Problem Set 7

Assigned: Nov. 20
Due: Nov. 29

Problem 1

Consider the problem of sorting the following array (length=14):

\[ [42, 6, 69, 17, 14, 38, 75, 52, 2, 11, 8, 56, 17, 85] \]

a. Produce a trace of mergesort working on this array. Use the in-place version of mergesort, and produce a trace in the style of Example 2 in the class notes on mergesort. Assume that smallSize=4.

b. Produce a trace of quicksort working on this array. 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=4.

Note: it is much easier to generate these traces using a text editor, where you can use copy/paste, than to write them out by hand. However, be careful to avoid copy/paste errors.

Problem 2

Let us say that an array \( a \) of length \( n \) is almost sorted with errors of distance \( k \) 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 array

\[ [50, 80, 70, 60, 150, 120, 110, 200, 160, 250, 300, 350, 320] \]

is almost sorted with errors of distance 2. For example \( a[3] = 60 \) is less than \( a[1] = 80 \), 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 \( k \).

b. If the input array is almost sorted with errors of size \( k \), what is the running time of insertion sort, as a function of \( k \) and \( n \)? Explain your answer.

Problem 3

In a different sense, we can say that an numerical array \( a \) is almost sorted with errors of size \( d \), if, for all indices \( j > i \), \( a[j] \leq a[i] + d \). For instance, the same array from problem 2 is almost sorted with errors of size 40, since \( a[7]=200 \) and \( a[8]=160 \) but no two elements that are out of order have a difference greater than 40.

Suppose that the keys are all floating point numbers between two bounds \( b \) and \( t \). Assume that \( d \gg (t-b)/n \). Show how bucket sort (my bucket sort, not GTG’s) can be modified to sort the keys efficiently.
Problem 4

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 5

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:

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