Quicksort Partitioning

	i	j	0	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	6	5	E	C	A	I	E	K	L	P	U	T	M	Q	R	X	O	S

v

a[i]

Partitioning trace (array contents before and after each exchange)

Quicksort 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]`.

K	R	A	T	E	L	E	P	U	I	M	Q	C	X	O	S

lo i j

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

Quicksort 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]`.

K	R	A	T	E	L	E	P	U	I	M	Q	C	X	O	S
↑ lo	↑ i													↑ j	

Quicksort partitioning

Repeat until `i` and `j` pointers cross.

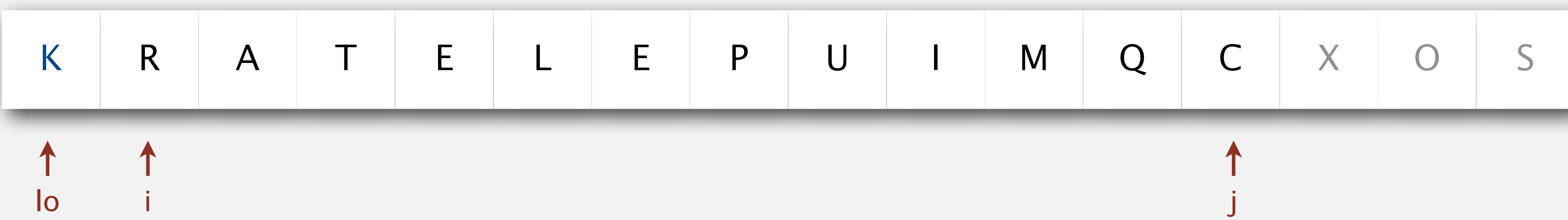
- Scan `i` from left to right so long as `a[i] < a[lo]`.
- Scan `j` from right to left so long as `a[j] > a[lo]`.
- Exchange `a[i]` with `a[j]`.

K	R	A	T	E	L	E	P	U	I	M	Q	C	X	O	S
lo	i												j		

Quicksort 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]`.



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

Quicksort partitioning

Repeat until `i` and `j` pointers cross.

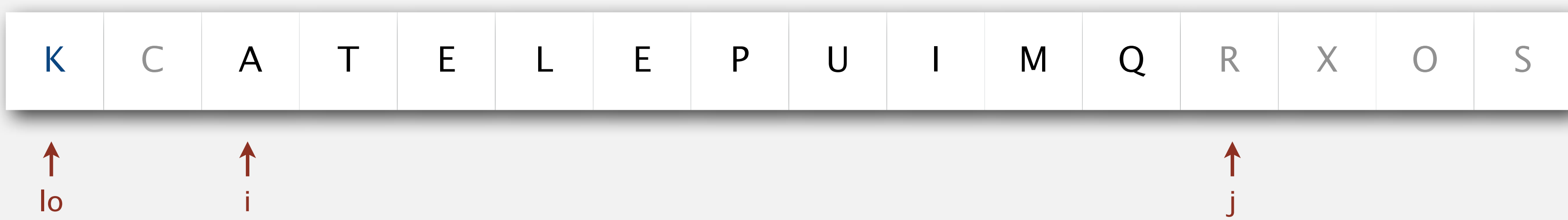
- Scan `i` from left to right so long as `a[i] < a[lo]`.
- Scan `j` from right to left so long as `a[j] > a[lo]`.
- Exchange `a[i]` with `a[j]`.

K	C	A	T	E	L	E	P	U	I	M	Q	R	X	O	S
↑ lo	↑ i											↑ j			

Quicksort partitioning

Repeat until `i` and `j` pointers cross.

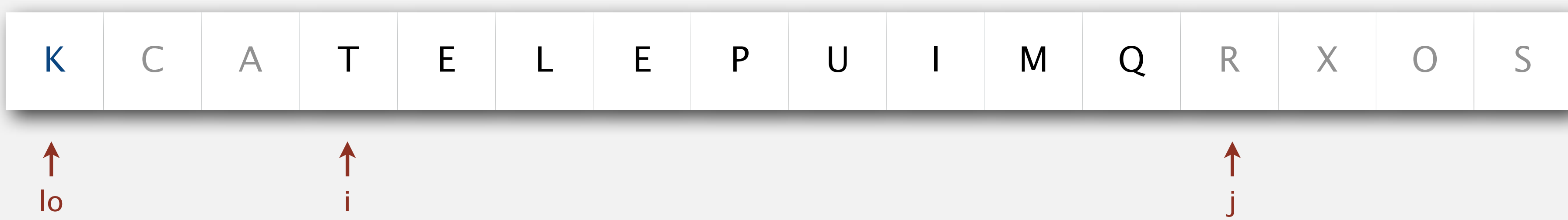
- Scan `i` from left to right so long as `a[i] < a[lo]`.
- Scan `j` from right to left so long as `a[j] > a[lo]`.
- Exchange `a[i]` with `a[j]`.



Quicksort partitioning

Repeat until `i` and `j` pointers cross.

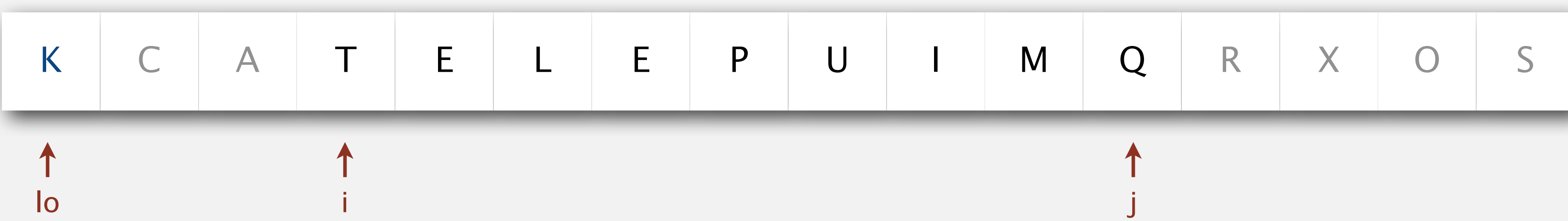
- Scan `i` from left to right so long as `a[i] < a[lo]`.
- Scan `j` from right to left so long as `a[j] > a[lo]`.
- Exchange `a[i]` with `a[j]`.



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

Repeat until `i` and `j` pointers cross.

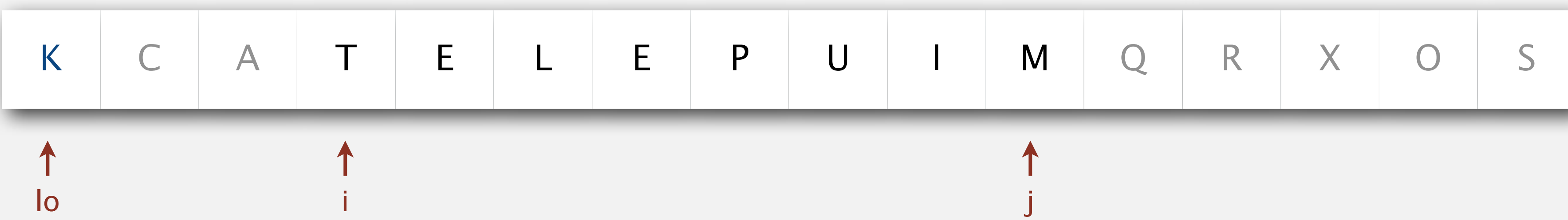
- Scan `i` from left to right so long as `a[i] < a[lo]`.
- Scan `j` from right to left so long as `a[j] > a[lo]`.
- Exchange `a[i]` with `a[j]`.



Quicksort 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]`.



Quicksort 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]`.

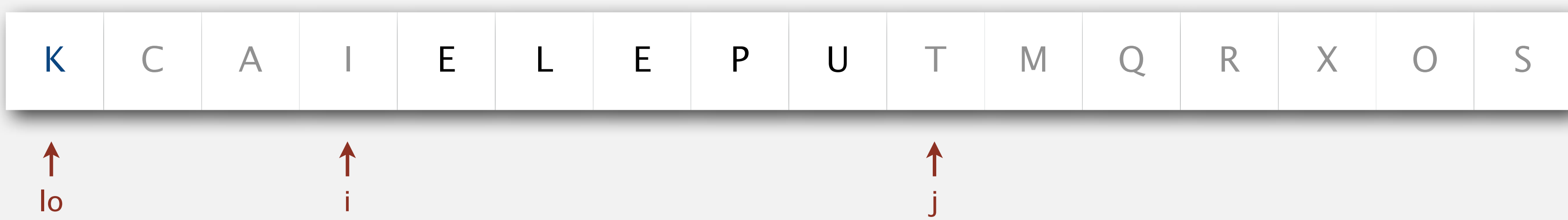
K	C	A	T	E	L	E	P	U	I	M	Q	R	X	O	S
↑ lo			↑ i						↑ j						

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

Quicksort partitioning

Repeat until `i` and `j` pointers cross.

