Data Structures (IT205) 2013-14
Final exam
28^{th} November, 2013

Time: 2 hours **marks: 80**

This question paper consists of four questions each worth 20 marks, printed on two back-to-back pages. Please check that your question paper is complete. Attempt all questions.

PART *A*

1. (a) Draw the graph given by the following adjacency lists:

$$\begin{array}{|c|ccccccccccc}
\hline
 & & & & & & & & & & \\
A & \leftarrow & B & \leftarrow & C & & & & & & \\ \hline
B & \leftarrow & A & \leftarrow & C & \leftarrow & D & \leftarrow & E & & \\ \hline
C & \leftarrow & A & \leftarrow & B & \leftarrow & F & \leftarrow & G & \leftarrow & H \\ \hline
D & \leftarrow & B & \leftarrow & E & & & & & & \\ \hline
E & \leftarrow & B & \leftarrow & D & \leftarrow & H & & & & \\ \hline
F & \leftarrow & C & \leftarrow & G & & & & & & \\ \hline
G & \leftarrow & C & \leftarrow & F & & & & & & \\ \hline
H & \leftarrow & C & \leftarrow & E & & & & & & \\ \hline
\end{array}$$

   (b) Draw the BFS tree using vertex A as the root. List the cross edges and classify them into vertical and horizontal.

   (c) Draw the DFS tree. Does any node have more than one child in the DFS tree? If yes then which vertex? Is it a cut vertex?

2. (a) Write a recursive code for computing the size of each subtree of a given binary tree.
   Input: A binary tree with a root and parent and child pointers at each node
   Output: The list set of n subtree sizes of each subtree

   (b) Now reverse the problem by interchanging the input and output of part (a). That is, given a list of n positive integers first find out whether the list corresponds to the subtree sizes of some binary tree. If the answer is yes then construct the required binary tree. Attempt it in the following three parts:

   - Give an example of an instance when it can be constructed (with the construction) and an instance when it cannot be constructed.
   - List any mathematical observations or properties you can of the n element input sequence which indicate it is a yes instance or a no instance.

   (c) Develop an algorithm using the ideas of part (b) to construct a corresponding binary tree or report that none exists.

3. (a) Given a hash table of size 17 implemented as an array $T[0..16]$ effect the following sequence of operations **insert** 7, **insert** 734, **insert** 289, **insert** 100, **insert** 909, **insert** 566, **insert** 332, **insert** 139, **insert** 154, **insert** 374, **insert** 559. Then **delete** 909, **delete** 7, **delete** 374, **delete** 559. **insert** 38, **insert** 600, **insert** 21. For all this use the hash function $h(i) \equiv i \bmod 17$, and linear hashing by probing, with jumps of 1. Draw the hash table description at the end of each monotonic subsequence of inserts or deletes. Now give the index sequence to search for the following elements: $1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17$.
   [Remember you need to distinguish between *nil* and *deleted*.]

   (b) If we insert 1000 elements into a hash table of size 100 using hashing by linked lists (to handle collisions) then give the lowest value of worst case search-time for an element in the table. (the answer is the same as the length of the longest linked list for successful search and length of the longest linked list plus 1 for a failed search).

4. You are given a sequence of stacks $S_1, \ldots, S_t$, containing $n_1, \ldots, n_t$ elements. The elements in all the stacks are numbers from the set $\{0, \ldots, 9\}$. The bottom most element of a stack is not allowed to be 0. The i^{th} stack can thus be thought of as decimal integer of n_i digits where the most significant digit is at the bottom of the stack and the least significant digit is at the top. You need to sort the stacks in numerical order. You may use two extra stacks of sufficient capacity (ignore overflow error) to decide if a particular stack is less than another stack (according to the previous definition) and then write the output in an array A of length t. This array will indicate the permutation of the input stacks according to sorted order. For example Stack 5 is the minimum, Stack 1 is the second minimum, Stack t is the third minimum etc. Write an efficient algorithm for this and analyse its running time in terms of the input sizes given. Notice that you can use any standard optimal algorithm like merge sort or heap sort. Just focus on the subroutine for comparing two stacks (whose running time is now a function of n_i and n_j, and not $O(1)$). For the overall algorithm find the total cost according to the underlying sorting algorithm you select. At the end of the algorithm the stacks must contain the original elements in the same order and only the stack index permutation must be written into array A.