Sorting algorithms

Johan Montelius

KTH

HT22

selection sort

- an $O(n^2)$ algorithm
- few write operations
- useful when iteratively searching for the smallest elements

selection sort

- an $O(n^2)$ algorithm
- few write operations
- useful when iteratively searching for the smallest elements

selection sort

- an $O(n^2)$ algorithm
- few write operations
- useful when iteratively searching for the smallest elements

selection sort

- an $O(n^2)$ algorithm
- few write operations
- useful when iteratively searching for the smallest elements

insertion sort

- an $O(n^2)$ algorithm
- many write operations (in an array)
- useful when data set almost sorted
- always used when inserting a new element in a sorted data set

insertion sort

- an $O(n^2)$ algorithm
- many write operations (in an array)
- useful when data set almost sorted
- always used when inserting a new element in a sorted data set

insertion sort

- an $O(n^2)$ algorithm
- many write operations (in an array)
- useful when data set almost sorted
- always used when inserting a new element in a sorted data set

insertion sort

- an $O(n^2)$ algorithm
- many write operations (in an array)
- useful when data set almost sorted
- always used when inserting a new element in a sorted data set

insertion sort

- an $O(n^2)$ algorithm
- many write operations (in an array)
- useful when data set almost sorted
- always used when inserting a new element in a sorted data set

merge sort

```
private static void sort(int[] org, int[] aux, int lo, int
   if (lo != hi) {
     int mid = lo + (hi-lo)/2;
     sort(org, aux, lo, mid);
     sort(org, aux, mid+1, hi);
     merge(org, aux, lo, mid, hi);
    }
}
```

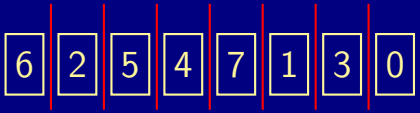
merge sort work

merge sort work

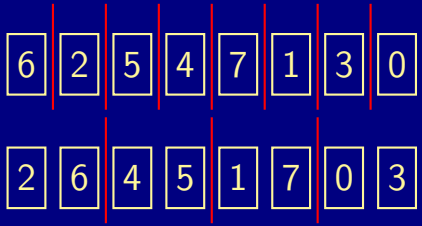
merge sort work

merge sort work

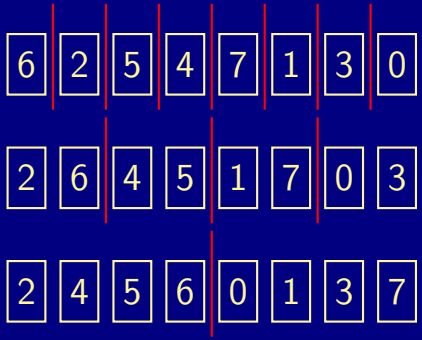
merge sort work



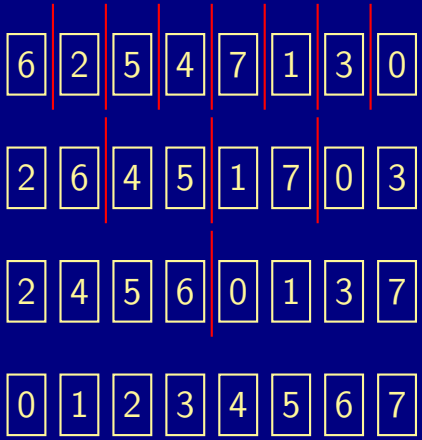
merge sort work



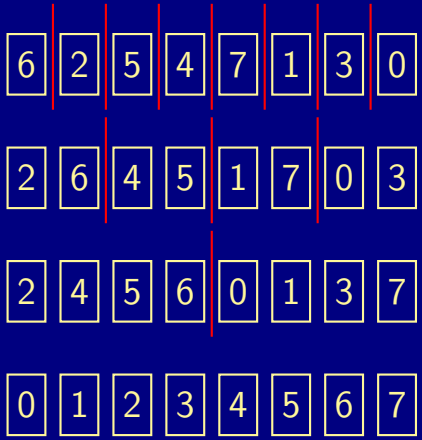
merge sort work



merge sort work

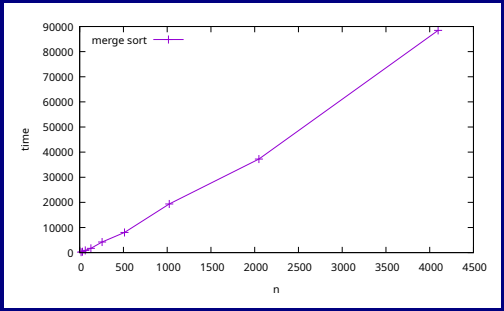


merge sort work

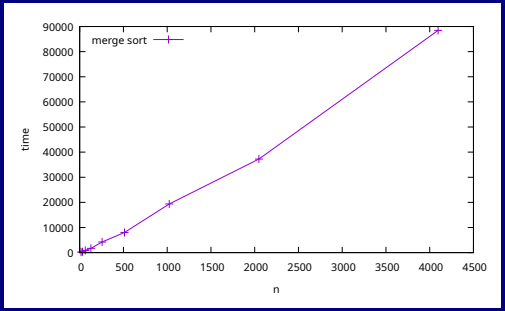


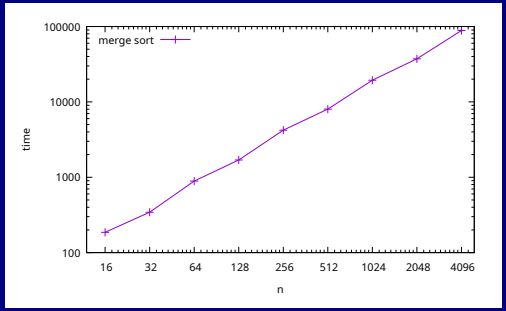
merge sort executes in $O(n \times lg(n))$

