Quicksort

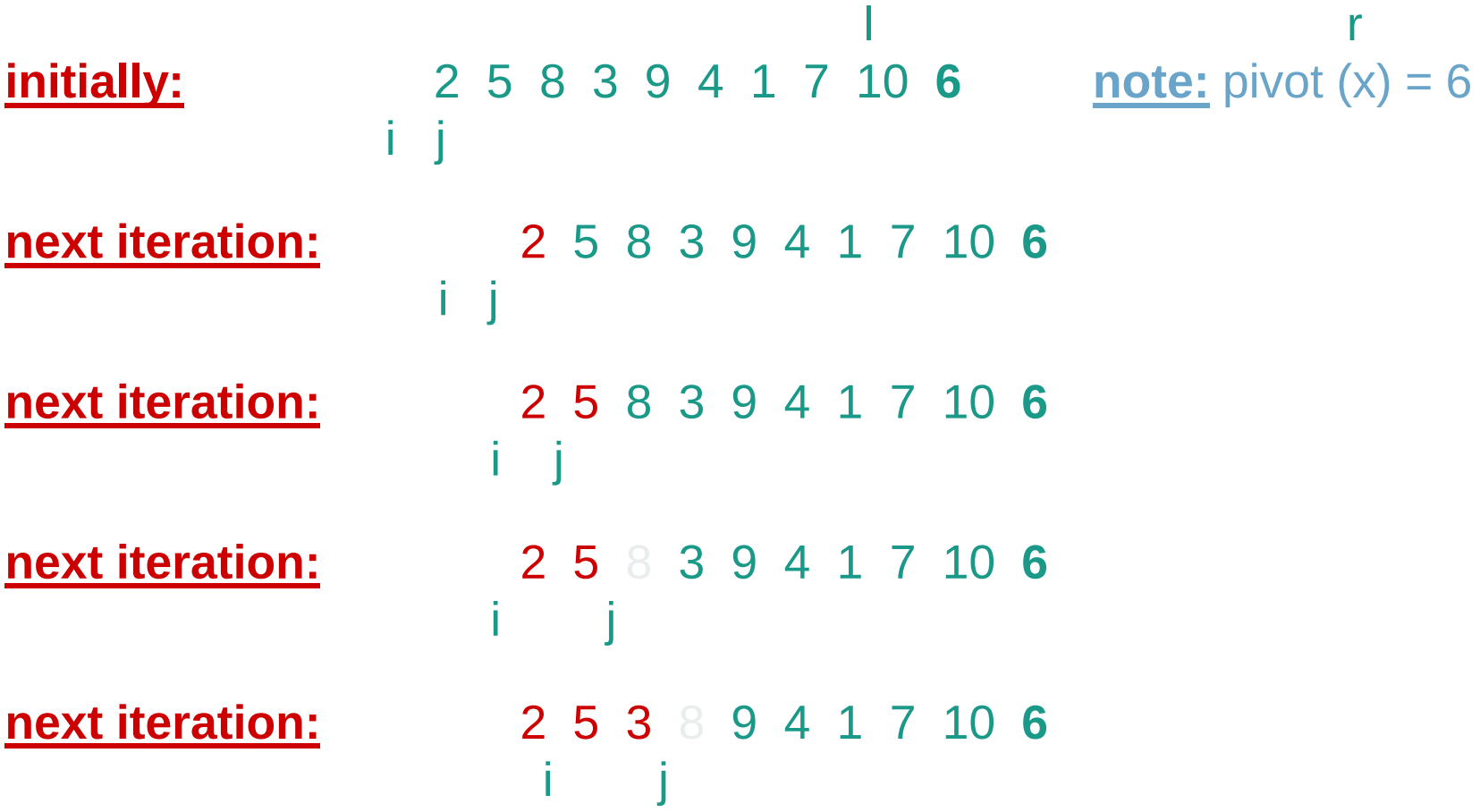
QuickSort Design

- Follows the **divide-and-conquer** paradigm.
- ***Divide*:** Partition (separate) the array $A[l..r]$ into two (possibly empty) subarrays $A[l..p-1]$ and $A[p+1..r]$.
  - Each element in $A[l..p-1] < A[p]$.
  - $A[p] <$ each element in $A[p+1..r]$.
  - Index p is computed as part of the partitioning procedure.
- ***Conquer*:** Sort the two subarrays by recursive calls to quicksort.
- ***Combine*:** The subarrays are sorted in place – no work is needed to combine them.

Partitioning

- Select the last element A[*r*] in the subarray *A*[*l*..*r*] as the *pivot* – the element around which to partition.
- As the procedure executes, the array is partitioned into four (possibly empty) regions.
  1. *A*[*l*..*i*] — All entries in this region are **< *pivot***.
  2. *A*[*i*+1..*j* – 1] — All entries in this region are **> *pivot***.
  3. *A*[*r*] = *pivot*.
  4. *A*[*j*..*r* – 1] — Not known how they compare to *pivot*.
- The above hold before each iteration of the *for* loop, and constitute a *loop invariant*.

Example


l
r
initially:
2 5 8 3 9 4 1 7 10 6
note: pivot (x) = 6
i j
next iteration:
2 5 8 3 9 4 1 7 10 6
i j
next iteration:
2 5 8 3 9 4 1 7 10 6
i j
next iteration:
2 5 8 3 9 4 1 7 10 6
i j
next iteration:
2 5 3 8 9 4 1 7 10 6
i j

Example (Continued)

next iteration: 2 5 3 8 9 4 1 7 10
6

i j

next iteration: 2 5 3 8 9 4 1 7 10
6

i j

next iteration: 2 5 3 4 9 8 1 7 10
6

i j

next iteration: 2 5 3 4 1 8 9 7 10
6

i j

next iteration: 2 5 3 4 1 8 9 7 10
6

i j

next iteration: 2 5 3 4 1 8 9 7 10

Algorithm

```
Quicksort(A, l, r)
        if l < r then
                p := Partition(A, l, r);
                Quicksort(A,
```

```
partition(a, l, r):
   i = (l-1)
   pivot = a[r]
   for j = l to r-1:
        if a[j] <= pivot:
            i = i+1
            swap(a[i], a[j])
    swap(a[i+1], a[r])
    return (i+1)
```