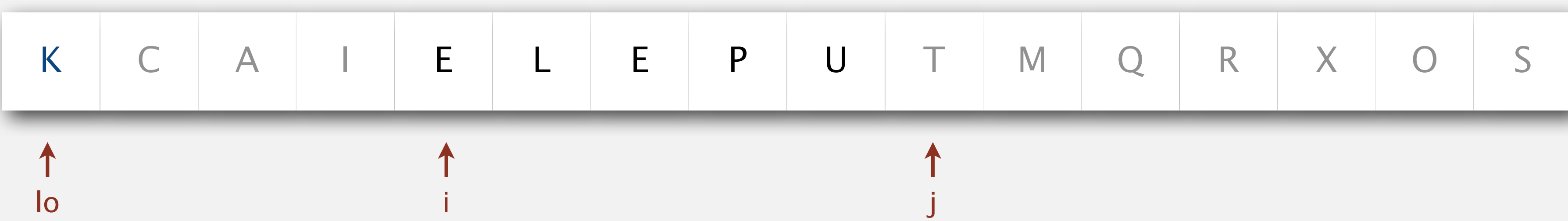
- Scan `i` from left to right so long as `a[i] < a[lo]`.
- Scan `j` from right to left so long as `a[j] > a[lo]`.
- Exchange `a[i]` with `a[j]`.



Quicksort partitioning

Repeat until `i` and `j` pointers cross.

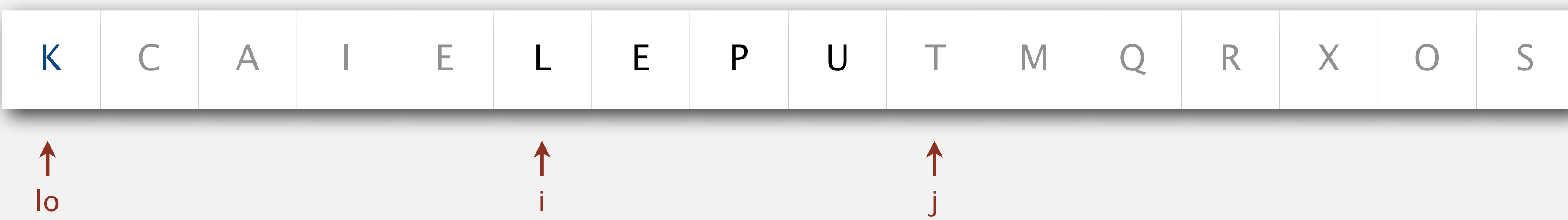
- Scan `i` from left to right so long as `a[i] < a[lo]`.
- Scan `j` from right to left so long as `a[j] > a[lo]`.
- Exchange `a[i]` with `a[j]`.



Quicksort 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]`.

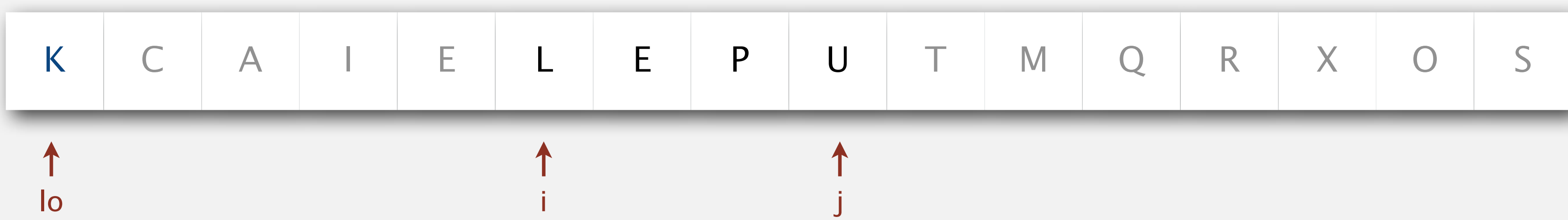


stop `i` scan because `a[i] >= a[lo]`

Quicksort partitioning

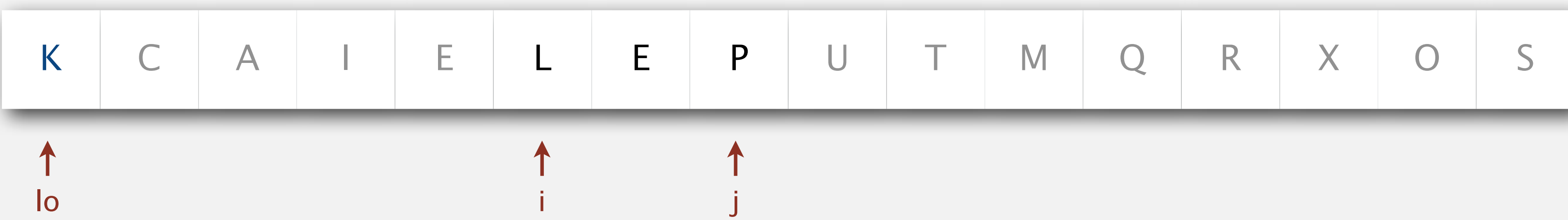
Repeat until `i` and `j` pointers cross.

- 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]`.



Repeat until `i` and `j` pointers cross.

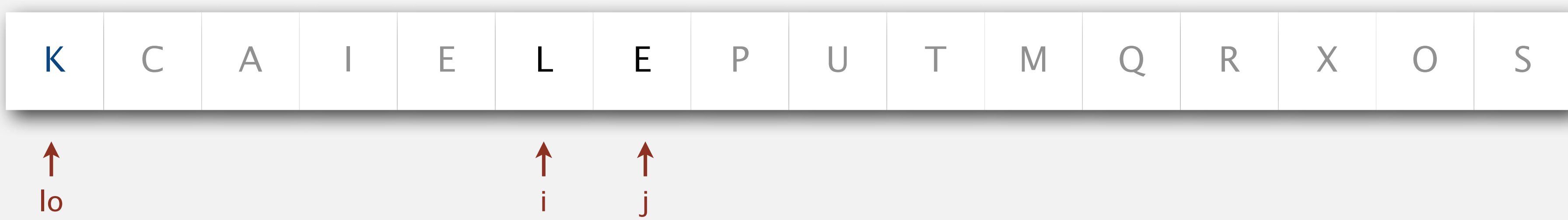
- Scan `i` from left to right so long as `a[i] < a[lo]`.
- Scan `j` from right to left so long as `a[j] > a[lo]`.
- Exchange `a[i]` with `a[j]`.



Quicksort 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]`.

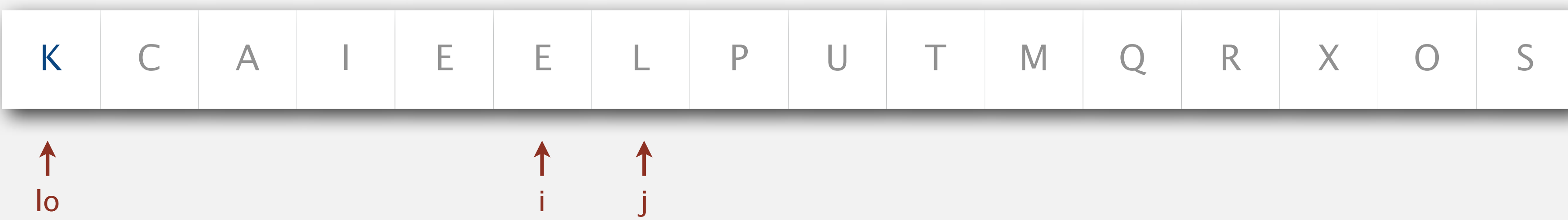


stop `j` scan and exchange `a[i]` with `a[j]`

Quicksort partitioning

Repeat until `i` and `j` pointers cross.

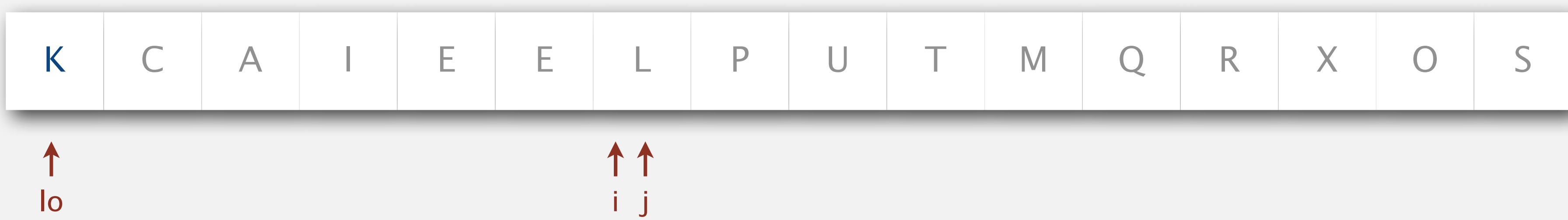
- Scan `i` from left to right so long as `a[i] < a[lo]`.
- Scan `j` from right to left so long as `a[j] > a[lo]`.
- Exchange `a[i]` with `a[j]`.