benchmark


merge sort
time
n
0
10000
20000
30000
40000
50000
60000
70000
80000
90000
0
500
1000
1500
2000
2500
3000
3500
4000
4500

benchmark


merge sort
time
n
0
10000
20000
30000
40000
50000
60000
70000
80000
90000
0
500
1000
1500
2000
2500
3000
3500
4000
4500


merge sort
time
n
100
1000
10000
100000
16
32
64
128
256
512
1024
2048
4096

benchmark

n	t	t/n
16	186	11.6
32	343	10.7
64	889	13.9
128	1695	13.2
256	4217	16.5
512	8018	15.7
1024	19358	18.9
2048	37242	18.2
4096	88452	21.6

benchmark

n	t	t/n	$t/(n \times log(n))$
16	186	11.6	4.2
32	343	10.7	3.1
64	889	13.9	3.3
128	1695	13.2	2.7
256	4217	16.5	3.0
512	8018	15.7	2.5
1024	19358	18.9	2.7
2048	37242	18.2	2.4
4096	88452	21.6	2.6

merge sort

merge sort

- an $O(n \times lg(n))$ algorithm

merge sort

- an $O(n \times lg(n))$ algorithm
- requires additional space (could be avoided but complicated)

merge sort

- an $O(n \times lg(n))$ algorithm
- requires additional space (could be avoided but complicated)
- robust i.e. no worst case scenarios

quick sort

Merge sort divides the array on the way down and sorts it on the way up.

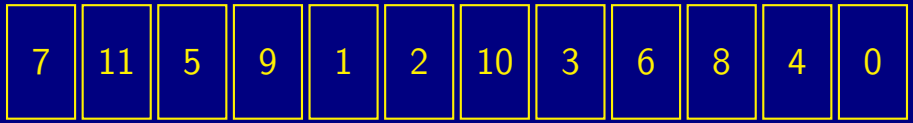
quick sort

Merge sort divides the array on the way down and sorts it on the way up.

Quick sort sorts the array on the way down and assemble it on the way up.

