Problem Set 2

This problem set is due in class on October 12th (October 14th for CVN students).

When a problem asks to give an algorithm, in your solution: (i) describe shortly and informally the main ideas in your solution; (ii) give a detailed description of the algorithm, using a style similar to (but possibly more concise than) the pseudo-code of CLR; (iii) prove the correctness of the algorithm; (iv) prove a bound on the time complexity of the algorithm.

In a data structure problem, give a single informal description of your ideas and then, for each operator, give separate detailed descriptions, times bounds, and correctness.

Problem 1. More divide-and-conquer

Let $A$ and $B$ be two sorted (in non decreasing order) vectors with $n$ elements each. You can assume that the elements on each vector are distinct, and that the vectors do not have common elements.

Give a $O(\log n)$ algorithm for finding the median of $A \cup B$.

Problem 2. Intersection

Let $A$ and $B$ be two sorted (in non decreasing order) vectors with $n$ elements each. You can assume that the elements on each vector are distinct.

Give an $O(n)$ algorithm that receiving as input $A$ and $B$ outputs $A \cap B$.

Problem 3. Data Structures

Consider the Set data structure, and assume that the keys are in the range $1, \ldots, M$. Consider the following implementation. The Set is represented as a Boolean vector $V$ with $M$ entries. Insertion of element $i$ is implemented by setting $V[i] = 1$; deletion of element $i$ is implemented by setting $V[i] = 0$; testing whether element $i$ belongs to the set is done by testing whether $V[i] = 1$. Unfortunately initializing all elements of $V[]$ to 0 requires $O(M)$ time.

Devise an alternative implementation that still uses $O(M)$ memory, and where each operation, including initialization, is doable in $O(1)$ worst case time. The data structure should
be able to handle an unbounded number of insertion/deletion. Partial credit will be given to a solution that only implements find and insert, or to a solution that can handle only a sequence of $O(M)$ insertions/deletions.

You can assume that in constant time you can allocate memory for a vector of arbitrary size, but the content of the vector will be unpredictable. Your implementation must work correctly independently of the initial vectors (if any) that you allocate.

[Hint: store the elements into a stack $S$; also use a vector $V$ of $M$ pointers into the stack. Be careful in implementing find, so that it cannot mistakenly find an element that wasn’t previously inserted. Be very careful in implementing delete.]