Problem Set 2

Assigned: June 3
Due: June 10

Problem 1.
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 \( \log(n) \), but much larger than 1.

a. Describe how selection sort, insertion sort, mergesort, and heapsort 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. You may use the recursive version of mergesort.

b. Find the worst-case running times of these algorithms as functions of \( k \) and \( n \).

Problem 2.
Give an \( O(n \cdot \log(k)) \) time algorithm to merge \( k \) sorted lists into one sorted list, where \( n \) is the total number of elements in all the input lists. (Hint: Use a heap for \( k \)-way merging.)

Problem 3.
(Siegel) Consider the following sorting problem. The input is a sequence of \( n \) integers with many duplications, such that the number of distinct integers in the sequence is \( O(\log(n)) \). Design a sorting algorithm to sort such sequences using at most \( O(n \log \log n) \) comparisons in the worst case.

Problem 4.
(Siegel) Design an efficient algorithm to determine whether two unsorted sets of \( m \) and \( n \) integers are disjoint. Assume that \( m < n \). Full credit will be given for an algorithm that runs in time \( O(n \log m) \); half credit will be given for an algorithm that runs in time \( O(n \log n + m \log m) \) (which is the same thing as \( O(n \log n) \).