Algorithms

 \checkmark

ROBERT SEDGEWICK | KEVIN WAYNE

Algorithms

Robert Sedgewick | Kevin Wayne

http://algs4.cs.princeton.edu

2.3 PARTITIONING DEMOS

Sedgewick 2-way partitioning

- Dijkstra 3-way partitioning
- Bentley-McIlroy 3-way partitioning
- dual-pivot partitioning

2.3 PARTITIONING DEMOS

Bentley-McIlroy 3-way partitioning

Sedgewick 2-way partitioning

Dijkstra 3-way partitioning

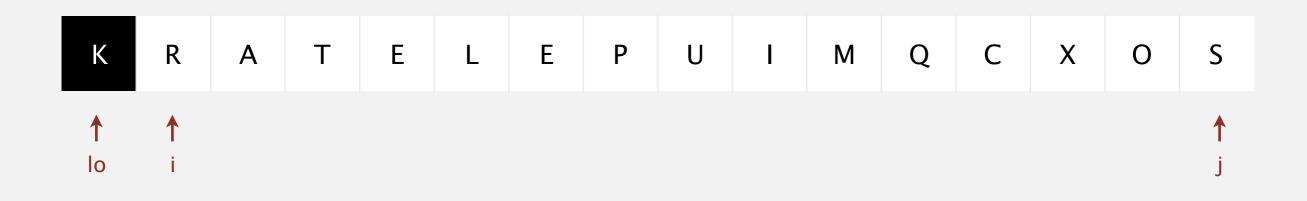
dual-pivot partitioning

Algorithms

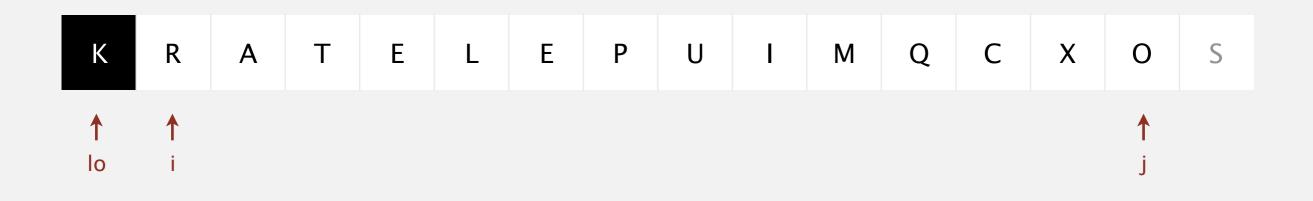
Robert Sedgewick | Kevin Wayne

http://algs4.cs.princeton.edu

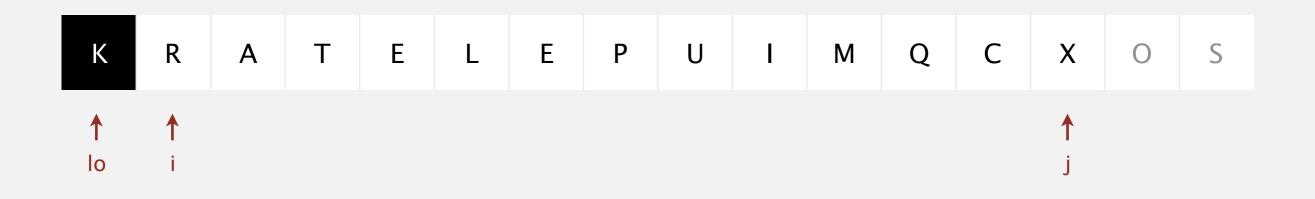
- Scan i from left to right so long as (a[i] < a[10]).
- Scan j from right to left so long as (a[j] > a[lo]).
- Exchange a[i] with a[j].



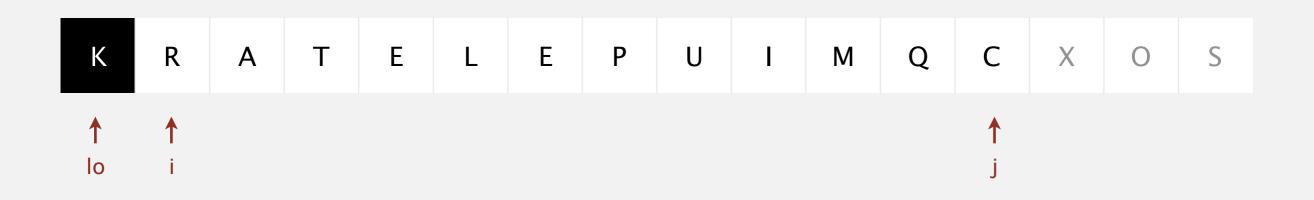
- Scan i from left to right so long as (a[i] < a[lo]).
- Scan j from right to left so long as (a[j] > a[lo]).
- Exchange a[i] with a[j].



- 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].

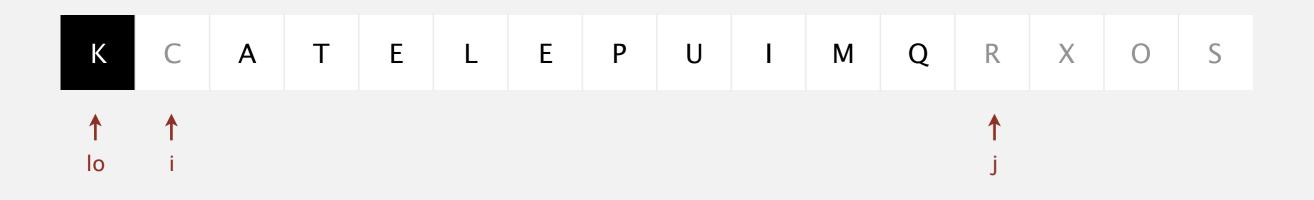


- Scan i from left to right so long as (a[i] < a[10]).
- Scan j from right to left so long as (a[j] > a[lo]).
- Exchange a[i] with a[j].

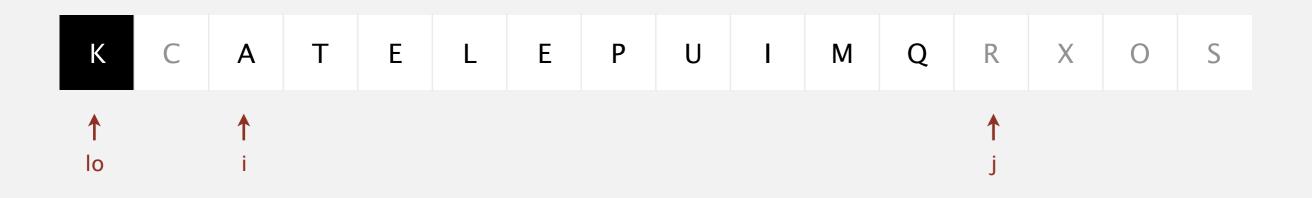


stop j scan and exchange a[i] with a[j]

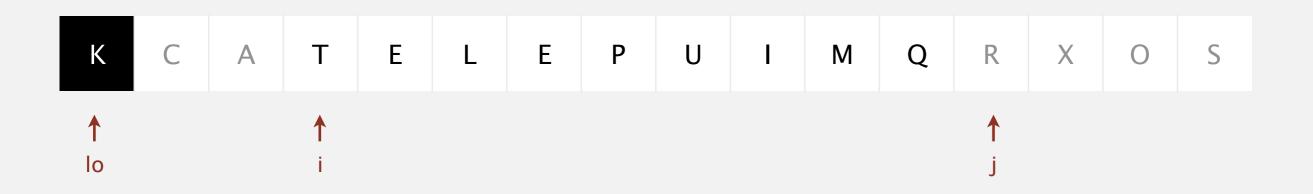
- Scan i from left to right so long as (a[i] < a[lo]).
- Scan j from right to left so long as (a[j] > a[lo]).
- Exchange a[i] with a[j].



- 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].

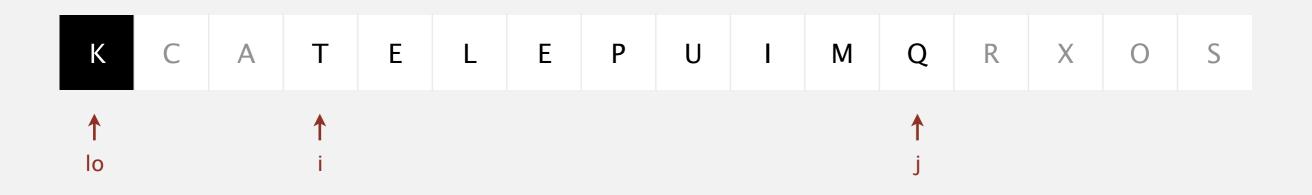


- Scan i from left to right so long as (a[i] < a[10]).
- Scan j from right to left so long as (a[j] > a[lo]).
- Exchange a[i] with a[j].

