CS062

DATA STRUCTURES AND ADVANCED PROGRAMMING

19: Quicksort

Alexandra Papoutsaki
LECTURES

Mark Kampe
LABS

Lecture 19: Quicksort

- Quicksort

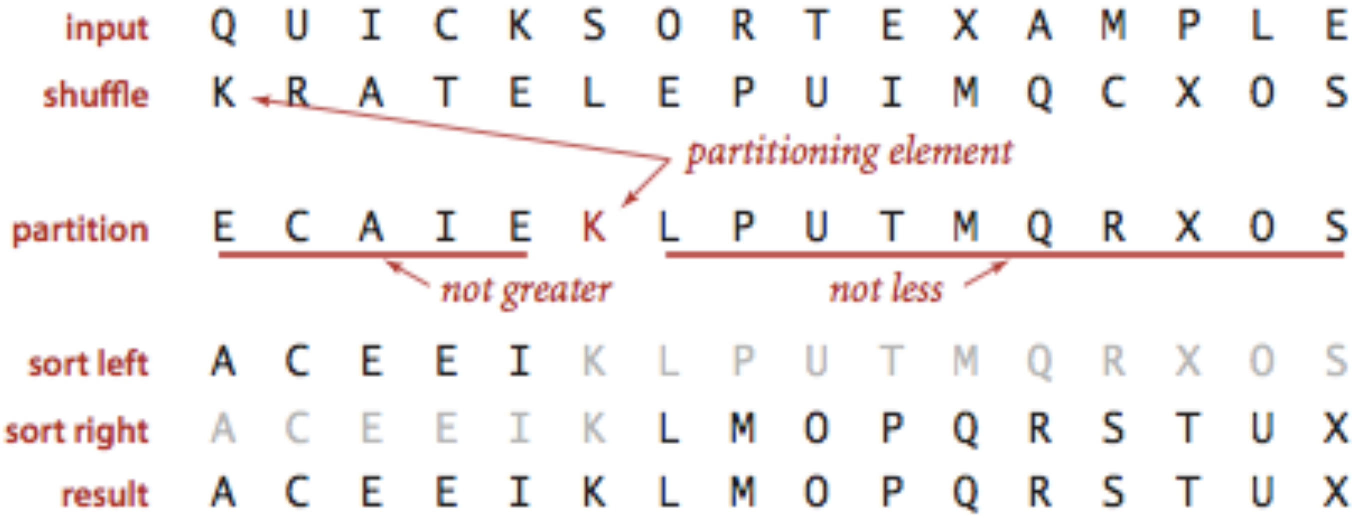
Some slides adopted from Algorithms 4th Edition or COS226

Mergesort and Quicksort: the classics

- Mergesort used in Java to sort objects.
- Quicksort used in Java to sort primitives.
- Quicksort was invented by Sir Tony Hoare in 1959.
  - Wanted to sort Russian words before looking them up in dictionary.
  - Came up with quicksort but did not know how to implement it.
  - Learned Algol 60 and recursion and implemented it.
  - Won the 1980 Turing Award.
- Bob Sedgewick (author of your textbook) refined and analyzed many versions of quicksort.

Basic Plan

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



Quicksort overview

Partition

- Partition the subarray `a[lo…hi]` so that `a[lo…j-1]<=a[j]<=a[j+1…hi]`
- Start with pointer `i` at `lo` and pointer `j` at `hi+1`.
- Repeat the following until pointers `i` and `j` cross:
  - Scan `i` from left to right as long as `a[i]<a[lo]`.
  - Scan `j` from right to left as long as `a[j]>a[lo]`.
  - Exchange `a[i]` with `a[j]`.
- Exchange `[lo]` and `a[j]`. Return `j`.

