Quicksort

Revised based on textbook author’s notes.

Quick Sort

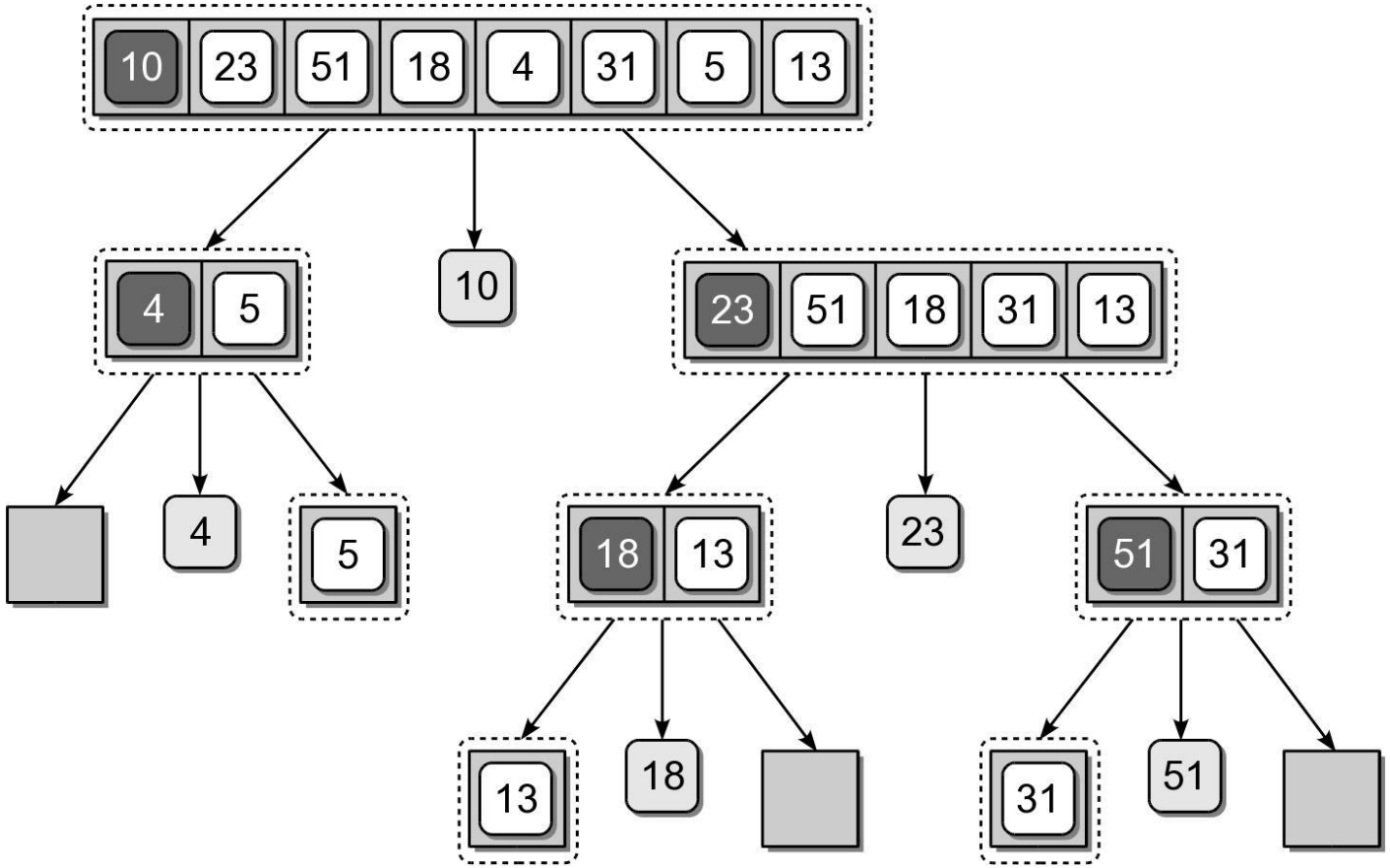
- Uses a divide and conquer strategy to sort the keys stored in a sequence.
  - Partitions the sequence by dividing it into two segments based on a **pivot key**.
  - Uses virtual subsequences without the need for temporary storage.