Quicksort 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]`.

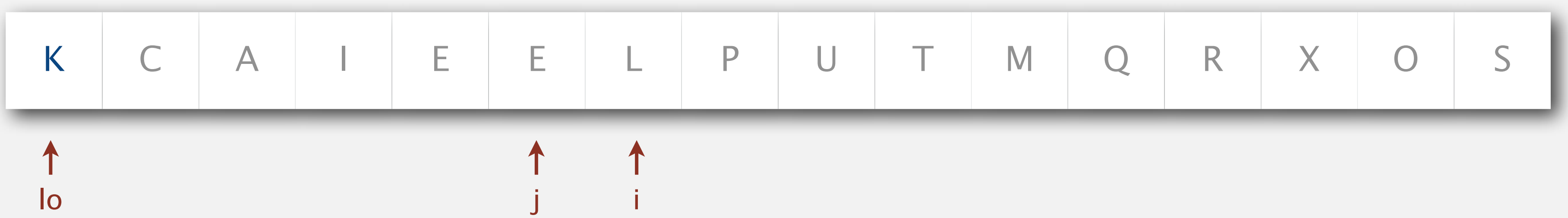


stop `i` scan because `a[i] >= a[lo]`

Quicksort 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]`.



stop `j` scan because `a[j] <= a[lo]`

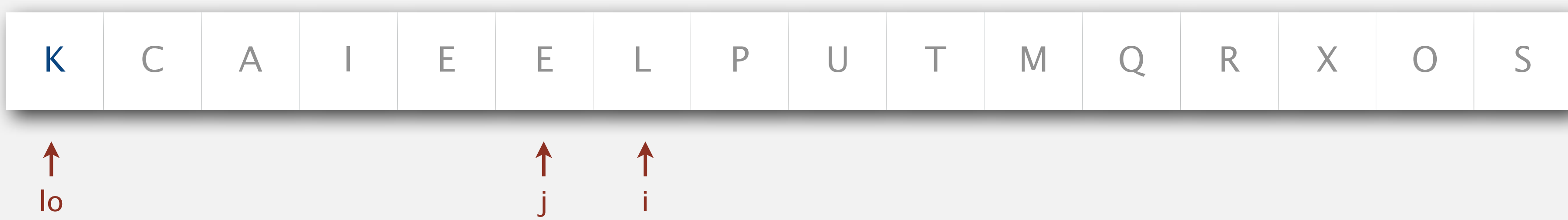
Quicksort partitioning

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]`.



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

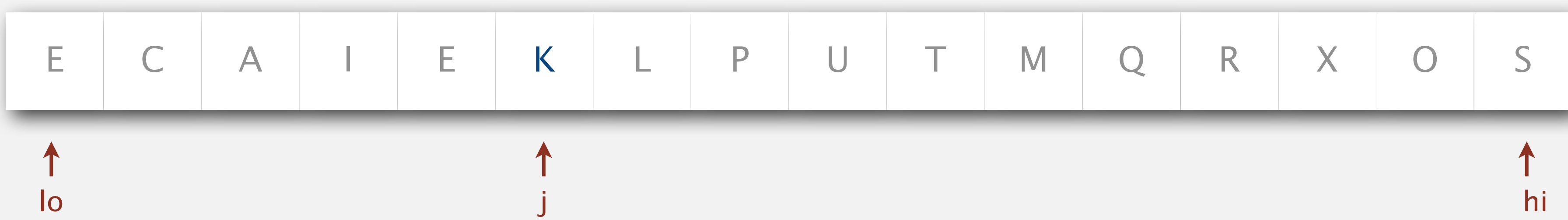
Quicksort partitioning

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]`.



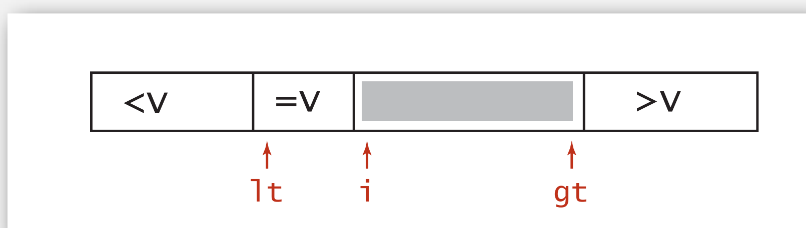
partitioned!

Dijkstra 3-Way Partitioning

Dijkstra 3-way partitioning

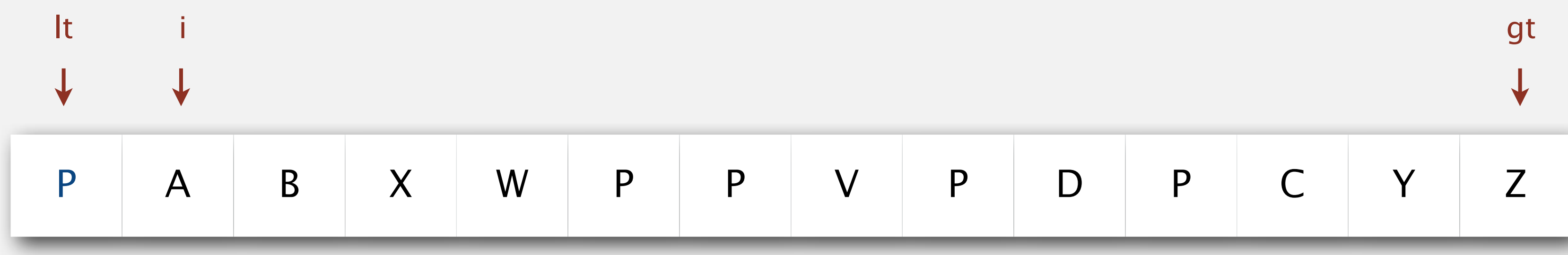
- Let `v` be partitioning element `a[lo]`.
- Scan `i` from left to right.
  - `(a[i]  < v)`: exchange `a[lt]` with `a[i]` and increment both `lt` and `i`
  - `(a[i]  > v)`: exchange `a[gt]` with `a[i]` and decrement `gt`
  - `(a[i] == v)`: increment `i`