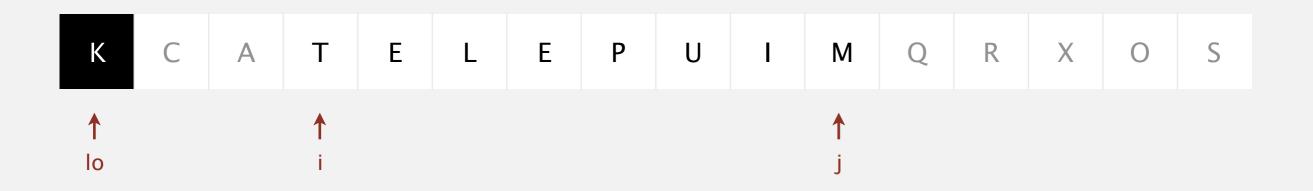


stop i scan because a[i] >= a[lo]

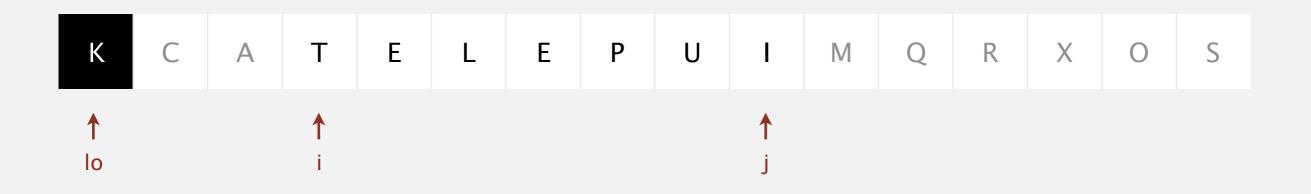
- Scan i from left to right so long as (a[i] < a[10]).
- Scan j from right to left so long as (a[j] > a[lo]).
- Exchange a[i] with a[j].



- Scan i from left to right so long as (a[i] < a[10]).
- Scan j from right to left so long as (a[j] > a[lo]).
- Exchange a[i] with a[j].

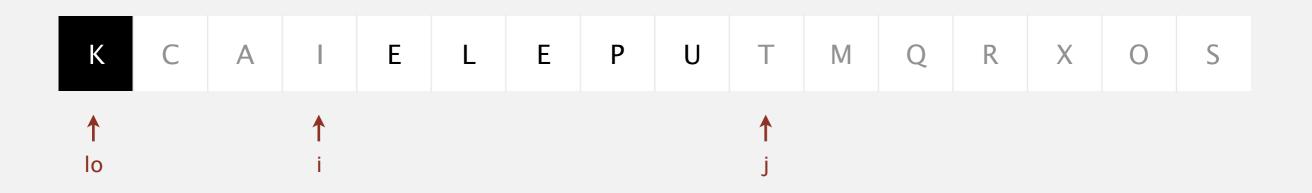


- Scan i from left to right so long as (a[i] < a[10]).
- Scan j from right to left so long as (a[j] > a[lo]).
- Exchange a[i] with a[j].

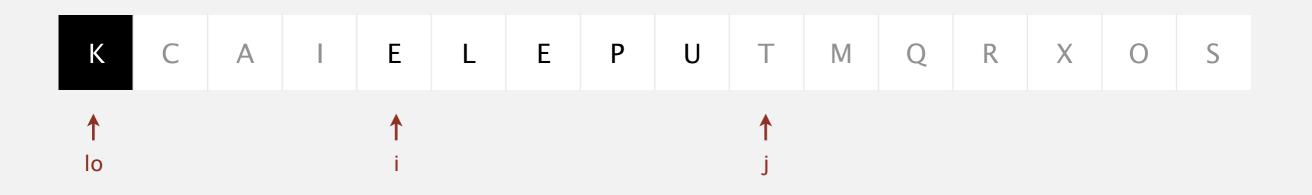


stop j scan and exchange a[i] with a[j]

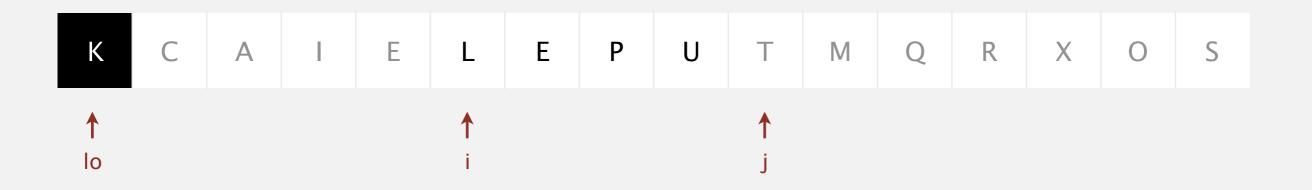
- Scan i from left to right so long as (a[i] < a[lo]).
- Scan j from right to left so long as (a[j] > a[lo]).
- Exchange a[i] with a[j].



- Scan i from left to right so long as (a[i] < a[10]).
- Scan j from right to left so long as (a[j] > a[lo]).
- Exchange a[i] with a[j].



- Scan i from left to right so long as (a[i] < a[10]).
- Scan j from right to left so long as (a[j] > a[lo]).
- Exchange a[i] with a[j].



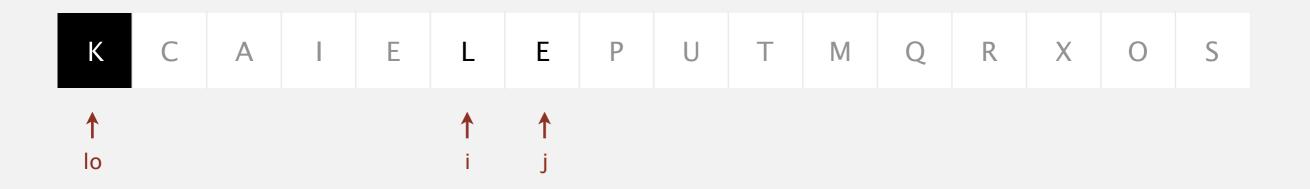
- Scan i from left to right so long as (a[i] < a[10]).
- Scan j from right to left so long as (a[j] > a[lo]).
- Exchange a[i] with a[j].



- Scan i from left to right so long as (a[i] < a[10]).
- Scan j from right to left so long as (a[j] > a[lo]).
- Exchange a[i] with a[j].

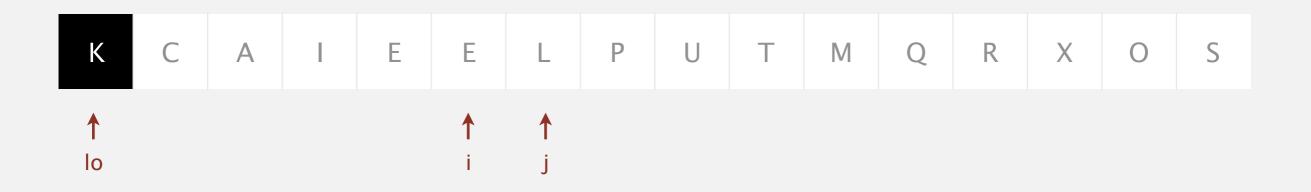


- Scan i from left to right so long as (a[i] < a[10]).
- Scan j from right to left so long as (a[j] > a[lo]).
- Exchange a[i] with a[j].

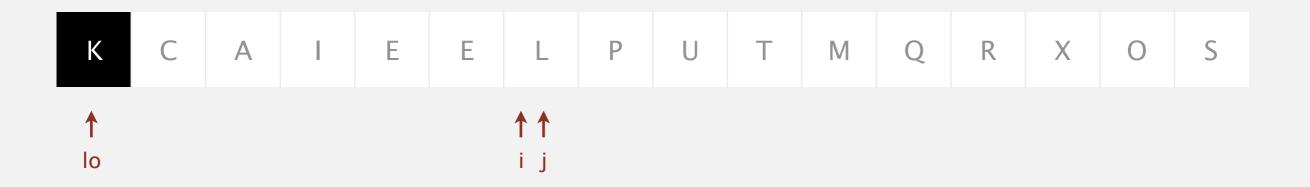


stop j scan and exchange a[i] with a[j]

- Scan i from left to right so long as (a[i] < a[10]).
- Scan j from right to left so long as (a[j] > a[lo]).
- Exchange a[i] with a[j].

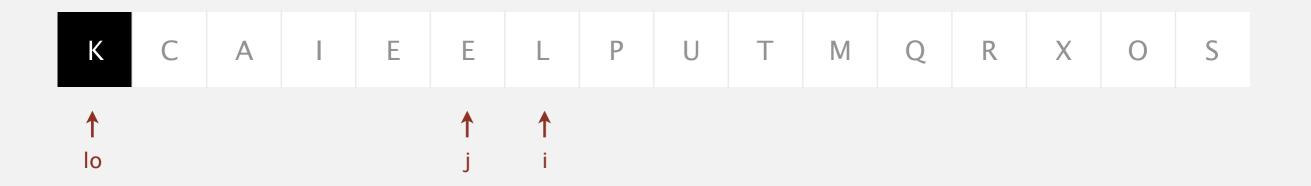


- Scan i from left to right so long as (a[i] < a[10]).
- Scan j from right to left so long as (a[j] > a[lo]).
- Exchange a[i] with a[j].



stop i scan because a[i] >= a[lo]

- Scan i from left to right so long as (a[i] < a[10]).
- Scan j from right to left so long as (a[j] > a[lo]).
- Exchange a[i] with a[j].



