DEPARTMENT OF COMPUTER SCIENCE & ENGINEEING
MAJOR : CSL201
(Data Structures)

Max. Time – 2 hrs. **Max. Marks 70** **Date: 28/Nov/06**

Note: Answers of all the questions with all parts in the sequence.

Q1. (24)

a. Write an algorithm to reverse the order of items in a queue Q using stack S. Use standard functions of queue & stack. (4)
b. Suppose that you have to implement MIN-STACK which supports the stack operations **PUSH** and **POP** and a third operation **FindSum** which returns the sum of all elements in the stack, all in O(1) worst case time. Specify the data structures used along with implementation of all these three operations. (1+4)
c. Design an efficient algorithm to find the middle node of a singly linked list. [Hint use multiple pointers and a single loop]. (4)
d. Given a list of 'n' integers along with count field with each key containing the count (number of keys < the key), write an efficient algorithm to sort this list without using additional array. State the complexity of your algorithm. (4+1)
e. You are given two AVL trees T1 and T2. Write an algorithm to merge T1 and T2 to get a height balanced tree. (6)

Q2. (18) Please read parts (a to d) of this question carefully and answer all of them.

Rather than writing a stack and queue implementation from scratch, you decide to use a priority queue that is already implemented. You realize that by assigning the right priorities to data items when they are inserted, you can make data come out in either a LIFO or FIFO order. Your priority queue class has the following prototype:

```
class Pqueue
{
public:
void insert(int priority, dataType D); // Insert D with priority
dataType getmax();      // Get the max element and remove it
int empty();            // Return 1 if Pqueue is empty, 0 otherwise
Pqueue();               // Create an empty priority queue
};
```

and the insert definition says:

```
void Pqueue::insert(int priority, dataType D)
{ // pre: priority is any integer, positive or negative...
```

Both your stack and your queue should have member functions

```
void insert(dataType D); // Insert data into the stack/queue
dataType getnext(); // Get & remove the next element (in LIFO/FIFO order)
int empty();  // Return 1 if stack/queue is empty, 0 otherwise
```

a. Write the class prototype for STACK and QUEUE which includes private data fields, and use a comment to describe each entry of the class: (2+2)

b. Write the **getnext()** member funetion for the STACK. Briefly describe how you would ehange it if you were to implement a queue. (3+2)

e. Write the **insert(dataType D)** member function for the STACK. Briefly deseribe how you would ehange it if you were to implement a QUEUE. (3+2)

d. If Pqueue class is implemented with a HEAP, then give the tightest big-O bounds you ean for
   i. **getnext()** for your QUEUE class,
   ii. **insert()** for your QUEUE class

Assume N elements in the queue. Bricfly explain your answer. (2+2)

Q3. (13)

a. Given the following Adjaceney Matrix, draw unweighted direeted graph labelling nodes numerieally starting at 1. (4)

	1		1		1
		1		1	
1	1		1		1
				1	
1	1		1		
		1	1		

b. Perform a depth first seareh on the above graph. Clearly show the tree edges. Start the seareh at node 1 and always seleet successors in numerieally increasing order. What is the maximum size of the staek during this DFS run? (4+2)

e. What is the maximum degree of any vertex in the graph? Does the graph have any directed cyeles? (2+1)

Q4. (15)

a. Starting with an empty B-Tree of order 4, insert elements into the B-tree with the following keys, showing the insertion at each step and each split operation: (5)

   55, 100, 90, 75, 10, 5, 15, 40, 65, 60, 50, 20, 30, [illegible]

b. Show the result of inserting 2,1,4,5,8,3,6,7 into an empty splay tree.[Show the tree at the end of each insertion] (4)

c. Show the result of deleting node 8 from the splay tree eonstrueted in (b). (2)

d. Suppose we want to use an array to implement a stack. Howevcr, we do not know in advance the size to which the stack can grow. Indeed there may not bc any apriori limit to thc size of the stack. Suppose we start with a 'small' array A of size say 10. Now as long as our array bounds are not exeecded, all we need is a variable "top" that points to top of stack (so A[top] is the next free eell). Each of PUSH and POP operations are O(1). However, when the array is full and we need to push a new element on, we have a problem! In that case we can alloeate a new larger array of double the size [say], copy the old one over and then go on from there. This is going to be an expensive operation, so a push that requires us to do this is going to cost a lot. But maybe we ean "amortize" the cost over the previous eheap operations that got us to this point. (4)

   Prove that with the above implementation of a staek, the cost of an arbitrary sequence of 'n' stack[push/pop] operations to the staek costs at most O(n) and uses at most O(n) memory.