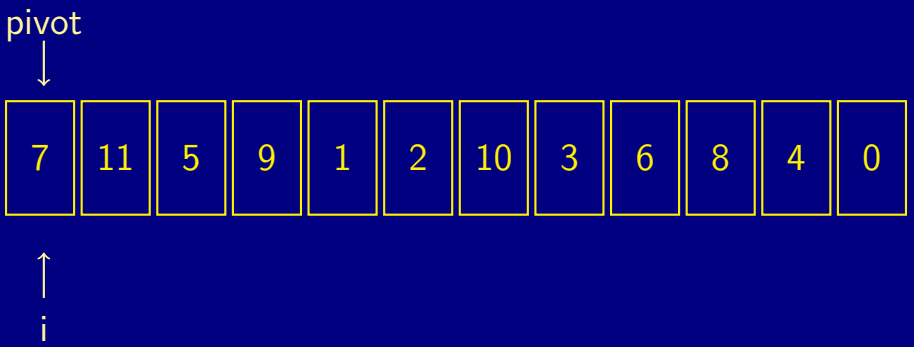
quick sort

quick sort

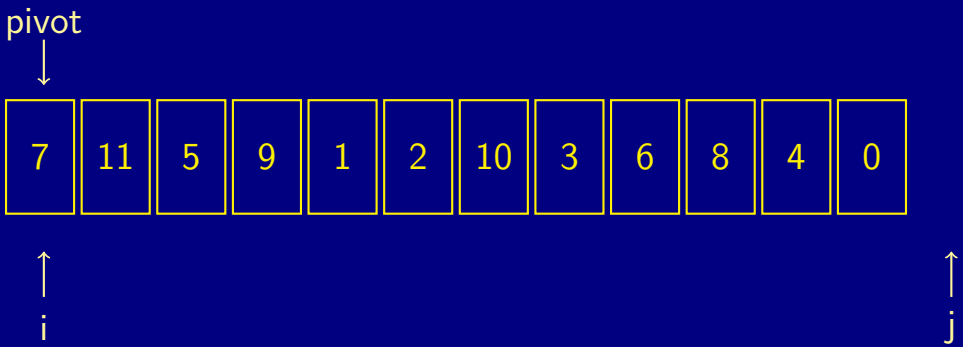


quick sort

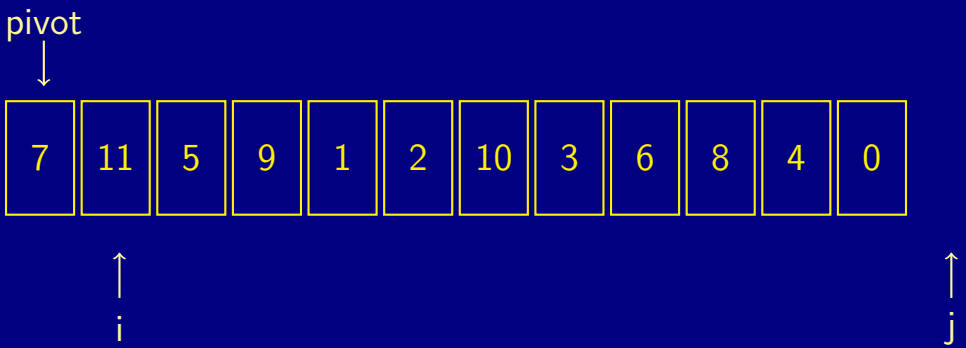
quick sort



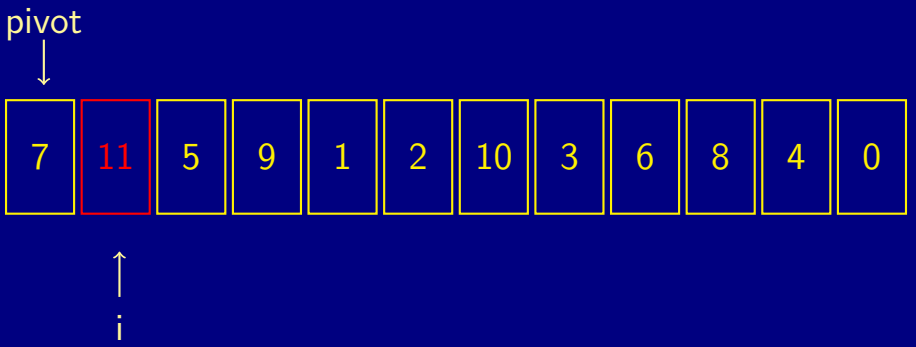
quick sort



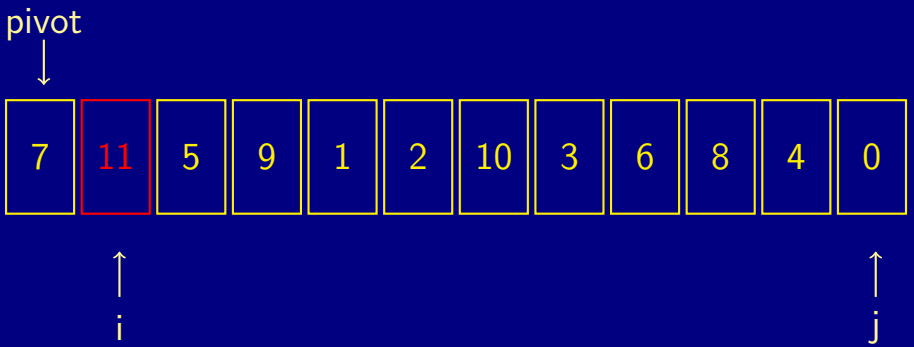
quick sort



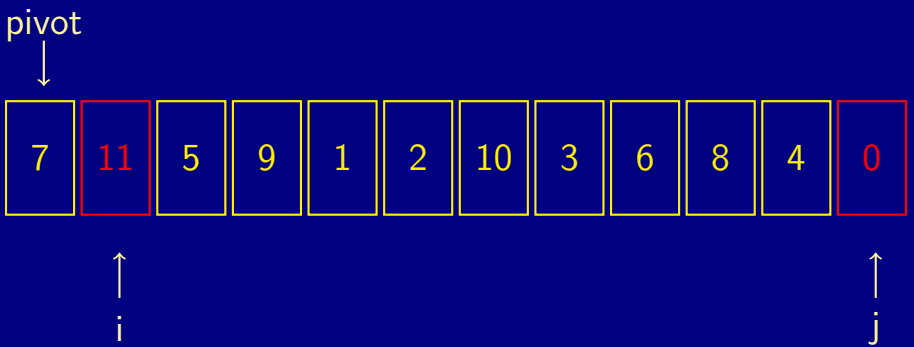
quick sort



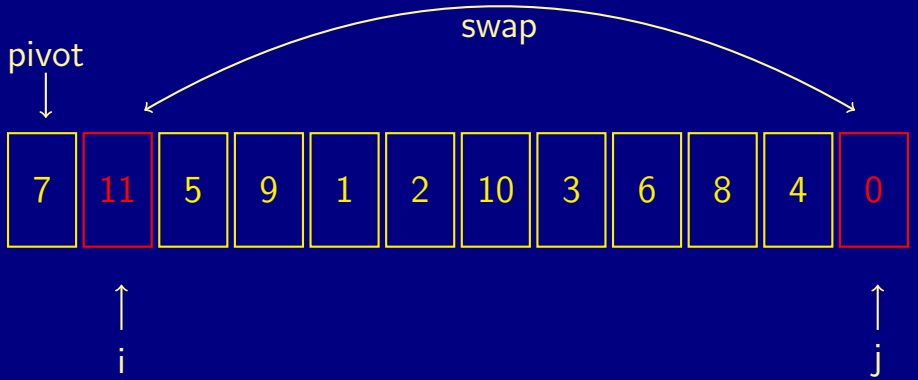
quick sort



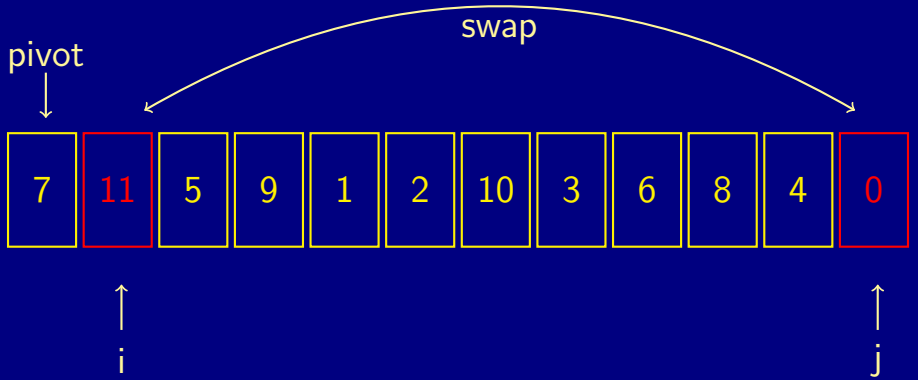
quick sort



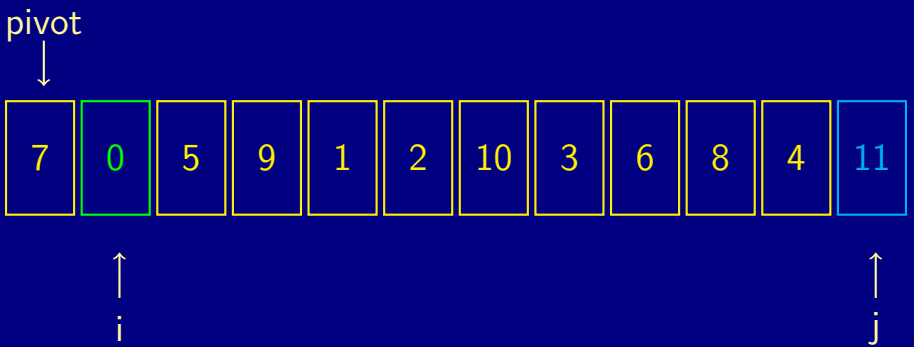
quick sort



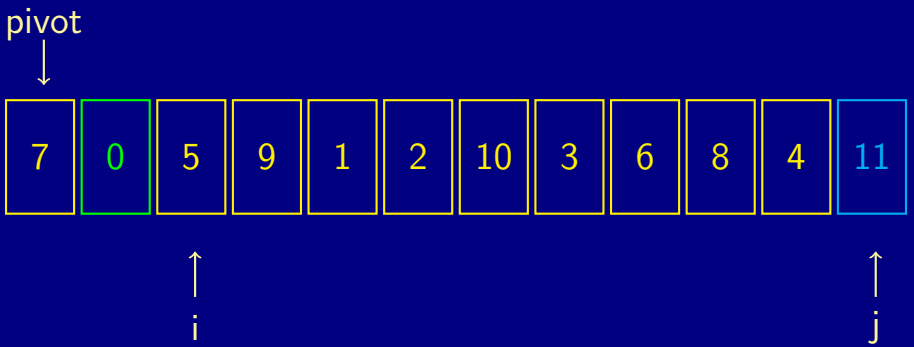
quick sort