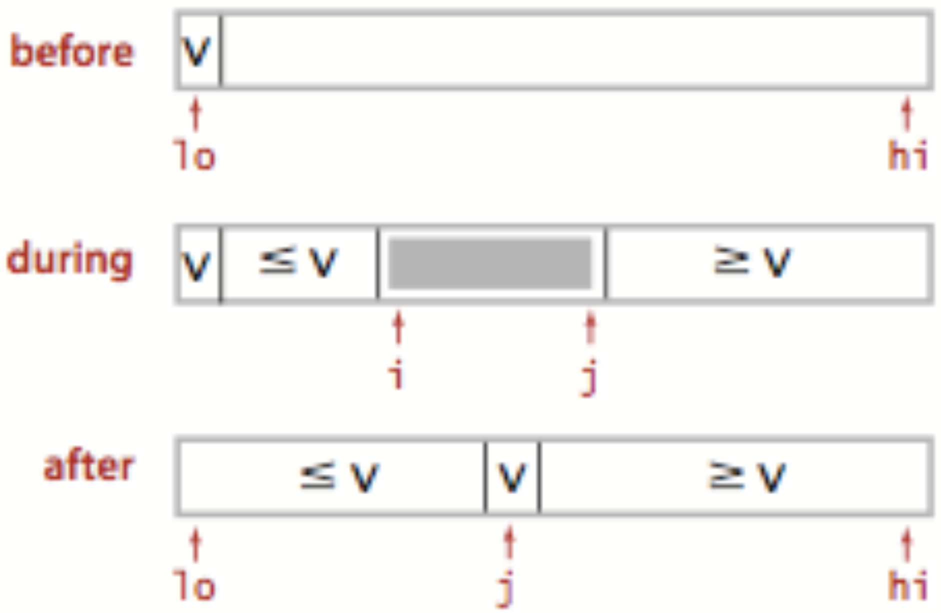
Partition Example

	i	j	a[0] (v)	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
initial values	0	16	K	R	A	T	E	L	E	P	U	I	M	Q	C	X	O	S
scan left, scan right	1	12	K	R	A	T	E	L	E	P	U	I	M	Q	C	X	O	S
exchange	1	12	K	C	A	T	E	L	E	P	U	I	M	Q	R	X	O	S
scan left, scan right	3	9	K	C	A	T	E	L	E	P	U	I	M	Q	R	X	O	S
exchange	3	9	K	C	A	I	E	L	E	P	U	T	M	Q	R	X	O	S
scan left, scan right	5	6	K	C	A	I	E	L	E	P	U	T	M	Q	R	X	O	S
exchange	5	6	K	C	A	I	E	E	L	P	U	T	M	Q	R	X	O	S
scan left, scan right	6	5	K	C	A	I	E	E	L	P	U	T	M	Q	R	X	O	S
final exchange	6	5	E	C	A	I	E	K	L	P	U	T	M	Q	R	X	O	S
result		5	E	C	A	I	E	K	L	P	U	T	M	Q	R	X	O	S

Partitioning trace (array contents before and after each exchange)

Partition Code

```
// partition the subarray a[lo..hi] so that a[lo..j-1] <= a[j] <= a[j+1..hi]
// and return the index j.
private static int partition(Comparable[] a, int lo, int hi) {
    int i = lo;
    int j = hi + 1;
    Comparable v = a[lo];
    while (true) {

        // find item on lo to swap
        while (less(a[++i], v)) {
            if (i == hi) break;
        }

        // find item on hi to swap
        while (less(v, a[--j])) {
            if (j == lo) break;      // redundant since a[lo] acts as sentinel
        }

        // check if pointers cross
        if (i >= j) break;

        exch(a, i, j);
    }

    // put partitioning item v at a[j]
    exch(a, lo, j);

    // now, a[lo .. j-1] <= a[j] <= a[j+1 .. hi]
    return j;
}
```



Quicksort partitioning overview

Quicksort Code

```java
/**
 * Rearranges the array in ascending order, using the natural order.
 * @param a the array to be sorted
 */
public static void sort(Comparable[] a) {
    StdRandom.shuffle(a);
    sort(a, 0, a.length - 1);
}

// quicksort the subarray from a[lo] to a[hi]
private static void sort(Comparable[] a, int lo, int hi) {
    if (hi <= lo) return;
    int j = partition(a, lo, hi);
    sort(a, lo, j-1);
    sort(a, j+1, hi);
}
```