- Quick sort is a recursive algorithm.

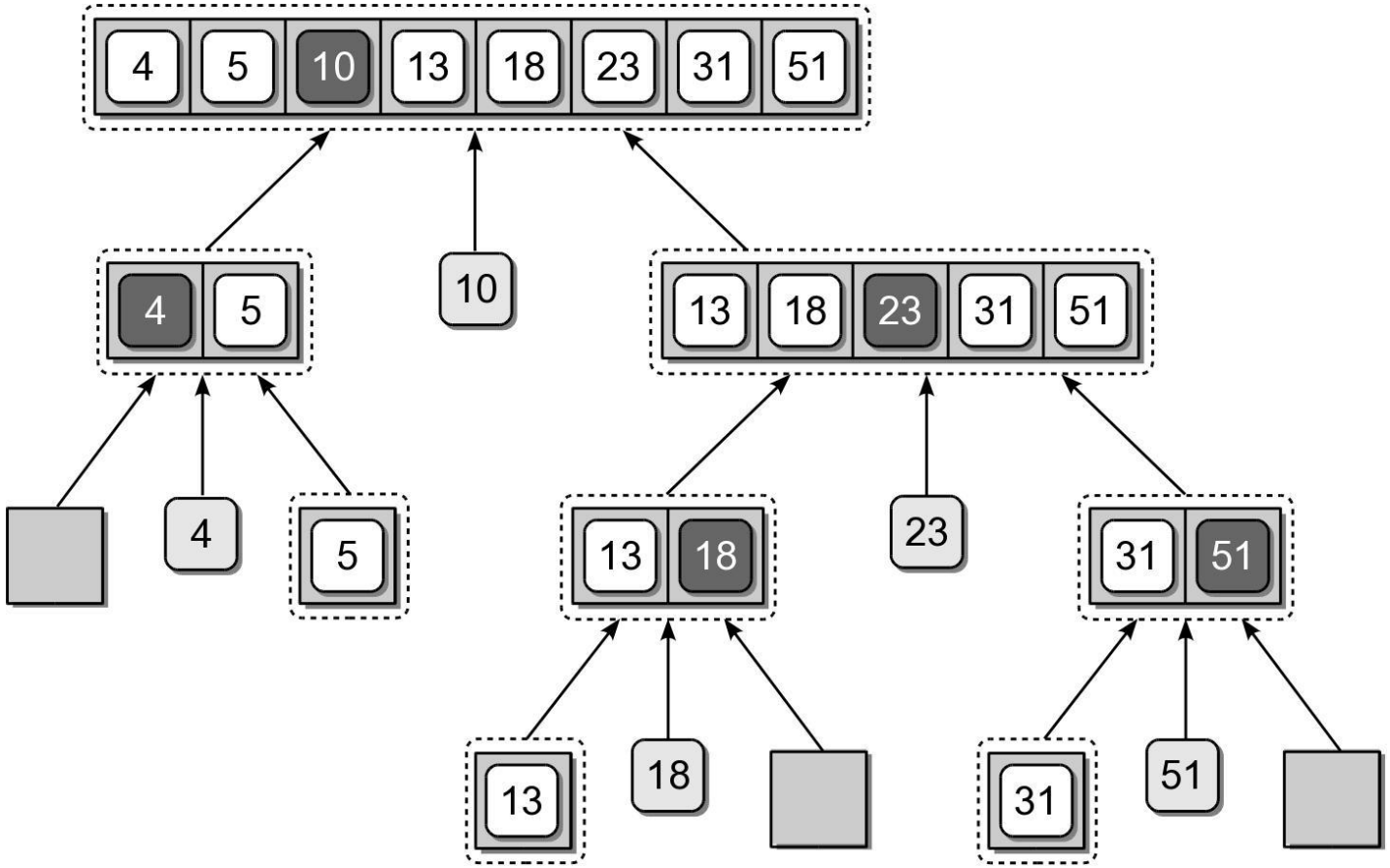
Quick Sort – Description

- Select the first key as the pivot (p)
- Partition the sequence into segments L and G.
  - L contains all keys less than p
  - G contains all keys greater than or equal to p.
- Recursively apply the same operation on L & G.
  - Continues until the sequence contains 0 or 1 key.
- Merge the pivot and two segments back together.