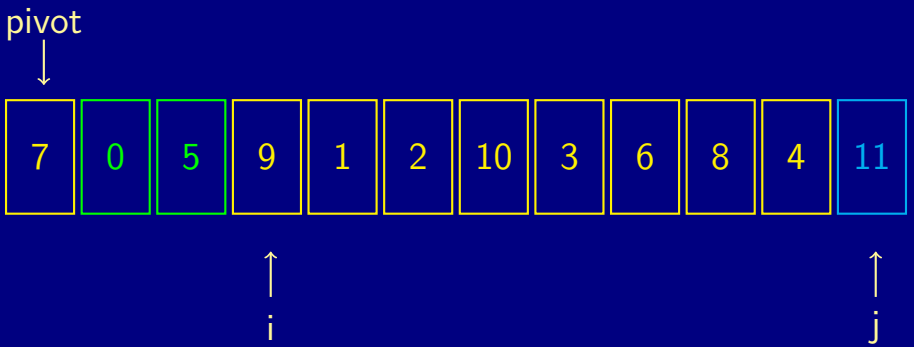
quick sort



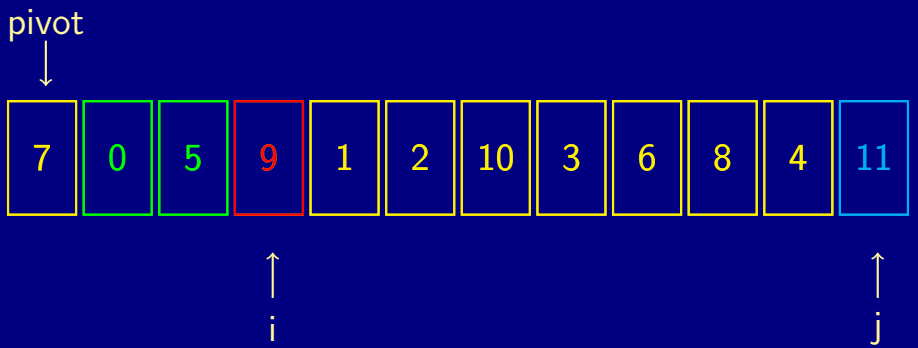
quick sort



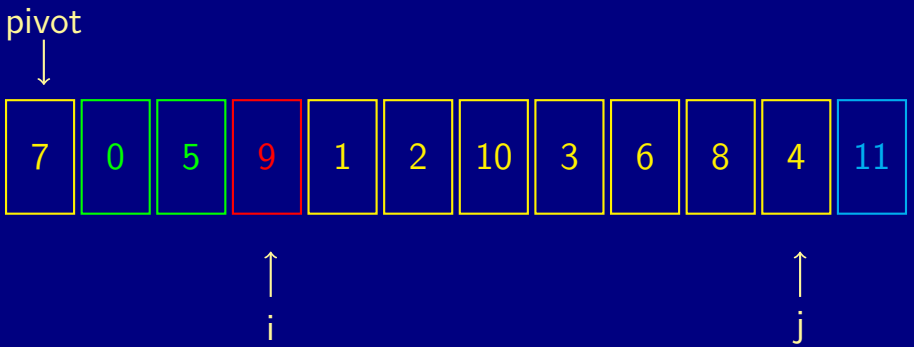
quick sort



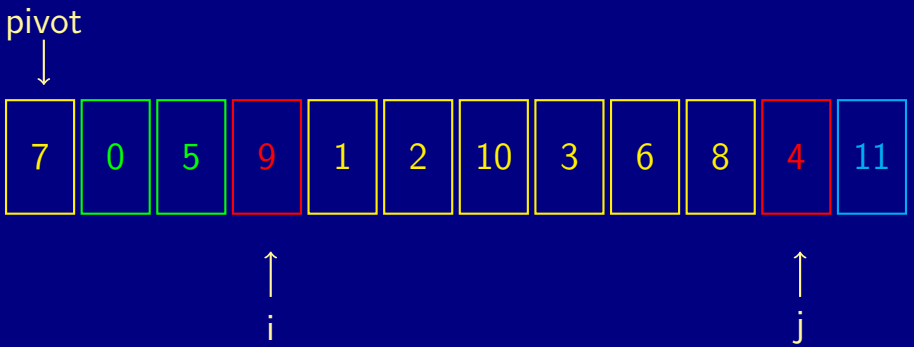
quick sort



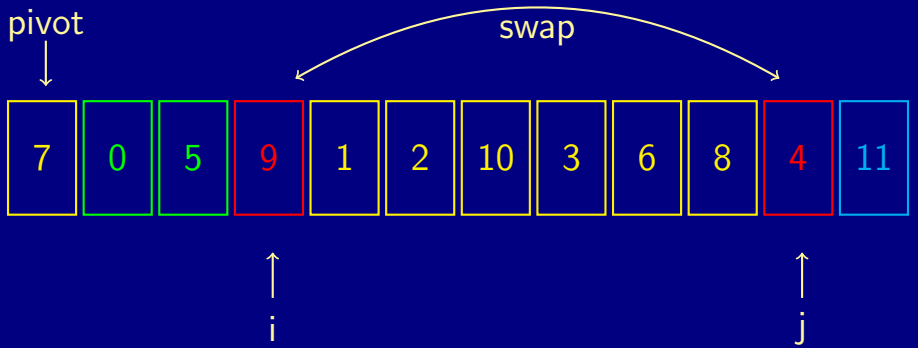
quick sort



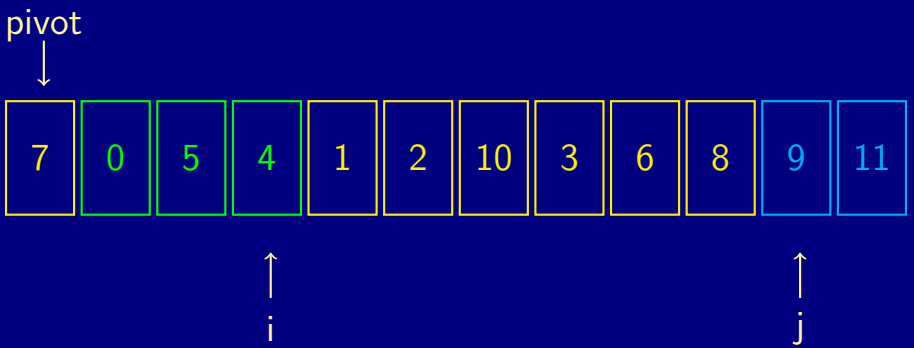
quick sort



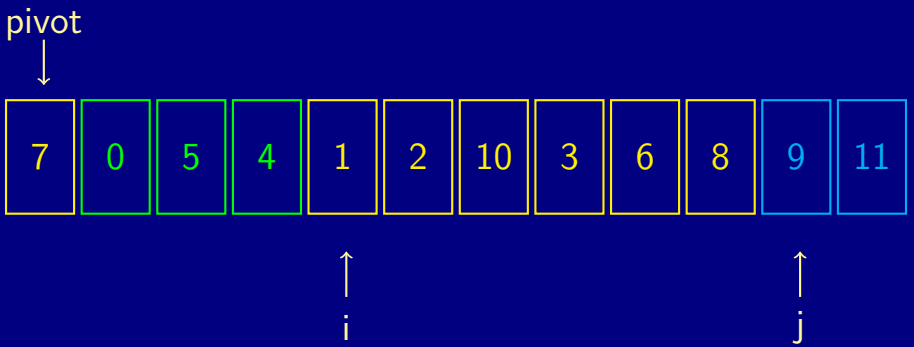
quick sort



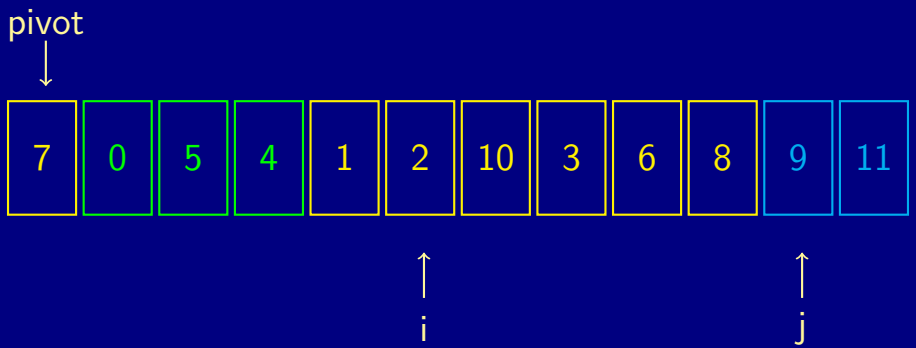
quick sort



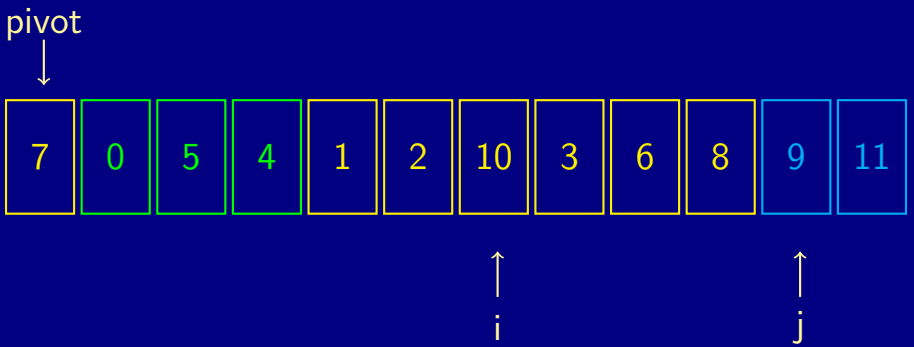
quick sort



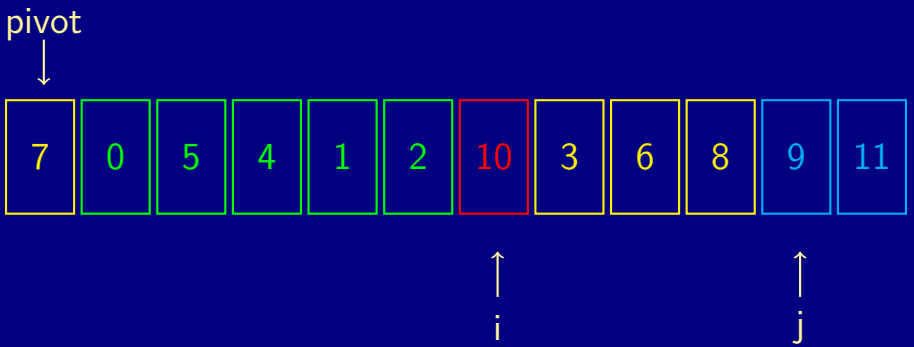
quick sort



