Problem Set 8

Assigned: Nov. 9
Due: Nov. 28

Problem 1
Show a trace of quicksort operating on the following array (length=18):

\[34,14,43,22,50,4,18,62,36,26,6,29,54,71,25,39,8,2\]

Use the in-place version of quicksort, and produce a trace in the style of Example 2 in the class notes on Quicksort. Assume that smallSize=3.

Problem 2
Suppose that you are given the problem of returning in sorted order the \( k \) smallest elements in an array of size \( n \), where \( k \) is much smaller than \( n \), but much larger than 1.

Describe how quicksort can be adapted to this problem. Your description need not give the pseudo-code for the modified algorithms; it is enough simply to describe what changes can be made, as long as your description is clear.

Problem 3
Let us say that an array \( A \) of length \( n \) is *almost sorted with errors of size* \( k \) for \( k < n \) if, for any \( I, J \), if \( J - I > k \) then \( A[J] \geq A[I] \). Thus, the array does not have to be completely ordered, but any two elements in the array that are out of order cannot be more than \( k \) places apart. For example, the list

\[50, 80, 10, 60, 150, 120, 110, 200, 190, 250, 300, 350, 320\]

is almost sorted with errors of size 2. \( A[3] = 10 \) is less than \( A[1] = 50 \), and \( 3 - 1 = 2 \), but there are no elements out of order that are 3 or more steps apart.

a. Show how quicksort can be modified to produce a list that is almost sorted with errors of size \( k \). What is the *best case* running time of this modified quicksort?

b. If the input array is almost sorted with errors of size \( k \), what is the running time of insertion sort? Justify your answer.
Problem 4

Consider the problem of finding the \( k \)th smallest element in an array.

This problem can be solved by an algorithm of a similar structure to quicksort as follows. We will use a recursive method \( \text{findKth}(a, l, u, k) \) which looks between indices \( l \) and \( u \) for the \( k \)th smallest element of \( a \).

We can write \( \text{findKth} \) as follows:

\[
\text{findKth}(a, l, u, k) \{
\text{if (} k < l \text{ || } k > u \text{) raise error; // invalid value of } k
\text{if (} l == u \text{) return } a[l]; \quad \text{// base case}
\text{m = partition}(a, l, u);
\text{if (???) return } a[m];
\text{else if (???)
\quad \text{return } \text{findKth}(a,???,???,???); \quad \text{// search in the first part}
\text{else}
\quad \text{return } \text{findKth}(a,???,???,???); \quad \text{// search in the second part}
\}
\]

The \text{partition} function here is the same one as in the usual quicksort. It returns the index in \( a \) where the pivot ends up.

Fill in the question marks in the code above.