Quick Sort – Divide



Quick Sort – Merge



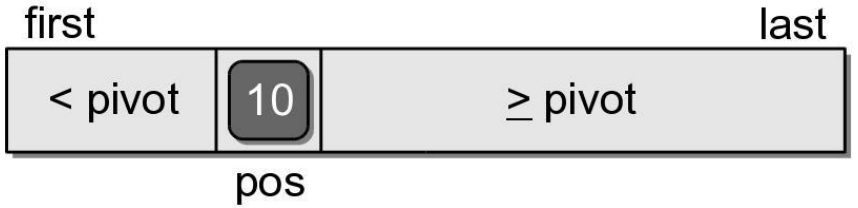
Quick Sort – Implementation

- An efficient solution can be designed.

```
def quickSort( theSeq ):
  n = len( theSeq )
  recQuickSort( theSeq, 0, n-1 )

def recQuickSort( theSeq, first, last ):

  if first >= last :
    return
  else :
     # Partition the sequence and obtain the pivot position.
    pos = partitionSeq( theSeq, first, last )

     # Repeat the process on the two subsequences.
    recQuickSort( theSeq, first, pos - 1 )
    recQuickSort( theSeq, pos + 1, last )
```

Quick Sort – Partition

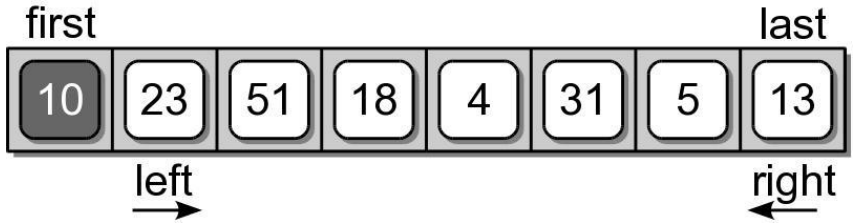
- The partitioning step can be done without having to use temporary storage.
  - Rearranges the keys within the sequence structure.



  - The pivot will be in its correct position within the sequence.
  - Position of the pivot indicates the position where the split occurred.

Quick Sort – Partition

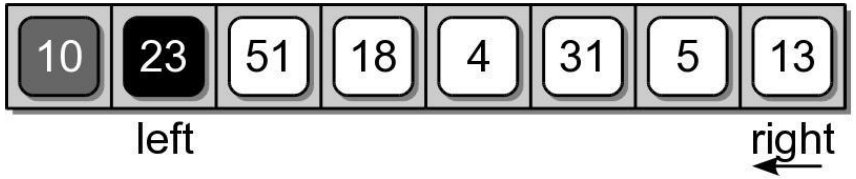
- For illustration, we step through the first complete partitioning.
  - Pivot value is the first key in the segment.
  - Two markers (`left` and `right`) are initialized.



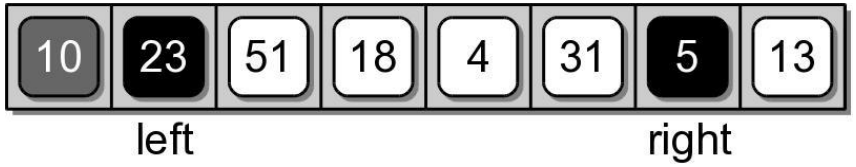
  - The markers will be shifted left and right until they cross each other.

Quick Sort – Partition

- The `left` marker is shifted right until a key value larger than the pivot is found.