Quicksort Demo

	lo	j	hi	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
initial values				Q	U	I	C	K	S	O	R	T	E	X	A	M	P	L	E
random shuffle				K	R	A	T	E	L	E	P	U	I	M	Q	C	X	O	S
	0	5	15	E	C	A	I	E	K	L	P	U	T	M	Q	R	X	O	S
	0	3	4	E	C	A	E	I	K	L	P	U	T	M	Q	R	X	O	S
	0	2	2	A	C	E	E	I	K	L	P	U	T	M	Q	R	X	O	S
	0	0	1	A	C	E	E	I	K	L	P	U	T	M	Q	R	X	O	S
	1		1	A	C	E	E	I	K	L	P	U	T	M	Q	R	X	O	S
	4		4	A	C	E	E	I	K	L	P	U	T	M	Q	R	X	O	S
	6	6	15	A	C	E	E	I	K	L	P	U	T	M	Q	R	X	O	S
	7	9	15	A	C	E	E	I	K	L	M	O	P	T	Q	R	X	U	S
	7	7	8	A	C	E	E	I	K	L	M	O	P	T	Q	R	X	U	S
	8		8	A	C	E	E	I	K	L	M	O	P	T	Q	R	X	U	S
	10	13	15	A	C	E	E	I	K	L	M	O	P	S	Q	R	T	U	X
	10	12	12	A	C	E	E	I	K	L	M	O	P	R	Q	S	T	U	X
	10	11	11	A	C	E	E	I	K	L	M	O	P	Q	R	S	T	U	X
	10		10	A	C	E	E	I	K	L	M	O	P	Q	R	S	T	U	X
	14	14	15	A	C	E	E	I	K	L	M	O	P	Q	R	S	T	U	X
	15		15	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

no partition for subarrays of size 1

Quicksort Considerations

- Partitioning in-place:Using an extra array makes partitioning easier (and stable), but it is not worth the cost.
- Terminating the loop: Testing whether the pointers cross is trickier than it might seem.
- Equal keys: When duplicate keys are present, it is (counter-intuitively) better to stop scans on keys equal to the partitioning item's key.
- Preserving randomness: Shuffling is needed for performance guarantee.
  - Equivalent alternative: Pick a random partitioning item in each subarray.

Great algorithms are better than good ones

- Your laptop executes 10^8 comparisons per second
- A supercomputer executes 10^{12} comparisons per second

	Insertion sort			Mergesort			Quicksort		
Computer	**Thousand inputs**	**Million inputs**	**Billion inputs**	**Thousand inputs**	**Million inputs**	**Billion inputs**	**Thousand inputs**	**Million inputs**	**Billion inputs**
Home	Instant	2 hours	300 years	instant	1 sec	15 min	Instant	0.5 sec	10 min
Supercomputer	Instant	1 second	1 week	instant	instant	instant	instant	instant	Instant

Quicksort analysis: best case

- Quicksort divides everything exactly in half.
- Similar to merge sort
- Number of compares is $\sim n \log n$

Quicksort analysis: worst case

- Data are already sorted.
- Number of compares is ~$1/2n^2$ - quadratic!
- Extremely unlikely if we shuffle and our shuffling is not broken.

Quicksort - things to remember

- $\sim 2n \ln n$ or $1.39n \log n$ compares on average
    - 39% more compares than merge sort but in practice is faster because it does not move data much.
    - If good implementation, even in sorted arrays it can be linearithmic. If not, we end up with quadratic.
  - $1/3n \ln n$ exchanges.
  - We won’t do the analysis.
- In-place sorting.
- **Not** stable

Quicksort practical improvements

- Use insertion sort for small subarrays.
  - Too much overhead for tiny subarrays.
  - Cutoff to insertion sort usually around 10 items.
- Best choice of pivot is the median

Lecture 19: Quicksort

- Quicksort

Readings:

- Textbook:
  - Chapter 2.3 (Pages 288-296)
- Website:
  - Quicksort: https://algs4.cs.princeton.edu/23quicksort/
  - Code: https://algs4.cs.princeton.edu/23quicksort/Quick.java.html

Practice Problems:

- 2.3.1-2.3.4