Tulsiramji Gaikwad-Patil College of Engineering & Technology

Department of Master in Computer Application

Subject Notes

Academic Session: 2018 – 2019

Subject: Data Structure

Semester: I **UNIT: I**

1. (a) How are the integer, Real and Character data stored in computer memory?

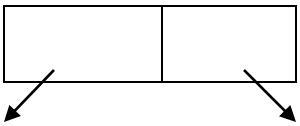
INTEGER: The most widely used method for interpreting bit settings as nonnegative integers is the binary number system. In this system each bit position represents a power of 2. The rightmost bit position represents 2^0 which equal 1. There are two widely used methods for representing negative binary numbers. In the first method called ones complement notation, a negative number is represented by changing each bit in its absolute value to the opposite bit setting. For example since 00100110 represents 38, 11011001 is used to represent -38. This means the leftmost bit of a number is no longer used to represent a power of 2 but is reserved for sign of a number. The second method of representing negative binary number is called twos complement notation. In this notation, 1 is added to the ones complement representation of a negative nu mber. For example since, 11011001 represents -38 in ones complement notation, 11011010 is used to represent -38 in twos complement notation. Binary number system is by no means the only method by which bits can be used to represent integers. For ex: in this system the bit string 00100110 is separated into two strings of four bits each: 0010 and 0110. This entire string represents 26. This representation is called binary coded decimal.

REAL: The usual method used to represent real numbers is floating point notation. There are many varieties of floating point notation and each has individual characteristics. The key concept is that real number is represented by a number, called a mantissa, times a base raised to an integer power, called an exponent. The base is usually fixed, and the mantissa and exponent vary to represent different real numbers. For example, if the base is fixed at 10, the number 387.53 could be represented as 38753×10^{-2}. Other possible representations are $.38753 \times 10^3$ and 387.53×10^0.

CHARACTER: Information is usually represented in character string form. For example, in some computers the eight bits 00100110 are used to represent the character '&'. A different eight bit pattern is used to represent the character 'A', another to represent 'B" and still another for each character that has a representation in a particular method. If eight bits are used to represent a character, up to 256 characters can be represented, since there are 256 different eight bit patterns.

1. (b) What is linked list? What are the advantages of linked list over an array in case of insertion and deletion operation? Write an algorithm to insert the KEY value after the given ITEM in single linked list.

A *linked list* is an ordered collection of data in which each element contains the location of the next element; that is, each element contains two parts: data and link.



Data Link

The data part holds the useful information, the data to be processed. The link is used to chain the data together. It contains a *pointer* (an address) that identifies the next element in the list.
Advantages of linked list over array are:

- Information can be placed at random location using linked list.
- Requires few manipulation of address during insertion and deletion of element.
- Memory is utilized properly
- Memory allocation takes place during run time and can be allocated as and when required and hence run time maximized.
- Memory can be reclaimed during execution of program.

Algorithm to insert key value after given item:

INSERT_SL_ANY(HEADER,ITEM ,KEY)
INPUT: HEADER is the pointer to the header node and ITEM is the data of the node to be inserted, and KEY being the data of the key node after which the node has to be inserted.
Output: A single linked list with newly inserted node having data ITEM after the node with the data key.
Data Structure: A single linked list whose address of the starting node is known from HEADER.

1. new=GETNODE(NODE)
2. If (new = NULL) then
    1. print "Memory is insufficient : Insertion is not possible"
    2. EXIT
3. Else
    1. ptr=HEADER
    2. While(ptr.DATA ≠KEY) and (ptr.LINK ≠NULL) do
        1. ptr=ptr.LINK
    3. EndWhile
    4. If (ptr.LINK=NULL)then
        1. Print"KEY is not available in the list"
        2. Exit
    5. ELSE
        1. new.LINK=ptr.LINK
        2. new.DATA=ITEM
        3. ptr.LINK=new
    6.EndIf

4. EndIF
5. Stop

(c) Explain the Abstract data type specification. Give the ADT for rational numbers.

A useful tool for specifying the logical properties of a data type is the abstract data type, or ADT. The term ADT refers to the basic mathematical concept that defines the data type. There are number of methods for specifying an ADT. The method that we use is semiformal and borrows heavily from C notation but extends notation where necessary. For example ADT RATIONAL, which corresponds to the mathematical concept of a rational number? A rational number is a number that can be expressed as the quotient of two integers. The operations on rational numbers that we define are the creation of rational number from two integers, addition and testing for equality. The following is an initial specification of this ADT:

```
/* Value definition */
Abstract typedef <integer, integer? RATIONAL;
Condition RATIONAL[1]=0;
/*operator definition*/
abstract RATIONAL makerational(a,b)
Int a,b;
Precondition b!=0;
Postcondition makerational[0]==a;
                    makerational[1]==b;

    abstract  RATIONAL add(a,b)
    RATIONAL a,b;
    Precondition add[1] == a[1] * b[1];
                 Add[0]== a[0] * b[1] + b[0] * a[1];

     abstract equal(a,b)
     RATIONAL a,b;
     Post condition equal ==(a[0] *b[1] == b[0]*a[1]);
```

An ADT consists of two parts: a value definition and an operator definition. The value definition defines the collection of values for the ADT and consists of two parts: a definition clause and a condition clause. The keywords abstract tyepdef introduce a value definition and the keyword condition is used to specify any condition on the newly defined type. in this definition, the condition specifies that th denominator may not be 0.the definition clause is required but the condition clause may not be necessary for every ADT.

Immediately following the value definition comes the operator definition. Each operator is defined as an abstract function with three parts, a header, the operational preconditions and the post conditions. For example, the operator definition of the ADT RATIONAL includes the operations of creation (make rational), addition (add) and equality (equal).

1.(d) What is double linked list? What are the advantages of double linked list over single linked list in case of insertion and searching operation? Write an algorithm to delete the given element from the double linked list.

In a single linked list one can move starting from the header (first node) to any node in one direction only(from Left to right). This why, a single linked List is called as one-way list.
The double linked list is two-way list because one can move in either Direction, from Left to Right or from Right to Left.

Advantages of double linked list over single linked list are:

- Traversing is possible from both direction
- Any operation towards the end will be easy for ex:
- Adding node towards end or at the end

Deleting last node
Searching the node towards the end
For any operation using link list the time complexity of double link list is better than of single linked list.

Algorithm to delete a given element from double linked list:

Deletion of a node at any position in a double linked list. The algorithm is as follows:

Algorithm DELETE_DL_ANY(KEY)
Input: A double linked list with data, and KEY, the data content of the key node to be deleted.
Output: A double linked list without a node having data content KEY, if any.
Data structure: Double linked list structure whose pointer to the header node is HEADER.

Steps:

1. ptr = HEADER.RLINK //Move to the first node
2. If (ptr = NULL) then
   1. Print "List is empty: No deletion is made"
   2. Exit
3. EndIf //Quit the program
4. While (ptr.DATA ≠ KEY) and (ptr.RLINK ≠ NULL) do
   //Move to the desired node
   1. ptr = ptr.RLINK
5. EndWhile
6. If (ptr.DATA = KEY) then //If the node is found
   1. ptr1 = ptr.LLINK //Track to the predecessor node
   2. ptr2 = ptr.RLINK //Track to the successor node
   3. ptr1.RLINK = ptr2 //Change the pointer as shown 1 in Figure 3.12(c)
   4. If (ptr2 ≠ NULL) then //If the deleted node is the last node
      1. ptr2.LLINK = ptr1 //Change the pointer shown as 2 in Figure 3.12(c)
   5. EndIf
   6. RETURNNODE(ptr) //Return the free node to the memory bank
7. Else
   Print "The node does not exist in the given list"
8. EndIf
9. Stop

2. (a) What is stack? What are the applications of stacks in computer science? How the stacks are can be represented as linked list?

It is an ordered group of homogeneous items of elements. Elements are added to and removed from the top of the stack (the most recently added items are at the top of the stack). The last element to be added is the first to be removed (LIFO: Last In, First Out).

Applications of stack in computer science:

- Keeping track of order of function calls.
- Computation of polish expression
- Conversion of decimal system to octal/binary/hexadecimal system
- Solving the recursion problems
- Quick sort
- Parenthesis matching problem
- Computer uses the concept of stack in syntax analysis

- Tree traversal technique and in depth first search of a graph.

Linked representation of stack:

In stack elements can be added and deleted from only one end called top. By adding node at the beginning of a list we can add a new node at the top of stack. Similarly, by applying the concept of deleting the node from the beginning we can implement the concept of deleting the node from the stack

Algo - To insert element in to the stack using linked list :

1. new=GETNODE(NODE)
2. If (new = NULL) then
    1. print "Memory underflow : No Insertion"
    2. EXIT
3. Else
    1. new.LINK = HEADER.LINK
    2. NEW.DATA=x
    3. HEADER.LINK=new
4. EndIf
5. Stop

Algo To delete element in to the stack using linked list :

1. ptr=HEADER.LINK
2. If (ptr = NULL) then
    1. print "The list is empty : No deletion"
    2. EXIT
3. Else
    1. ptr1=ptr.LINK
    2. HEADER.LINK=ptr1
    3. RETURNNODE(ptr)
4. EndIf
5. Stop

2.(b) What is queue? What are the overflow and underflow condition in circular queue? Write an algorithm to delete the element from the circular queue.

It is an ordered group of homogeneous items of elements.
Queues have two ends:

- Elements are added at one end.
- Elements are removed from the other end.

The element added first is also removed first (FIFO: First In, First Out).

Underflow:
Circular Queue is empty
Front=0
Rear =0
Overflow:
Circular Queue is full
Front =(REAR Mod Length)+1

Algorithm to delete the element from circular queue:

Algorithm DECQUEUE( )
Input: A queue CQ with elements. Two pointers FRONT and REAR are known.
Output: The deleted element is ITEM if the queue is not empty.
Data structures: CQ is the array representation of circular queue.

Steps:

1. If (FRONT = 0) then
   1. Print "Queue is empty"
   2. Exit
2. Else
   1. ITEM = CQ[FRONT]
   2. If (FRONT = REAR) then //If the queue contains single element
      1. FRONT = 0
      2. REAR = 0
   3. Else
      1. FRONT = (FRONT MOD LENGTH) + 1
   4. EndIf
3. EndIf
4. Stop

2. (c) Write an algorithm to translate the infix expression into postfix expression.

1) Examine the next element in the input.
2) If it is operand, output it.
3) If it is opening parenthesis, push it on stack.
4) If it is an operator, then
i) If stack is empty, push operator on stack.
ii) If the top of stack is opening parenthesis, push operator on stack
iii) If it has higher priority than the top of stack, push operator on stack.
iv) Else pop the operator from the stack and output it, repeat step 4
5) If it is a closing parenthesis, pop operators from stack and output them until an opening parenthesis is encountered. pop and discard the opening parenthesis.
6) If there is more input go to step 1
7) If there is no more input, pop the remaining operators to output.

OR
Read the tokens from a vector infixVect of tokens (strings) of an infix expression
When the *token* is an operand
Add it to the end of the vector postfixVect of token (strings) that is used to store the corresponding postfix expression
When the *token* is a left or right parenthesis or an operator
If the *token x* is "("
Push_back the token x to the end of the vector stackVect of token (strings) that simulates a stack
if the *token* x is ")"
Repeatedly pop_back a token y from stackVect and push_back that token y to postfixVect until "(" is encountered in the end of stackVect. Then pop_back "(" from stackVect.
If stackVect is already empty before finding a "(", that expression is not a valid expression.
if the *token x* is a regular operator
Step 1: Check the token y currently in the end of stackVect.
Step 2: If (case 1) stackVect is not empty and (case 2) y is not "(" and (case 3) y is an operator of higer or equal precedence than that of x, then pop_back the token y from stackVect and push_back the token y to postfixVect, and go to Step 1 again.
Step 3: If (case 1) stackVect is already empty or (case 2) y is "(" or (case 3) y is an operator of lower precedence than that of x, then push_back the token x into stackVect.
When all tokens in infixVect are processed as described above, repeatedly pop_back a token y from stackVect and push_back that token y to postfixVect until stackVect is empty.

2.(d) define the priority queue and deque. Give ADT specification for the queue.

It is another variation of queue structure. Each element has been assigned a value called priority of element. An element is inserted at any position in queue according to priority. Priority queue may be divided into two types mainly, ascending or descending priority. In De-queue both insertion and deletion operation can be made at either end of the structure. Actually the term is originated from Double Ended Queue.

ADT specification for queue:

```
abstract typedef <eltype> QUEUE(eltype);
abstract empty(q)
QUEUE(eltype) q;
Postcondtions empty == (len(q)==0);
Abstract eltype rmpve(q)
QUEUE(eltype) q;
Precondition empty(q)==FALSE;
Postcondition remove == first(q`);
                Q== sub(q`, 1, len(q`) -1);
Abstract insert(q,elt)
QUEUE(eltype) q;
Eltype elt;
```

Postcondition q==q` + <elt>;

3.(a) What is recursion? Write an iterative and recursive for finding the greatest common devisor of two numbers.

When the function calls the function itself within a body of function then this type of structure is called a recursion of function. A procedure is termed as recursive if the procedure is defined by itself. Recursion is a technique that solves a problem by solving a smaller problem of the same type

Algorithm using iterative method:

GCD(a,b)
Where a and b are two positive integers and assume a>b
1. Repeat while b!=0
R= a%b
A=b
B=r
(end while)
2. Return(a)

Recursive algorithm:

GCD(a,b)
If (a<b)
GCD(b,a)
(end if)
If (b=0)
Return (a)
(end if)
r= a % b
GCD(b,r)
Return.

3.(b) Consider the following recursive function :

$$A(m,n)=\begin{cases} n+1 & \text{if } m=0 \\ A(m-1,1) & \text{if } m\neq 0 \text{ but } n=0 \\ A(m-1, A(m,n-1)) & \text{if } m\neq 0 \text{ and } n\neq 0 \end{cases}$$

Find (1) A(1,4)
(2) A(2,3)

A(1,4)= A(0,A(1,3)) as m≠0,n≠0
= A(0,A(0,(A(1,2))) as m≠0,n≠0
= A(0,A(0,A(0,A(1,1)))) as m≠0,n≠0
=A(0,A(0,A(0,(A(0,A(1,0))))) as m≠0
= A(0,A(0,A(0,A(0,A(0,1))))) as m≠0,n=0
=A(0,A(0,A(0,A(0,2)))) as m=0
=A(0,A(0,A(0,3))) as m=0
=A(0,A(0,4)) as m=0

=A(0,5) as m=0
=6.
A(2,3) = A(1,A(2,2))
= A(1,A(1,A(2,1)))
=A(1,A(1,A1,A(2,0))))
=A(1,A(1,A(1,A(1,1))))
=A(1,A(1,(A,1(A(0,A(1,0))))
.=A(1,A(1,A(1,(A(0,A(0,1))))
.=A(1,A(1,A(1,3)))
= A(1,A(1,(A(0,(A(1,2)))
= A(1,A(1,(A(0,A(0,A(1,1))))
= A(1,A(1,(A(0,A(0,(A(0,A(1,0)))))
= A(1,A(1,(A(0,A(0,A(0,A(0,1)))))
= A(1,A(1,(A(0,A(0,A(0,2))))
= A(1,A(1,(A(0,A(0,3)))
= A(1,A(1,(A(0,4))
=A(1,A(1,5))
=A(1,A(0,A(1,4)))
=A(1,A(0, A(0,A(1,3))
=A(1,A(0, A(0,A(0,(A(1,2)))
=A(1,A(0, A(0,A(0,A(0,A(1,1))))
=A(1,A(0, A(0,A(0,A(0,(A(0,A(1,0)))))
=A(1,A(0, A(0,A(0,A(0,A(0,A(0,1)))))
=A(1,A(0, A(0,A(0,A(0,A(0,2))))
=A(1,A(0, A(0,A(0,A(0,3)))
=A(1,A(0, A(0,A(0,4))
=A(1,A(0, A(0,5)))
=A(1,A(0,6))
=A(1,7)
=A(0,(1,6))
=A(0,A(0,,A(1,5)))
=A(0,A(0,A(0,A(1,4)))
=A(0,A(0,A(0, A(0,A(1,3))
=A(0,A(0,A(0, A(0,A(0,(A(1,2)))
=A(0,A(0,A(0, A(0,A(0,A(0,A(1,1))))
=A(0,A(0,A(0, A(0,A(0,A(0,(A(0,A(1,0)))))
=A(0,A(0,A(0, A(0,A(0,A(0,A(0,A(0,1)))))
=A(0,A(0,A(0, A(0,A(0,A(0,A(0,2))))
=A(0,A(0,A(0, A(0,A(0,A(0,3)))
=A(0,A(0,A(0, A(0,A(0,4))
=A(0,A(0,A(0, A(0,5)))
=A(0,A(0,A(0,6))
=A(0,A(0,7))
=A(0,8)
=9.

3.(c) Explain the mechanism of calling the function and returning from the function. How are the stacks used in the implementation of the recursive solution?

The act of calling the function may be divided into three parts:

- Passing arguments
- Allocating and initializing local variables

Transferring control to the function

1. Passing arguments: A copy of argument is made locally within the function and any changes in to the parameter are made to that local copy. The effect of this scheme is that original input argument cannot be allocated. In this method storage for the argument is allocated within the data area of function.

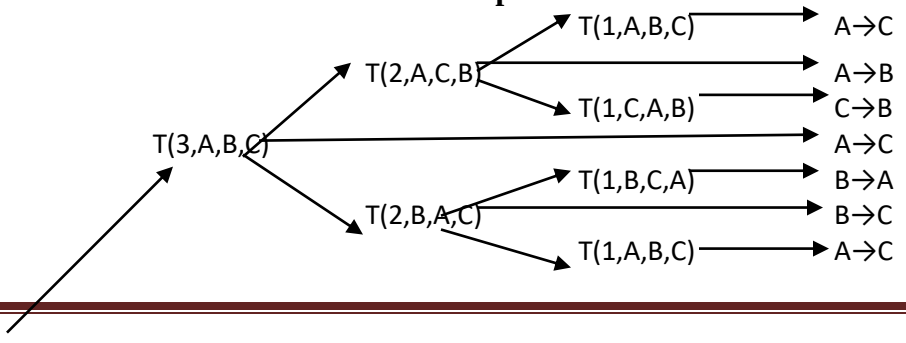
2. Allocating and initializing local variables: After arguments have been passed, the local variables of the function are allocated. These local variables include all those declared directly in the function and any temporaries that must be created during the course of execution.

