University of
Nottingham
UK | CHINA | MALAYSIA

Sorting

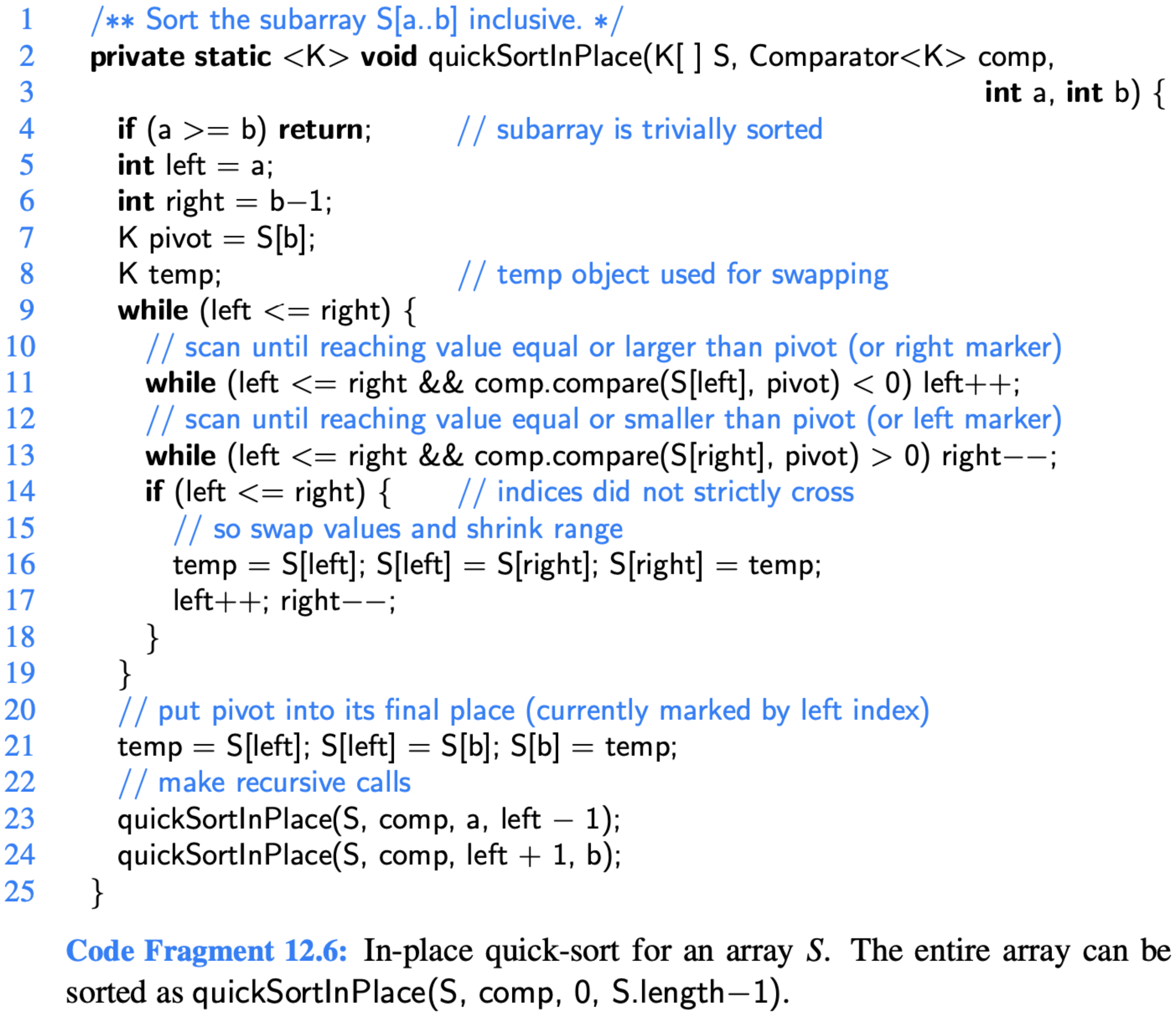
Exercise 1: In-Place Quick Sort

The in-place quick sort takes an array S and two indices a and b as input, and sort the subarray $S[a..b]$. The pivot is $S[b]$.

Apply in-place quick sort over the array below. Show the sorting process stey by step.

85	**24**	**63**	**45**	**17**	**31**	**96**	**50**

In-Place Quick Sort



```
/** Sort the subarray S[a..b] inclusive. */
private static <K> void quickSortInPlace(K[ ] S, Comparator<K> comp,
                                                                   int a, int b) {
  if (a >= b) return;           // subarray is trivially sorted
  int left = a;
  int right = b-1;
  K pivot = S[b];
  K temp;                       // temp object used for swapping
  while (left <= right) {
    // scan until reaching value equal or larger than pivot (or right marker)
    while (left <= right && comp.compare(S[left], pivot) < 0) left++;
    // scan until reaching value equal or smaller than pivot (or left marker)
    while (left <= right && comp.compare(S[right], pivot) > 0) right--;
    if (left <= right) {        // indices did not strictly cross
      // so swap values and shrink range
      temp = S[left]; S[left] = S[right]; S[right] = temp;
      left++; right--;
    }
  }
  // put pivot into its final place (currently marked by left index)
  temp = S[left]; S[left] = S[b]; S[b] = temp;
  // make recursive calls
  quickSortInPlace(S, comp, a, left - 1);
  quickSortInPlace(S, comp, left + 1, b);
}
```

Code Fragment 12.6: In-place quick-sort for an array *S*. The entire array can be sorted as quickSortInPlace(S, comp, 0, S.length−1).