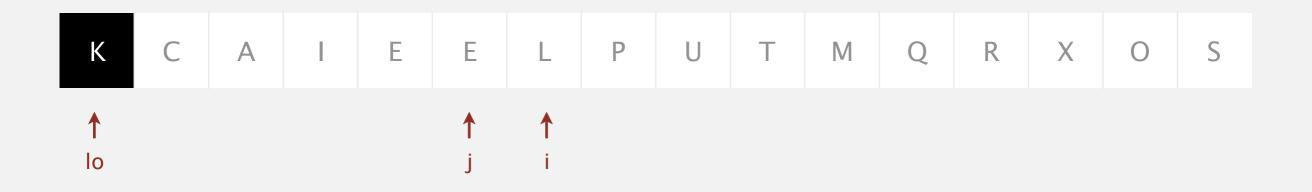
Quicksort partitioning demo

Repeat until i and j pointers cross.

- Scan i from left to right so long as (a[i] < a[10]).
- 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].

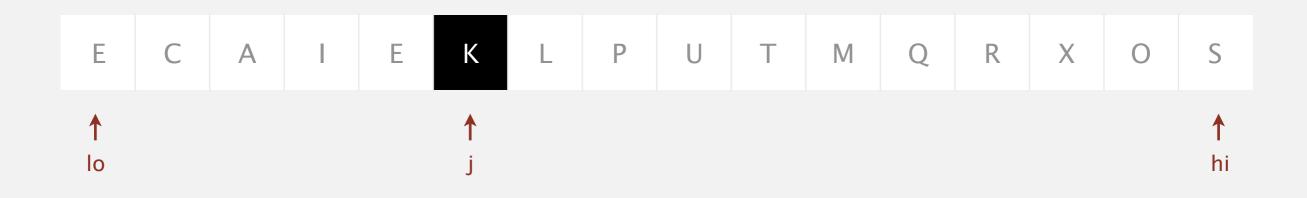


pointers cross: exchange a[lo] with a[j]

- Scan i from left to right so long as (a[i] < a[10]).
- 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].



2.3 PARTITIONING DEMOS

Bentley-McIlroy 3-way partitioning

Sedgewick 2-way partitioning

Dijkstra 3-way partitioning

dual-pivot partitioning

Algorithms

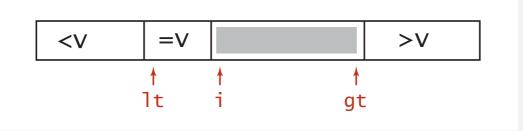
Robert Sedgewick | Kevin Wayne

http://algs4.cs.princeton.edu

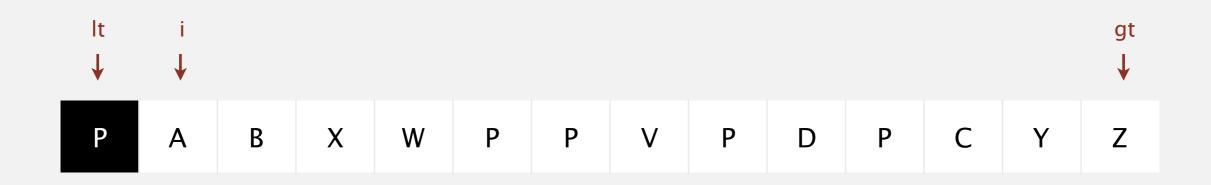
- Let v be partitioning item a[lo].
- Scan i from left to right.
 - (a[i] < v): exchange a[lt] with a[i]; increment both lt and i
 - (a[i] > v): exchange a[gt] with a[i]; decrement gt
 - (a[i] == v): increment i



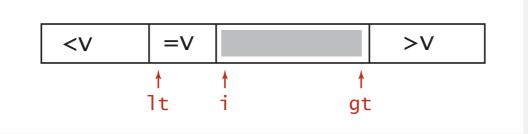
invariant



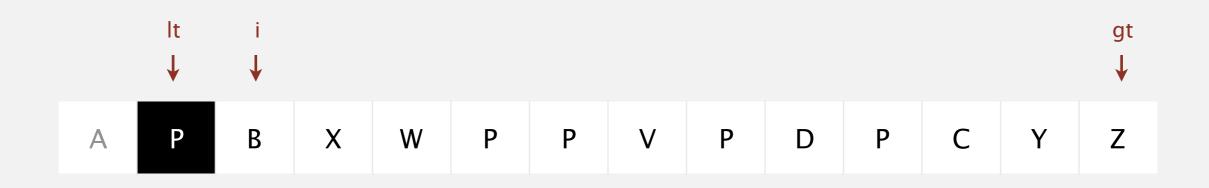
- Let v be partitioning item a[lo].
- Scan i from left to right.
 - (a[i] < v): exchange a[lt] with a[i]; increment both lt and i
 - (a[i] > v): exchange a[gt] with a[i]; decrement gt
 - (a[i] == v): increment i



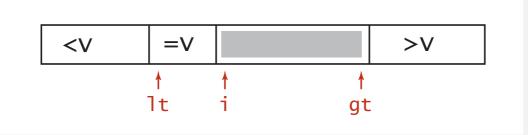
invariant



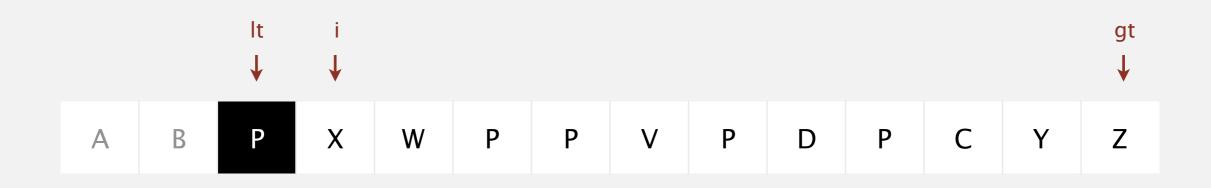
- Let v be partitioning item a[lo].
- Scan i from left to right.
 - (a[i] < v): exchange a[lt] with a[i]; increment both lt and i
 - (a[i] > v): exchange a[gt] with a[i]; decrement gt
 - (a[i] == v): increment i



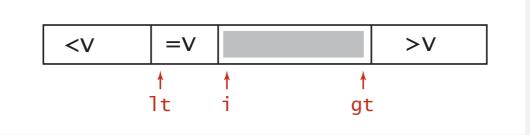
invariant



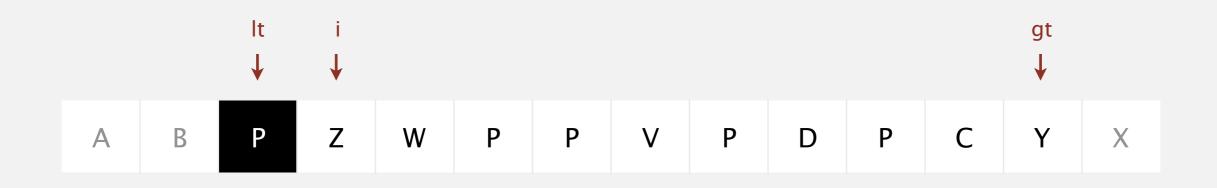
- Let v be partitioning item a[lo].
- Scan i from left to right.
 - (a[i] < v): exchange a[lt] with a[i]; increment both lt and i
 - (a[i] > v): exchange a[gt] with a[i]; decrement gt
 - (a[i] == v): increment i



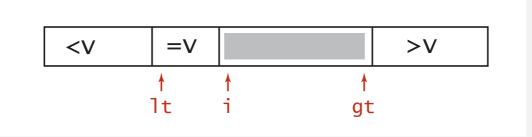
invariant



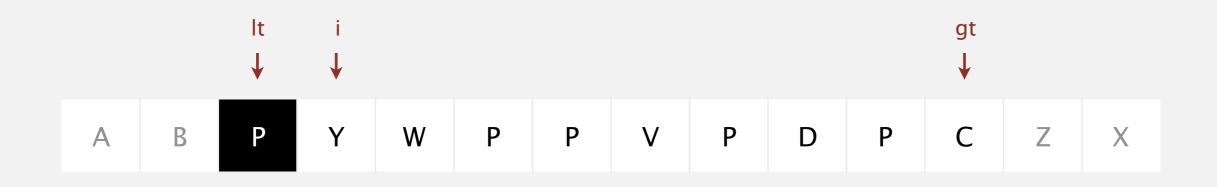
- Let v be partitioning item a[lo].
- Scan i from left to right.
 - (a[i] < v): exchange a[lt] with a[i]; increment both lt and i
 - (a[i] > v): exchange a[gt] with a[i]; decrement gt
 - (a[i] == v): increment i