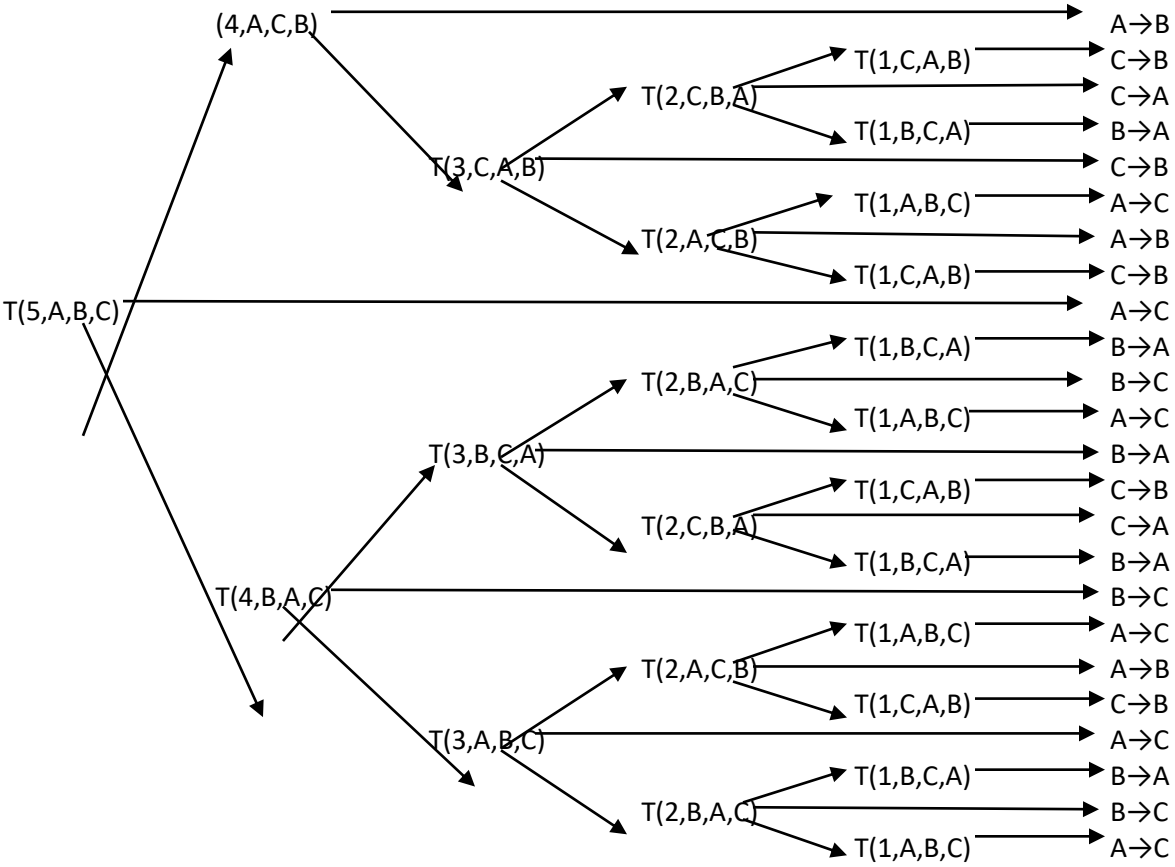
3. Transferring control to the function: At this point control may still not be passed to the function because provision has not yet been made for saving the return address. If a function is given control, it must eventually restore control to the calling routine by means of a branch. However it cannot execute that branch unless it knows the location to which it must return. Since this location is within the calling routine and not within the function, the only way that the function can know this address is to have it passed as an argument. This is exactly what happens, aside from explicit arguments specified by the programmer, there is also a set of implicit arguments that contain information necessary for the function to execute and return correctly. The function stores this address within its own data area. When it is ready to return control to the calling program, the function retrieves the return address and braches to that location.

Return from function: when the function returns, three actions are performed. First the return address is retrieved and stored in a safe location. Second the functions data area is freed. This data area contains all local variables, temporaries, and the return address. Finally a branch is taken to the return address which had been previously saved. This restores control to the calling routine at the point immediately following the instruction that initiated the call. In addition if the function returns a value, that value is placed in a secure location from which the calling program may retrieve it.

Stacks are used in recursion to keep the successive generations of local variables and parameters. This stack is maintaining by the system and is kept invisible to the user. Each time that a recursive function is entered, a new allocation of its variables is pushed on top of the stack. Any reference to a local variable or parameter is through the current top of the stack. When the function returns, the stack is popped the top allocation is freed, and previous allocation becomes the current stack top to be used for referencing local variables.

3.(d) Give the recursive solution for the towers of Hanoi problem for n=5 disks.

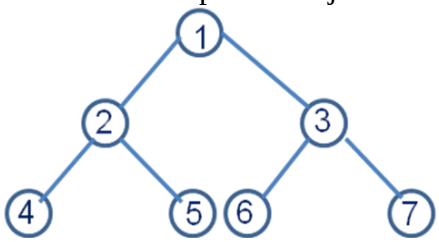




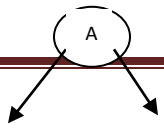
4.(a) explain the binary tree, threaded binary tree and height balance tree with suitable example. Give the linked list representation of the binary tree.

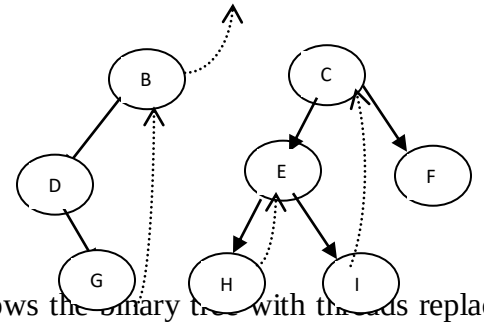
A binary tree T is defined as finite set of elements, called nodes. Such that:

- T is empty (called the null tree or empty tree)
- T contains a distinguished node R, called root of T, and the remaining nodes of T from an ordered pair of disjoint binary trees T1 and T2.



Threaded binary tree: Instead of containing a NULL pointer in its right field, a node with an empty right subtree contained in its right field a pointer to the node that would be on top of the stack at that point. There be would longer be a need for stack since the last node visited during traversal of a left subtree points directly to its inorder successor. Such a pointer is called a thread and must be differentiable from a tree pointer that is used to link a node to its left or right subtree.



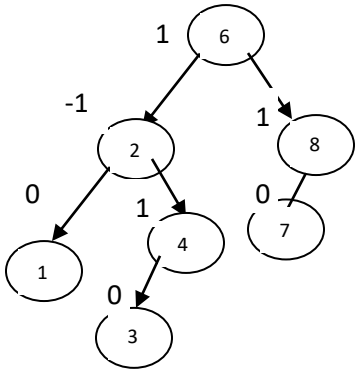


Above figure shows the binary tree with threads replacing NULL pointers in nodes with empty right subtrees. The threads are drawn with doted lines to differentiate them from tree pointers. Rightmost node in each tree still has a NULL right pointer, since it has no longer successor. Such a trees are called right in-threaded binary trees.

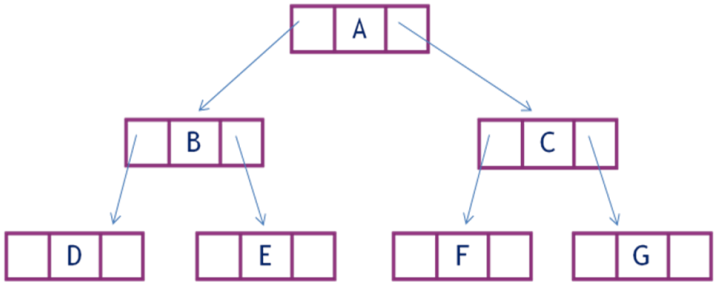
A left in-threaded binary tree may be defined similarly, as one in which such NULL left pointer is altered to contain a thread to that nodes inorder predecessor. An in-threaded binary tree may be then defined as binary tree that is both left in-threaded and right in-threaded.

Height balanced binary tree: A binary tree is said to be height balanced binary tree if all nodes have a balance factor of 1, 0 or -1. That is

$|bf|=|h_L - h_R| <= 1$. for every node in the tree.



Linked list representation of binary tree:



4.(b) Write an algorithm to delete the node containing value KEY from the graph.

Input: Dgptr, the pointer to the graph, key the label of the vertex which has to be removed from the graph.

Let N be the number of vertices presently available in the graph.

Output: the reduced graph without the vertex key and its associated edges.

Data structure: linked structure of undirected graph and Dgptr is the pointer to it.

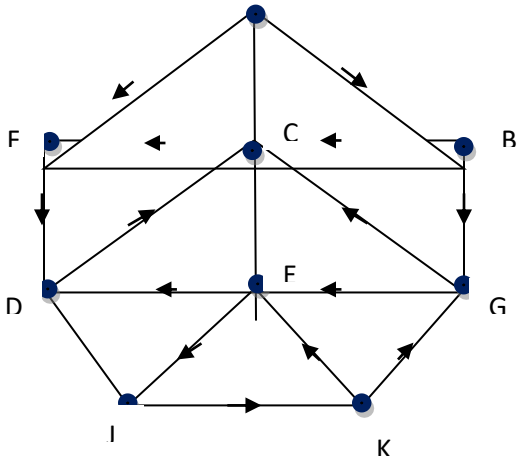
Steps:

```
If (N=0) then
Print "graph is empty : no deletion"
Exit
Endif
Ptr=Dgptr[key],link
Dgptr[key].link=NULL
Dgptr[key].label=NULL
N=N-1
Return_node(ptr)
For i=1 to N do
Delete_sl_any(dgptr[i].key)
Endfor
stop
```

4.(c) Write a non recursive algorithm for traversing the tree in postorder.

```
POSTORDER(INFO,LEFT,RIGHT,ROOT)
1.  [Initially Push NULL onto STACK and initialize PTR]
        Set TOP = 1,STACK[1] = NULL and PTR: = ROOT
Repeat  Steps 3 TO 5 While PTR ≠ NULL.
  Set TOP = TOP + 1 and STACK[TOP] = PTR
4.  If  RIGHT[PTR] ≠ NULL then
                Set: TOP=TOP+1 and
       STACK[TOP]= - RIGHT[PTR]
    [End of If Structure]
5.  Set  PTR = LEFT[PTR] .
    [End of Step 2 loop]
Set PTR = STACK[TOP] and TOP = TOP-1.
7.  Repeats  While PTR > 0
          (a) Apply PROCESS to INFO[PTR]
          (b) Set PTR = STACK[TOP] and TOP = TOP-1.
    [End Of Loop]
8.  If PTR < 0, then:
              (a) Set PTR = - PTR.
              (b) Go to Step 2.
    [End of If Structure]
9.  EXIT.
```

4. (d) consider the following graph



Find the minimum path from node A to node J by using the breadth first search method.
Input: V is the starting vertex.
Output: A list VISIT giving the order of visit of vertices during the traversal.
Data Structure: Linked structure of graph. Gptr is the pointer to a graph.
Steps:

```
1.If (GPTR =NULL) then
        1.print"Graph is Empty"
        2.Exit
2.EndIf
3.u=V
4.OPEN.ENQUEUE(u)
5.While(OPENQ.STATUS( ) ≠ EMPTY )do
        1.u=OPENQ.DEQUEUE( )
        2.If (SEARCH_SL_END (VISIT,u)
                1.INSERT_SL_END(VISIT,u)
                2.ptr=Gptr[u]
                3.While (ptr.Link ≠ Null) do
                        1.vptr = ptr.LINk
                        2.OPENQ.ENQUEUE(vptr.LABEL)
                4.EndWhile
        3.EndIf
6.EndWhile
7.Return(VISIT)
8.Stop
```

The minimum path P can be found by using breadth first search method.
We will keep track of origin of each edge by using an array orig together with the array queue. The steps are as follows.
Initially, add A to queue and add NULLto orig as follows
Front =1 queue=A
Rear=1orig= Ø
Remove front element A from queue by setting front=front+1 and add to queue the neighbors of A as follows
Front = 2 queue:A,F,C,B

Rear=4orig: Ø,A,A,A
Remove front element F and its neighbors
Front=3 queue:A,F,C,B,D
Rear=5orig: Ø,A,A,A,F
Remove element C and add its neighbors
Front=4 queue:A,F,C,B,D
Rear=5orig: Ø,A,A,A,F
Remove element C and add its neighbors
Front=4 queue:A,F,C,B,D
Rear=5orig: Ø,A,A,A,F
Remove element B and add its neighbors
Front=4 queue:A,F,C,B,D,G
Rear=5orig: Ø,A,A,A,F,B
Remove element D and add its neighbors
Front=4 queue:A,F,C,B,D,G
Rear=5orig: Ø,A,A,A,F,B
Remove element G and add its neighbors
Front=4 queue:A,F,C,B,D,G,E
Rear=5orig: Ø,A,A,A,F,B,G
Remove element E and add its neighbors
Front=4 queue:A,F,C,B,D,G,E,J
Rear=5orig: Ø,A,A,A,F,B,G,E
We stop here as soon as J is added to queue, since J is our final destination. We now backtrack from j using array orig to find path P, thus
J←E←G←B←A is required path P.

5.(a) explain the quick sort method with suitable example. How stacks are used in quick sort method? Discuss the complexity of quick sort method.

Divide:
Pick any element p as the pivot, e.g, the first element
Partition the remaining elements into
FirstPart, which contains all elements < p
SecondPart, which contains all elements ≥ p
Recursively sort the FirstPart and SecondPart
Combine: no work is necessary since sorting is done in place

```
Quick-Sort(A, left, right)
        if   left ≥ right  return
 else
                middle ← Partition(A, left, right)
                Quick-Sort(A, left, middle–1 )
                Quick-Sort(A, middle+1, right)
 end if
Partition(A, left, right)
        x ← A[left]
        i ← left
        for j ← left+1 to right
```

```
            if A[j] < x then
                    i ← i + 1
                    swap(A[i], A[j])
            end if
    end for j
    swap(A[i], A[left])
    return i
```

Example:
Suppose A having list of 12 number.
44, 33, 11, 55, 77, 90, 40, 60, 99, 22, 88, 66
keyElement=44
scan from Right to Left
Comparing each no. with 44 and stop at first no. less than 44.
Interchange 44 and 22.

22, 33, 11, 55, 77, 90, 40, 60, 99, 44, 88, 66
Now being with 22, key Element=44
scan from Left to Right
Comparing each no. with 44 and stop at first no. greater than 44.
Interchange 44 and 55.
22, 33, 11, 44, 77, 90, 40, 60, 99, 55, 88, 66
Beginning with 55,
key Element=44
scan from Right to Left
22, 33, 11, 44, 77, 90, 40, 60, 99, 55, 88, 66
Comparing each no. with 44 and stop at first no. less than 44.
Interchange 44 and 40.
22, 33, 11,40, 77, 90, 44, 60, 99, 55, 88, 66
Now being with 40, key Element=44
scan from Left to Right
22, 33, 11,40, 77, 90, 44, 60, 99, 55, 88, 66
Comparing each no. with 44 and stop at first no. greater than 44.
Interchange 44 and 77.
22, 33, 11,40, 44, 90,77, 60, 99, 55, 88, 66
Beginning with 77,
key Element=44
scan from Right to Left
22, 33, 11,40, 44, 90,77, 60, 99, 55, 88, 66

Beginning with 77,scan the number less then 44.we do not meet
Such number before meeting 44.This means all numbers less than 44
now form the sublist of number to the left of 44, and all numbers greater
than 44 now form the sublist of number to right of 44.

Thus 44 is correctly placed in its position,

The original list A has been reduced to the task of sorting each above Sublist.

The above reduction step is repeated with each sublist containing 2 or more elements.

Complexity:

Ave. Case:

The average case complexity of quick sort comes from the following fact . In this method each reduction step of algorithm. Produce 2 sub lists. Accordingly that

1)Reducing the initial list place one elements & produce 2 sub lists.

2) Reducing the two sub list place 2 elements & produces 4 sublists.

3) Reducing 4 sublist place 4 elements & produce 8 sublist.

The reduction at each level

each level uses at most n elements

$f(n) = n+(n-1)+\ldots..2+1 = \frac{n(n+1)}{2}$

$= \frac{n^2}{2} + O(n) = O(n^2)$

$= O(n \log n)$

5.(b) write an algorithm to insert the element KEY in sorted array N elements by using the binary search method.

(Binary Search) BINARY(DATA, LB, UB, ITEM, LOC)
Here DATA is a sorted array with lower bound LB and upper bound UB, and ITEM is a given item of information. The variables BEG, END and MID denote, respectively, the beginning, end and middle locations of a segment of elements of DATA. This algorithm finds the location LOC of ITEM in DATA or sets LOC = NULL.

1. [Initialize segment variables.]
   Set BEG := LB, END := UB and MID = INT((BEG + END)/2).
2. Repeat Steps 3 and 4 while BEG ≤ END and DATA[MID] ≠ ITEM.
3. If ITEM < DATA[MID], then:
   Set END := MID − 1.
   Else:
   Set BEG := MID + 1.
   [End of If structure.]
4. Set MID := INT((BEG + END)/2).
   [End of Step 2 loop.]
5. If DATA[MID] = ITEM, then:
   Set LOC := MID.
   Else:
   End=end+1
   Data[end]=item
6. Exit.

End=end+1
Data[end]=item

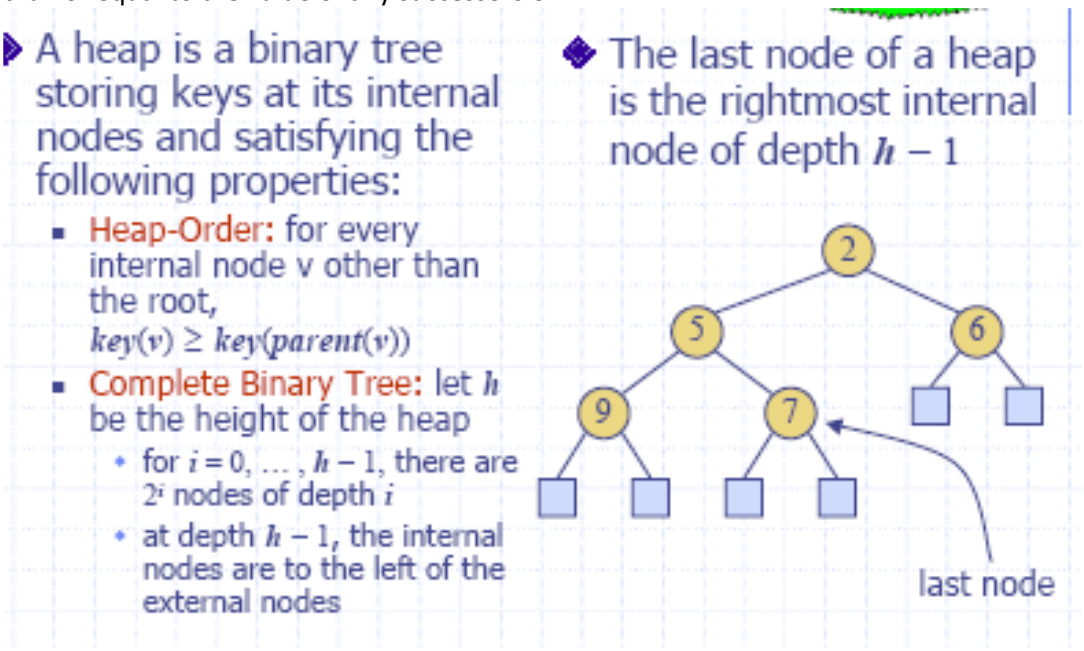
5.(c) what is heap? Write an algorithm for deleting the element from heap.

Suppose H is a complete binary tree. Then it will be termed as heap tree, if it satisfies the following properties:

For each node N in H, the value at N is greater than or equal to the value of each of the children of N.

Or in other words, N has the value which is greater than or equal to the value of every successor of N.

