Sample Assignment 9

Not graded

Pseudocode for Quicksort. This variant of Quicksort uses the leftmost element of the input area as a pivot. It is the same as we have in the lecture slides, but may differ from some other Quicksort flavors (randomized etc.) that you may encouter in other sources.

```
QUICKSORT($A[\ell \ \dots \ r]$) :
 1  if $l < r$ :
 2      $i = \ell$        (i increases from the left and searches elements $\geq$ than pivot)
 3      $j = r + 1$   (j decreases from the right and searches elements $\leq$ than pivot.)
 4      $v = A[\ell]$    (v is the pivot.)
 5      while $i < j$ :
 6          $i = i + 1$
 7          while $i < r$ and $A[i] < v$ :
 8              $i = i + 1$
 9          $j = j - 1$
10          while $j > \ell$ and $A[j] > v$ :
11              $j = j - 1$
12          $A[i] \leftrightarrow A[j]$  (Undo the extra swap at the end)
13      $A[i] \leftrightarrow A[j]$  (Undo the extra swap at the end)
14      $A[j] \leftrightarrow A[\ell]$  (Move pivot to its proper place)
15      QUICKSORT($A[\ell \ \dots \ j-1]$)
16      QUICKSORT($A[j+1 \ \dots \ r]$)
```

(A) Run this pseudocode for one invocation QUICKSORT($A[0..11]$), where the table to sort is the following:

$$13, 0, 23, 1, 8, 9, 29, 16, 8, 24, 6, 11$$

Draw the state of the array every time you swap two elements (i.e. execute $A[k_1] \leftrightarrow A[k_2]$ for any k_1, k_2).

(B) Continue with the first recursive call of QUICKSORT() (the original call QUICKSORT($A[0..11]$) is assumed to be the 0th call of this function). Draw the state of the array every time you swap two elements.

(C) Decide which is the second recursive call of QUICKSORT() and draw the state of the array every time you swap two elements. Show the end-result after this second recursive call at the very end.

Solution.
Your answer can be simple lists of numbers (without any grid lines or additional markings). Just try to keep the lists of numbers aligned.

(A)

	0	1	2	3	4	5	6	7	8	9	10	11
	13	0	23	1	8_2	9	29	16	8_1	24	6	11
	13	0	**11**	1	8_2	9	29	16	8_1	24	6	**23**
after Line 12	13	0	11	1	8_2	9	**6**	16	8_1	24	**29**	23
	13	0	11	1	8_2	9	6	**8_1**	**16**	24	29	23
	13	0	11	1	8_2	9	6	**16**	**8_1**	24	29	23
Line 13	13	0	11	1	8_2	9	6	**8_1**	**16**	24	29	23
Line 14	8_1	0	11	1	8_2	9	6	13	16	24	29	23

Figure 1: Swaps during the 0th call.

(B) *Since this example contains two elements equal to 8, we added subscripts to them (to show clearly, where every one is being swapped). As integer numbers they are fully identical to the Quicksort algorithm. (Still, the Quicksort algorithm does redundant swaps on them.)*

	8_1	0	11	1	8_2	9	6	13	16	24	29	23
	8_1	0	**6**	1	8_2	9	**11**	13	16	24	29	23
	8_1	0	6	1	**8_2**	9	11	13	16	24	29	23
Line 13	8_1	0	6	1	**8_2**	9	11	13	16	24	29	23
Line 14	8_2	0	6	1	8_1	9	11	13	16	24	29	23

Figure 2: Swaps during the first recursive call.

(C) *Notice that the second recursive call happens within the first recursive call (sorting the left side of the left half).*

	8_2	0	6	1	8_1	9	11	13	16	24	29	23
	8_2	0	6	**1**	8_1	9	11	13	16	24	29	23
Line 13	8_2	0	6	**1**	8_1	9	11	13	16	24	29	23
Line 14	1	0	6	8_2	8_1	9	11	13	16	24	29	23

Figure 3: Swaps during the second recursive call.