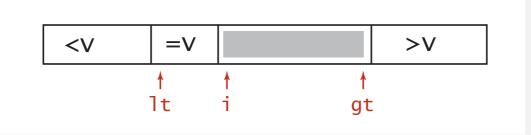




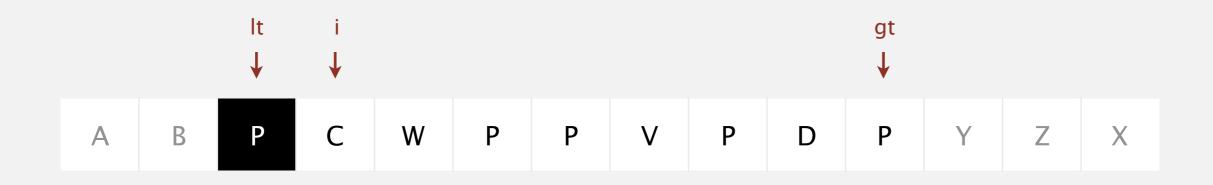
- Let v be partitioning item a[lo].
- Scan i from left to right.
 - (a[i] < v): exchange a[lt] with a[i]; increment both lt and i
 - (a[i] > v): exchange a[gt] with a[i]; decrement gt
 - (a[i] == v): increment i



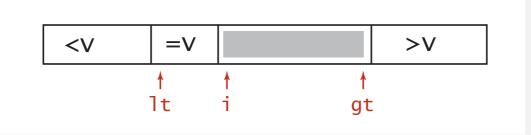
invariant



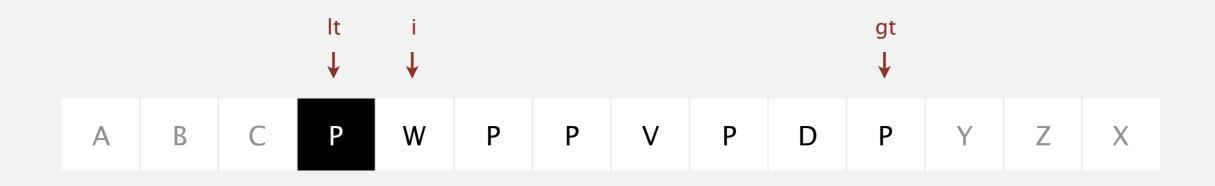
- Let v be partitioning item a[lo].
- Scan i from left to right.
 - (a[i] < v): exchange a[lt] with a[i]; increment both lt and i
 - (a[i] > v): exchange a[gt] with a[i]; decrement gt
 - (a[i] == v): increment i



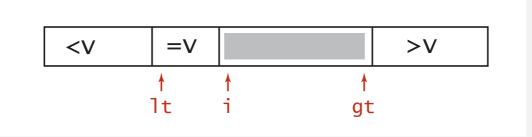
invariant



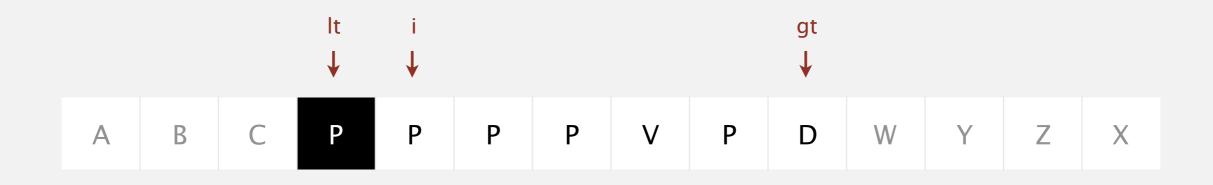
- Let v be partitioning item a[lo].
- Scan i from left to right.
 - (a[i] < v): exchange a[lt] with a[i]; increment both lt and i
 - (a[i] > v): exchange a[gt] with a[i]; decrement gt
 - (a[i] == v): increment i



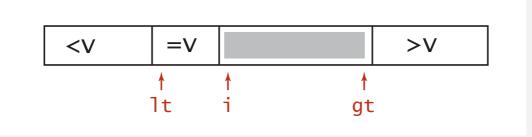




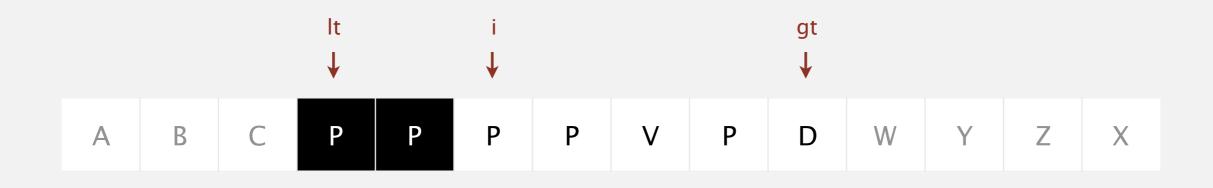
- Let v be partitioning item a[lo].
- Scan i from left to right.
 - (a[i] < v): exchange a[lt] with a[i]; increment both lt and i
 - (a[i] > v): exchange a[gt] with a[i]; decrement gt
 - (a[i] == v): increment i



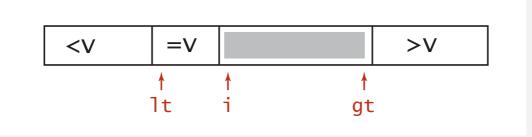
invariant



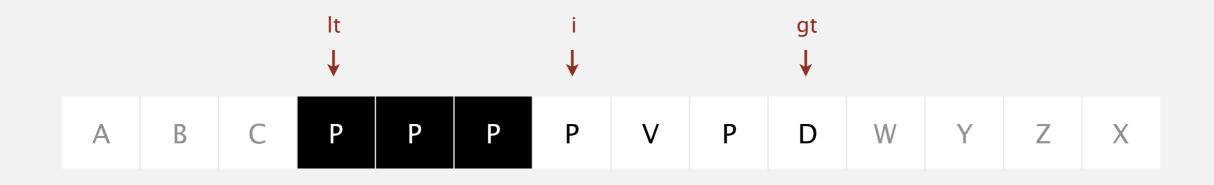
- Let v be partitioning item a[lo].
- Scan i from left to right.
 - (a[i] < v): exchange a[lt] with a[i]; increment both lt and i
 - (a[i] > v): exchange a[gt] with a[i]; decrement gt
 - (a[i] == v): increment i



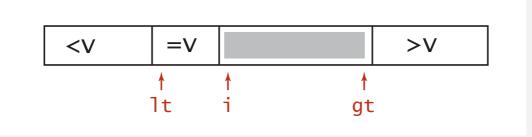
invariant



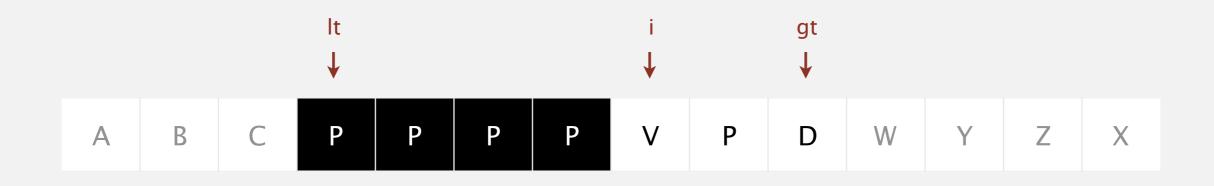
- Let v be partitioning item a[lo].
- Scan i from left to right.
 - (a[i] < v): exchange a[lt] with a[i]; increment both lt and i
 - (a[i] > v): exchange a[gt] with a[i]; decrement gt
 - (a[i] == v): increment i



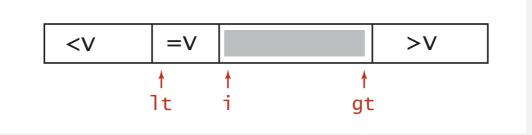
invariant



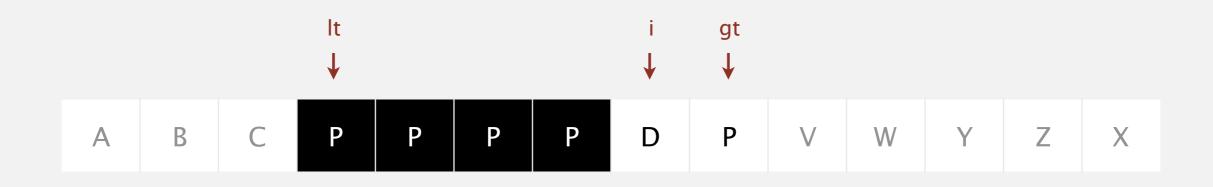
- Let v be partitioning item a[lo].
- Scan i from left to right.
 - (a[i] < v): exchange a[lt] with a[i]; increment both lt and i
 - (a[i] > v): exchange a[gt] with a[i]; decrement gt
 - (a[i] == v): increment i