Such a heap tree is called max heap. Similarly, min heap is possible where any node N has the value less than or equal to the value of any successors of N.



```
element delete_max_heap(int *n)
{
/* delete element with the highest key from the heap */
   int parent, child;
   element item, temp;
   if (HEAP_EMPTY(*n)) {
      fprintf(stderr, "The heap is empty\n");
      exit(1);
   }
   /* save value of the element with the highest key */
   item = heap[1];
   /* use last element in heap to adjust heap */
   temp = heap[(*n)--];
   parent = 1;
   child = 2;
   while (child <= *n) {
      /* find the larger child of the current parent */
      if    (child    <    *n)    &&    (heap[child].key
     heap[child+1].key)
         child++;
      if (temp.key >= heap[child].key) break;
      /* move to the next lower level */
      heap[parent] = heap[child];
      parent = child;
      child *= 2;
   }
   heap[parent] = temp;
   return item;
}
```

5.(d) explain the insertion sort method with suitable example. Write an algorithm for sorting the array A of N elements using insertion sort method.

Suppose an array A with n elements A[1], A[2],…..,A[N] is in memory. The insertion sort algorithm scans A from A[1] to A[N], inserting each element A[K] into its proper position in the previously sorted sub array A[1], A[2],……,A[K-1].
Pass 1: A[1] by itself is trivially sorted.
Pass 2: A[2] is inserted either before or after A[1] so that: A[1], A[2] is sorted.
Pass 3: A[3] is inserted into its proper place in A[1], A[2], that is, before A[1], between A[1] and A[2], or after A[2], so that A[1], A[2], A[3] is sorted.
Pass 4: A[4] is inserted into into proper place in A[1], A[2], A[3] so that: A[1], A[2], A[3], A[4] is sorted.

Suppose an array A contains 8 elements as follows:

77, 33, 44, 11, 88, 22, 66, 55

Pass	A[0]	A[1]	A[2]	A[3]	A[4]	A[5]	A[6]	A[7]	A[8]
K = 1:	−∞	(77)	33	44	11	88	22	66	55
K = 2:	−∞	77	(33)	44	11	88	22	66	55
K = 3:	−∞	33	77	(44)	11	88	22	66	55
K = 4:	−∞	33	44	77	(11)	88	22	66	55
K = 5:	−∞	11	33	44	77	(88)	22	66	55
K = 6:	−∞	11	33	44	77	88	(22)	66	55
K = 7:	−∞	11	22	33	44	77	88	(66)	55
K = 8:	−∞	11	22	33	44	66	77	88	(55)
Sorted:	−∞	11	22	33	44	55	66	77	88

SS/K7/S/1212

SECOND SEMESTER MASTER IN COMPUTER APPLICATION

DATA STRUCTURES

Paper- 2CSA1

1.(a) explain the representation of two dimensional array in memory. A magic square of 5 rows and 5 columns contain different elements. Write and algorithm to verify whether the sum of each individual column elements, sum of each individual row elements and sum of diagonal elements is equal or not.

The memory of a computer is linear and not a matrix like a 2D array. So, the elements of the array are stored either by row, called "row-major", or by column, called "column-major". Row-major order is used most notably in C and C++ during static declaration of arrays.
In C, since the length of each row is always known, the memory can be filled row one row at a time, one after the other. Example:
a[i][j] =
1 2 3 4 5 6 7 8 9
Representation in the memory: In row-major: 1 2 3 4 5 6 7 8 9 In column-major: 1 4 7 2 5 8 3 6 9
Address calculation of an element: Row-Major : addr (i,j) = B + W * (Nc * (i - Lr) + (j-Lc)) Col-Major : addr (i,j) = B + W * ((i - Lr) + Nr * (j-Lc)) i,j = subscript number. B = Base address W = width (size) of each element Nc = Number of Columns Nr = Number of Rows Lc = Lower-bound of Column Lr = Lower-bound of Row
In above example, for element (6), i.e., a(1,2) in row-major or a(2,1) in col-major, B = 200 (say) W = 2 Lr=Lc=0 Nc=Nr=3
addr (1,2) = 210; addr (2,1) = 214
A Magic Square is a square matrix where the sum of all columns, rows and diagonals is constant.

Consider the following 3rd order Magic square :

8	1	6
3	5	7
4	9	2

S = 15

Fig 1 : A 3rd order magic square

The above square shows a simple 3rd order Magic Square. 8 other combinations are possible by shifting Rows/Colums Up/Right.

Algorithm for Odd Order Magic Squares.

Top left cell = (X,Y) = (0,0)
Step 1 : Pick any X , Y. (I take this as the center of the first row)
Step 2 : N = 1
Step 3 : While N <= Sqr(Order)
Step 4 : Write N at (X,Y)
Step 5 : TempX = (X++) mod N
Step 6 : TempY = (Y--) mod N
Step 7 : IF (TempX,TempY) is Empty X = TempX, Y = TempY
ELSE Y++
Step 8 : Loop Step 3
Step 9 : S = (1/2)(order)(1 + sqr(order))

Graphically shown for plotting order '3' Magic Square:

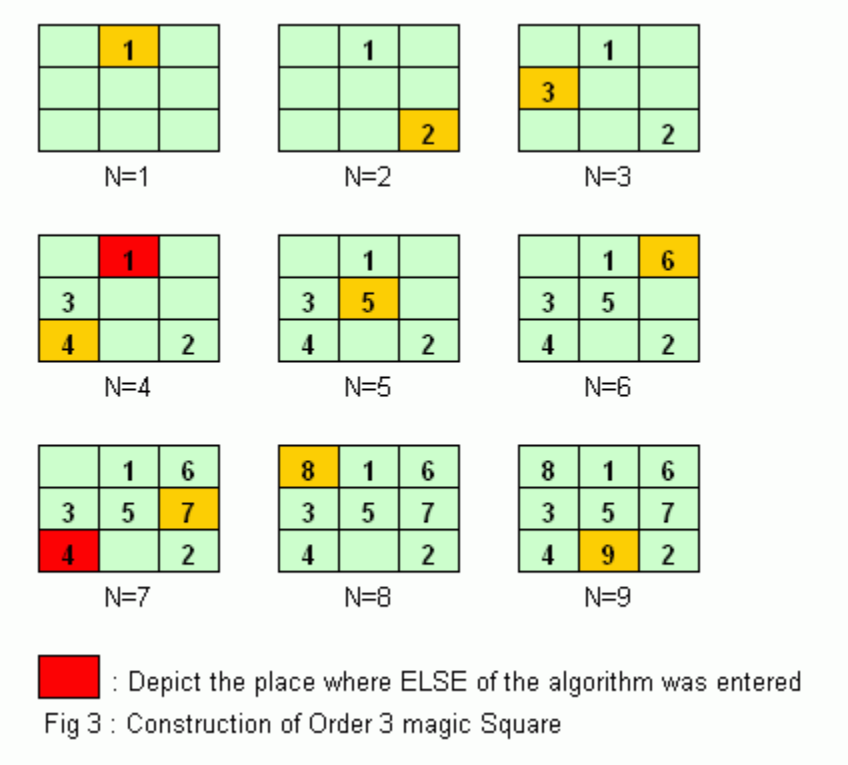


Fig 3 : Construction of Order 3 magic Square

Similarly, view how the same algorithm was used in Fig.2 5th and 7th order Magic Squares

Also for odd ordered magic squares thus constructed, the sum of rows, colums, diagonals is given by S = (1/2)(order)(1 + sqr(order)).

```
//C++ code to generate odd-order Magic Squares
//Porting should be a POC
//(c) osix.net/~anilg
#include <iostream>
#define MAX 15
using namespace std;

int main()
{
 int order, n, X, Y, i, j, a[MAX][MAX],s;

 cout<<"Enter the order of matrix (odd) :";
 cin>>order;

 if(!order%2 || order>MAX) return -1; // for those who chose to be stubborn

 for(i=0;i<order;i++)
  for(j=0;j<order;j++)
   a[i][j]=0;
```

```
 X = 0;
 Y = order/2;

 for(n=1;n<=(order*order);n++)
 {
   a[X][Y] = n;
   if (a[(X+order-1)%order][(Y+order+1)%order])
     X = (X+order+1)%order ;
   else
     X = (X+order-1)%order , Y = (Y+order+1)%order ;
 }

 for(i=0;i<order; i++)
 {
  cout<<'n' ;
  for(j=0;j<order; cout<<a[i][j++]<<'t'); //for order>9,i'd change 't' to ' '
 }

 s= (1/2.0)*order*(1+order*order);
 cout<<"nnSum of Rows,Coulums,Diagonals = "<<s;

 system("pause>nul"); //getch() equivalent
                 // non-windows ppl remove this
 return 0;
}
```

1.(b) write an algorithm to delete the given ITEM from singly linked list.

DELETE_SL_ANY(HEADER)
INPUT: HEADER is the pointer to the header node.KEY is the data content of the node to be deleted.
Output: A single linked list except the node with data content as KEY.
Data Structure: A single linked list whose address of the starting node is known from HEADER.
ptr1=HEADER
2. ptr=ptr1.LINK
3. While (ptr ≠ NULL) do
    1.if (ptr.DATA ≠KEY)THEN
        1.ptr1=ptr
        2.ptr=ptr.LINK
    2. Else
        1.ptr1.LINK=ptr.LINK
        2.RETURNNODE(ptr)
        3.Exit
    3.EndIf
4. EndWhile
5. if(ptr=NULL) hen
    1.Print”Node with KEY does not exist :No Deletion”
6. EndIf

7. STOP

1.(c) Write algorithm to merge the two singly linked list.

Algorithm: MERGE_SL(HEADER1,HEADER2,HEADER)
Input: HEADER1 and HEADER2are pointers to header nodes of lists
(L1and L2)to be merged.
Output: HEADER is the pointer to the resultant list.
Data Structure: Single Linked List Structure.
Steps:
1. ptr=HEADER1
2. While(ptr.LINK ≠ NULL)do
    1. ptr=ptr.LINK
3. EndWhile
4. ptr.LINK=HEADER2.LINK
5. RETURNNODE(HEADER2)
6. HEADER=HEADER1
7. STOP

1.(d) what are the limitations of array implementation of the linked list? Write an algorithm to insert new element at the given location LOC in doubly linked list.

Limitations of array implementation:
Data stored without using pointers
Static
Blocks memory
Restricts size
Memory wastage

Algorithm INSERT_DL_ANY(*X*, KEY)

Input: *X* be the data content of the node to be inserted, and KEY the data content of the node after which the new node to be inserted.

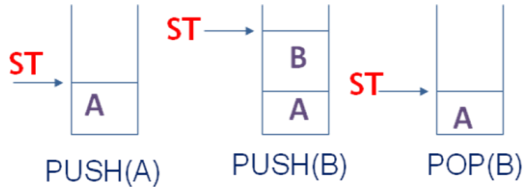
Output: A double linked list enriched with a node containing data as *X* after the node with data KEY, if any.

Data structure: Double linked list structure whose pointer to the header node is HEADER.

Steps:

```
1. ptr = HEADER
2. While (ptr.DATA ≠ KEY) and (ptr.RLINK ≠ NULL) do   // Move to the key node if the current
                                                      //node is not the KEY node or list reaches at the end
   1. ptr = ptr.RLINK
3. EndWhile
4. new = GETNODE(NODE)                                //Get a new node from the pool of free storage
5. If (new = NULL) then                               //When the memory is not available
   1. Print "Memory is not available"
   2. Exit                                            //Quit the program
6. EndIf
7. If (ptr.RLINK = NULL)                              //If the KEY node is at the end or not found in the list
   1. new.LLINK = ptr
   2. ptr.RLINK = new                                 //Insert at the end
   3. new.RLINK = NULL
   4. new.DATA = X                                    //Copy the information to the newly inserted node
8. Else                                               //The KEY is available
   1. ptr1 = ptr.RLINK                                //Next node after the key node
   2. new.LLINK = ptr                                 //Change the pointer shown as 2 in Figure 3.11(c)
   3. new.RLINK = ptr1                                //Change the pointer shown as 4 in Figure 3.11(c)
   4. ptr.RLINK = new                                 //Change the pointer shown as 1 in Figure 3.11(c)
   5. ptr1.LLINK = new                                //Change the pointer shown as 3 in Figure 3.11(c)
   6. ptr = new                                       //This becomes the current node
   7. new.DATA = X                                    //Copy the content to the newly inserted node
9. EndIf
10. Stop
```

2.(a) Explain stack with suitable example. Write an algorithm to insert the new element in stack.
Translate the following infix expression in to prefix and postfix from:
A*(B+D)/E-F*(G+H/K)
((A+B)*D)↑(E-F)
It is an ordered group of homogeneous items of elements.
Elements are added to and removed from the top of the stack (the most recently added items are at the top of the stack).
The last element to be added is the first to be removed (LIFO: Last In, First Out).



Algorithm to insert element into stack:

PUSH(STACK, TOP, MAXSTK, ITEM)
This procedure pushes an ITEM onto a stack.

1. [Stack already filled?]
   If TOP = MAXSTK, then: Print: OVERFLOW, and Return.
2. Set TOP := TOP + 1. [Increases TOP by 1.]
3. Set STACK[TOP] := ITEM. [Inserts ITEM in new TOP position.]
4. Return.

A*(B+D)/E-F*(G+H/K)
A*[BD+]/EF-]*(G+HK/)
[ABD+*]/[EF-]*[GHK/+]
[ABD+*]/[EF-GHK/+*]
ABD+*EF-GHK/+*/
((A+B)*D)↑(E-F)
(AB+)D*)↑(EF-)
(AB+)D*) (EF-)↑
AB+DEF-↑*

2.(b) Explain deque with suitable example, suppose each data structure is stored in circular array with N memory cells
(a) find the number of elements in queue in terms of front and rear
(b) find the number of elements in a deque iterms of left and right.
I
De-queue both insertion and deletion operation can be made at either end of the structure . Actually the term is originated from Double Ended Queue.

Count number of elements using front and rear

1 2 3 4 5
F=4 R=5
In circular queue since the position of front is less than rear i.e. front<rear, total number of elements are calculated by following formula
Num=rear-front+1
=5-4+1
=2.
In circular if position of front is equal to rear i.e front =rear total number of elements are calculated by following formula
Num = rear-front + 1.
In circular queue if position of front is greater than rear i.e front>rear total number of elements are calculated by following formula
Num=maxq+(rear-front+1)

number of elements in a deque iterms of left and right:
Deque is maintained by circular array with pointers left and right, which points two ends of the deque. The term circular array comes from the fact that we assume that deque[1] comes after deque[n] in the array. the condition left=NULL will be used to indicate that a deque is empty.
If(left<=right)
Num=(left-right+1)
Else
Num=maxq+(left-right+1)
endif

2.(c) Write an algorithm to translate infix expression to postfix form.
Refer 08 summer 2(c).
2.(d) Suppose priority queue is maintained as singly linked list, write an algorithm which adds an item with priority number N to the queue.
Algorithm Insert-pq(item,p)
Input: the item and its priority P, value of a node that is to be inserted
Output: a new node inserted
Data structure: linked list structure of priority queue. Header as the pointer to the header
Steps:
Ptr=header
New=getnode(node)
New.data=item
New.priority=p
While(ptr.rlink≠NULL) and (ptr.priority<p) do
Ptr=ptr.rlink
End while
It(ptr.rlink=NULL)
Ptr.rlink=new
New.llink=ptr
New.rlink=NULL
Rear=new

Else
If(ptr.priority>=p) then
Ptr1=ptr.llink
Ptr1.rlink=new
New.rlink=ptr
Ptr.llink=new
New.llink=ptr1
Nedif
Endif
Front=header.rlink
stop

3.(a) Explain the tower of Hanoi problem. Write a recursive algorithm for the tower of Hanoi problem.
This problem has historical basis in the ritual of ancient Vietnam.
Suppose, there are 3 pillars A,B and C. There are N discs of decreasing size so that no two discs are of the same size. Initially all the discs are stacked on one pillar in their decreasing order of size. Let this be Pillar A. Other two pillars are empty.
The problem is to move all the discs from one pillar to other using the third pillar as auxiliary.

Recursive algorithm to solve tower of Hanoi problem:
MOVE(N, ORG, INT, DES)
If N=1
    1. move ORG to DES)
    2. return.
else
2. If N > 0 then
    1. Move(N-1, ORG, DES, INT)
    2. ORG-> DES(MOVE from ORG to DES)
    3. MOVE (N_1, INT, ORG, DES)
3. Endif
4. Stop.

3.(b) Let A be an integer array with N elements suppose X is an integer function defined by

$$X(K)=X(A,N,K) = \begin{cases} 0 & \text{if } K=0 \\ X(K-1)+A(K) & \text{if } 0<K<=N \\ X(K-1) & \text{if } K>N \end{cases}$$

Find X(5) for each of the following arrays:-
N=8, A : 3,7,-2,5,6,-4,2,7
N=3, A : 2,7,-4
What does this function do?

X(A,8,5) = X(5-1)+A(5) as 5<8
= X(4)+6 as A(5) = 6
= X(A,8,4) + 6
= X(4-1)+A(4)+6
= X(3)+5+6
= X(3-1)+A(3)+11

= X(2)+ -2+11
= X(2-1)+A(2)+9
= X(1)+7+9
= X(1-1)+A(1)+16
= X(0)+3+16
= 0+19 as K=0
=19. (i.e sum of 1st 5 elements)

X(A,3,5)= X(5-1) as k>n i.e 5>3
= X(4)
= X(A,3,4)
= X(4-1) as k>n i.e 4>3
= X(3)
= X(A,3,3)
= X(3-1)+A(3)
= X(2)+ -4
=X(A,3,2) - 4
= (X(2-1)+A(2)) -4
= X(1)+7-4
= X(A,3,1)+3
= [X(1-1)+A(1)]+3
= X(0)+2+3
= 0+05 as K=0
=5 (i.e sum of array elements)

3.(c) Explain the simulation of recursion.

Simulating recursion:

The act of calling the function may be divided into three parts:

Passing arguments

Allocating and initializing local variables

Transferring control to the function

1. Passing arguments : A copy of argument is made locally within the function and any changes in to the parameter are made to that local copy. The effect of this scheme is that original input argument cannot be allocated. In this method storage for the argument is allocated within the data area of function.

2. Allocating and initializing local variables : After arguments have been passed., the local variables of the function are allocated. These local variables include all those declared directly in the function and any temporaries that must be created during the course of execution.

3. Transferring control to the function : At this point control may still not be passed to the function because provision has not yet been made for saving the return address. If a function is given control, it must eventually restore control to the calling routine by means of a branch. However it cannot execute that branch unless it knows the location to which it must return. Since this location is within the calling routine and not within the function, the only way that the function can know this address is to have it passed as an argument. This is exactly what happens, aside from explicit arguments specified by the programmer, there is also a set of implicit arguments that contain information necessary for the function to execute and return correctly. The function stores this address within its own data area. When it is ready to return control to the calling program, the function retrieves the return address and braches to that location.