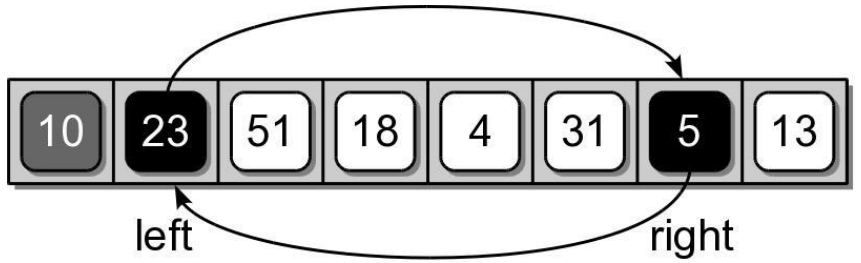


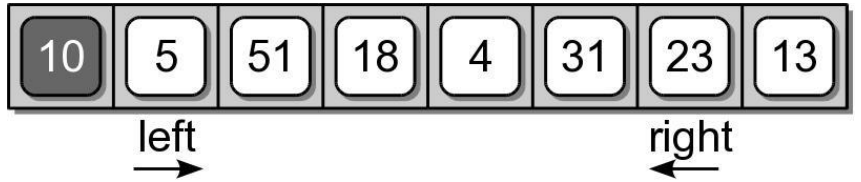
- The `right` marker is then shifted left until a key value less than the pivot is found.



Quick Sort – Partition

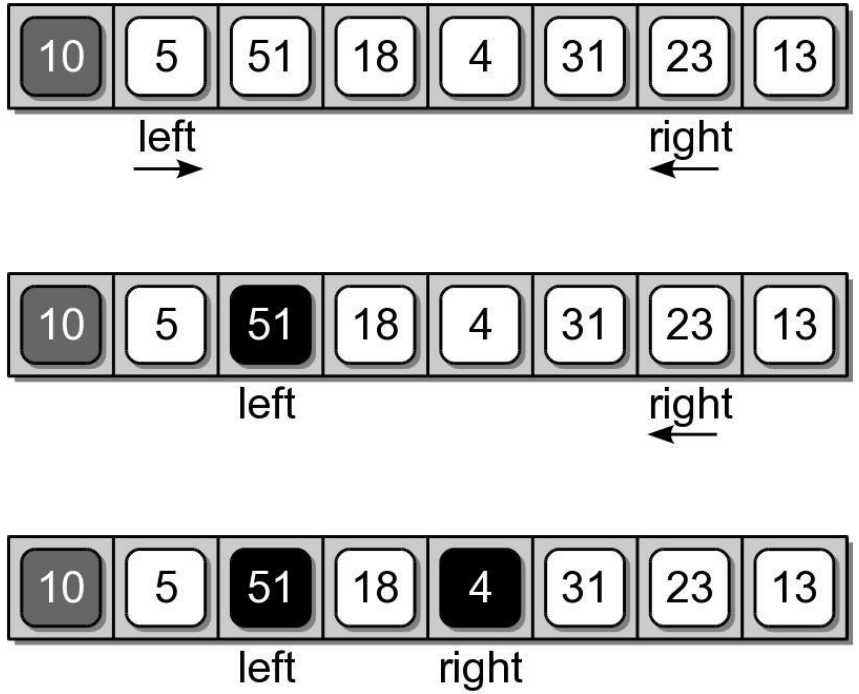
- The two keys at the positions of the `left` and `right` markers are swapped.





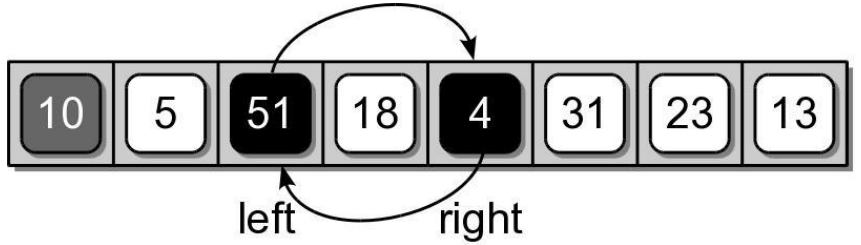
Quick Sort – Partition

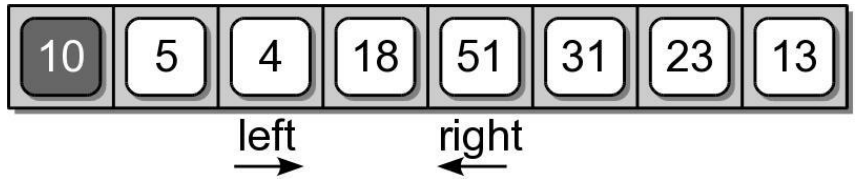
- The two markers are again shifted starting where they left off.