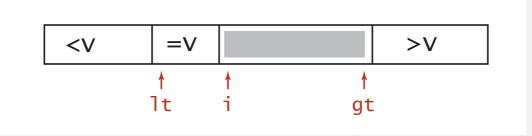
invariant



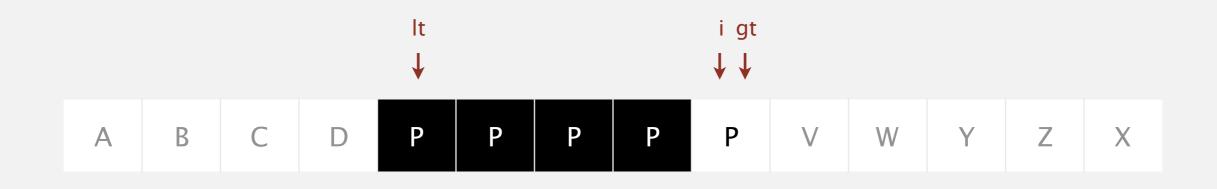
- Let v be partitioning item a[lo].
- Scan i from left to right.
 - (a[i] < v): exchange a[lt] with a[i]; increment both lt and i
 - (a[i] > v): exchange a[gt] with a[i]; decrement gt
 - (a[i] == v): increment i



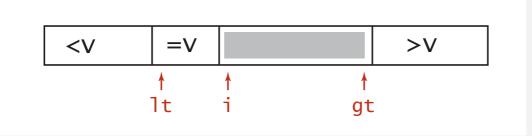
invariant



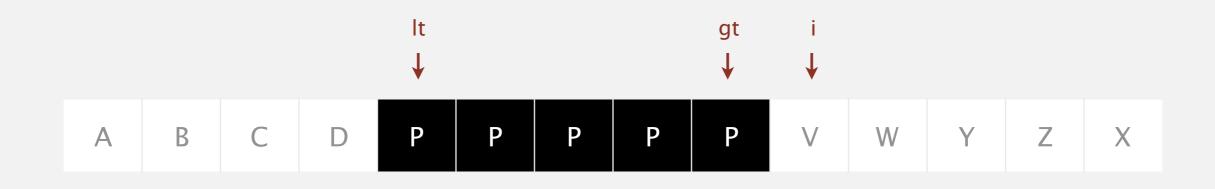
- Let v be partitioning item a[lo].
- Scan i from left to right.
 - (a[i] < v): exchange a[lt] with a[i]; increment both lt and i
 - (a[i] > v): exchange a[gt] with a[i]; decrement gt
 - (a[i] == v): increment i



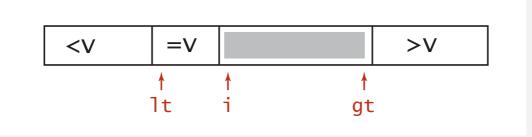




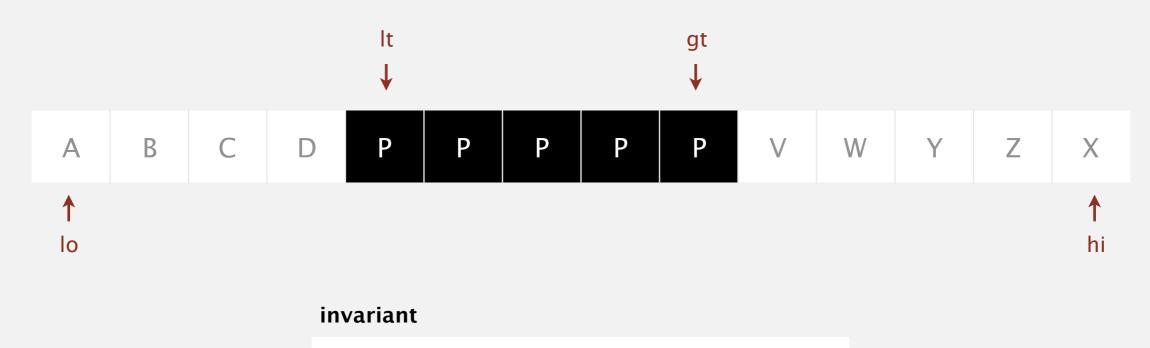
- Let v be partitioning item a[lo].
- Scan i from left to right.
 - (a[i] < v): exchange a[lt] with a[i]; increment both lt and i
 - (a[i] > v): exchange a[gt] with a[i]; decrement gt
 - (a[i] == v): increment i

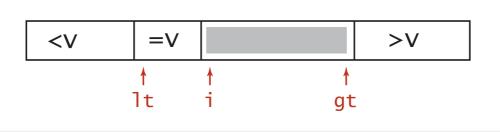






- Let v be partitioning item a[lo].
- Scan i from left to right.
 - (a[i] < v): exchange a[lt] with a[i]; increment both lt and i
 - (a[i] > v): exchange a[gt] with a[i]; decrement gt
 - (a[i] == v): increment i





2.3 PARTITIONING DEMOS

Sedgewick 2-way partitioning
Dijkstra 3-way partitioning

dual-pivot partitioning

Algorithms

Bentley-McIlroy 3-way partitioning

Robert Sedgewick | Kevin Wayne

http://algs4.cs.princeton.edu

- Scan i from left to right so long as (a[i] < a[lo]).
- Scan j from right to left so long as (a[i] > a[lo]).
- Exchange a[i] with a[j].
- If (a[i] == a[lo]), exchange a[i] with a[p] and increment p.
- If (a[j] == a[lo]), exchange a[j] with a[q] and decrement q.



- Scan i from left to right so long as (a[i] < a[lo]).
- Scan j from right to left so long as (a[i] > a[lo]).
- Exchange a[i] with a[j].
- If (a[i] == a[lo]), exchange a[i] with a[p] and increment p.
- If (a[j] == a[lo]), exchange a[j] with a[q] and decrement q.



- Scan i from left to right so long as (a[i] < a[lo]).
- Scan j from right to left so long as (a[i] > a[lo]).
- Exchange a[i] with a[j].
- If (a[i] == a[lo]), exchange a[i] with a[p] and increment p.
- If (a[j] == a[lo]), exchange a[j] with a[q] and decrement q.



- Scan i from left to right so long as (a[i] < a[lo]).
- Scan j from right to left so long as (a[i] > a[lo]).
- Exchange a[i] with a[j].
- If (a[i] == a[lo]), exchange a[i] with a[p] and increment p.
- If (a[j] == a[lo]), exchange a[j] with a[q] and decrement q.



Phase I. 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[i] > a[lo]).
- Exchange a[i] with a[j].
- If (a[i] == a[lo]), exchange a[i] with a[p] and increment p.
- If (a[j] == a[lo]), exchange a[j] with a[q] and decrement q.



exchange a[i] with a[j]

- Scan i from left to right so long as (a[i] < a[lo]).
- Scan j from right to left so long as (a[i] > a[lo]).
- Exchange a[i] with a[j].
- If (a[i] == a[lo]), exchange a[i] with a[p] and increment p.
- If (a[j] == a[lo]), exchange a[j] with a[q] and decrement q.



- Scan i from left to right so long as (a[i] < a[lo]).
- Scan j from right to left so long as (a[i] > a[lo]).
- Exchange a[i] with a[j].
- If (a[i] == a[lo]), exchange a[i] with a[p] and increment p.
- If (a[j] == a[lo]), exchange a[j] with a[q] and decrement q.



- Scan i from left to right so long as (a[i] < a[lo]).
- Scan j from right to left so long as (a[i] > a[lo]).
- Exchange a[i] with a[j].
- If (a[i] == a[lo]), exchange a[i] with a[p] and increment p.
- If (a[j] == a[lo]), exchange a[j] with a[q] and decrement q.



Phase I. 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[i] > a[lo]).
- Exchange a[i] with a[j].
- If (a[i] == a[lo]), exchange a[i] with a[p] and increment p.
- If (a[j] == a[lo]), exchange a[j] with a[q] and decrement q.



exchange a[i] with a[p] and increment p

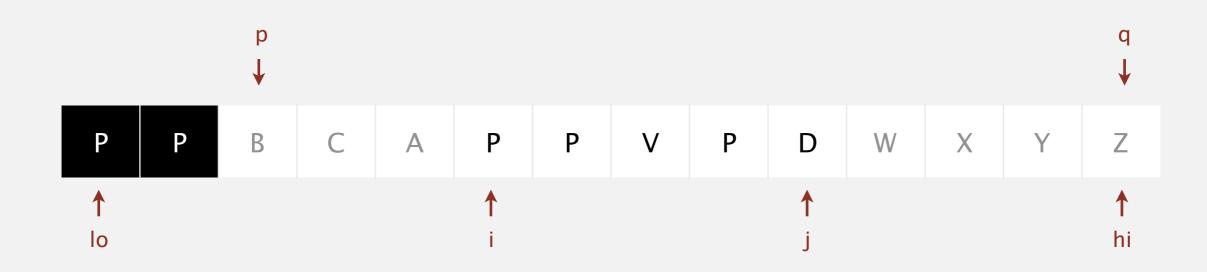
- Scan i from left to right so long as (a[i] < a[lo]).
- Scan j from right to left so long as (a[i] > a[lo]).
- Exchange a[i] with a[j].
- If (a[i] == a[lo]), exchange a[i] with a[p] and increment p.
- If (a[j] == a[lo]), exchange a[j] with a[q] and decrement q.