invariant

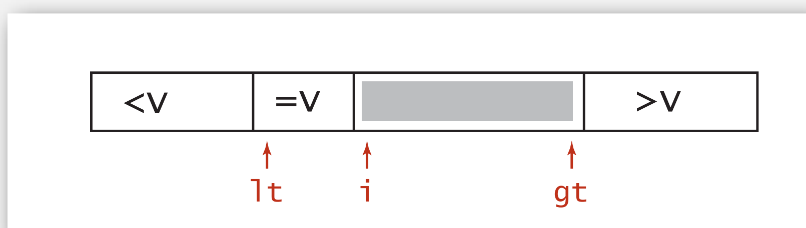


Dijkstra 3-way partitioning

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

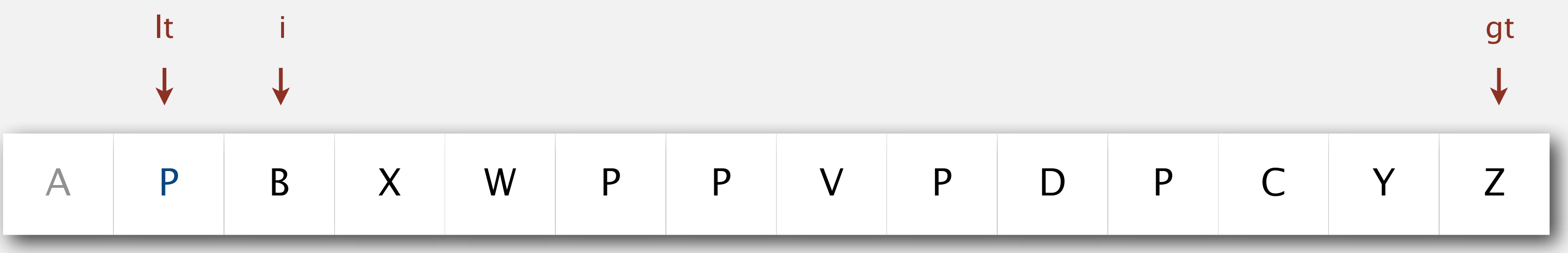


invariant

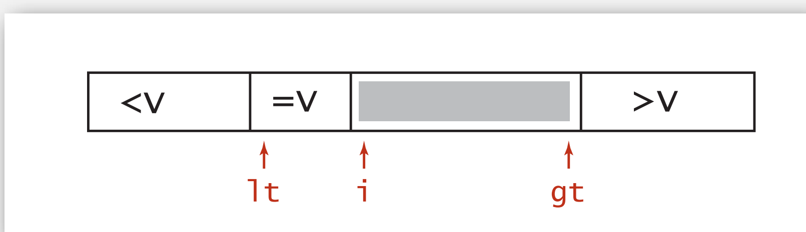


Dijkstra 3-way partitioning

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

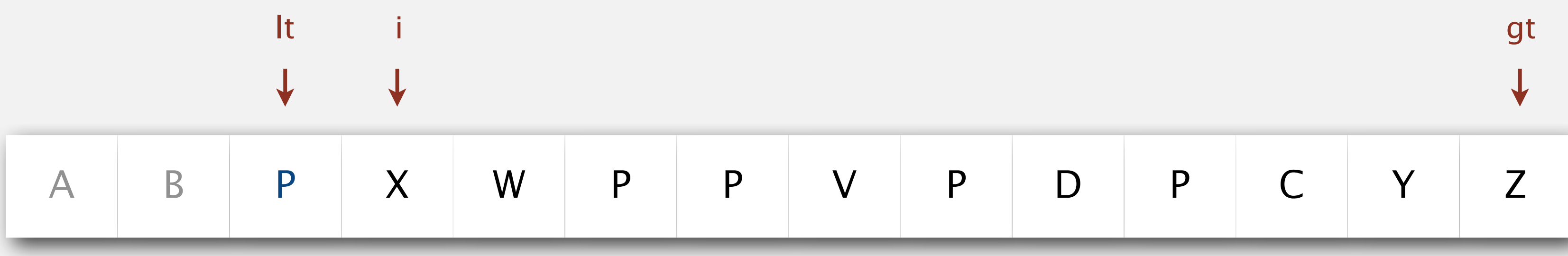


invariant

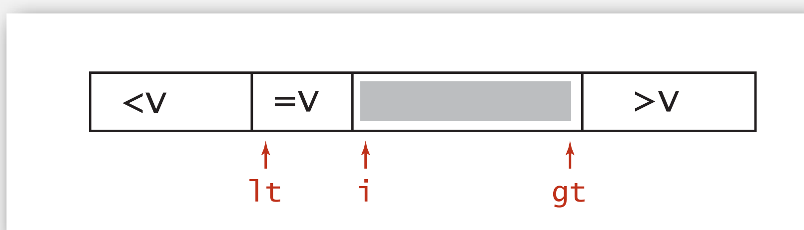


Dijkstra 3-way partitioning

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

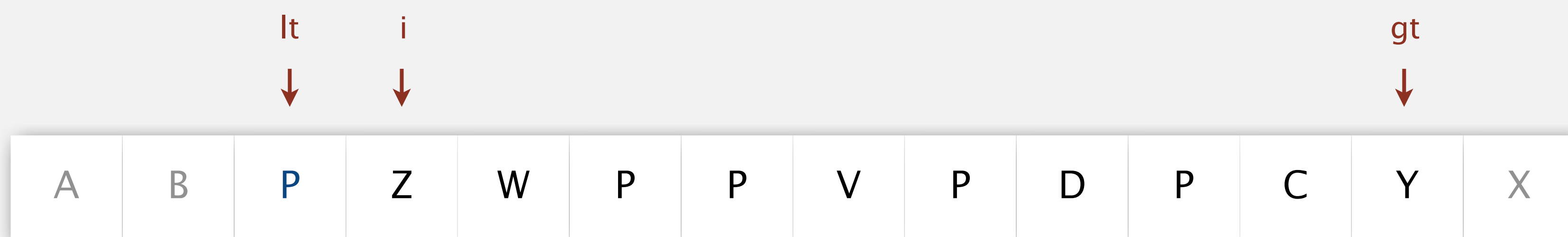


invariant

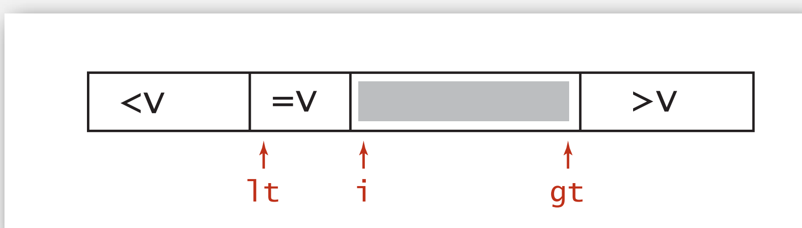


Dijkstra 3-way partitioning

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

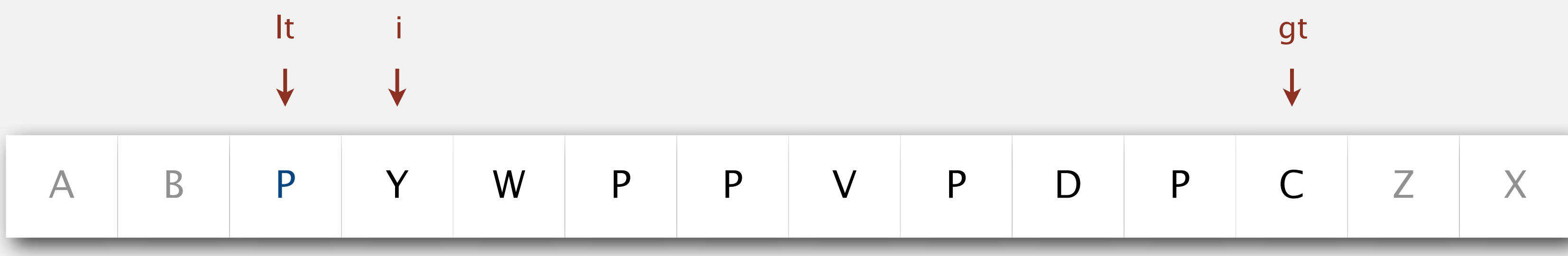


invariant

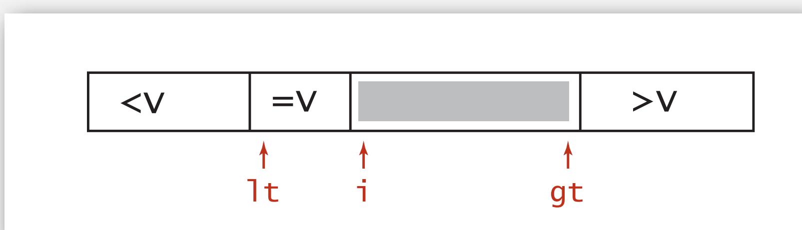


Dijkstra 3-way partitioning

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

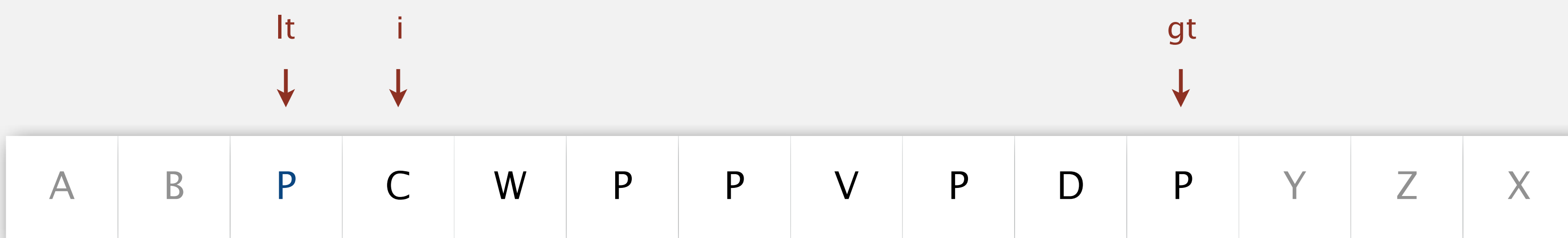


invariant

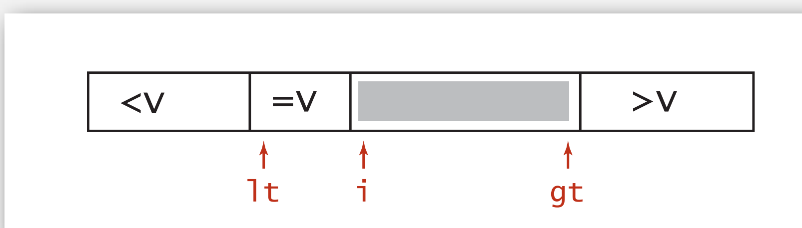


Dijkstra 3-way partitioning

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

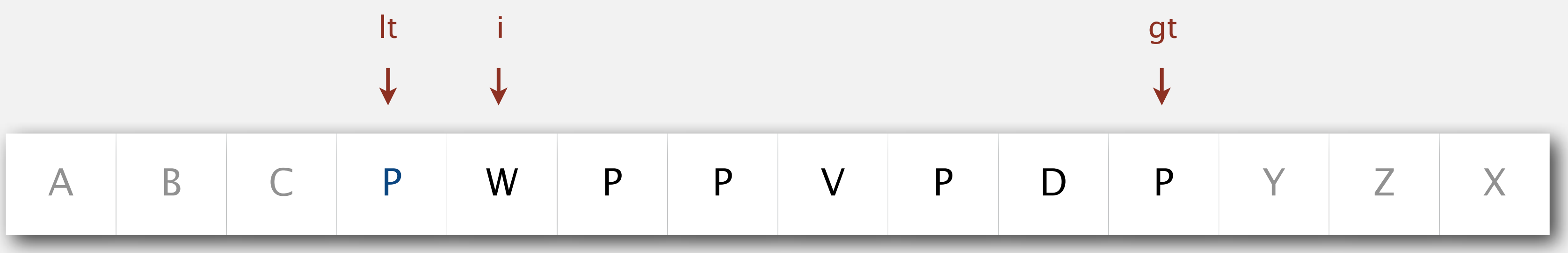


invariant

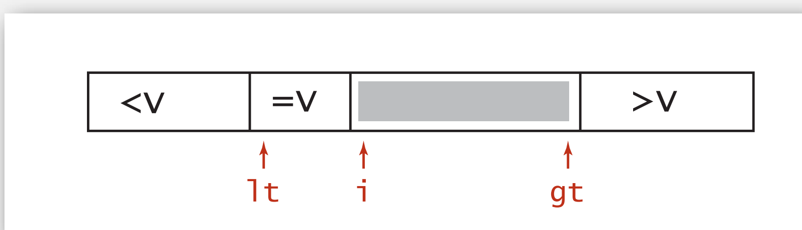


Dijkstra 3-way partitioning