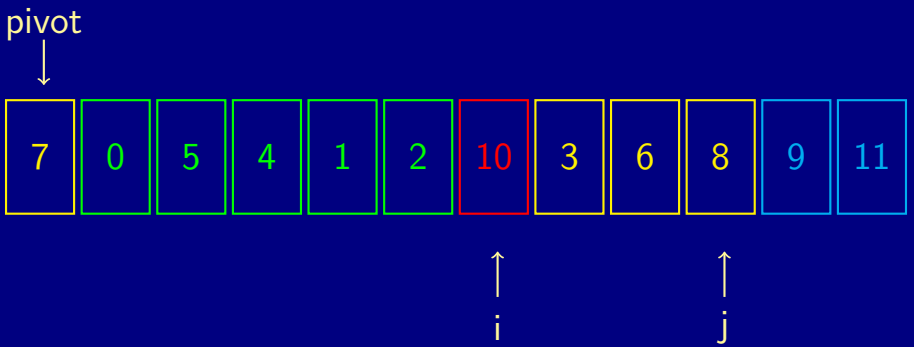
quick sort



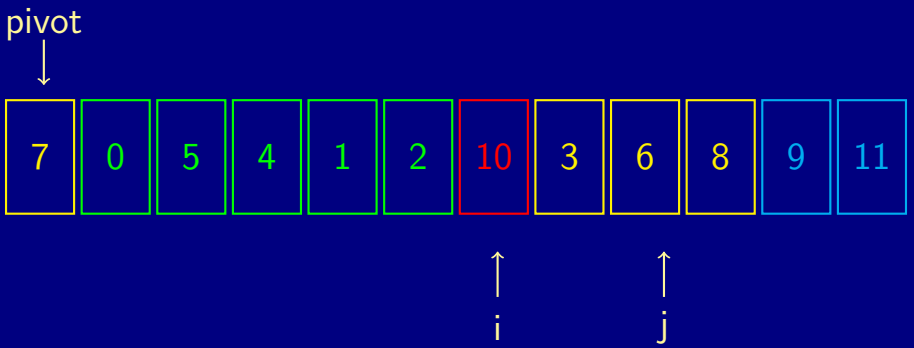
quick sort



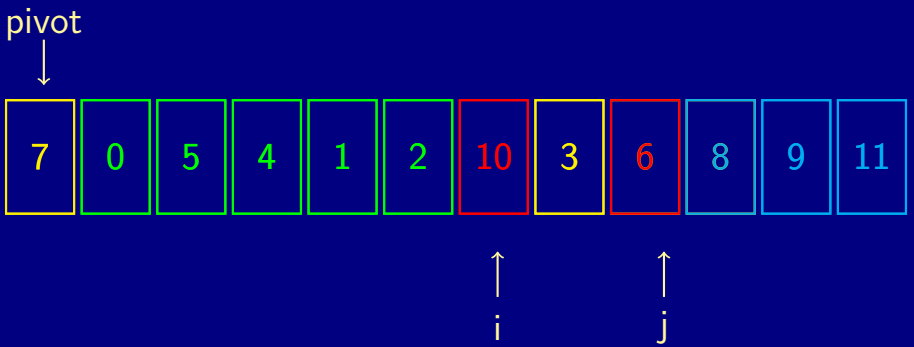
quick sort



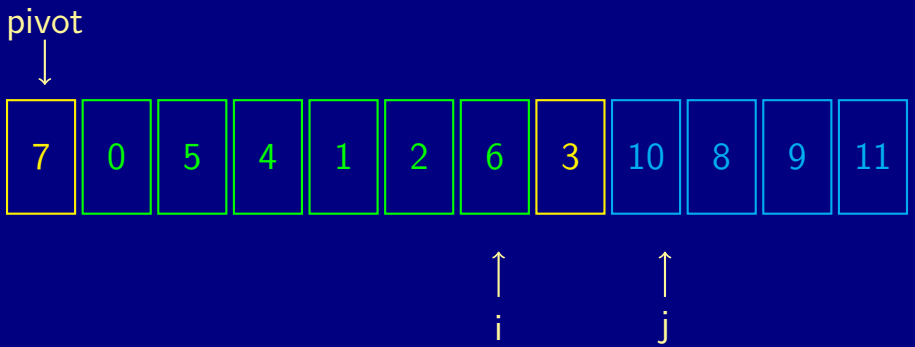
quick sort



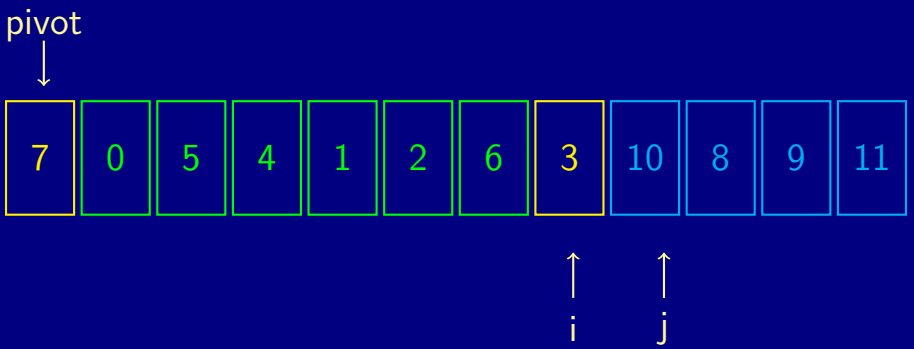
quick sort



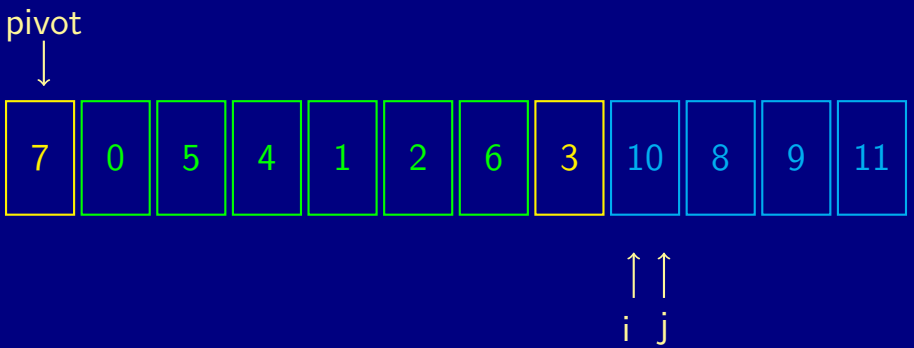
quick sort



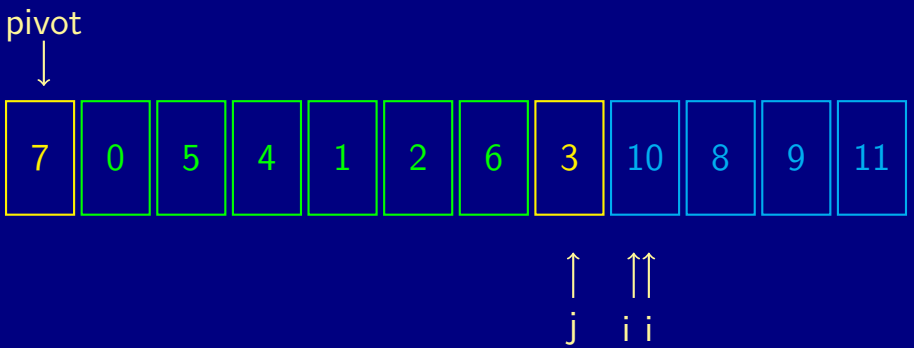
quick sort



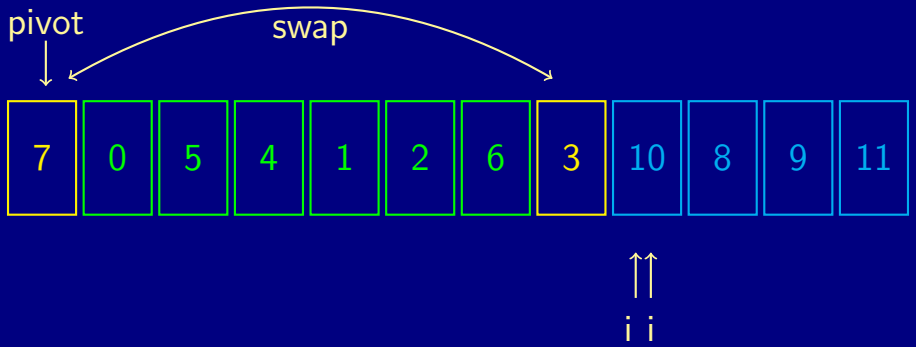
quick sort



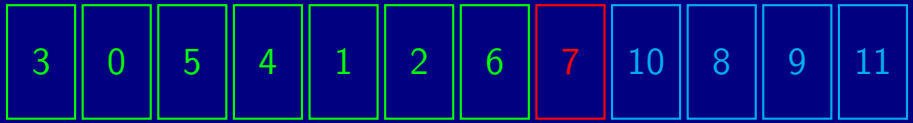
quick sort



quick sort



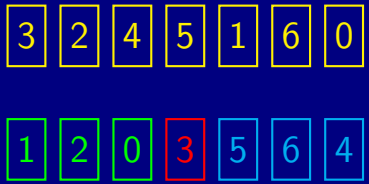
