Department of Computer Science and Engineering
Mid Semester Examination

Course No: CS 213 Course Name: Data Structures and Algorithms
Date: 5/10/2020 Time 8-00 a.m. to 11-00 a.m Marks: 30

Q1(a) Define the functions `min(x,y)` and `max(x,y)` for natural numbers `x,y`, using only the axioms of numbers. These are the usual minimum and maximum of two numbers. (3)

(b) Suppose addition is defined by `add(x,0) = x` and `add(x,next(y)) = next(add(x,y))` for all numbers `x,y`. Prove that for all numbers `x,y`, `add(x,y) = add(max(x,y),min(x,y))`. (3)

(c) Prove that `add(min(x,y),z) = min(add(x,z),add(y,z))` for all numbers `x,y,z`. The proofs must be complete and not assume anything other than the axioms and the definitions of the functions. (4)

Q2. A data structure is required to represent equivalence relations on the set $\{0, 1, \ldots, n-1\}$. This is the same as partitions of the set. The operations to be supported are given below. All the operations should only modify the relation to which they are applied (if required) and not return a new relation. The meet of two equivalence relations is their intersection (considering them as sets of ordered pairs), and their join is the transitive closure of their union, or the smallest equivalence relation that contains both. Describe an implementation so that the required operations can be done in the time specified. Describe clearly the variables used, and how they are changed in each operation. (10)

1. `initialize(n)`: create a partition with n elements, each in a separate part, in $O(n)$ time.
2. `split(i)`: split the part containing element `i`, so that `i` becomes a part by itself and all other elements are unchanged, in $O(1)$ time. If `i` was already a part by itself, this does nothing.
3. `shift(i,j)`: move element `i` to the part containing element `j` in $O(1)$ time. There is no change if they are already in the same part.
4. `num_parts()`: return the number of parts in the partition in $O(1)$ time. Note that a part, by definition, is not empty.
5. `max_part()`: return the size of the largest part in the partition in $O(1)$ time.
6. `meet(Q)`: The new value is the meet with partition `Q` in $O(n)$ time.
7. `join(Q)`: The new value is the join with partition `Q` in $O(n)$ time.

Q3. Let $a_0, a_1, \ldots, a_{n-1}$ be a sequence of integers. A substring $a_i, a_{i+1}, \ldots, a_j$ for $0 \leq i \leq j < n$ is said to be well-formed if a_i is the smallest and a_j the largest element in the substring. In other words, $a_i \leq a_k \leq a_j$ holds for all k such that $i \leq k \leq j$. A well-formed substring is said to be maximal if it is not a proper substring of any other well-formed substring.

(a) Prove that every element in the sequence is contained in exactly one maximal well-formed substring. Thus every sequence can be written uniquely as the concatenation of maximal well-formed substrings of the sequence. You do not have to prove by induction, although that is also possible. (3)

(b) Describe an $O(n)$ time algorithm to find all the maximal well-formed substrings of a given sequence. Explain briefly why your algorithm is correct and why it takes $O(n)$ time, but do not have to prove it formally. (7)