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

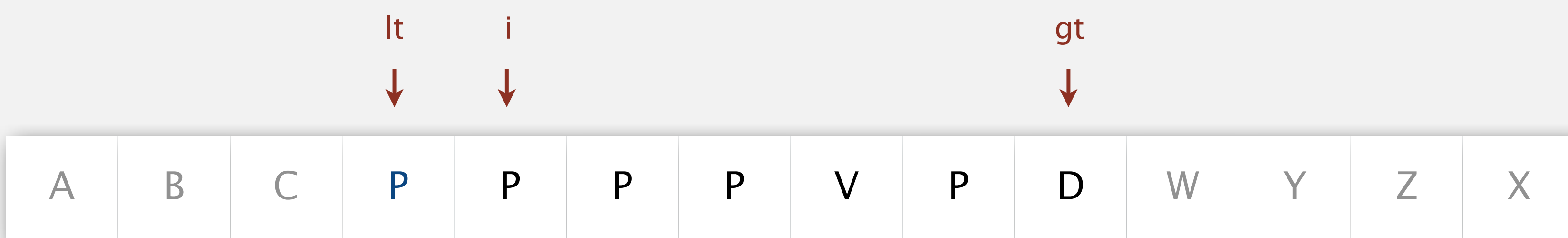


invariant

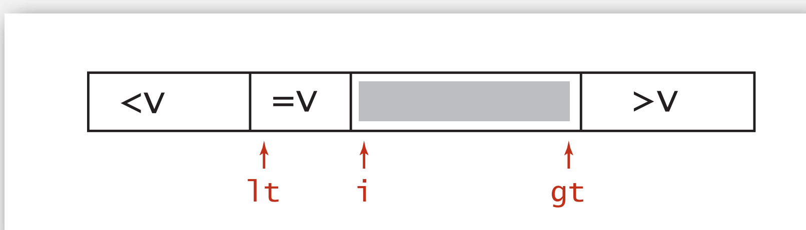


Dijkstra 3-way partitioning

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

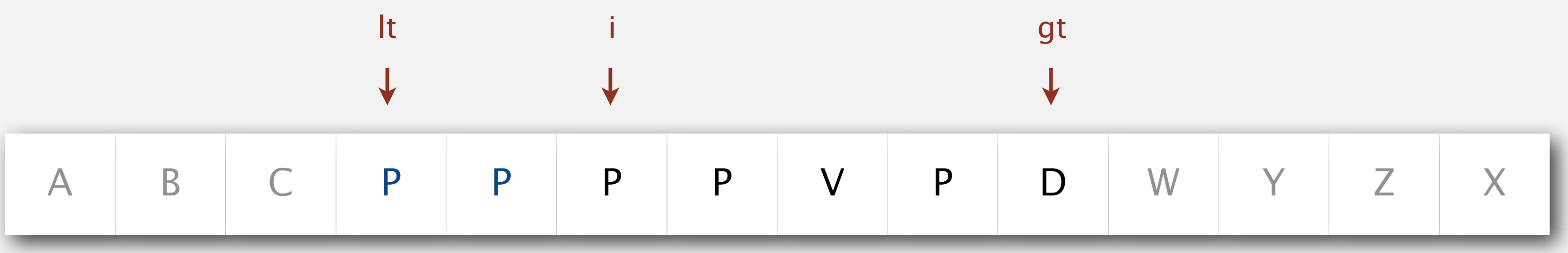


invariant

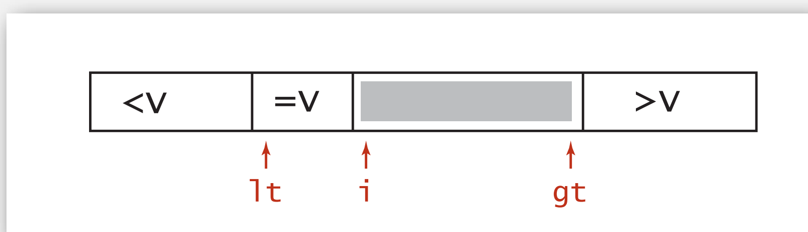


Dijkstra 3-way partitioning

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

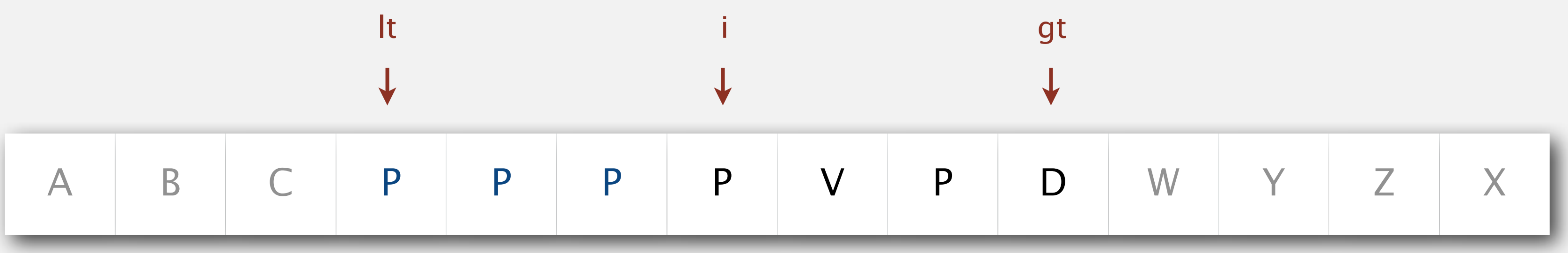


invariant

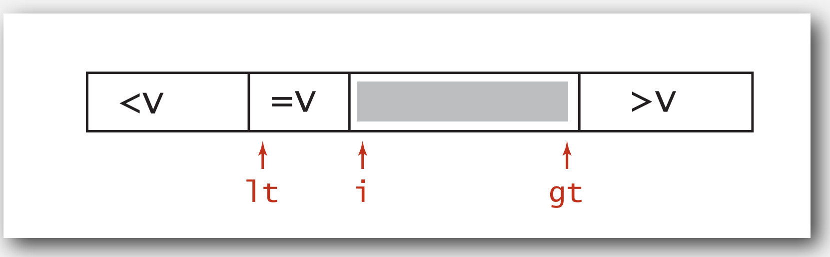


Dijkstra 3-way partitioning

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

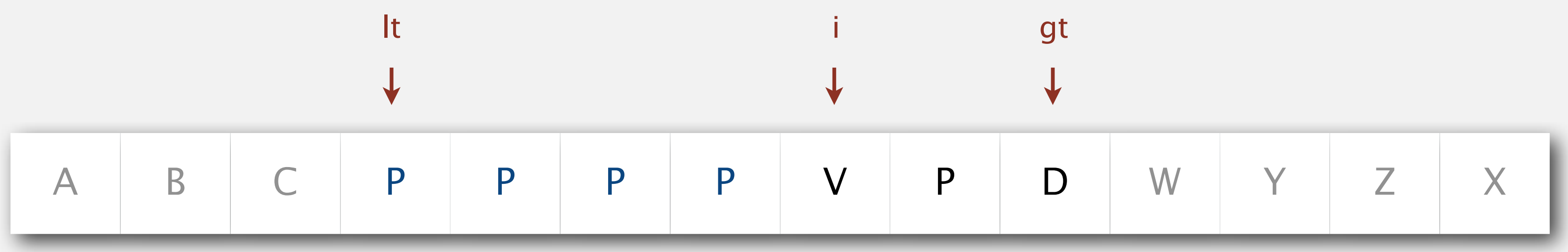


invariant

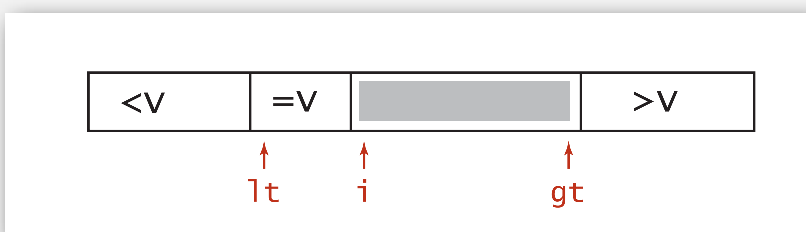


Dijkstra 3-way partitioning

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

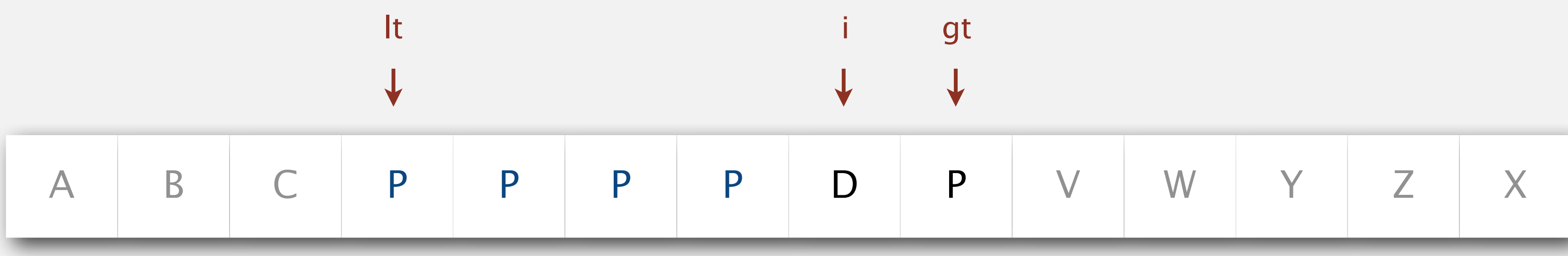


invariant

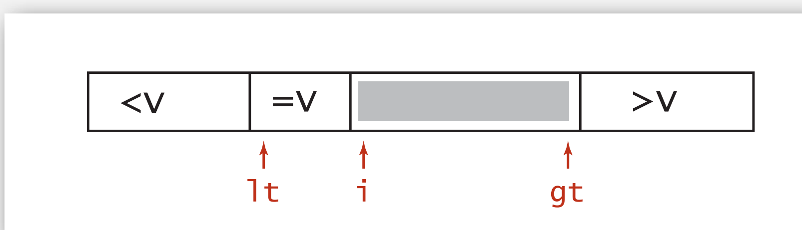


Dijkstra 3-way partitioning

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

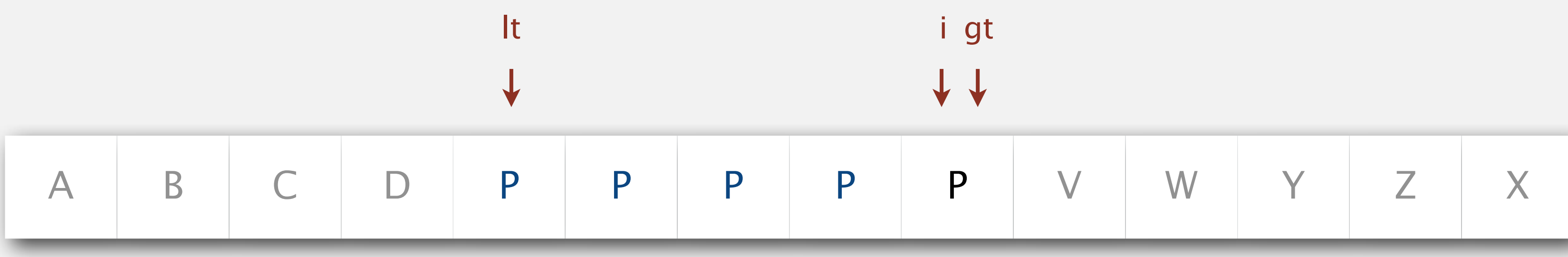


invariant

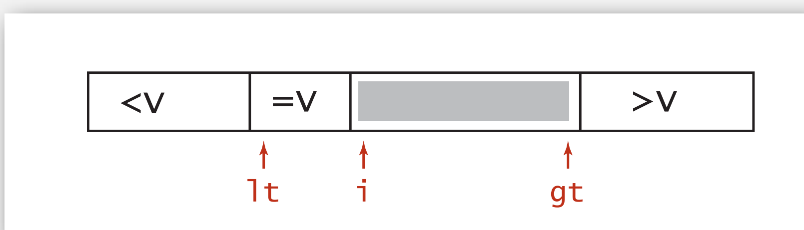


Dijkstra 3-way partitioning

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

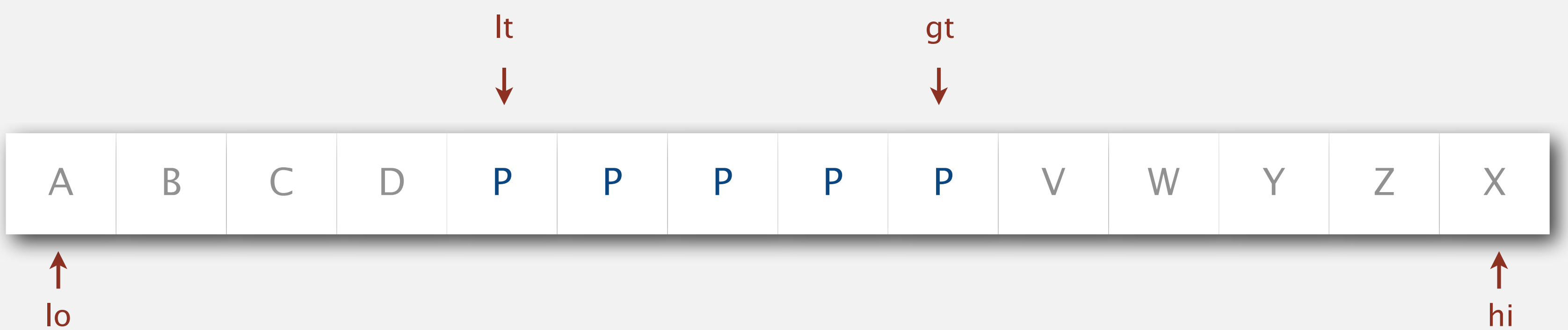


invariant

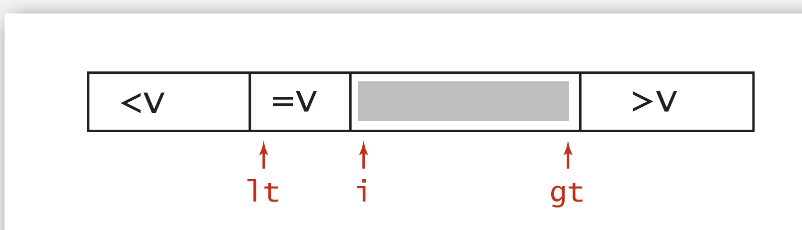


Dijkstra 3-way partitioning

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



invariant



Bentley-McIlroy 3-Way Partitioning

Bentley-McIlroy 3-way 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]`.
- 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`.

Bentley-McIlroy 3-way 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]`.
- 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`.

Bentley-McIlroy 3-way 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]`.
- 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`.

Bentley-McIlroy 3-way 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]`.
- 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`.

Bentley-McIlroy 3-way 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]`.
- 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]`

Bentley-McIlroy 3-way 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]`.
- 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`.

Bentley-McIlroy 3-way 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]`.
- 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`.