quick sort



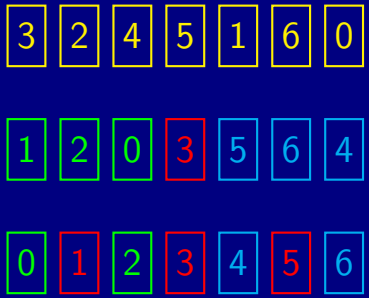
quick sort work

quick sort work

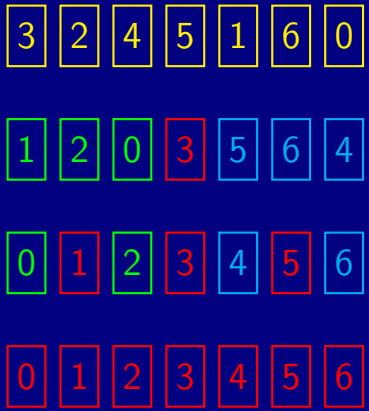
quick sort work



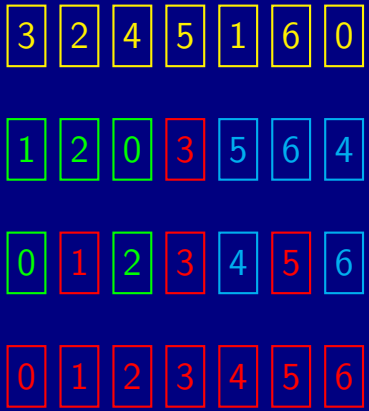
quick sort work



quick sort work



quick sort work



quick sort executes in $O(n \times lg(n))$

quick sort

quick sort

- an $O(n \times lg(n))$ algorithm

quick sort

- an $O(n \times lg(n))$ algorithm
- no additional space

quick sort

- an $O(n \times lg(n))$ algorithm
- no additional space
- worst case scenario if array is sorted - $O(n^2)$

stability