Return from function: when the function return, three actions are performed. First the return address is retrieved and stored in a safe location. Second the functions data area is freed. This data area contains

all local variables, temporaries, and the return address. Finally a branch is taken to the return address which had been previously saved. This restores control to the calling routine at the point immediately following the instruction that initiated the call. In addition if the function returns a value, that value is placed in a secure location from which the calling program may retrieve it.
Stacks are used in recursion to keep the successive generations of local variables and parameters. This stack is maintaining by the system and is kept invisible to the user. Each time that a recursive function is entered, a new allocation of its variables is pushed on top of the stack. Any reference to a local variable or parameter is through the current top of the stack. When the function returns, the stack is popped the top allocation is freed, and previous allocation becomes the current stack top to be used for referencing local variables.

3.(d) Write a recursive algorithm to generate n Fibonacci sequence.
Recursive Algorithm to generate Fibonacci series
Fibo(n)
If n=0 or n=1
Return (n)
Endif
Fiba=fibo(n-1)
Fibb=fibo(n-2)
Fib=fiba+fib
Return(fib)

4.(a) Write an algorithm to insert an item into threaded binary tree.

Algorithm insert_thread(x,n)
Input: header, the pointer to the header node of the threaded binary tee
X is the data of a node to be inserted
N is the data of a node to be inserted
Output: if x exists in the tree then n is inserted after x
Data structure: linked structure of threaded binary tree

Steps:
Ptr=header.lchild
Flag=false
While(ptr≠header) and (flag=false)
If(ptr.data=x) then
Xptr=ptr
Flag=true
2.else
1. ptr=inscc(ptr)
3. endif
4. endwhile
5. if(flag=false) then
1. print “node does not exist, no insertion”
2. exit
6. endif
7. read option = l/r or left(l) or right(r) child
8. nptr=getnode(node)

9. case: option ='l'
    1. if(xptr.ltag=true) then
        1. nptr.lchild=xptr.lchild
        2. nptr.ltag=true
        3. xptr.lchild=nptr
        4. xptr.ltag=false
        5. nptr.rchild=xptr
        6. nptr.rtag=true
    Else
        1. lptr.lchild=xptr.lchild
        2.. xptr.lchild=true
        3. xptr.ltag=0
        4. nptr.rchild=xptr
        5. nptr.rtag=true
        6. .nptr.lchild=lptr
        7.. nptr.lchild=false
        8. ptr=inpred(xptr)
        9. ptr.rchild=nptr
    endif
10. case: option ='r'
    1. if(xptr.rtag=true) then
        1. nptr.rchild=xptr.rchild
        2. nptr.rtag=true
        3. xptr.rchild=nptr
        4. xptr.rtag=false
        5. nptr.lchild=xptr
        6. nptr.ltag=true
    2. Else
        1. rptr =xptr.rchild
        2.. xptr.rchild=nptr
        3. xptr.rtag=false
        4. nptr.lchild=xptr
        5. nptr.ltag=true
        6. .nptr.rchild=rptr
        7.. nptr.rchild=false
        8. ptr. =insucc(xptr)
        9. ptr.lchild=nptr
    3. Endif
Endcase
stop

4.(b) write an algorithm to create a minimal spanning tree, where the graph is represented as linked list

- A **spanning** tree of a graph G is a tree that contains all vertices of G
- A MST for a weighted graph is a spanning tree with minimum weight
- If G is not connected, it does not have a minimum spanning tree MST
  (instead it has a minimum spanning forrest)
- From now on, we assume that G is connected (if not, we would be able to find its connected components and then find their respective minimum spanning trees)

- Starts from an arbitrary vertex u
- Selects vertex v so that the edge (u,v) is in MST

- Let $A = (a_{ij})$ be the matrix representation of $G = (V,E,w)$
- Let V_T be the set of vertices found to be in the MST
- Let d[1..n] be a vector. For each $v \in (V - V_T)$, $d[v]$ holds the weight of the edge with least weight from any vertex in V_T to v
- In each iteration, a new v is chosen with the minimum $d[v]$

Cost:

- The while loop is executed $n-1$ times
- The min operationen (row 10) and the for loop (rows 12-13) both requires $O(n)$ steps

Totally: $\Theta(n^2)$ steps are required

```
1.  procedure PRIM_MST(V, E, w, r)
2.  begin
3.      V_T := {r};
4.      d[r] := 0;
5.      for all v ∈ (V − V_T) do
6.          if edge (r, v) exists set d[v] := w(r, v);
7.          else set d[v] := ∞;
8.      while V_T ≠ V do
9.      begin
10.         find a vertex u such that d[u] = min{d[v]|v ∈ (V − V_T)};
11.         V_T := V_T ∪ {u};
12.         for all v ∈ (V − V_T) do
13.             d[v] = min{d[v], w(u, v)};
14.     endwhile
15. end PRIM_MST
```

Init: vertices with edge to r (start) is assigned cost = weight of that edge. Other vertices are given weight = inf

For all corners outside V_T with edge to V_T chose the one with lowest cost

Include the vertex

Update d-values

4.(c) Write an algorithm to copy a given binary tree.

In postorder, the root is visited *last*

Here's a postorder traversal to make a complete copy of a given binary tree:

```
public BinaryTree copyTree(BinaryTree bt) {
   if (bt == null) return null;
   BinaryTree left = copyTree(bt.leftChild);
   BinaryTree right = copyTree(bt.rightChild);
   return new BinaryTree(bt.value, left, right);
}
```

```
mynode *copy(mynode *root)
{
  mynode *temp;

  if(root==NULL)return(NULL);

  temp = (mynode *) malloc(sizeof(mynode));
  temp->value = root->value;

  temp->left  = copy(root->right);
  temp->right = copy(root->left);

  return(temp);
}
```

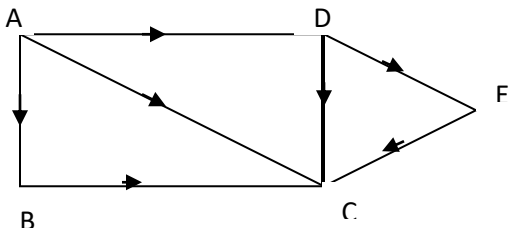
This code will will only print the mirror of the tree

```
void tree_mirror(struct node* node)
{
  struct node *temp;

  if (node==NULL)
  {
    return;
  }
  else
  {
    tree_mirror(node->left);
    tree_mirror(node->right);

    // Swap the pointers in this node
    temp = node->left;
    node->left = node->right;
    node->right = temp;
  }
}
```

4.(d) Explain depth first method for traversing the graph with suitable example

DFS

1.Push the Starting vertex into the stack OPEN
2.While OPEN is not empty do
    1.POP a vertex V
    2.If V is not in VISIT
        1.Visit the vertex V
        2.Store V in VISIT
        3.Push all the adjacent vertex of V onto OPEN
3.EndWhile
4.Stop



We use stack for dfs.
Stack : A (push)
Write: A, stack:B, C, D (pop A, push its neighbor)
Write D, stack: B,C,E

Write E, stack: B,C
Write C, stack: B
Write B, stack: (empty)

Thus DFS is now complete from starting at node A. nodes that are printed : A,D, E, C, B.

5.(a) Write an algorithm to find the location of given item from an array A of N elements using binary search method.

Algorithm 4.6: (Binary Search) BINARY(DATA, LB, UB, ITEM, LOC)

Here DATA is a sorted array with lower bound LB and upper bound UB, and ITEM is a given item of information. The variables BEG, END and MID denote, respectively, the beginning, end and middle locations of a segment of elements of DATA. This algorithm finds the location LOC of ITEM in DATA or sets LOC = NULL.

```
1. [Initialize segment variables.]
   Set BEG := LB, END := UB and MID = INT((BEG + END)/2).
2. Repeat Steps 3 and 4 while BEG ≤ END and DATA[MID] ≠ ITEM.
3.     If ITEM < DATA[MID], then:
           Set END := MID − 1.
       Else:
           Set BEG := MID + 1.
       [End of If structure.]
4.     Set MID := INT((BEG + END)/2).
   [End of Step 2 loop.]
5. If DATA[MID] = ITEM, then:
       Set LOC := MID.
   Else:
       Set LOC := NULL.
   [End of If structure.]
6. Exit.
```

5.(b) Write heap sort algorithm to sort the array A of N elements. Discuss the complexity of heap sort algorithm.

Steps:

1. CREATE_MAX_HEAP(A) //To create the heap H from the set of data
2. $i = N$
3. While ($i > 1$) do //Continue until heap contains single element
   1. SWAP ($A[1]$, $A[i]$) //Delete the root node and replace it by the last node
   2. $i = i - 1$ //Pointer to the last node shifted towards the left
   3. $j = 1$ //To rebuild the heap
   4. While ($j < i$) do
      1. lchild = $2 * j$ //Left child
      2. rchild = $2 * j + 1$ //Right child
      3. If ($A[j] < A$[lchild]) and (A[lchild] > A[rchild]) then
         1. SWAP($A[j]$, A[lchild])
         2. j = lchild
      4. Else
         1. If ($A[j] < A$[rchild]) and (A[rchild] > A[lchild]) then
            1. SWAP($A[j]$, A[rchild])
            2. j = rchild
         2. Else
            1. Break( ) //Rebuild is completed: break the loop
         3. EndIf
      5. EndIf
   5. EndWhile
4. EndWhile
5. Stop

Complexity:
Average case : O(n log n)
Worst case : O(n logn)

5.(c) Write quick sort algorithm to sort the array A of N elements. Discuss the complexity of heap sort algorithm.

```
Quick-Sort(A, left, right)
        if   left ≥ right  return
 else
                middle ← Partition(A, left, right)
                Quick-Sort(A, left, middle–1 )
                Quick-Sort(A, middle+1, right)
 end if
Partition(A, left, right)
        x ← A[left]
        i ← left
        for j ← left+1 to right
                if A[j] < x then
                        i ← i + 1
                        swap(A[i], A[j])
                end if
```

```
        end for j
        swap(A[i], A[left])
        return i
```

Complexity of quick sort algorithm:
Time
Most of the work done in partitioning.
Average case takes $\Theta(n \log(n))$ time.
Worst case takes $\Theta(n^2)$ time
Space
Sorts in-place, i.e., does not require additional space

5.(d) Explain the selection sort method with suitable example.

Procedure 9.2: MIN(A, K, N, LOC)

An array A is in memory. This procedure finds the location LOC of the smallest element among A[K], A[K + 1], . . . , A[N].

1. Set MIN := A[K] and LOC := K. [Initializes pointers.]
2. Repeat for J = K + 1, K + 2, . . . , N:
   If MIN > A[J], then: Set MIN := A[J] and LOC := A[J] and LOC := J.
   [End of loop.]
3. Return.

The selection sort algorithm can now be easily stated:

EXAMPLE 9.5

Suppose an array A contains 8 elements as follows:

77, 33, 44, 11, 88, 22, 66, 55

Applying the selection sort algorithm to A yields the data in Fig. 9-4. Observe that LOC gives the location of the smallest among A[K], A[K + 1], . . . , A[N] during Pass K. The circled elements indicate the elements which are to be interchanged.

There remains only the problem of finding, during the Kth pass, the location LOC of the smallest among the elements A[K], A[K + 1], . . . , A[N]. This may be accomplished by using a variable MIN to hold the current smallest value while scanning the subarray from A[K] to A[N]. Specifically, first set MIN := A[K] and LOC := K, and then traverse the list, comparing MIN with each other element A[J] as follows:

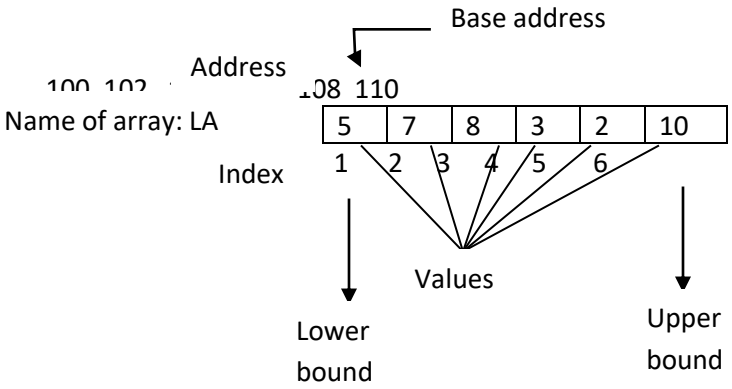
Pass	A[1]	A[2]	A[3]	A[4]	A[5]	A[6]	A[7]	A[8]
K = 1, LOC = 4	(77)	33	44	(11)	88	22	66	55
K = 2, LOC = 6	11	(33)	44	77	88	(22)	66	55
K = 3, LOC = 6	11	22	(44)	77	88	(33)	66	55
K = 4, LOC = 6	11	22	33	(77)	88	(44)	66	55
K = 5, LOC = 8	11	22	33	44	(88)	77	66	(55)
K = 6, LOC = 7	11	22	33	44	55	(77)	(66)	88
K = 7, LOC = 7	11	22	33	44	55	66	(77)	88
Sorted:	11	22	33	44	55	66	77	88

Fig. 9-4 Selection sort for $n = 8$ items.

RNL/SK6-1682
SECOND SEMESTER MASTER IN COMPUTER APPLICATION
DATA STRUCTURES
Paper- 2CSA1

1.(a) What is an array? Explain the representation of one dimensional array in memory. Write an algorithm to count the given item in sorted array of N elements.
An array is a group of related data item that share a common name..
An array is a collection of similar elements.These similar elements could be all i*nts* or all *floats*, or all *chars* etc. The array of characters is called a 'string', whereas an array of *ints* or *floats* is called an array.

Memory representation of one dimensional array:



Count(la,lb,ub,item)
Where la- linear array
Lb- lower bound
Ub-upper bound
Item- given item to be counted
Repeat for j=lb to ub step 1
If (la[j]=item
Count=count+1
Endif
Endfor
Write "number of item ",count
Return

1.(b) write an algorithm to inert item in sorted linked list.
Inserting an element in a sorted linked list.
First will find the location then insert an element.
So algorithm:

Finda(info,link,start,item,loc)

This procedure finds the location loc of the last node in a sorted list such that info[loc]<item or sets loc=null
1.if start =null then set loc=null and return
2. if item<info[start] then set loc=null and return
3. set save=start and ptr=link[start]
4. repeat steps 5 and 6 while ptr≠null
5. if item<info[ptr] then
Set loc=save and return
End of if structure
6. set save=ptr and ptr=link[ptr]
End of step 4 loop
7.set loc=save
8. return

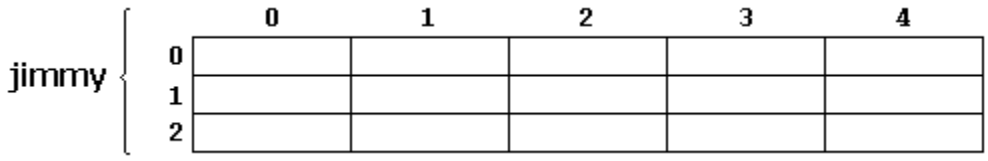
Insloc(info,link.start,avail,loc,item)
This algorithm inserts item so that item follows the node location loc
If avail=null then write overflow and exit
Set new=avail and avail=link[avail]
Set info[new]=item
If loc=null then
Set link[new]=start and start=new
Else
Set link[new]=link[loc] and link[loc]=new
End if
Exit

Insert(info,link,start,avail,item)
This algorithm inserts item into a sorted linked list
Call finda(info,link,start,item,doc)
Call insloc(info,link,start,avail,loc,item)
Exit.

1.(c) explain the representation of two dimensional array in memory. Write an algorithm to delete the last node from double linked list.

Each time an element of array is accessed in multidimension, it requires a transformation from row-or-column major to linear address and hence it is less efficient than one dimensional array and such transformation is known as mapping function.
Multidimensional arrays

Multidimensional arrays can be described as "arrays of arrays". For example, a bidimensional array can be imagined as a bidimensional table made of elements, all of them of a same uniform data type.

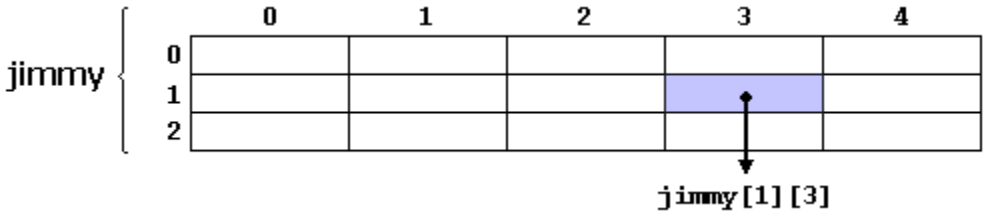


jimmy represents a bidimensional array of 3 per 5 elements of type int. The way to declare this array in C++ would be:

```
int jimmy [3][5];
```

and, for example, the way to reference the second element vertically and fourth horizontally in an expression would be:

```
jimmy[1][3]
```



(remember that array indices always begin by zero).

Multidimensional arrays are not limited to two indices (i.e., two dimensions). They can contain as many indices as needed. But be careful! The amount of memory needed for an array rapidly increases with each dimension. For example:

```
char century [100][365][24][60][60];
```

declares an array with a char element for each second in a century, that is more than 3 billion chars. So this declaration would consume more than 3 gigabytes of memory!

Multidimensional arrays are just an abstraction for programmers, since we can obtain the same results with a simple array just by putting a factor between its indices:

```
1 int jimmy [3][5];   // is equivalent to
2 int jimmy [15];     // (3 * 5 = 15)
```

With the only difference that with multidimensional arrays the compiler remembers the depth of each imaginary dimension for us. Take as example these two pieces of code, with both exactly the same result. One uses a bidimensional array and the other one uses a simple array:

Deletion of a node at the end of a double linked list. The algorithm is as under:

Algorithm DELETE_DL_END( )
Input: A double linked list with data.
Output: A reduced double linked list.
Data structure: Double linked list structure whose pointer to the header node is HEADER.

Steps:

1. ptr = HEADER
2. While (ptr.RLINK ≠ NULL) do //Move to the last node
   1. ptr = ptr.RLINK
3. EndWhile
4. If (ptr = HEADER) then //If the list is empty
   1. Print "List is empty: No deletion is made"
   2. Exit //Quit the program
5. Else
   1. ptr1 = ptr.LLINK //Pointer to the last but one node
   2. ptr1.RLINK = NULL //Change the pointer shown as 1 in Figure 3.12(b)
   3. RETURNNODE(ptr) //Return the node to the memory bank
6. EndIf
7. Stop

1.(d) write an algorithm to merge two sorted single linked list

```
/*
while both lists are not empty...
  {
   if the top of list1 is less than the top of list2...
     insert the value at the top of list1 into the union
   else if the top of list1 is less than the top of list2...
     insert the top of list2 into the union
   else
     insert the top of list1 into the intersection
  }
while list1 is not empty...
  insert the top of list1 into the union
while list2 is not empty...
  insert the top of list2 into the union
*/
```

2.(a) Write an algorithm to :-
(i) check whether stack is empty or not
(ii) to insert element in stack
(iii)to delete the element from the stack using the dynamic storage implementation of linked list.

Algorithm ot check stack is empty or not:

1. [Stack has an item to be removed?]
   If TOP = 0, then: Print: UNDERFLOW, and Return.

(ii) to insert element in stack

PUSH(STACK, TOP, MAXSTK, ITEM)
This procedure pushes an ITEM onto a stack.