- Scan i from left to right so long as (a[i] < a[lo]).
- Scan j from right to left so long as (a[i] > a[lo]).
- Exchange a[i] with a[j].
- If (a[i] == a[lo]), exchange a[i] with a[p] and increment p.
- If (a[j] == a[lo]), exchange a[j] with a[q] and decrement q.

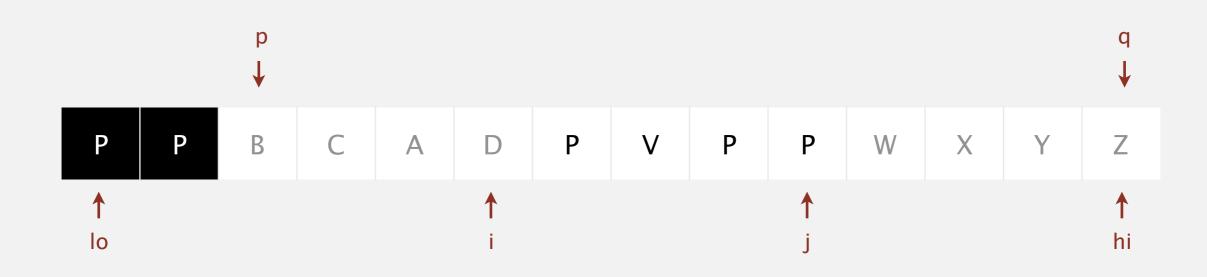


- Scan i from left to right so long as (a[i] < a[lo]).
- Scan j from right to left so long as (a[i] > a[lo]).
- Exchange a[i] with a[j].
- If (a[i] == a[lo]), exchange a[i] with a[p] and increment p.
- If (a[j] == a[lo]), exchange a[j] with a[q] and decrement q.



Phase I. 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[i] > a[lo]).
- Exchange a[i] with a[j].
- If (a[i] == a[lo]), exchange a[i] with a[p] and increment p.
- If (a[j] == a[lo]), exchange a[j] with a[q] and decrement q.



exchange a[j] with a[q] and decrement q

- Scan i from left to right so long as (a[i] < a[lo]).
- Scan j from right to left so long as (a[i] > a[lo]).
- Exchange a[i] with a[j].
- If (a[i] == a[lo]), exchange a[i] with a[p] and increment p.
- If (a[j] == a[lo]), exchange a[j] with a[q] and decrement q.

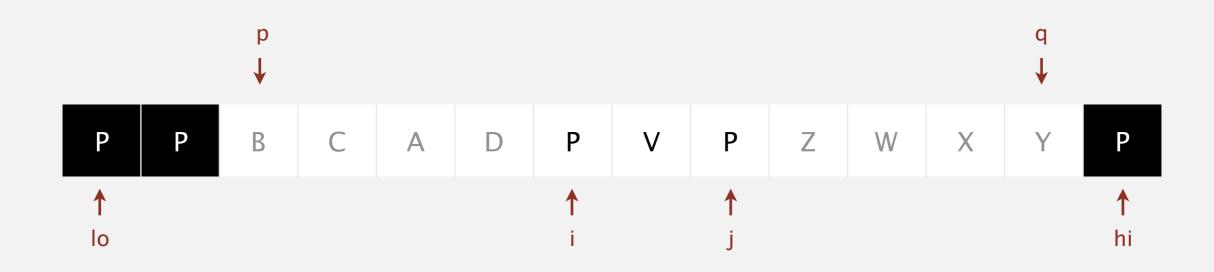


- Scan i from left to right so long as (a[i] < a[lo]).
- Scan j from right to left so long as (a[i] > a[lo]).
- Exchange a[i] with a[j].
- If (a[i] == a[lo]), exchange a[i] with a[p] and increment p.
- If (a[j] == a[lo]), exchange a[j] with a[q] and decrement q.



Phase I. Repeat until i and j pointers cross.

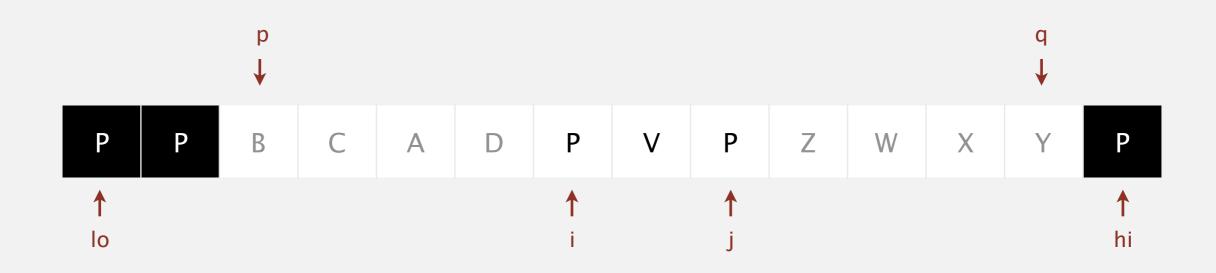
- Scan i from left to right so long as (a[i] < a[lo]).
- Scan j from right to left so long as (a[i] > a[lo]).
- Exchange a[i] with a[j].
- If (a[i] == a[lo]), exchange a[i] with a[p] and increment p.
- If (a[j] == a[lo]), exchange a[j] with a[q] and decrement q.



exchange a[i] with a[j]

Phase I. Repeat until i and j pointers cross.

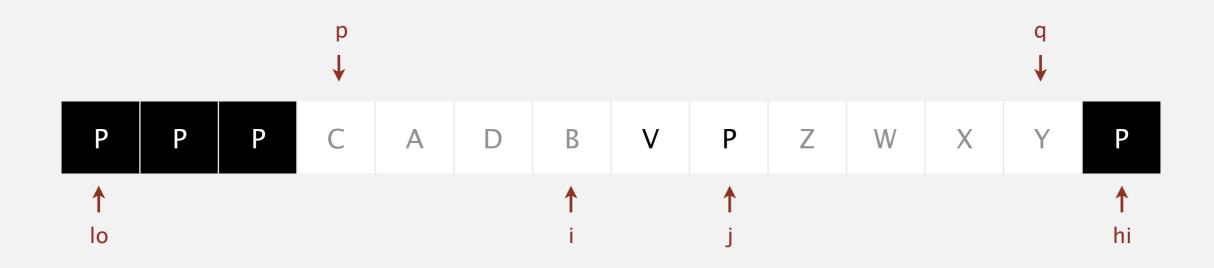
- Scan i from left to right so long as (a[i] < a[lo]).
- Scan j from right to left so long as (a[i] > a[lo]).
- Exchange a[i] with a[j].
- If (a[i] == a[lo]), exchange a[i] with a[p] and increment p.
- If (a[j] == a[lo]), exchange a[j] with a[q] and decrement q.



exchange a[i] with a[p] and increment p

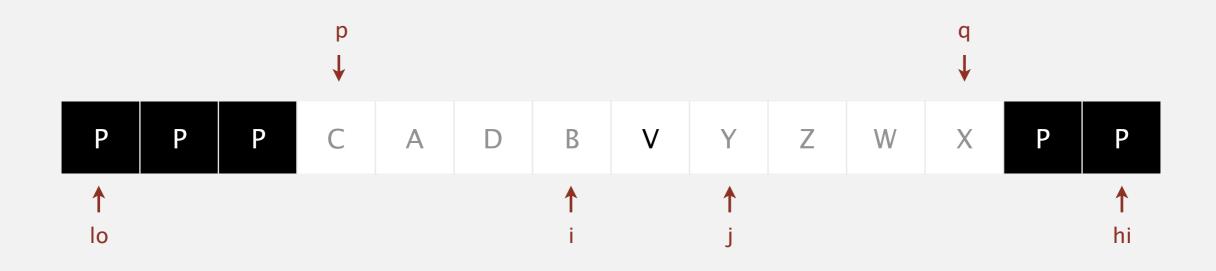
Phase I. Repeat until i and j pointers cross.

- Scan i from left to right so long as (a[i] < a[lo]).
- Scan j from right to left so long as (a[i] > a[lo]).
- Exchange a[i] with a[j].
- If (a[i] == a[lo]), exchange a[i] with a[p] and increment p.
- If (a[j] == a[lo]), exchange a[j] with a[q] and decrement q.



exchange a[j] with a[q] and decrement q

- Scan i from left to right so long as (a[i] < a[lo]).
- Scan j from right to left so long as (a[i] > a[lo]).
- Exchange a[i] with a[j].
- If (a[i] == a[lo]), exchange a[i] with a[p] and increment p.
- If (a[j] == a[lo]), exchange a[j] with a[q] and decrement q.



- Scan i from left to right so long as (a[i] < a[lo]).
- Scan j from right to left so long as (a[i] > a[lo]).
- Exchange a[i] with a[j].
- If (a[i] == a[lo]), exchange a[i] with a[p] and increment p.
- If (a[j] == a[lo]), exchange a[j] with a[q] and decrement q.



