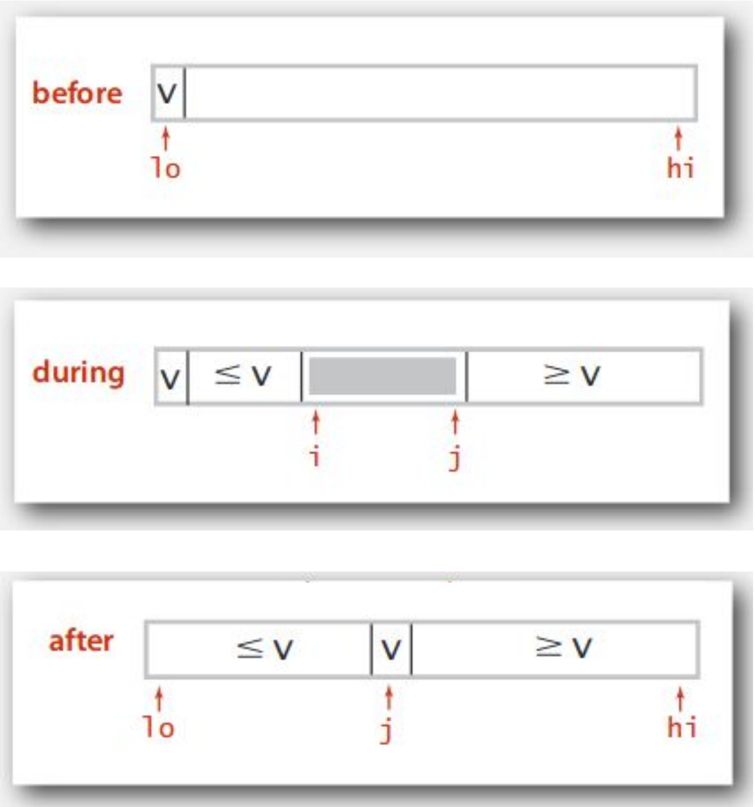
Quick sort partitioning

- Repeat until `i` and `j` pointers cross:
  - Scan `i` from left to right so long as (`a[i] < a[lo]`)
  - Scan `j` from right to left so long as (`a[j] > a[lo]`)
  - Exchange `a[i]` with `a[j]`
- When pointers cross:
  - Exchange `a[lo]` with `a[j]`



Quick sort partition() implementation

```
def partition(arr, lo, hi):
    i = lo + 1
    j = hi
    pivot = arr[lo]
    while True:
        while arr[i] <= pivot:
            if i == hi:
                break
            i += 1
        while arr[j] >= pivot:
            if j == lo:
                break
            j -= 1
        if i >= j:
            break
        arr[i], arr[j] = arr[j], arr[i]
    arr[lo], arr[j] = arr[j], arr[lo]
    return j
```



Quick sort implementation

```
def sort_req(arr, lo, hi):
   if hi <= lo:
       return
   j = partition(arr, lo, hi)
   sort_req(arr, lo, j - 1)
   sort_req(arr, j + 1, hi)


def quick_sort(arr):
   shuffle(arr)
   sort_req(arr, 0, len(arr) - 1)
```

Quick sort. Performance

- Worst-case: $O(n^2)$
- Best-case: $O(n \log(n))$
- On average: $O(n \log(n))$

Quick sort. Important characteristics

- One of the biggest advantage over merge sort is that quick sort doesn't take extra space. The sort is done in-place
- Number of comparisons is greater than in merge sort, but quick sort is faster in practice because of less data movements
- Probabilistic guarantee against worst case
- Quick sort is not stable

Quick sort implementation improvements

- Use insertion sort for small subarrays:
    - Quick sort has too much overhead for tiny subarrays
    - Cutoff to insertion sort for ~10 items
- Use median of sample:
    - Best choice of pivot item = median
    - Estimate true median by taking median of sample
    - Median of 3 random items