1. [Stack already filled?]
   If TOP = MAXSTK, then: Print: OVERFLOW, and Return.
2. Set TOP := TOP + 1. [Increases TOP by 1.]
3. Set STACK[TOP] := ITEM. [Inserts ITEM in new TOP position.]
4. Return.

(iii)to delete the element from the stack using the dynamic storage implementation of linked list.
Pop(info,link,top,avail,item)
If top=null then write underflow and exit
Set item=info[top]
Set temp=top and top=link[top]
Set link[temp]=avail and avail=temp
Exit

2.(b) What set of conditions are necessary and sufficient for a sequence of insert and remove operations on a single empty queue to leave the queue empty without causing underflow? What set of conditions are necessary and sufficient for such a sequence to leave a non empty queue unchanged?

Algorithm ENQUEUE(ITEM)
Input: An element ITEM that has to be inserted.
Output: The ITEM is at the REAR of the queue.
Data structure: Q is the array representation of queue structure; two pointers FRONT and REAR of the queue Q are known.

Steps:

```
1. If (REAR = N) then                          //Queue is full
   1. Print "Queue is full"
   2. Exit
2. Else
   1. If (REAR = 0) and (FRONT = 0)            //Queue is empty
      1. FRONT = 1
   2. EndIf
   3. REAR = REAR + 1                          //Insert the item into the queue at REAR
   4. Q[REAR] = ITEM
3. EndIf
4. Stop
```

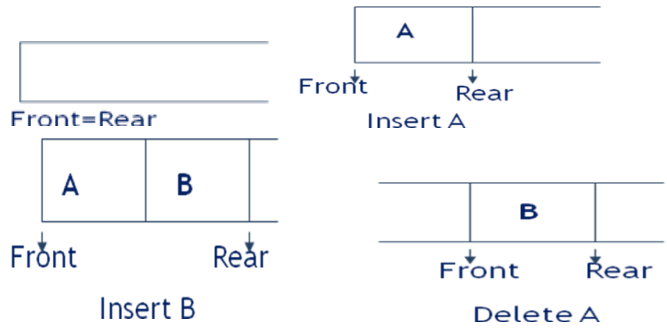
Algorithm DEQUEUE( )
Input: A queue with elements. FRONT and REAR are the two pointers of the queue Q.
Output: The deleted element is stored in ITEM.
Data structures: Q is the array representation of queue structure.

Steps:

```
1. If (FRONT = 0) then
   1. Print "Queue is empty"
   2. Exit
2. Else
   1. ITEM = Q[FRONT]                          //Get the element
   2. If (FRONT = REAR)                        //When queue contains single element
      1. REAR = 0                              //The queue becomes empty
      2. FRONT = 0
   3. Else
      1. FRONT = FRONT + 1
   4. EndIf
3. EndIf
4. Stop
```

2.(c) write an algorithm to translate the given infix expression to postfix expression.
Refer summer 2008 2(c)
2.(d) explain queues, priority queue and deque with suitable example.
It is an ordered group of homogeneous items of elements.
Queues have two ends:

Elements are added at one end.
Elements are removed from the other end.
The element added first is also removed first (FIFO: First In, First Out).



circular array : For queue representation using array when the REAR pointer reaches at a end , insertion will be denied even if room is available at the front. One way to remove this using a circular array.

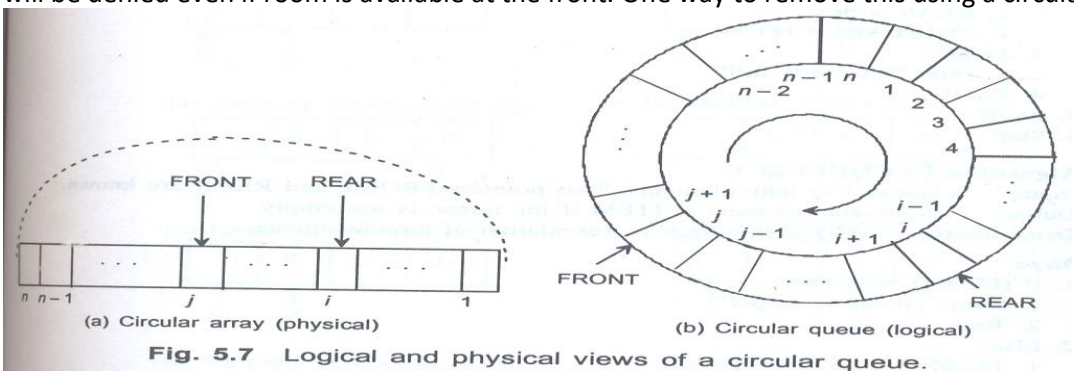
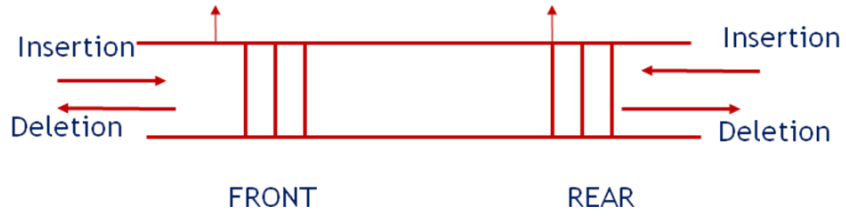


Fig. 5.7 Logical and physical views of a circular queue.

Deque: In De-queue both insertion and deletion operation can be made at either end of the structure . Actually the term is originated from Double Ended Queue.



Types of Deque :- Input restricted queue and Output restricted
Priority queue : variation of queue structure. Each element has been assigned a value called priority of element. An element is inserted at any position in queue according to priority.

Priority queue may be divided into two types mainly, ascending or descending priority.

Element →		A	B			X			P
Priority position →		P_1	P_2			P_i			P_n

3.(a) Explain recursion. Write an iterative and recursive algorithms to evaluate a*b by using addition, where a and b are non negative integers.

When the function calls the function itself within a body of function then this type of structure is called a recursion of function. A procedure is termed as recursive if the procedure is defined by itself.
Sometimes, the best way to solve a problem is by solving a smaller version of the exact same problem first
Recursion is a technique that solves a problem by solving a smaller problem of the same type

Algorithm to evaluate a*b:

Recursive	Iterative
Mul(a,b) If (b=0) Return (0) Endif' S=a + mul(a,b-1) Return(s)	Mul(a,b) S=0 Repeat for i= 1 to b S=s+a Endfor Return(s)

3.(b) Suppose S is string with N characters. Let SUB(S,J,L) denote the substring of S beginning in the position J and having length L. Let a||b denote the concatenation of string A and B. suppose REV (S,N) is recursively defined by

$$REV(S,N) = \begin{cases} S \text{ if } N=1 \\ SUB(S,N,1) \,||\, REV(SUB(S,1,N-1),N-1) \text{ otherwise} \end{cases}$$

Find REV(S,N) when
N=3, S= abc
N=5, S =ababc

REV("abc",3) = (SUB("abc",3,1)||REV(SUB("abc", 1,2),2))
= ("c" || REV("ab",2))
= ("c" || (SUB("ab",2,1)||REV(SUB("ab", 1,1),1))
= ("c" || ("b" || (REV("a",1)))
= ("c" || "b" || "a")
= cba.
Therefore REV("abc",3) is cba.

REV("ababc",5) = (SUB("ababc",5,1)||REV(SUB("ababc", 1,4),4))
= ("c" || REV("abab",4))
= ("c" || (SUB("abab",4,1)||REV(SUB("abab", 1,3),3))
= ("c" || "b" || REV("aba",3)
= ("c"||"b"|| (SUB("aba",3,1)||REV(SUB("aba", 1,2),2))
= ("c" || "b" || "a" || REV("ab",2))
= ("c" || "b" ||"a" ||(SUB("ab",2,1)||REV(SUB("ab", 1,1),1)))
= ("c"|| "b" || "a" || "b" || REV("a",1))
= (" c" || "b" || "a" || " b" || "a")
= cbaba.
3.(c) write iterative and recursive algorithm to generate n Fibonacci sequence.

ITERATIVE DEFINITION	RECURSIVE DEFINITION
FACTORIAL(N) Steps: Fact = 1 For (I = 1 to n) do fact= I * fact 3. EndFor 4. Return(fact) 5. Stop	FACTORIAL(N) Steps: If (N = 0) then Fact = 1 2. Else fact= N * FACTORIAL(N-1) 3. EndIf 4. Return(fact) 5. Stop

3.(d) Explain the simulation of recursion.
Refer summer 2007 3(c)

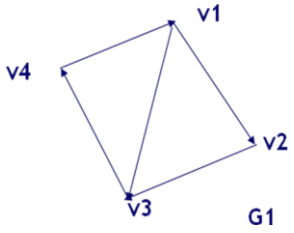
4.(a) Write an algorithm to delete item from threaded binary tree.
Algorithm:
Input: ptr is the pointer of a node that has to be deleted
Output: the threaded binary search tree eliminated with a node ptr or its data content
Data structure: linked structure of threaded binary tree
Steps:
Parent=parent(ptr) //get parent of node ptr
If(ptr.ltag=true) and (ptr.rtag=true) then
Case=1
Else
If(ptr.tag=false) and (ptr.rtag=false) then
Case=3
2.else
1. case=2
3.endif
4.endif
5. do case=1
1. if (parent.rchild=ptr) then
1. parent.rchild = ptr.rchild
2. parent.rtag=true
2. else
1. if (parent.lchild=ptr) then
1. parent.lchild = ptr.rchild
2. parent.ltag=true
2.endif
3.endif
4. return_node(ptr)
5. exit
6.enddo
7. do case=2
1. if (ptr.rtag.false) then
1.child=ptr.rchild

2. else
    1. if (ptr.ltag.false) then
    2. child=ptr.lchild
3.endif
4. if (parent.rchild=ptr) then
    1. parent.rchild = child
5. else
    1. if (parent.lchild=ptr) then
    2. parent.lchild =child
6.endif
7. if (ptr.rtag.false) then
    1. succ=insucc(ptr)
    2. succ.lchild=parent
8. else
    1. if (ptr.ltag.false) then
        1. pred=inpred(ptr)
        2. pred.rchild=parent
    2. endif
9.endif
10.return_node(ptr)
11.exit

8. enddo
9. do case = 3
    1. succ=insucc(ptr)
    2.ptr.data=succ.data
    3. delete_bst_thread(succ)
10. enddo
11.stop.

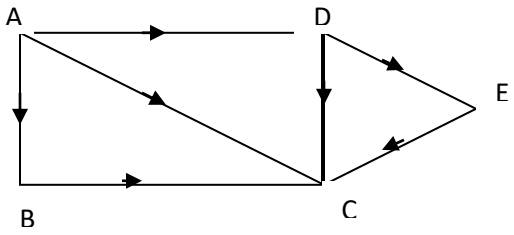
4.(b) define diagraph with suitable example. Explain the depth-first traversal of graph with suitable example.

A Directed graph G is called digraph if edges e of G is assigned a direction.
A digraph is also called a directed graph. It is a graph G,such that G=<V,E>,Where V ia the set of all vertices and E is the set ordered pair of elements from V. For example graph G1 is a digraph where V={v1,v2,v3,v4} E={(v1,v2),(v1,v3),(v2,v3),(v3,v4),(v4,v1)}



DFS
1.Push the Starting vertex into the stack OPEN
2.While OPEN is not empty do

1.POP a vertex V
2.If V is not in VISIT
1.Visit the vertex V
2.Store V in VISIT
3.Push all the adjacent vertex of V onto OPEN
3.EndWhile
4.Stop



We use stack for dfs.
Stack : A (push)
Write: A, stack:B, C, D (pop A, push its neighbor)
Write D, stack: B,C,E
Write E, stack: B,C
Write C, stack: B
Write B, stack: (empty)
Thus DFS is now complete from starting at node A. nodes that are printed : A,D, E, C, B.
4.(c) Write an algorithm for the preorder traversal of the binary tree.
Preorder Traversal
Preorder (ROOT, LEFT, RIGHT)
1.Process the Root R
2.Traverse the Left sub tree of R in Preorder.
3.Traverse the Right sub tree of R in Preorder.
PREORDER(INFO,LEFT,RIGHT,ROOT)
1. [Initially Push NULL onto STACK and initialize PTR]
Set TOP=1,STACK[1]=NULL and PTR:=ROOT
2. Repeat Steps 3 To 5 While PTR ≠ NULL.
Apply PROCESS to INFO[PTR]
[Right Child?]
If RIGHT[PTR] ≠NULL, then:
Set TOP=TOP+1 and
STACK[TOP]=RIGHT[PTR]
[End of If Structure.]
[Left Child?]
If LEFT[PTR] ≠NULL, then:
Set PTR=LEFT[PTR]
ELSE
Set PTR=STACK[TOP] and TOP=TOP-1.
[End of Step 2 loop.]
6.EXIT.

4.(d) suppose a graph G is maintained in memory in the form

Graph(node,next,adj, start,dest,link)
Write an algorithm to find indegegree indeg and the outdegree outdeg of each node of G.

Algorithm :
Degree(node, nextm adj, start, dest,link, indeg, outdeg)
This procedure finds the indegree indeg and outdegree outdeg of each node in the graph G in memory.
(Initialize arrays indeg and outdeg)
Set ptr =start
Repeat while ptr≠null
Set indeg[ptr]=0 and outdeg[ptr]=0
Set ptr=next[ptr]
[End of loop]
Set ptra=start
Repeat steps 4 to 6 while ptra≠null
Set ptrb=adj[ptr1]
Repeat while ptrb≠null
Set outdeg[ptra]=outdeg[ptr1]+1
Indeg[dest[ptrb]]=indeg[dest[ptrb]]+1
Set ptrb=next[ptrb]
Set ptr1=next[ptra]
Return

5.(a) write an algorithm to insert given item at a proper position in sorted array A of N elements.

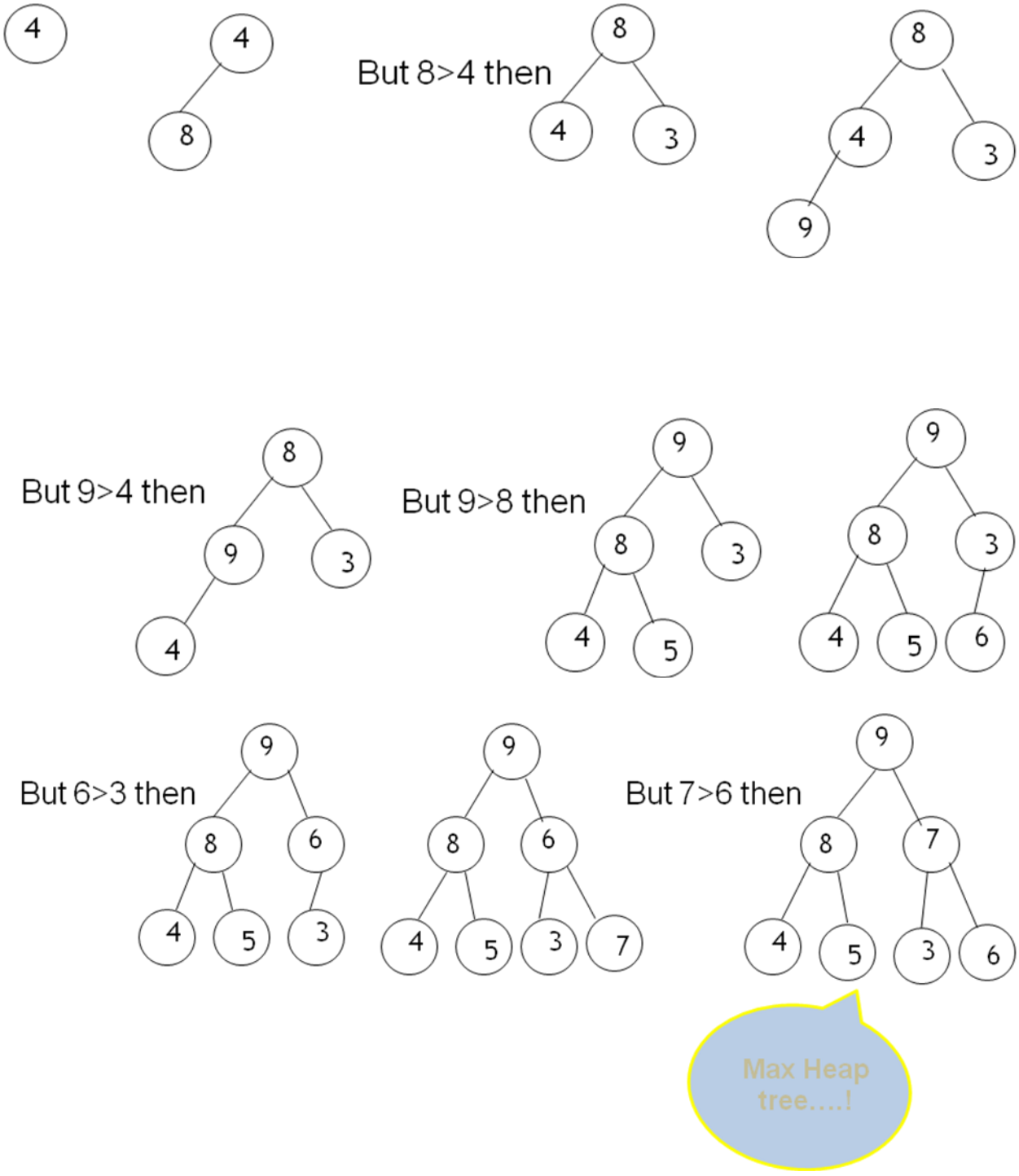
Algorithm INSERT(KEY, L( )
Input: KEY is the item, LO
Output: Array enriched with KEY.
Data structures: An array $A[L \ldots U]$. //L and U are the lower and upper bound of array index

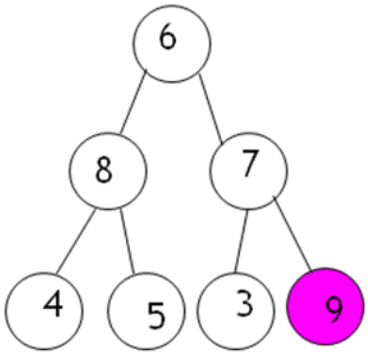
Steps:
1. If A[U] ≠ NULL then
   1. Print "Array is full: No insertion possible"
   2. Exit //End of execution
2. Else
   1. $i = U$ //Start pushing from bottom
   2. While $i >$ A[i]>key
      1. A[$i + 1$] = A[i]
      2. $i = i - 1$
   3. EndWhile
   4. A[LO(I] KEY //Put the element at desired location
   5. $U = U + 1$ //Update the upper index of the array
3. EndIf
4. Stop

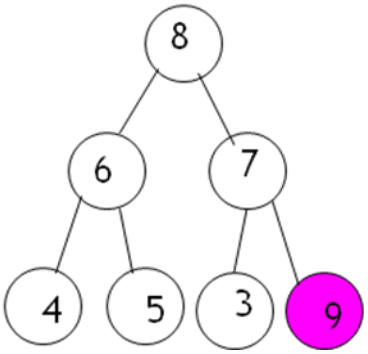
5.(b) Explain the heap sort with suitable example. Discuss the complexity of heap sort method.
The HEAP is a technique for arranging the elements in some proper order. Heap operation is start from left to right order.
Heap Property:

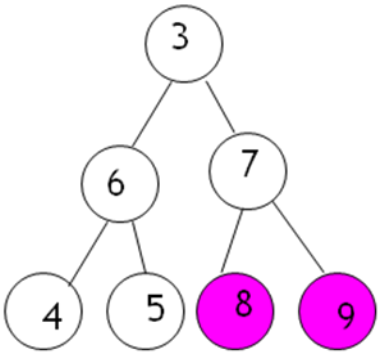
In a heap, for every node i other than the root, the value of a node is greater than or equal (at most) to the value of its parent. [PARENT (i)] ≥ A[i]
Thus, the largest element in a heap is stored at the root.
Example:

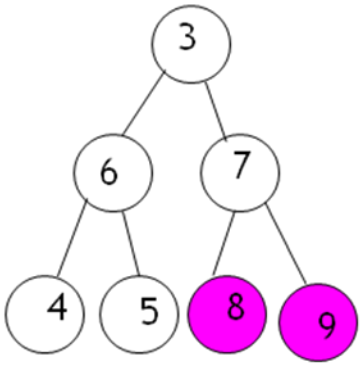
Consider the elements : 4,8,3,9,5,6,7

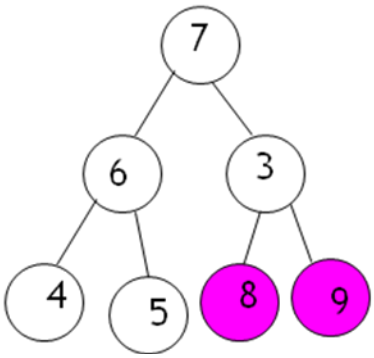


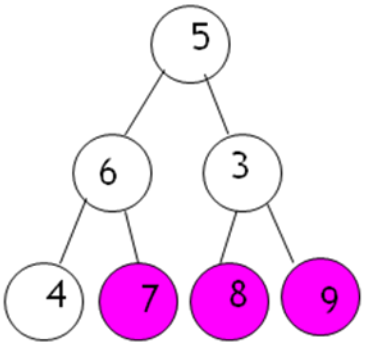




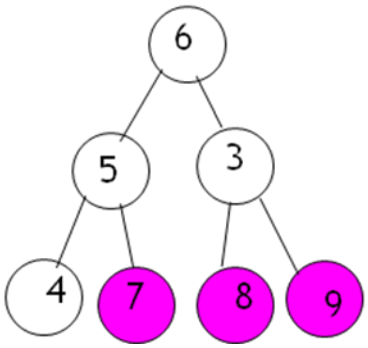


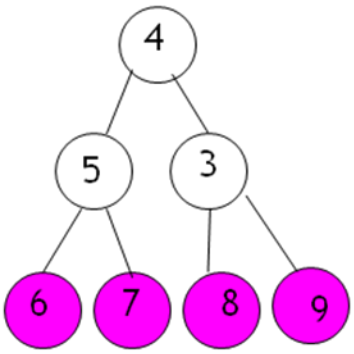




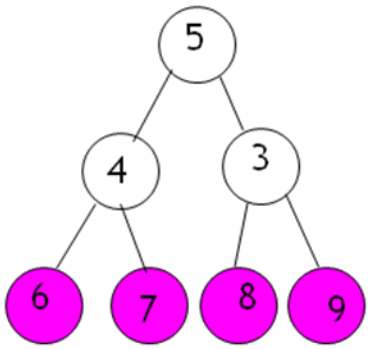


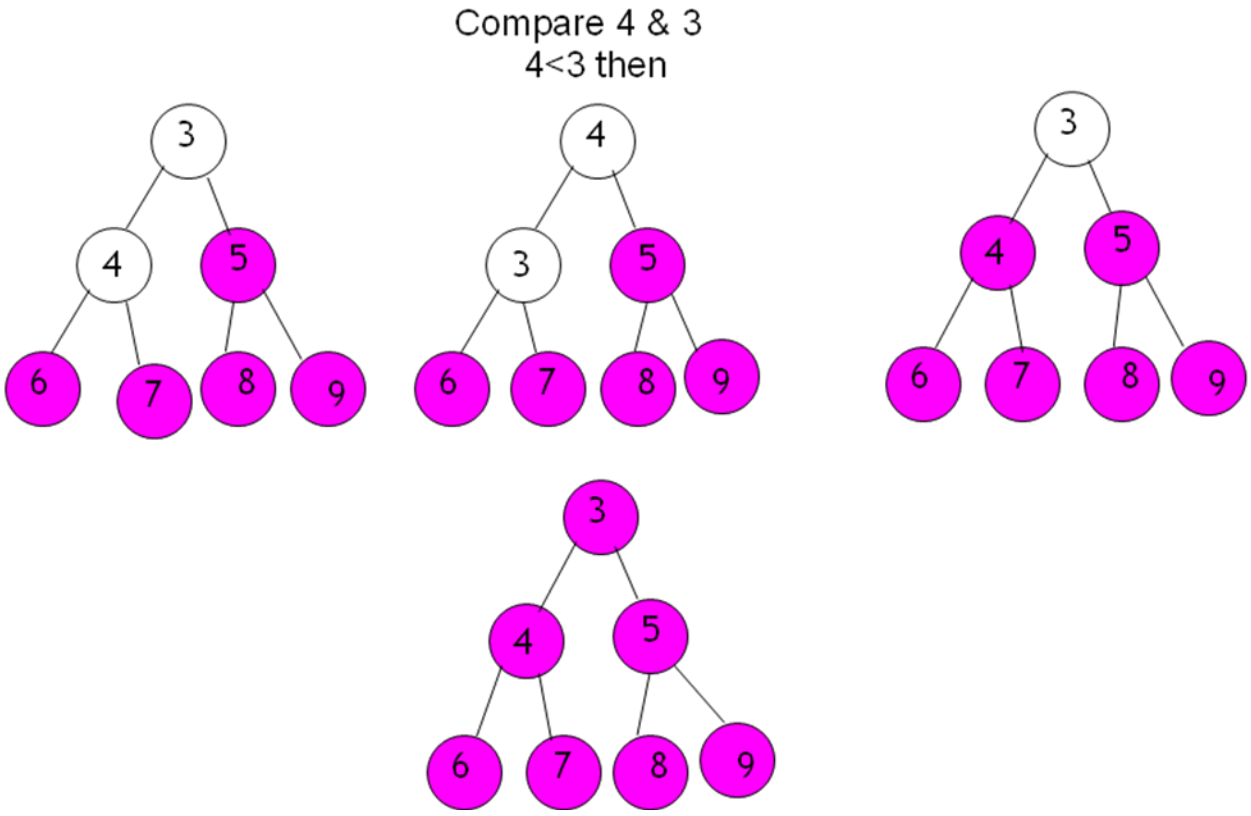
Compare 6 & 3
6<3





Compare 5 & 3
5>3





Complexity:
Average case : O(n log n)
Worst case : O(n logn)

5.(c) explain the bubble sort method with suitable example. Write an algorithm to sort array A of N elements in descending order using bubble sort method.
The bubble sort works by comparing each item in the list with the item next to it, and swapping them if required.
The algorithm repeats this process until it makes a pass all the way through the list without swapping any items (in other words, all items are in the correct order).
This causes larger values to "bubble" to the end of the list while smaller values "sink" towards the beginning of the list.

```
procedure bubbleSort( A : list of sortable items ) defined as:
  do
    swapped := false
    for each i in 0 to length(A) - 2 inclusive do:
      if A[i] > A[i+1] then
        swap( A[i], A[i+1] )
        swapped := true
      end if
    end for
  while swapped
end procedure
```

Step-by-step example

Let us take the array of numbers "5 1 4 2 8", and sort the array from lowest number to greatest number using bubble sort algorithm. In each step, elements written in bold are being compared.

First Pass:

(5 1 4 2 8) (1 5 4 2 8), Here, algorithm compares the first two elements, and swaps them.

(1 5 4 2 8) (1 4 5 2 8), Swap since 5 > 4

(1 4 5 2 8) (1 4 2 5 8), Swap since 5 > 2

(1 4 2 5 8) (1 4 2 5 8), Now, since these elements are already in order (8 > 5), algorithm does not swap them.

Second Pass:

(1 4 2 5 8) (1 4 2 5 8)

(1 4 2 5 8) (1 2 4 5 8)

(1 2 4 5 8) (1 2 4 5 8)

(1 2 4 5 8) (1 2 4 5 8)

Now, the array is already sorted, but our algorithm does not know if it is completed. The algorithm needs one whole pass without any swap to know it is sorted.

Third Pass:

(1 2 4 5 8) (1 2 4 5 8)

(1 2 4 5 8) (1 2 4 5 8)

(1 2 4 5 8) (1 2 4 5 8)

(1 2 4 5 8) (1 2 4 5 8)

Finally, the array is sorted, and the algorithm can terminate.

5.(d) Explain insertion sort method with suitable example. Write an algorithm to sort array A of N elements in ascending order using insertion sort method

Refer summer 2008 5(d)

YSS/K.5-708
SECOND SEMESTER MASTER IN COMPUTER APPLICATION
DATA STRUCTURES
Paper- 2CSA1

1.(a) write an algorithm and explain with neat diagram following operations on a linked list.

1) Insertions of a node at various position in a linked list
2) Insertion of a node at the end of a doubly linked list.

1) insertions node at front

1.new=GETNODE(NODE)
2.If (new = NULL) then
    1. print "Memory underflow : No Insertion"
    2. EXIT
3.Else
    1. new.LINK = HEADER.LINK
    2. NEW.DATA=x
    3. HEADER.LINK=new
4.EndIf
5.Stop

Insertion of node at end

. new=GETNODE(NODE)
2. If (new = NULL) then
    1. print "Memory is insufficient : Insertion is not possible"
    2. EXIT
3. Else
    1. ptr=HEADER
    2. While(ptr.LINK ≠NULL) do
        1. ptr=ptr.LINK
    3. EndWhile
    4. ptr.LINK=new
    5. new.DATA=X
    6. new.LINK=NULL
4. EndIf
5. Stop

Insertion of node at any position

. new=GETNODE(NODE)
2. If (new = NULL) then
    1. print "Memory is insufficient : Insertion is not possible"
    2. EXIT
3. Else
    1. ptr=HEADER
    2. While(ptr.DATA ≠KEY) and (ptr.LINK ≠NULL) do
        1. ptr=ptr.LINK
    3. EndWhile
    4. If (ptr.LINK=NULL)then

1. Print"KEY is not available in the list"
2. Exit

5. ELSE
    1. new.LINK=ptr.LINK
    2. new.DATA=X
    3. ptr.LINK=new

6.EndIf

4. EndIF
5. Stop

2) insertion of node at end of doubly linked list

Insertion of a node at the end. The algorithm INSERT_DL_END is to insert a node at the end into a double linked list.

Algorithm INSERT_DL_END(X)
Input: X the data content of the node to be inserted.
Output: A double linked list enriched with a node containing data as X at the end of the list.
Data structure: Double linked list structure whose pointer to the header node is HEADER.

Steps:

1. ptr = HEADER
2. While (ptr.RLINK ≠ NULL) do //Move to the last node
    1. ptr = ptr.RLINK
3. EndWhile
4. new = GETNODE(NODE) //Avail a new node
5. If (new ≠ NULL) then //If the node is available
    1. new.LLINK = ptr //Change the pointer shown as 1 in Figure 3.11(b)
    2. ptr.RLINK = new //Change the pointer shown as 2 in Figure 3.11(b)
    3. new.RLINK = NULL //Make the new node as the last node
    4. new.DATA = X //Copy the data into the new node
6. Else
    1. print "Unable to allocate memory: Insertion is not possible"
7. EndIf
8. Stop

1.(b) Write an algorithm to print the linked list in a reverse order.

Reverse_list(start)

Where start indicates the address of the first node and consist of two parts

Info and address of new node

Ptr- a local pointer variable which keep the track of current node

Ptr=start

Rptr=null

Repeat while ptr!=null

Temp=rptr

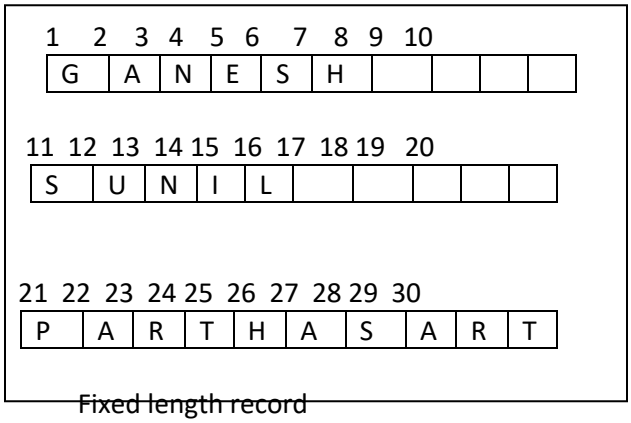
Rptr=ptr

Ptr=ptr→link //update pointer to next node

rptr→link=temp //link to preceeding node

(end while)
Start=rptr
Return
To display the list
Repeat while ptr!=null
  Process ptr→info
        Ptr=ptr→link
endwhile

1.(c) In what way records differ from a linear array? Explain the representation of records in a memory.
In a fixed length storage, all records have same length. But in case of array, each one can b of different length.
Disadvantages of record oriented structure : More time is required for reading an entire record if record is padded with blank space
When correction consists of few or more characters then string value has to be rewritten.
Representation of records in memory:



.

Fixed length record

1.(d) write an algorithm and explain with a neat diagram following operations on a linear array.
(i) Insertion of an element at a given location
(ii) Deletion of given element

Operations on linear array:
Traversing: Accessing each elements exactly once.
Searching: Finding the location of element.
Inserting: Adding new element.
Deleting: Removing a element.
Sorting: Arranging the elements in some logical order.
Merging: Combining the elements of two different sorted arrays into a single array.
Insertion of element at given element

Algorithm INSERT(KEY, LOCATION)
Input: KEY is the item, LOCATION is the index of the element where it is to be inserted.
Output: Array enriched with KEY.
Data structures: An array $A[L \ldots U]$. //L and U are the lower and upper bound of array index

Steps:

1. If A[U] ≠ NULL then
   1. Print "Array is full: No insertion possible"
   2. Exit //End of execution
2. Else
   1. $i = U$ //Start pushing from bottom
   2. While $i >$ LOCATION do
      1. A[$i + 1$] = A[i]
      2. $i = i - 1$
   3. EndWhile
   4. A[LOCATION] = KEY //Put the element at desired location
   5. $U = U + 1$ //Update the upper index of the array
3. EndIf
4. Stop

Deletion of a given element

Algorithm DELETE(KEY)
Input: KEY the element to be deleted.
Output: Slimed array without KEY.
Data structures: An array A[$L \ldots U$]. //L and U are the lower and upper bound of //array index

Steps:

1. i = SEARCH_ARRAY(A, KEY) //Perform the search operation on A and return //the location
2. If ($i = 0$) then
   1. Print "KEY is not found: No deletion"
   2. Exit //Exit the program
3. Else
   1. While $i < U$ do
      1. A[i] = A[$i + 1$] //Replace the element by its successor
      2. $i = i + 1$
   2. EndWhile
4. EndIf
5. A[U] = NULL //The bottom most element is made empty //(*see* Figure 2.6)
6. $U = U - 1$ //Updated the upper bound now
7. Stop

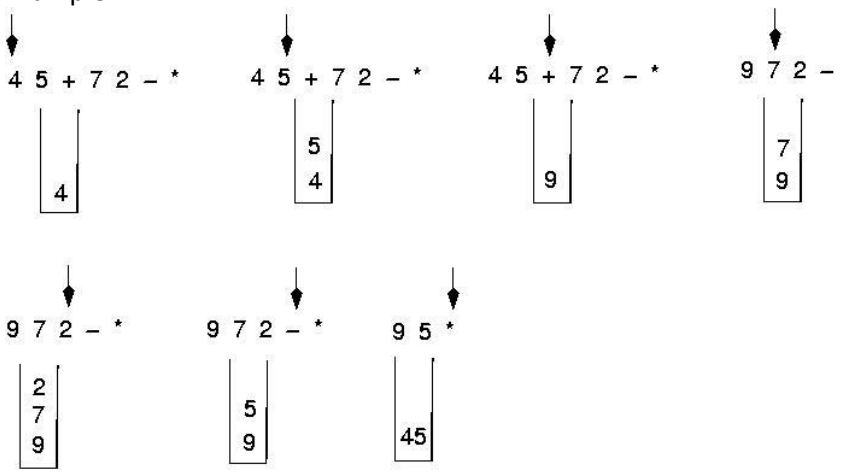
2.(a) What is priority queue? Explain the array representation of priority queue in memory.

Refer page no: 51 2(d) summer 2006

2.(b)Define stack? How is stack useful for an arithmetic expression? Explain in brief.
A stack is simply a list of elements with insertion & deletion permitted at one end called the Stack Top.

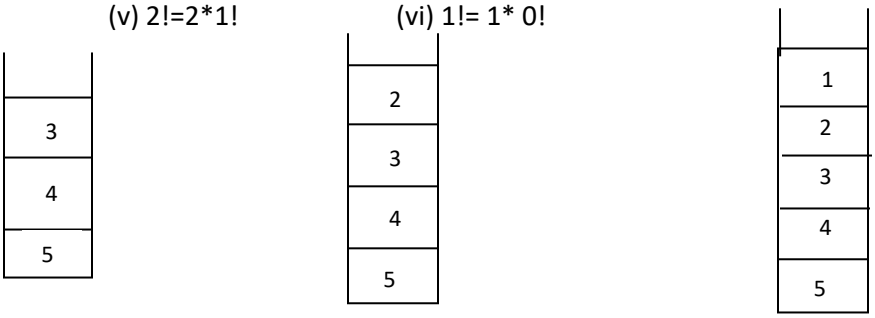
The stack is also called LIFO, because it is possible to remove elements from a stack in reverse order from the insertion of elements into the stack.
PUSH and POP are the operation that are provide for insertion of an elements into the stack and the removal of an elements from stack.