- Scan i from left to right so long as (a[i] < a[lo]).
- Scan j from right to left so long as (a[i] > a[lo]).
- Exchange a[i] with a[j].
- If (a[i] == a[lo]), exchange a[i] with a[p] and increment p.
- If (a[j] == a[lo]), exchange a[j] with a[q] and decrement q.



- Scan i from left to right so long as (a[i] < a[lo]).
- Scan j from right to left so long as (a[i] > a[lo]).
- Exchange a[i] with a[j].
- If (a[i] == a[lo]), exchange a[i] with a[p] and increment p.
- If (a[j] == a[lo]), exchange a[j] with a[q] and decrement q.



Phase II. Swap equal keys to the center.

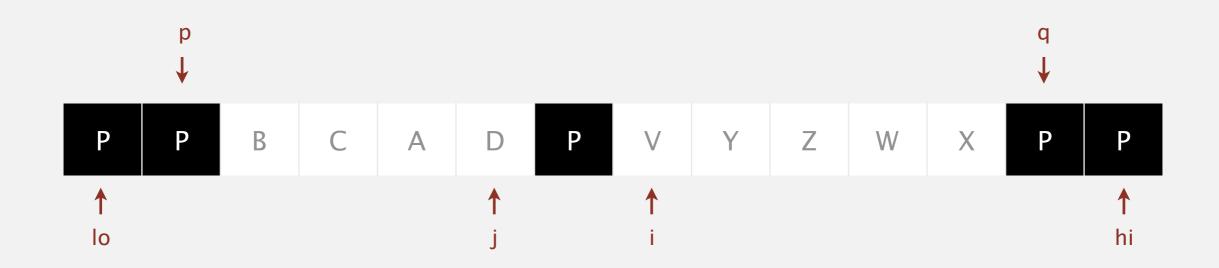
- Scan j and p from right to left and exchange a[j] with a[p].
- Scan i and q from left to right and exchange a[i] with a[q].



exchange a[j] with a[p]

Phase II. Swap equal keys to the center.

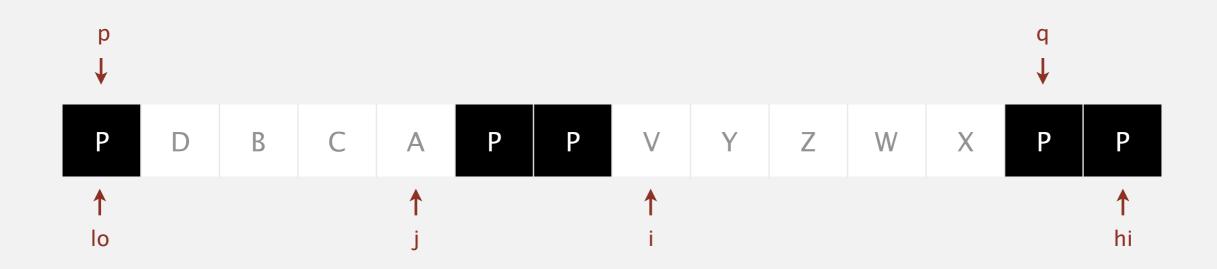
- Scan j and p from right to left and exchange a[j] with a[p].
- Scan i and q from left to right and exchange a[i] with a[q].



exchange a[j] with a[p]

Phase II. Swap equal keys to the center.

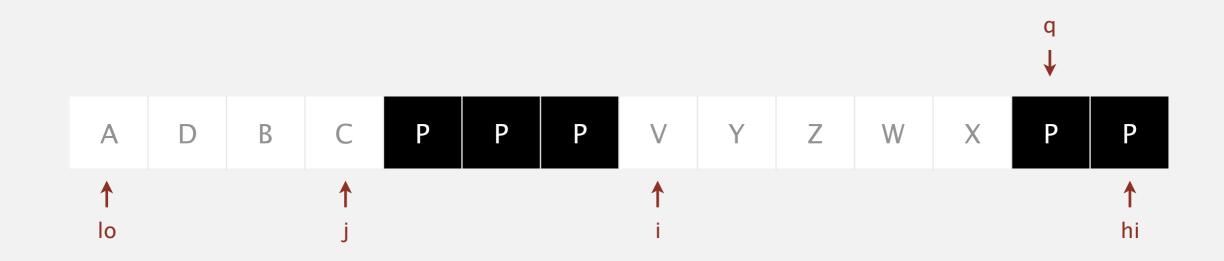
- Scan j and p from right to left and exchange a[j] with a[p].
- Scan i and q from left to right and exchange a[i] with a[q].



exchange a[j] with a[p]

Phase II. Swap equal keys to the center.

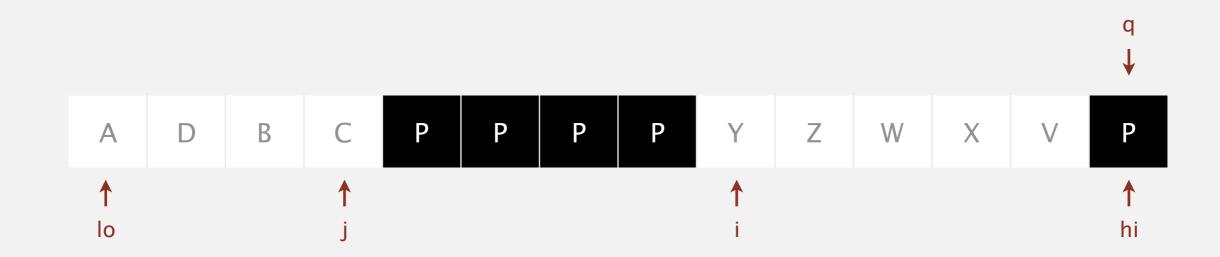
- Scan j and p from right to left and exchange a[j] with a[p].
- Scan i and q from left to right and exchange a[i] with a[q].



exchange a[i] with a[q]

Phase II. Swap equal keys to the center.

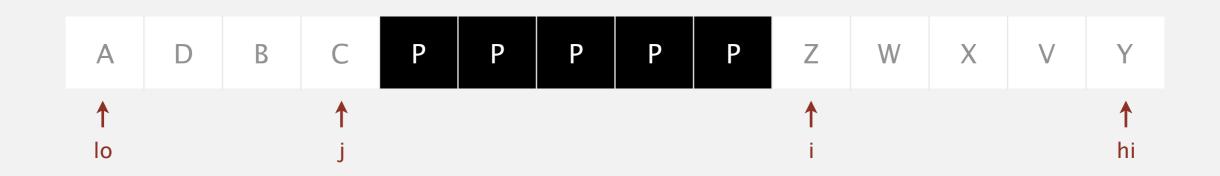
- Scan j and p from right to left and exchange a[j] with a[p].
- Scan i and q from left to right and exchange a[i] with a[q].



exchange a[i] with a[q]

Phase II. Swap equal keys to the center.

- Scan j and p from right to left and exchange a[j] with a[p].
- Scan i and q from left to right and exchange a[i] with a[q].



3-way partitioned

2.3 PARTITIONING DEMOS

Bentley-McIlroy 3-way partitioning

Sedgewick 2-way partitioning

Dijkstra 3-way partitioning

Algorithms

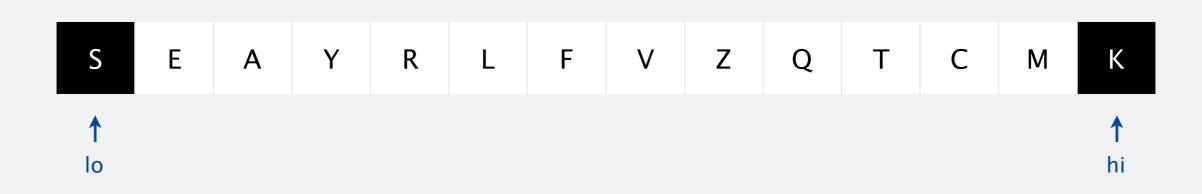
dual-pivot partitioning

ROBERT SEDGEWICK | KEVIN WAYNE

http://algs4.cs.princeton.edu

Initialization.

- Choose a[lo] and a[hi] as partitioning items.
- Exchange if necessary to ensure $a[lo] \le a[hi]$.

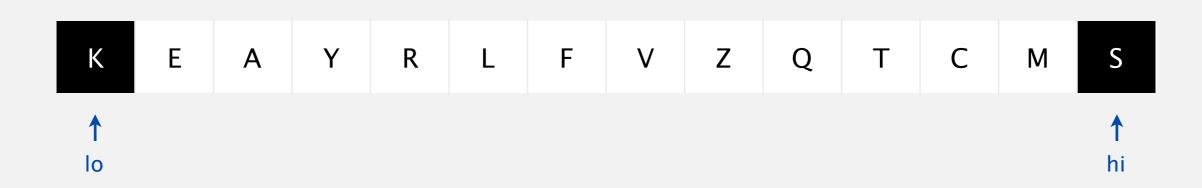


exchange a[lo] and a[hi]

Dual-pivot partitioning demo

Initialization.