A sorting algorithm is *stable* if the given order is preserved when sorting two equal elements.

stability

Why stability matters:

Gustav Bengtsson
Adam Bengtsson
Fredrik Andersson

stability

Why stability matters:

Gustav Bengtsson
Adam Bengtsson
Fredrik Andersson

Adam Bengtsson
Fredrik Andersson
Gustav Bengtsson

stability

Why stability matters:

Gustav Bengtsson
Adam Bengtsson
Fredrik Andersson

Adam Bengtsson
Fredrik Andersson
Gustav Bengtsson

Fredrik Andersson
Adam Bengtsson
Gustav Bengtsson

stability

Why stability matters:

Gustav Bengtsson
Adam Bengtsson
Fredrik Andersson

Adam Bengtsson
Fredrik Andersson
Gustav Bengtsson

Fredrik Andersson
Gustav Bengtsson
Adam Bengtsson

sorting algorithms

stable algorithms

sorting algorithms

stable algorithms

- insertion sort

sorting algorithms

stable algorithms

- insertion sort
- merge sort

unstable algorithms

sorting algorithms

stable algorithms

- insertion sort
- merge sort

unstable algorithms

- selection sort

sorting algorithms

stable algorithms

- insertion sort
- merge sort

unstable algorithms

- selection sort
- quick sort