Use a stack to evaluate an expression in postfix notation.
The postfix expression to be evaluated is scanned from left to right.
Variables or constants are pushed onto the stack.
When an operator is encountered, the indicated action is performed using the top elements of the stack, and the result replaces the operands on the stack.
Each operator in a postfix string refers to the previous two operands in the string.
Suppose that each time we read an operand we push it into a stack. When we reach an operator, its operands will then be top two elements on the stack
We can then pop these two elements, perform the indicated operation on them, and push the result on the stack.
So that it will be available for use as an operand of the next operator.
Example:

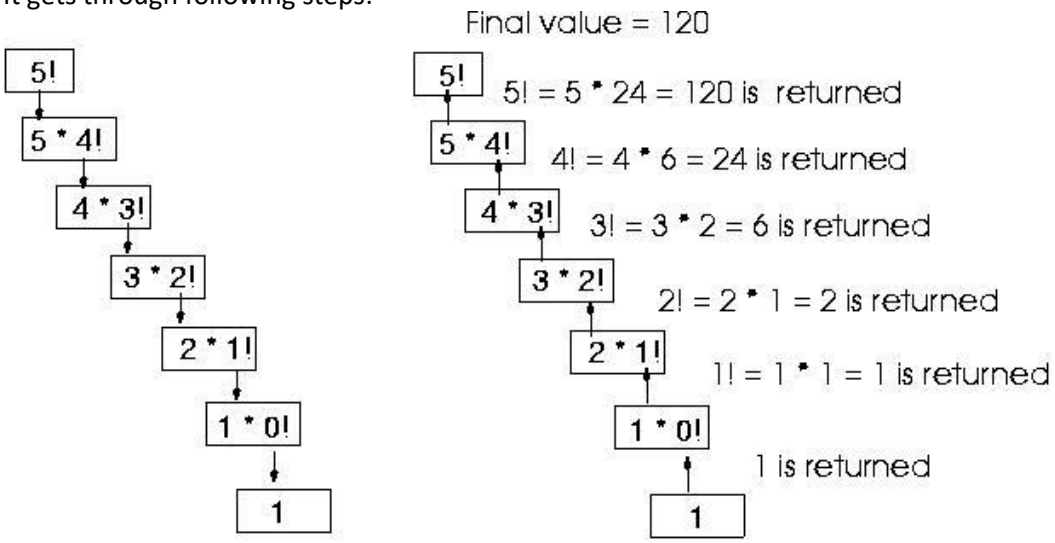


2.(d) How stack can be used for factorial calculation? Explain.
For factorial calculation we can use recursion technique. Recursion is an application of stack
RECURSIVE
DEFINITION
FACTORIAL(N)
Steps:
If (N = 0) then
Fact = 1
2. Else
fact= N * FACTORIAL(N-1)
3. EndIf
4. Return(fact)
5. Stop
Initially stack is empty (ii) 5! = 4*3! (iii) 4!= 4 *3!

3!= 3*2! (v) 2!=2*1! (vi) 1!= 1* 0!



It gets through following steps:



3(a) Write an algorithm/program for translation from prefix to postfix using recursion.

```
Void conver(char prefix[], char poftfix[])
{
        Char opnd1[maxlenght],opnd2[maxlength];
        Char post1[maxlength], post2[maxlenght];
        Car temp[maxlength];
        Car op[1];
        Int length;
        Int I,j,m,n;
        If(length = strlrn(prefix) == 1) {
                If(isalpha(prefix[0])) {
                /* the prefix is a single letter*/
                Postfix[0]=prefix[0];
                Postfix[i]='\0';
                Return;
                } /*endif*/
                Print("illegal prefix string*/
                Exit(1);
                } /* end ir */
//the prefix string is longer than a single character, extraxt the operator and the two operand lengths*/
        Op[0]=prefix[0];
        Op[1]='\0';
```

```
                Substr(prefix,1,length-1,temp);
                M=find(temp);
                Substr(prefix,m+1,length-m-1,temp);
                N=find(temp);
                If((op[0]!= '+' && op[0] != '-' && op[0] != '+' && op[0] != '*' && op[0] != '/' &&
                                || (m==0) || (n==0) || (m+n+1) != length)) {
                                Print("illegal prefix string");
                                Exit(1);
                        } //end if
Substr(prefix,1,m,opnd1);
Substr(prefix,m+1,n,opnd2);
Convert(opnd1,post1);
Convert(opnd2,post2);
Strcat(post1,post2);
Strcat(post1, op);
Substr(post1,0,lemgth,postfix);
} //end convert//
```

3.(b)Let A be an array of integers. Presetm recursive algorithm to compute:

(i) the maximum element of an array

(ii) the sum of the elements of array

(iii)The average of the elements of the array

Recursive algorithm to find maximum element in an array

```
Max(I,j,max)
Where I,j – lower and upper bound of index of array
Max- largest value in array
La is global array with N element 1<=i<=<=j<=n
If(i=j)
Max=la[i]
Return
        Endif
If(i=j-1)
If(la[i]>la[j])
        Max=la[i]
Else
        Max=la[j]
Endif
Endif
Mid=int((i+j)/2))
Max(I,mid,max)
Max(mid+1,j,max1)
        If(max1>max)
                Max=max1
        Endif
Return
```

Recursive algorithm to find sum of elements

```
Sum(a,n)
If (n=0)
 Return(1)
Endif
s=a[n]+sum(a,n-1)
Return(s)
```

Recursive algorithm to find average of elements in array

```
Average(a,n,m)
Where a is linear array with n elements
M is integer and ititally contains 0
If(n=0)
Return(1)
Endif
Sum=a[n] +average(a,n-1,m+1)
Return(sum/m)
```

3.(c) Explain and write a recursive algorithm for the multiplication of natural numbers.

```
Mul(n)
Where n is positive integer number
If(n=1)  //base crieteria
Return(1)
End if
M=n* mul(n-1)  //arithmetic call
Return(m)
```

```
Mul(3) = 3 *maul(2)
         =3*2*mul(1)
         =3*2*1
         =6,
```

3.(d) Explain simulation of recursive factorial function.

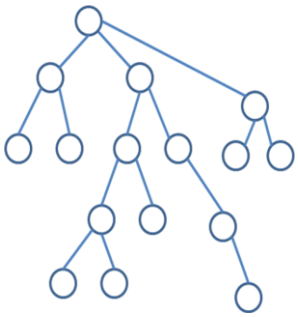
Refer 2(d) page no. 67

4.(a) define:-

(i) Tree (ii) binary tree (iii) full binary tree (iv) complete binary tree

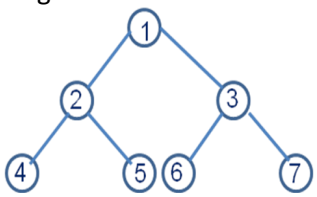
Explain the different properties of binary tree.

Trees are very useful data structure, where elements appear in a non-linear fashion, which require two dimensional representations.

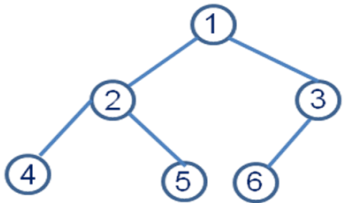


(ii) A binary tree T is defined as finite set of elements, called nodes. Such that:
T is empty (called the null tree or empty tree)
T contains a distinguished node R, called root of T, and the remaining nodes of T from an ordered pair of disjoint binary trees T1 and T2.

(iii) Full Binary Tree : If it contains maximum possible number of nodes in all level. Ex: binary tree of height 3



(iv) Complete Binary Tree: If all its level, except possibly the last level, have maximum number of possible nodes, and all the nodes at the last level appear as far left as possible.



Properties of binary tree:
Node : Stores actual data and links to other node.
External node and Internal node
Parent : Immediate predecessor of a node.
Child : If the immediate predecessor of a node is the parent of the node then all immediate successors of a node are known as child.
Link : Pointer to a node in a tree. There may be more than two links of a node.
Root : Specially designated node which has no parent.

Structure of a node in a tree

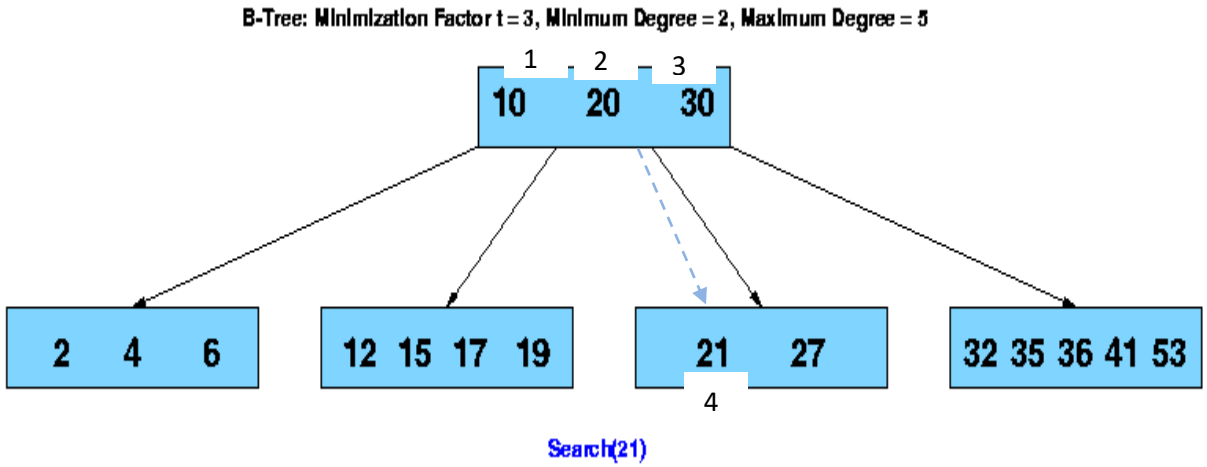
4.(b) Define B tree. What are the various operations that can be done on B-tree? Explain any one of them with neat diagram.
-trees are balanced trees that are optimized for situations when part or all of the tree must be maintained in secondary storage such as a magnetic disk. Since disk accesses are expensive (time consuming) operations, a b-tree tries to minimize the number of disk accesses. For example, a b-tree with a height of 2 and a branching factor of 1001 can store over one billion keys but requires at most two disk accesses to search for any node

A B tree of order m, if o empty is an –way search tree I which
The root as at least two child odes and at most child odes
The iteral odes except the root have at least [m/2] child ad at most m child odes
The number of keys in each internal node is one less than the number of child nodes and these keys partition the keys in the subtrees of the node in a manner similar to tat m-way search trees.
All leaf nodes are on the same level

A B-tree of order 3 is referred to as 2-3 tree since the internal nodes are of degree 2 or 3 only
Operations on B-Trees
The algorithms for the *search*, *create*, and *insert* operations. Note that these algorithms are single pass; in other words, they do not traverse back up the tree. Since b-trees strive to minimize disk accesses and the nodes are usually stored on disk, this single-pass approach will reduce the number of node visits and thus the number of disk accesses. Simpler double-pass approaches that move back up the tree to fix violations are possible.
Since all nodes are assumed to be stored in secondary storage (disk) rather than primary storage (memory), all references to a given node be be preceeded by a read operation denoted by *Disk-Read*. Similarly, once a node is modified and it is no longer needed, it must be written out to secondary storage with a write operation denoted by *Disk-Write*. The algorithms below assume that all nodes referenced in parameters have already had a corresponding *Disk-Read* operation. New nodes are created and assigned storage with the *Allocate-Node* call. The implementation details of the *Disk-Read*, *Disk-Write*, and *Allocate-Node* functions are operating system and implementation dependent.

```
B-Tree-Search(x, k)
i <- 1
while i <= n[x] and k > key_i[x]
   do i <- i + 1
if i <= n[x] and k = key_i[x]
   then return (x, i)
if leaf[x]
   then return NIL
   else Disk-Read(c_i[x])
      return B-Tree-Search(c_i[x], k)
```

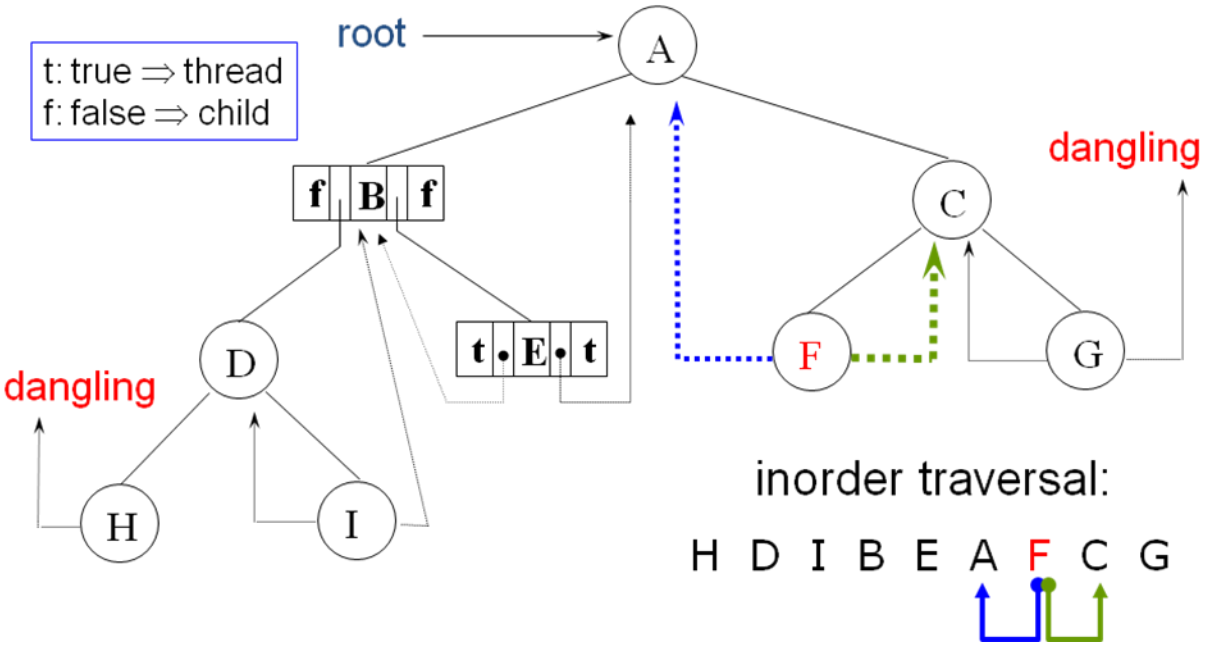
The search operation on a b-tree is analogous to a search on a binary tree. Instead of choosing between a left and a right child as in a binary tree, a b-tree search must make an n-way choice. The correct child is chosen by performing a linear search of the values in the node. After finding the value greater than or equal to the desired value, the child pointer to the immediate left of that value is followed. If all values are less than the desired value, the rightmost child pointer is followed. Of course, the search can be terminated as soon as the desired node is found. Since the running time of the search operation depends upon the height of the tree, *B-Tree-Search* is $O(\log_t n)$.
Searching a B-Tree for Key 21



4.(c) What is graph traversal? Explain in details BFS traversal algorithm.
Traversal: To visit all the nodes in a graph exactly once.
Refer 0age no:17. Summer2008 4(d)

4.(d) List the different types of binary tree. Draw and explin representation of threaded binary tree. What are advantages of threaded binary tree again non- threaded binary tree.
Different types of binary tree:
Expression tree
Binary Search Tree
Heap Tree
Threaded Binary Tree
Height Balanced Tree (AVL tree)
Huffman Tree
Decision Tree

Threads
drawback of the binary tree:
Too many null pointers in current representation of binary trees
n: number of nodes
number of non-null links: *n*-1
total links: 2*n*
null links: 2*n*-(*n*-1) = *n*+1
Solution: replace these null pointers with some useful “threads”
Rules for constructing the threads
If ptr->left_child is null, replace it with a pointer to the node that would be visited *before* ptr in an *inorder traversal*
If ptr->right_child is null, replace it with a pointer to the node that would be visited *after* ptr in an *inorder traversal*



Two additional fields of the node structure, left-thread and right-thread
If ptr->left-thread=*TRUE*,
then ptr->left-child contains a thread;
Otherwise it contains a pointer to the left child.
Similarly for the right-thread

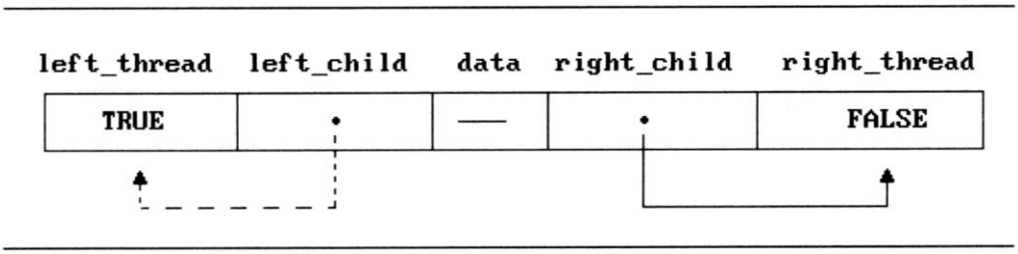


Figure 5.22: An empty threaded tree

If we don't want the left pointer of *H* and the right pointer of *G* to be dangling pointers, we may create root node and assign them pointing to the root node

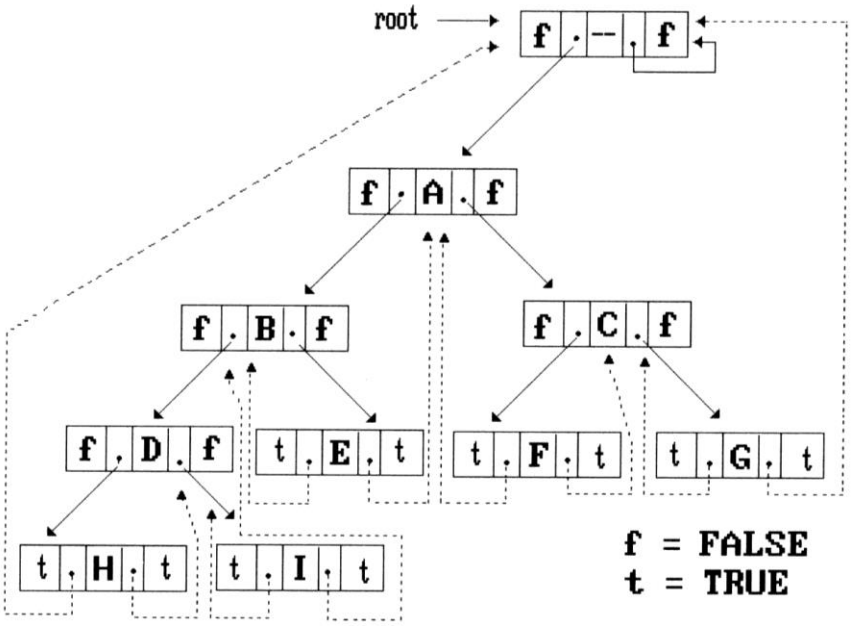


Figure 5.23: Memory representation of a threaded tree

5.(a) what is heap? Write an algorithm to insert an element into heap.
The heap abstract data type
Definition: A *max*(*min*) *tree* is a tree in which the key value in each node is no smaller (larger) than the key values in its children. A *max* (*min*) *heap* is a complete binary tree that is also a *max* (*min*) *tree*
Basic Operations:
creation of an empty heap
insertion of a new elemrnt into a heap
deletion of the largest element from the heap
The examples of max heaps and min heaps
Property: The root of max heap (min heap) contains the largest (smallest) element

