Question 1. (25 points) Let the procedure `Median`(S) return the median of a set S, that is, the $(|S|/2)$th smallest element in S; S is not given in sorted order. Let `Select`(S,k) return the kth smallest element of S (here too, S is not given in sorted order). Prove that the problem of designing an $O(|S|)$ time algorithm for `Median`(S) is equivalent to the problem of designing an $O(|S|)$ time algorithm for `Select`(S,k). That is, you have to show that a linear time algorithm for either of these problems, implies a linear time algorithm for the other (in both directions – of course one of the two directions is trivial but write it down nevertheless).

Question 2. (25 points) Given two sorted (by increasing order) arrays A and B of n elements each, give an $O(\log n)$ time algorithm for finding the nth smallest item among the $2n$ elements in $A \cup B$. Assume that n is a power of 2, and that the $2n$ items are distinct (i.e., that no two of them are equal).

Question 3. (25 points) Let $S = \{I_1, \ldots, I_n\}$ be a set of intervals where $I_i = [l_i, r_i]$, $l_i < r_i$. The intervals are given sorted according to their r_is (hence $r_1 < r_2 < \ldots < r_n$). For simplicity, we assume that no two of the l_is and r_is coincide (i.e., there are $2n$ distinct interval endpoints). Each interval has a positive weight w_i (that may be quite different from the length $r_i - l_i$ of that interval). The weight of a subset of S is the sum of the weights of the intervals in that subset. A subset of S is *acceptable* if none of its intervals overlap. Give an $O(n^2)$ time algorithm for computing the maximum weight of an acceptable subset $\hat{S}$ of S. You need only compute the maximum possible weight, not the subset $\hat{S}$ that achieves it.

Hint. Use dynamic programming.

Comment. An $O(n \log n)$ time solution exists but you are not responsible for it.

Question 4. (25 points) Let A be an array of n arbitrary and distinct numbers. A has the following property: If we imagine B as being the sorted version of A, then any element that is at position i in array A would, in B, be at a position j such that $|i - j| \leq k$. In other words, each element in A is not farther than k positions away from where it belongs in the sorted version of A. Suppose you are given such an array A, and you are told that A has this property for a particular value of k (that value of k is also given to you). Design an $O(n \log k)$ time algorithm for sorting A. Briefly explain why your algorithm is correct.

Hint. Do $O(n/k)$ times something that runs in $O(k \log k)$ time each – hence the total time is $O(n \log k)$ time.

Date due: September 13