Part I: Algorithm

1. (15%) Write an algorithm $DELETE(p)$ (in pseudo-code or any language) to delete a cell pointed to by p in a doubly-linked circular list which uses two special pointers *head* and *tail* (if not nil) to point to the first and last cells of the list, respectively. Each cell in the doubly-linked circular list contains a data item and two pointers *next* and *back*.

2. (15%) Show that in the worst case *quicksort* takes $O(n^2)$ time to sort an array of n elements.

3. (20%) Write a recursive algorithm $REVERSE(A)$ (in pseudo-code or any language) to reverse an array A. For example, if A = "*abcdef*" then $REVERSE(A)$ = "*fedcba*". You can use the following procedures:

   (a) $empty(A)$: returns TRUE is A is empty; FALSE otherwise.

   (b) $end(A)$: returns the position of the last element of A; 0 if $empty(A)$ is TRUE.

   (c) $retrieve(ith, A)$: retrieves the i^{th} element of A.

   (d) $delete(ith, A)$: deletes the i^{th} element of A.

   (e) $insert(x, ith, A)$: inserts an element x into the i^{th} position of A.

Part II: Data Structure

1.(15%) A hash function is to hash a large file into partitions such that within each partition the hashed values of all the data are equivalent. For instance, the file {3, 8, 9, 4, 2, 13, 7} will be hashed into the following partitions if it is hashed by using the hash function H(x)=x mod 3.

partition 1 (with H(x)=0) is {3, 9}
partition 2 (with H(x)=1) is {4, 13, 7}
partition 3 (with H(x)=2) is {8, 2}

The hash table after hashing this file will look like this:

Hash value	data list within a partition
0	$3 \rightarrow 9 \rightarrow$ nil
1	$4 \rightarrow 13 \rightarrow 7 \rightarrow$ nil
2	$8 \rightarrow 2 \rightarrow$ nil

Write a pseudo-code procedure for building a hash table of hashing a file $V=\{v_1, v_2, \ldots, v_n\}$ by using the hash function H(v). Assume that the return value of H(v) belong to {1, 2, ...,m}.

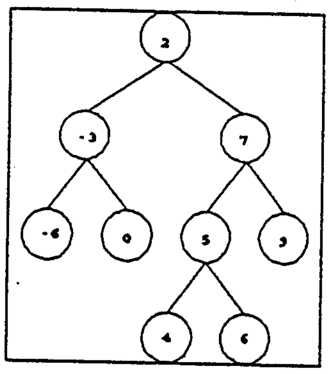
2.(20%) The *diameter* of a tree is defined as the maximum number of hops (i.e., edges) from the root to a leaf node of the tree. Now, let us define the diameter of a weighted tree (each edge has a weight) as the maximum "distance" from the root to a leaf node of the tree, where the distance between a node and its successor is the sum of the weights of all edges between these two nodes.

(a) Give an algorithm that finds the diameter of a weighted tree. Note that your algorithm should be of as low complexity as possible.

(b) Analyze the complexity of your algorithm.

3. Answer the following questions about trees.

(a) (5%) We all know that the purpose of using a tree is to speed up the search of a data item. The figure shows an example of a binary tree. If we want to find value 5, for example, we can follow the nodes of the tree to find that value. However, if we already know what we want to find (e.g., value 5), then why do we use the tree to find the value? If we do not know what we will find, then a tree does not help. So, what is a tree used for?



(b) (10%) We all know that the shorter the height (i.e., the number of levels) of a tree, the less time we will spend in finding a data item in the tree. Then, why do not use a one-level tree (in which the children of the root node are all leave nodes) to index data items? Do you think this tree will work or will not work? Why?