Quick Sort – Partition

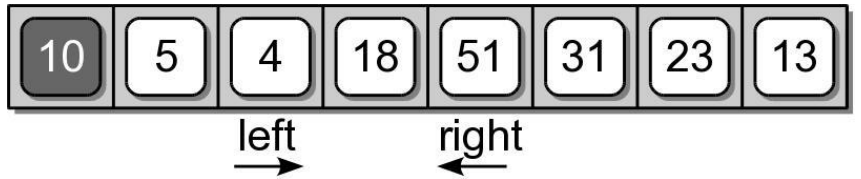
- After the markers are shifted, the corresponding keys are swapped as before.

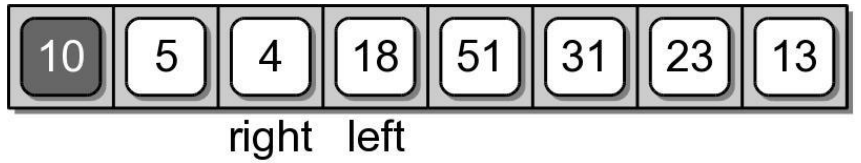




Quick Sort – Partition

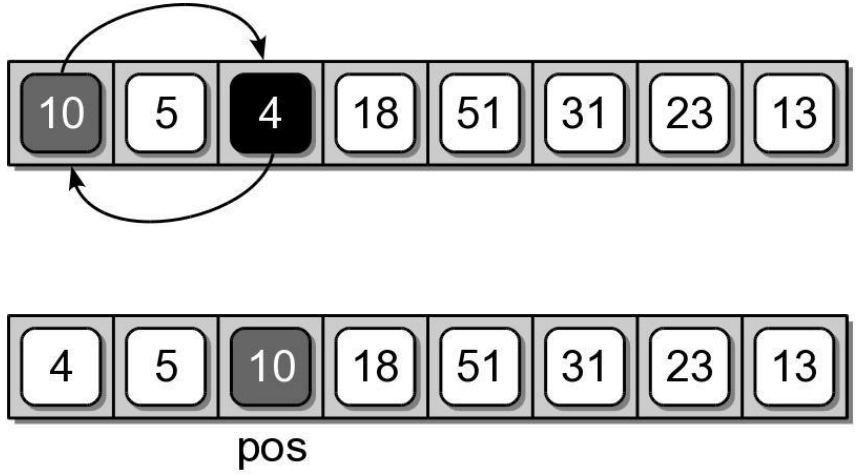
- The shifting and swapping continues until the two markers cross each other.





Quick Sort – Partition

- When the two markers cross, the `right` marker indicates the final position of the pivot value.
  - The pivot value and the value at the right marker have to be swapped.



Quick Sort – Partition

```
def partitionSeq( theSeq, first, last ):
  pivot = theSeq[first]
  left = first + 1
  right = last
  while left <= right :
    while left < right and theSeq[left] < pivot :
      left += 1

    while right >= left and theSeq[right] >= pivot :
      right -= 1

    if left < right :
      tmp = theSeq[left]
      theSeq[left] =  theSeq[right]
      theSeq[right] = tmp

  if right != first :
    theSeq[first] = theSeq[right]
    theSeq[right] = pivot

  return right
```

Pivot Key

- We are not limited to selecting the first key within the sequence as the pivot.
  - Using the first or last key is a poor choice in practice.
  - Choosing a key near the middle is a better choice.

Quick Sort – Efficiency

- The quick sort algorithm:
  - has a worst case time of $O(n^2)$
  - but an average case time of $O(n \log n)$
- It does not require additional storage (in-place).
- Commonly used in language libraries.
  - Earlier versions of Python used quick sort.
  - Current versions use a hybrid that combines the insertion and merge sort algorithms.