Bentley-McIlroy 3-way 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]`.
- 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]`

Bentley-McIlroy 3-way 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]`.
- 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`

Bentley-McIlroy 3-way partitioning

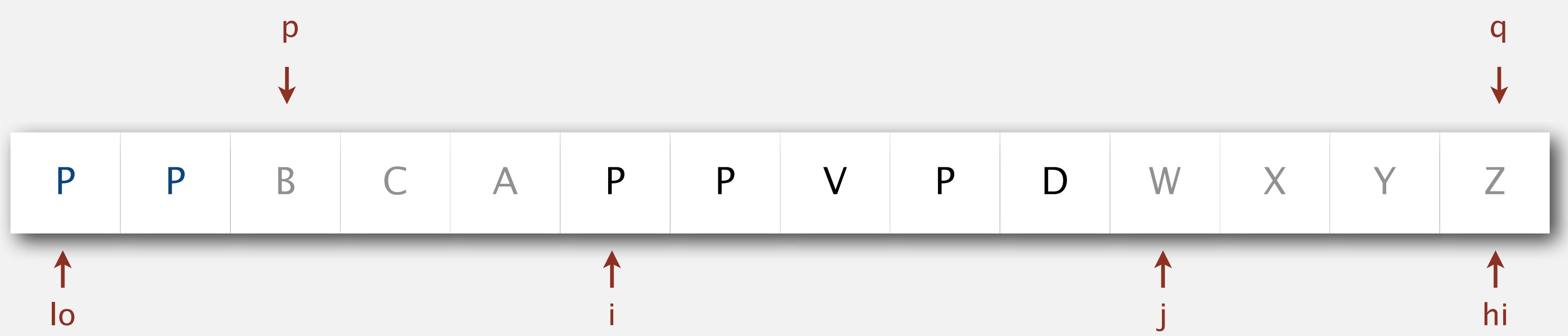
Repeat until `i` and `j` pointers cross.

- 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]`.
- 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`.

Bentley-McIlroy 3-way partitioning

Repeat until `i` and `j` pointers cross.

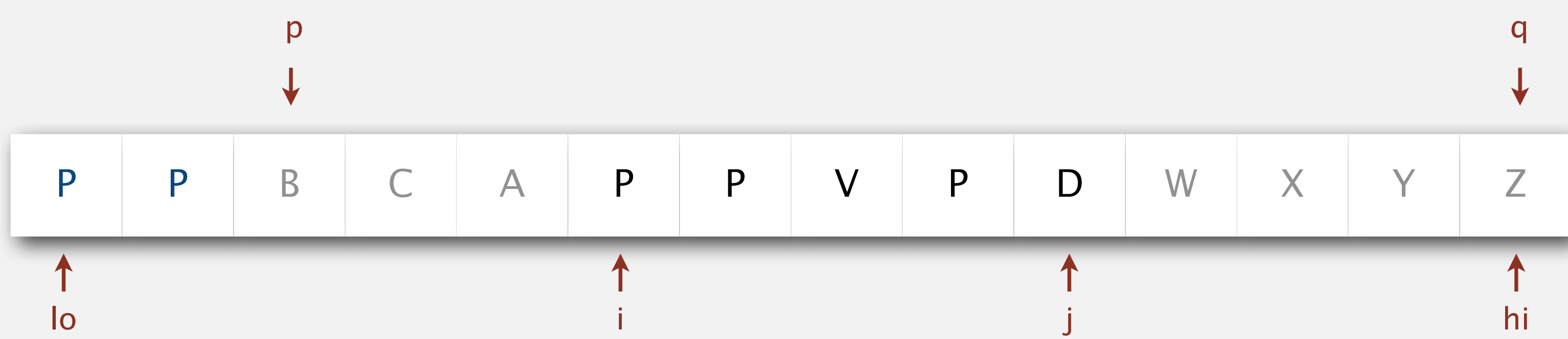
- Scan `i` from left to right so long as `a[i] < a[lo]`.
- Scan `j` from right to left so long as `a[j] > a[lo]`.
- Exchange `a[i]` with `a[j]`.
- If `a[i] == a[lo]`, exchange `a[i]` with `a[p]` and increment `p`.
- If `a[j] == a[lo]`, exchange `a[j]` with `a[q]` and decrement `q`.



Bentley-McIlroy 3-way 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]`.
- 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`.

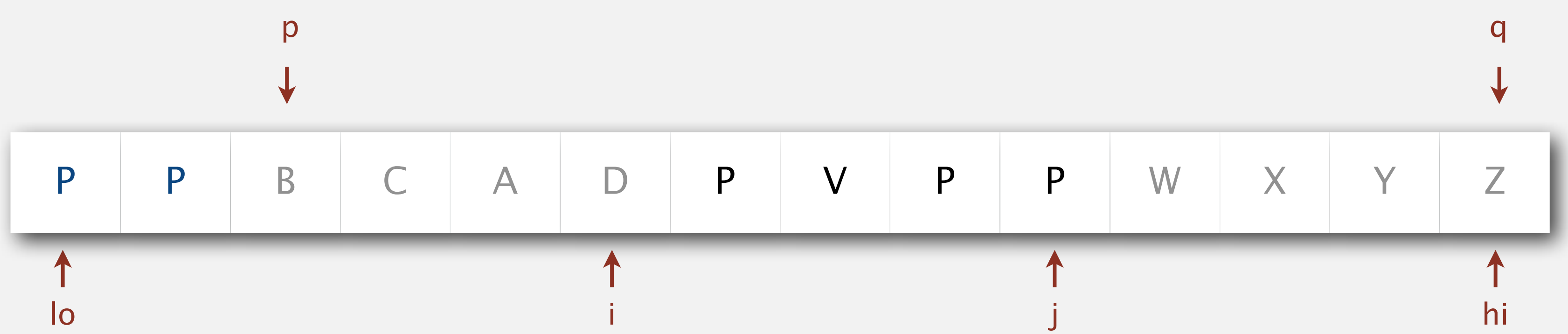


exchange `a[i]` with `a[j]`

Bentley-McIlroy 3-way 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]`.
- 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`.

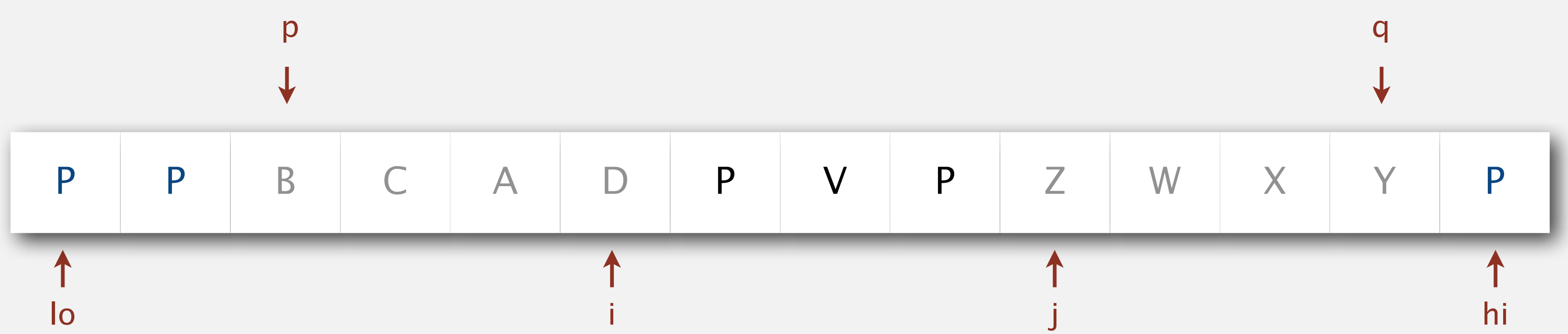


exchange `a[j]` with `a[q]` and decrement `q`

Bentley-McIlroy 3-way 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]`.
- 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`.



Bentley-McIlroy 3-way 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]`.
- 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`.

Bentley-McIlroy 3-way 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]`.
- 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]`

Bentley-McIlroy 3-way partitioning

Repeat until `i` and `j` pointers cross.

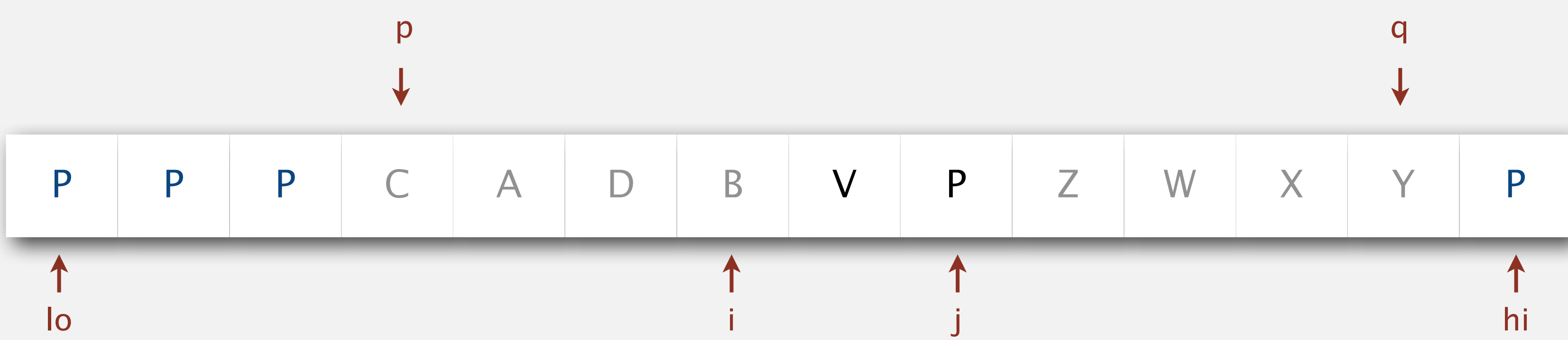
- Scan `i` from left to right so long as `a[i] < a[lo]`.
- Scan `j` from right to left so long as `a[j] > a[lo]`.
- Exchange `a[i]` with `a[j]`.
- If `a[i] == a[lo]`, exchange `a[i]` with `a[p]` and increment `p`.
- If `a[j] == a[lo]`, exchange `a[j]` with `a[q]` and decrement `q`.

