Exercise 7.1-2*

What value of \( q \) does PARTITION return when all elements in the array \( A[p..r] \) have the same value? Modify PARTITION so that \( q = \lfloor (p + r)/2 \rfloor \) when all elements in the array \( A[p..r] \) have the same value.

Solution:

(a).

PARTITION(A, p, r)
1   \( x = A[r] \)
2   \( i = p - 1 \)
3   \textbf{for} \( j = p \text{ to } r - 1 \)
4       \textbf{if} \( A[j] \leq x \)
5           \( i = i + 1 \)
6       \textbf{exchange} \( A[i] \) with \( A[j] \)
7   \textbf{exchange} \( A[i + 1] \) with \( A[r] \)
8   \textbf{return} \( i + 1 \)

Since the \textbf{if} test on the analogue of line 4 is thus successful when all entries are equal, in that case the value of \( k \) when the \textbf{for} loop terminates is \( k = r-1 \), so the value of \( q \) returned is \( q = k+1 = r \).

(b).

PARTITION'(A, p, r)  //modified version
1   \( x = A[r] \)
2   \( k = p - 1 \)
3   \( l = p - 1 \)
4   \textbf{for} \( j = p \text{ to } r - 1 \)
5       \textbf{if} \( A[j] < x \)
6           \( k = k + 1 \)
7       \textbf{exchange} \( A[j] \) with \( A[k] \)
8           \( l = l + 1 \)
9       \textbf{else}
10          \textbf{if} \( A[j] = x \)
11             \( l = l + 1 \)
12          \textbf{exchange} \( A[j] \) with \( A[l] \)
13          \textbf{exchange} \( A[l + 1] \) with \( A[r] \)
14   \textbf{return} \( \left( A, \lfloor (l + k)/2 \rfloor + 1 \right) \)
7.2-5

Suppose that the splits at every level of quicksort are in the proportion $1 - \alpha$ to $\alpha$, where $0 < \alpha \leq 1/2$ is a constant. Show that the minimum depth of a leaf in the recursion tree is approximately $-\log n / \log \alpha$ and the maximum depth is approximately $-\log n / \log(1 - \alpha)$.

(Don’t worry about integer round-off.)

**Solution.**

The minimum depth occurs for the path that always takes the smaller portion of the split, i.e., the nodes that take $\alpha$ proportion of work from the parent node. The first node in the path (after the root) gets $\alpha$ proportion of work (the size of data processed by this node is $\alpha n$), the second one gets $\alpha^2$ so on. The recursion bottoms out when the size of data becomes 1. Assuming the recursion ends at level $m$, we have:

$$a^m n = 1$$
$$a^m = 1/n$$
$$m = \log(1/n) / \log \alpha = -\log n / \log \alpha$$

Maximum depth $h$ is similar with minimum depth:

$$(1 - \alpha)^h n = 1$$
$$(1 - \alpha)^h = 1/n$$
$$h = \log(1/n) / \log (1 - \alpha) = -\log n / \log(1 - \alpha)$$

7-4 **Stack depth for quicksort**

The QUICKSORT algorithm of Section 7.1 contains two recursive calls to itself. After QUICKSORT calls PARTITION, it recursively sorts the left subarray and then it recursively sorts the right subarray. The second recursive call in QUICKSORT is not really necessary; we can avoid it by using an iterative control structure. This technique, called **tail recursion**, is provided automatically by good compilers. Consider the following version of quicksort, which simulates tail recursion:

```plaintext
TAIL-RECURSIVE-QUICKSORT (A, p, r)
1 while p < r
2    // Partition and sort left subarray.
3    q = PARTITION (A, p, r)
4    TAIL-RECURSIVE-QUICKSORT (A, p, q -1)
5    p = q + 1
```

**a.** Argue that TAIL-RECURSIVE-QUICKSORT($A$, 1, $A$.length) correctly sorts the array A.
Answer: The book proved that QUICKSORT correctly sorts the array A. TAIL-RECURSIVE-QUICKSORT differs from QUICKSORT in only the last line of the loop. It is clear that the conditions starting the second iteration of the while loop in TAIL-RECURSIVE-QUICKSORT are identical to the conditions starting the second recursive call in QUICKSORT. Therefore, TAIL-RECURSIVE-QUICKSORT effectively performs the sort in the same manner as QUICKSORT. Therefore, TAIL-RECURSIVE-QUICKSORT must correctly sort the array A.