- Choose a[lo] and a[hi] as partitioning items.
- Exchange if necessary to ensure $a[lo] \le a[hi]$.



- If (a[i] < a[lo]), exchange a[i] with a[lt] and increment It and i.
- Else if (a[i] > a[hi]), exchange a[i] with a[gt] and decrement gt.
- Else, increment i.

p ₁	<	p ₁	р	1≤ ar	nd ≤ p	02		- !	?		>	p ₂	p ₂
↑ Io			↑ It				↑ i			∱ gt			↑ hi
K	E	А	Y	R	L	F	V	Z	Q	т	С	Μ	S
↑ Io	↑↑ It i											↑ gt	∱ hi

- If (a[i] < a[lo]), exchange a[i] with a[lt] and increment It and i.
- Else if (a[i] > a[hi]), exchange a[i] with a[gt] and decrement gt.
- Else, increment i.

p1	<	p1	р	1≤ ar	nd ≤ p	02		Ĩ	?		>	p ₂	p ₂
∱ lo			↑ It				↑ i			↑ gt			↑ hi
K	Е	А	Y	R	L	F	V	Z	Q	т	С	Μ	S
↑ Io		↑↑ It i										↑ gt	∱ hi

- If (a[i] < a[lo]), exchange a[i] with a[lt] and increment It and i.
- Else if (a[i] > a[hi]), exchange a[i] with a[gt] and decrement gt.
- Else, increment i.

p1	<	p 1	р	ı≤ ar	nd ≤ p	02		÷	?		>	p ₂	p ₂
↑ Io			↑ It				↑ i			∱ gt			↑ hi
K	E	А	Y	R	L	F	V	Z	Q	т	С	Μ	S
↑ Io			↑↑ lt i									↑ gt	↑ hi

exchange a[i] and a[gt]; decrement gt

- If (a[i] < a[lo]), exchange a[i] with a[lt] and increment It and i.
- Else if (a[i] > a[hi]), exchange a[i] with a[gt] and decrement gt.
- Else, increment i.

p1	<	p ₁	р	$1 \leq ar$	nd ≤ p	02		÷	2		>	p ₂	p 2
↑ Io			↑ It				↑ i			∱ gt			↑ hi
К	Е	А	М	R	L	F	V	Z	Q	Т	С	Y	S
↑ Io			↑↑ It i								↑ gt		∱ hi

- If (a[i] < a[lo]), exchange a[i] with a[lt] and increment It and i.
- Else if (a[i] > a[hi]), exchange a[i] with a[gt] and decrement gt.
- Else, increment i.

p ₁	<	p ₁	p	$n_1 \leq ar$	nd ≤ p	02		Ĩ	?		>	p 2	p ₂
↑ Io			↑ It				↑ i			∱ gt			↑ hi
K	Е	А	Μ	R	L	F	V	Z	Q	Т	С	Y	S
↑ Io			↑ It	↑ i							↑ gt		∱ hi

increment i

- If (a[i] < a[lo]), exchange a[i] with a[lt] and increment It and i.
- Else if (a[i] > a[hi]), exchange a[i] with a[gt] and decrement gt.
- Else, increment i.

p ₁	<	p ₁	р	n≤ ar	id ≤ p	02		÷	?		>	p ₂	p ₂
↑ Io			↑ It				↑ i			↑ gt			↑ hi
К	Е	А	Μ	R	L	F	V	Z	Q	Т	С	Y	S
↑ Io			↑ It		↑ i						↑ gt		∱ hi

increment i

- If (a[i] < a[lo]), exchange a[i] with a[lt] and increment It and i.
- Else if (a[i] > a[hi]), exchange a[i] with a[gt] and decrement gt.
- Else, increment i.

p ₁	<	p ₁	р	$n \leq ar$	nd ≤ p	02		:	?		>	p ₂	p ₂
↑ Io			↑ It				↑ i			↑ gt			↑ hi
K	Е	А	Μ	R	L	F	V	Z	Q	т	С	Y	S
↑ Io			↑ It			↑ i					∱ gt		∱ hi

- If (a[i] < a[lo]), exchange a[i] with a[lt] and increment It and i.
- Else if (a[i] > a[hi]), exchange a[i] with a[gt] and decrement gt.
- Else, increment i.

p ₁	<	p ₁	р	$n_1 \leq an$	d ≤ p	0 2		7)		>	p ₂	p ₂
∱ Io			↑ It				↑ i			∱ gt			↑ hi
К	Е	А	F	R	L	Μ	V	Z	Q	т	С	Y	S
↑ Io				↑ It			↑ i				↑ gt		∱ hi

exchange a[i] and a[gt]; decrement gt

- If (a[i] < a[lo]), exchange a[i] with a[lt] and increment It and i.
- Else if (a[i] > a[hi]), exchange a[i] with a[gt] and decrement gt.
- Else, increment i.

p1	<	01	р	n≤ ar	id ≤ p	0 2		Ĩ	?		>	p ₂	p ₂
∱ Io			↑ It				↑ i			∱ gt			↑ hi
K	Е	А	F	R	L	Μ	С	Z	Q	Т	V	Y	S
↑ Io				↑ It			↑ i			↑ gt			∱ hi

- If (a[i] < a[lo]), exchange a[i] with a[lt] and increment It and i.
- Else if (a[i] > a[hi]), exchange a[i] with a[gt] and decrement gt.
- Else, increment i.

p ₁	<	p 1	р	ı≤ ar	nd ≤ p	02		Ī	?		>	p ₂	p ₂
∱ Io			↑ It				↑ i			∱ gt			↑ hi
K	Е	А	F	С	L	Μ	R	Z	Q	т	V	Y	S
↑ Io					↑ It			↑ i		∱ gt			∱ hi

exchange a[i] and a[gt]; decrement gt

- If (a[i] < a[lo]), exchange a[i] with a[lt] and increment It and i.
- Else if (a[i] > a[hi]), exchange a[i] with a[gt] and decrement gt.
- Else, increment i.

p ₁	<	p ₁	р	n≤ ar	nd ≤ p	02		Ī	2		>	p ₂	p ₂
↑ Io			↑ It				↑ i			∱ gt			↑ hi
К	Е	А	F	С	L	Μ	R	Т	Q	Z	V	Y	S
↑ Io					↑ It			↑ i	↑ gt				∱ hi

exchange a[i] and a[gt]; decrement gt

- If (a[i] < a[lo]), exchange a[i] with a[lt] and increment It and i.
- Else if (a[i] > a[hi]), exchange a[i] with a[gt] and decrement gt.
- Else, increment i.

p ₁	<	p ₁	р	₁≤ ar	nd ≤ p	02		?	,		>	p ₂	p ₂
∱ Io			↑ It				↑ i			∱ gt			↑ hi
K	E	А	F	С	L	Μ	R	Q	Т	Ζ	V	Y	S
↑ Io					↑ It			↑↑ i gt					↑ hi

- If (a[i] < a[lo]), exchange a[i] with a[lt] and increment It and i.
- Else if (a[i] > a[hi]), exchange a[i] with a[gt] and decrement gt.
- Else, increment i.

p ₁	<	p ₁	р	n≤ ar	nd ≤ p	D 2		-	?		>	p ₂	p ₂
↑ Io			↑ It				↑ i			∱ gt			↑ hi
K	Е	Α	F	С	L	М	R	Q	Т	Z	V	Y	S
↑ Io					∱ lt			↑ gt	↑ i				∱ hi

stop when pointers cross

Dual-pivot partitioning demo

Finalize.

- Exchange a[lo] with a[--lt].
- Exchange a[hi] with a[++gt].

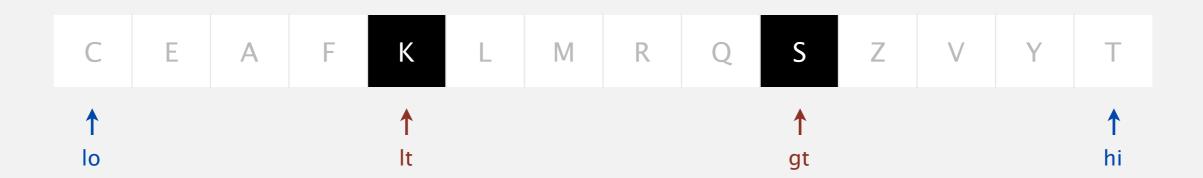
p ₁	< p1			$p_1 \leq and \leq p_2$				> p ₂				p ₂	
∱ Io					∱ lt			∱ gt					↑ hi
K	Е	А	F	С	L	М	R	Q	Т	Z	V	Y	S
↑ Io					↑ It			↑ gt					↑ hi

Dual-pivot partitioning demo

Finalize.

- Exchange a[lo] with a[--lt].
- Exchange a[hi] with a[++gt].

	< p1	p1	$p_1 \leq and \leq p_2$	p ₂	> p ₂	
∱ Io		∱ lt		↑ gt		↑ hi



3-way partitioned