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

Bentley-McIlroy 3-way 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]`.
- 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`.

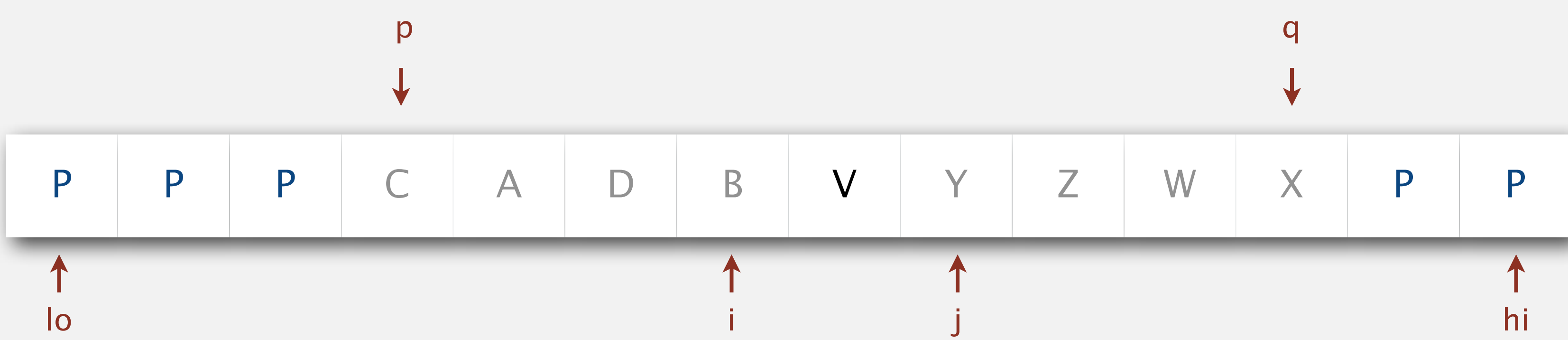


exchange `a[j]` with `a[q]` and decrement `q`

Bentley-McIlroy 3-way partitioning

Repeat until `i` and `j` pointers cross.

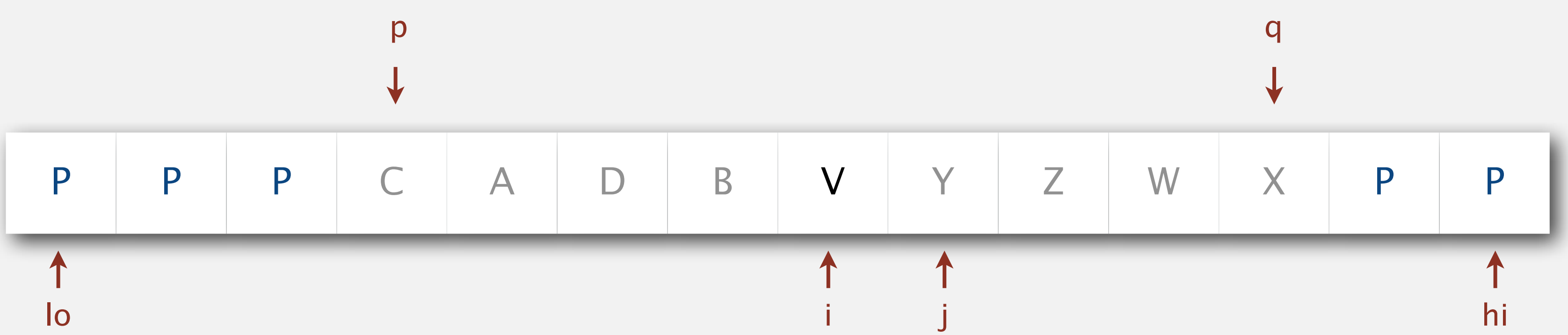
- Scan `i` from left to right so long as `a[i] < a[lo]`.
- Scan `j` from right to left so long as `a[j] > a[lo]`.
- Exchange `a[i]` with `a[j]`.
- If `a[i] == a[lo]`, exchange `a[i]` with `a[p]` and increment `p`.
- If `a[j] == a[lo]`, exchange `a[j]` with `a[q]` and decrement `q`.



Bentley-McIlroy 3-way 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]`.
- 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`.



Bentley-McIlroy 3-way 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]`.
- 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`.

Bentley-McIlroy 3-way 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]`.
- 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`.

pointers cross

Bentley-McIlroy 3-way partitioning

Afterwards, 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]`

Bentley-McIlroy 3-way partitioning

Afterwards, 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]`

Bentley-McIlroy 3-way partitioning

Afterwards, 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]`

Bentley-McIlroy 3-way partitioning

Afterwards, swap equal keys to the center.

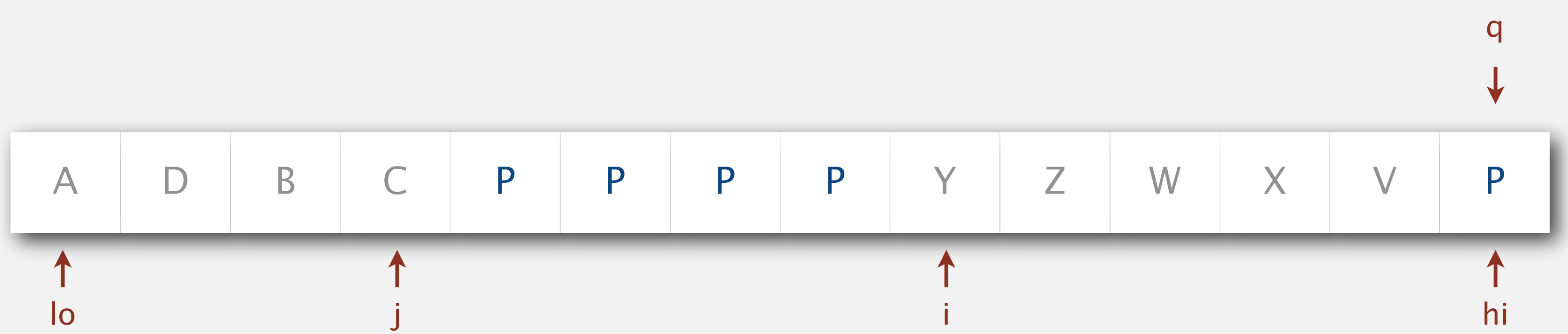
- Scan `j` and `p` from right to left and exchange `a[j]` with `a[p]`.
- Scan `i` and `q` from left to right and exchange `a[i]` with `a[q]`.

exchange `a[i]` with `a[q]`

Bentley-McIlroy 3-way partitioning

Afterwards, 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]`.

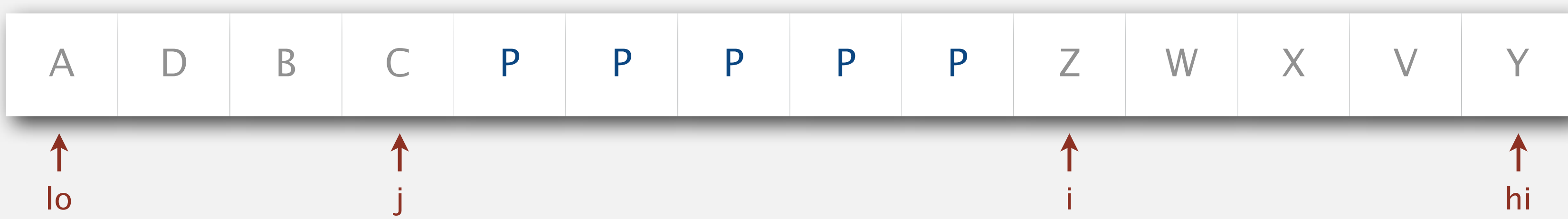


exchange `a[i]` with `a[q]`

Bentley-McIlroy 3-way partitioning

Afterwards, 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