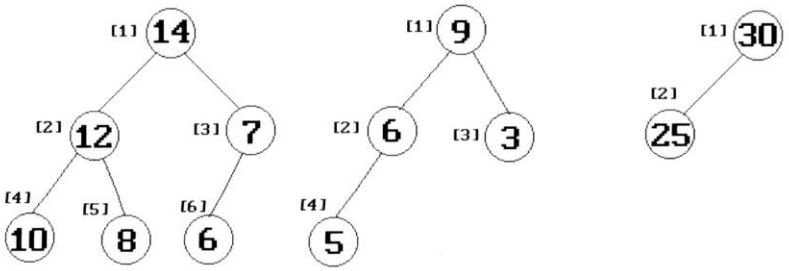


Figure 5.25: Sample max heaps

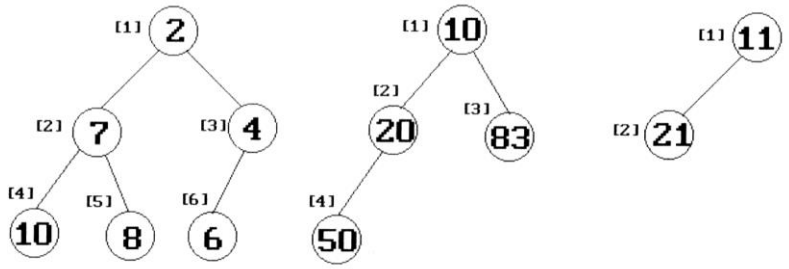


Figure 5.26: Sample min heaps

Insertion Into A Max Heap

Analysis of *insert_max_heap*

The complexity of the insertion function is $O(\log_2 n)$

```
void insert_max_heap(element item, int *n)
{
/*insert item into a max heap of current size *n */
   int i;
   if (HEAP_FULL(*n)){
      fprintf(stderr, "The heap is full. \n");
      exit(1);
   }
   i = ++(*n);
   while ((i != 1) && (item.key > heap[i/2].key)) {
      heap[i] = heap[i/2];
      i /= 2;
   }
   heap[i] = item;
}
```

Program 5.13: Insertion into a max heap

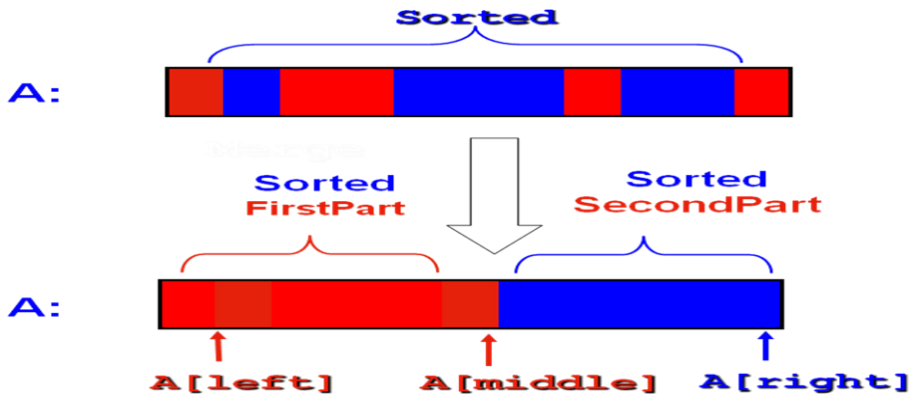
5.(b) Write an algorithm for implemention quick sort. Comment on the efficiency of quick sort.
Refer page no:18, summer 2008- 5.(a)

5.(c) Explain the logic of merge sort with suitable example.

```
Merge-Sort (A, left, right)
        if    left ≥ right   return
        else
middle ← b(left+right)/2⌋
Merge-Sort(A, left, middle)
Merge-Sort(A, middle+1, right)
Merge(A, left, middle, right)

Merge(A, left, middle, right)
n1 ← middle – left + 1
n2 ← right – middle
create array L[n1], R[n2]
for i ← 0 to n1-1 do L[i] ← A[left +i]
for j ← 0 to n2-1 do R[j] ← A[middle+j]
k ← i ← j ← 0
while i < n1 & j < n2
          if L[i] < R[j]
                  A[k++] ← L[i++]
          else
                  A[k++] ← R[j++]
while i < n1
        A[k++] ← L[i++]
```

while $j < n_2$

A[k++] ← R[j++]



Suppose the array A contains 14 elements as follows:

66, 33, 40, 22, 55, 88, 60, 11, 80, 20, 50, 44, 77, 30

Each pass of the merge-sort algorithm will start at the beginning of the array A and merge pairs of sorted subarrays as follows:

Pass 1. Merge each pair of elements to obtain the following list of sorted pairs:

33, 66 22, 40 55, 88 11, 60 20, 80 44, 50 30, 77

Pass 2. Merge each pair of pairs to obtain the following list of sorted quadruplets:

22, 33, 40, 66 11, 55, 60, 88 20, 44, 50, 80 30, 77

Pass 3. Merge each pair of sorted quadruplets to obtain the following two sorted subarrays:

11, 22, 33, 40, 55, 60, 66, 88 20, 30, 44, 50, 77, 80

Pass 4. Merge the two sorted subarrays to obtain the single sorted array

11, 20, 22, 30, 33, 40, 44, 50, 55, 60, 66, 77, 80, 88

The original array A is now sorted.

5.(d) Explain bubble sort? why it is named so? Write an algorithm to implement bubble sort.
Refer page no: 60 summer 2006- 5(c)

1.(a) Explain the double linked list with suitable example. Write an algorithm to insert item at the end of double lined list.
Refer page no:62 summer 2005- 1(a)

1.(b) What is an abstract data typre? Write an ADT specification for varying length data string.
We use sequence notation in defining an ADT, there are 4 basic operations normally included in system that support variable length string.
Length : a function that returns current length of string
Concat: a function that return the concatenation of its input string
Substr: a function that returns substring of a given string.
Pos: a function that returns the first position of one string as a substring of another

Abstract typedef<<char>>string;

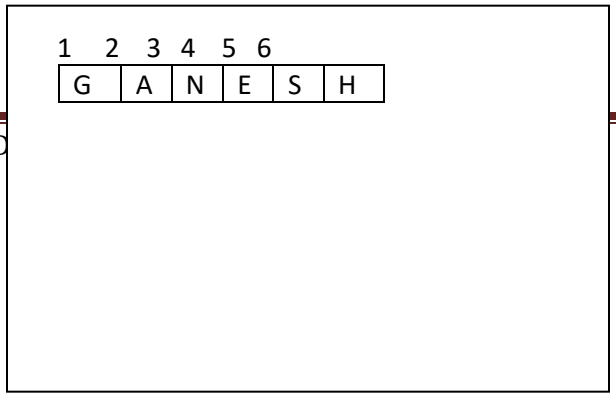
Abstract length(s)
String s;
Postcondition length==len(s)

Abstract string concat(s1,s2)
String s1,s2;
Postcondition concat==s1+s2

Abstract string substr(s1,i,j)
String s1;
Int I,j;
Precondition 0<=i<len(s1);
0<=j<=len(s1)-I;
Postcondition substr ==sub(s1,I,j);

Abstract pos(s1,s2)
String s1, s2;
Postcondition /*lastpos=len(s1)-len(s2)*/
((pos==-1) && (for(i=0;i<=lastpos;i++) (s2<>sub(s1,I,len(s2)))))
||
((pos>=0) && (pos<=lastpos) && (s2== sub(str1,pos,len(S2))
&& (for(i=1;i<pos;i++)
(s2<> sub(s1,I,len(s2)))));

Variable length with sentinals



D

11	12	13	14	15
S	U	N	I	L

.

21	22	23	24	25	26	27	28	29	30
P	*A*	*R*	*T*	*H*	*A*	*S*	*A*	*R*	*T*

1.(c) Explain the representation of single linked list in memory as an array. What are limitations of array implementations of the linked list? Write an algorithm to insert the item at the front of a single linked list.

Refer page no:2 summer 2008- 1(b)

Page no: 61 , summer 2005 – 1(a).

1.(d) write an algorithm to merge two sorted arrays into single array.

Refer page no-76, summer 2005- 5(c)

2.(a) What is stack? Explain the representation of stacks as an array and linked list in memory.

Refer page no-5, summer 2008-2(a)

Refer page no:29

2.(b) Explain the concept of circular queue. Write an algorithm to insert and remove the element from the circular queue.

Refer page no:6, summer 2008- 2(b)

Algorithm ENCQUEUE(ITEM)

Input: An element ITEM to be inserted into the circular queue.

Output: Circular queue with the ITEM at FRONT, if the queue is not full.

Data structures: CQ be the array to represent the circular queue. Two pointers FRONT and REAR are known.

Steps:

```
1. If (FRONT = 0) then                          //When queue is empty
   1. FRONT = 1
   2. REAR = 1
   3. CQ[FRONT] = ITEM
2. Else                                         //Queue is not empty
   1. next = (REAR MOD LENGTH) + 1
   2. If (next ≠ FRONT) then                    //If queue is not full
      1. REAR = next
      2. CQ[REAR] = ITEM
   3. ELSE
      1. Print "Queue is full"
   4. EndIf
3. EndIf
4. Stop
```

2.(c) Translate the following infix expression into prefix and postfix form:

(i) (A+B)*(C$(D-E)+F)-G

(II) A+((B-C)*(D-E)+F/G)$(H-J))

(A+B)*(C$(D-E)+f)-G
(AB+)*(C$(DE-)F+)-G
(AB+)*(CDE-$F+)-G
POSTFIX=AB+CDE-$F+G*-
PREFIX= -*+AB+$C-DEFG
(II) A+((B-C)*(D-E)+F/G)$(H-J))
A+((BC-)(DE-)*F+G/)HJ-$
POSTFIX=ABC-DE-*F+G/HJ-$+
PREFIX=+A$/+*-BC-DEFG-HJ.

2.(d) What is queue? Explain the sequential representation of queues in memory. Write ADT specification for the queue.
Refer page no:6 & 8, summer 08- 2(a) &a(d)
3.(a)What is recursion? Write an iterative and recursive algorithm to find the factorial of a given number.
Refer page no:50 & 52. Summer 06- 3(a) &3(c)
3.(b) Let A be an integer array with N elements suppose X is an integer function defined by

$$X(K)=X(A,N,K) = \begin{cases} 0 & \text{if } K=0 \\ X(K-1)+A(K) & \text{if } 0<K<=N \\ X(K-1) & \text{if } K>N \end{cases}$$

Find X(5) for each of the following arrays:-
N=8, A : 3,7,-2,5,6,-4,2,7
N=3, A : 2,7,-4
What does this function do?
Refer page no:34, summer 07- 3(b)

3.(c) Explain the tower of Hanoi problem. Write a recursive algorithm of tower of Hanoi problem.
Refer page no:33 summer 07-3(a)
3.(d) Let M and N be integers and suppose F(M,N) is recursively defined by,

$$F(M,N) = \begin{cases} 1 \text{ if } M=0 \text{ or } M>=N>=1 \\ F(M-1,N)+F(M-1,N-1) \text{ otherwise} \end{cases}$$

Find F(4,2), F(1,5) and F(2,4).

F(4,2) = 1 since M>=N>=1 i.e 4>=2>=1

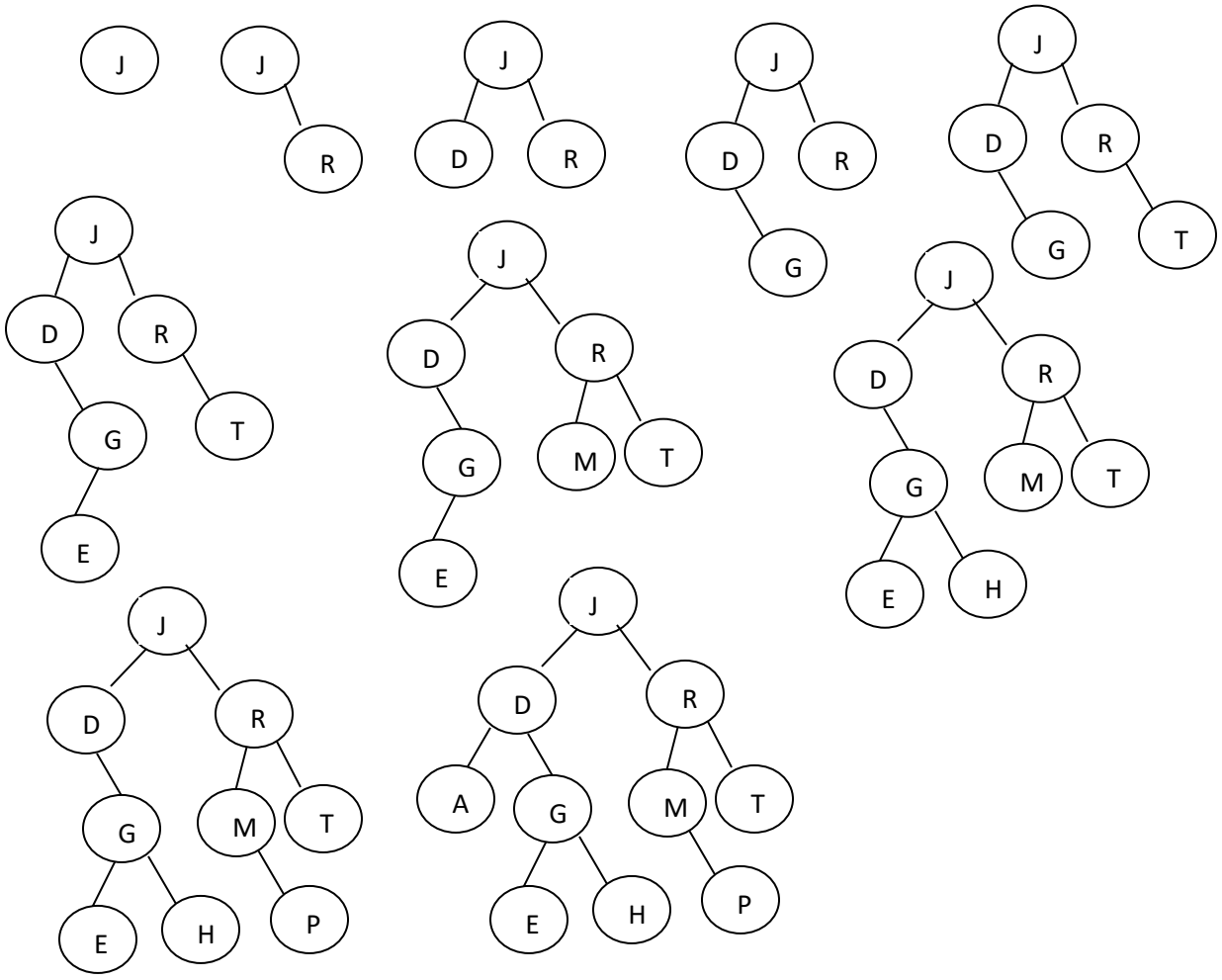
F(1,5) = F(M-1, N) + F(M-1, N-1)
= F(1-1,5) + F(1-1, 5-1)
= F(0,5) + F(0,4) since m=0 i.e F(0,5) =1
= 1+1
=2.

F(2,4) = F(M-1, N) + F(M-1, N-1)
= F(2-1, 4) + F(2-1,4-1)
= F(1,4) + F(1,3)

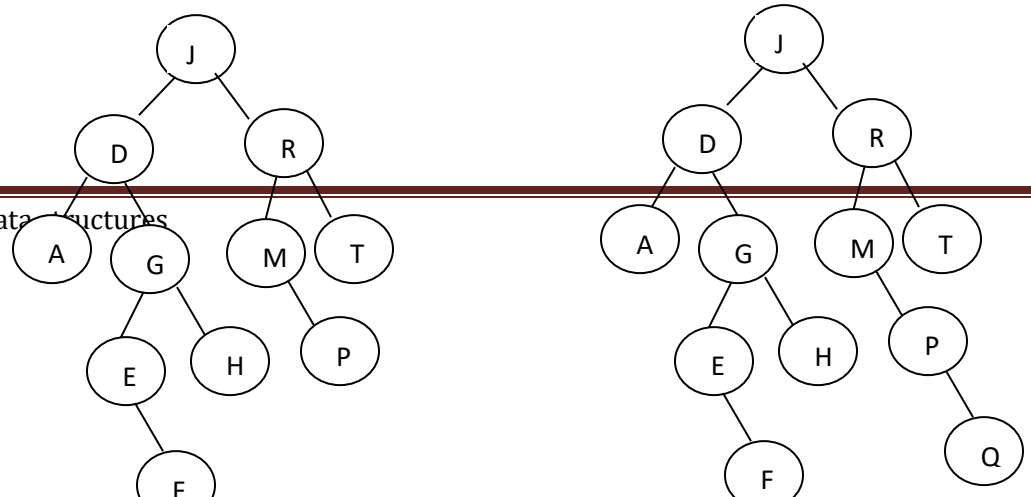
= F(1-1,4) + F(1-1, 3-1)
= F(0,4) + F(0,2) since m=0 i.e F(0,4) =1
= 1 +1
= 2.

4.(a) Define binary tree, suppose the following list of letters is inserted in order into an empty binary search tree:
J, R, D, G, T, E, M, H, P, A, F, Q
Find the final tree T
Find the inorder and postorder traversal binary tree T.

Refer page no-13.



Final tree T is :



(ii) Inorder traversal of tree T : AD FEGH J QPMRT
Postorder traversal of tree T : A FEHG DQP MTR J

4.(b) Write an algorithm to delete the item from the graph.
Refer page no: 15, summer 08- 4(b).

4.(c) Write an algorithm for postorder traversal of binary tree.
Refer page no: 16, summer 08- 4(c)

4.(d) Define connected graph. Explain breadth first traversal of graph with suitable example.
Refer Page no:17. Summer2008 4(d)
Refer page no: 55. 4(b)

5.(a) Explain any one method of searching an ordered table.
Refer page no:21 summer 08- 5(b)

5.(b) Explain quick sort method with suitable example.
Refer page no: 18 summer 08- 5(a)

5.(c) Write an algorithm to sort the array A of N elements using selection sort method. Discuss the complexity of selection sort method.
Refer page no: 43 summer 07- 5(c)

5.(d) explain the merge sort method with suitable example.
Refer page no: 77, summer 05 – 5(c)
NLR/W/05/708
SECOND SEMESTER MASTER IN COMPUTER APPLICATION
DATA STRUCTURES
Paper- 2CSA1
1.(a) Explain the following terms with respect to abstract data types :
1) value addition
2) operator definition
3) sequence
4) subsequence
An ADT is an informal specification of the logical properties of a data type such as its values and operations. It hides all the details of the embedded data structure and provides only the interface for manipulating data. Thus ADT consists of two part:

Value addition : it is a collection of values for ADT and it consists of two parts
Definition clause is mandatory and usually specifies data types of arguments
Condition clause is optional
/*value addition*/
Abstract <int, int>
Condition: rational [1] !=0;
Operator definition
Each operator is defined as abstract with three parts
Header i.e header definition of the operation that is going to perform just like function definition in C++
Pre-condition (optional)
Post condition specifies actual operation to be performed
/*operator definition*/
abstract RATIONAL makerational(a,b)
Int a,b;
Precondition b!=0;
Postcondition makerational[0]==a;
makerational[1]==b;

abstract RATIONAL add(a,b)
RATIONAL a,b;
Precondition add[1] == a[1] * b[1];
Add[0]== a[0] * b[1] + b[0] * a[1];

Finite sequence of one more character is alled string. The number of characters in a string is called length of string and the string with zero number of characters is called empty or null character string.
String can be stored in different ways
Fixed length, variable length with maximum, link structure

1.(b) Write an algorithm for the following:-
1) definition of a node from a singly linked list at the front
2) definition of a node from a singly linked list at the end
3) definition of a node from a singly linked list at any position

1) -which returns a node having pointer PTR to the free pool of storage.
Procedure:
RETURNNODE (ptr) //ptr is the pointer of the node to be returned
1. ptr1=AVAIL //start from the beginning of the free pool
2. while(ptr1.LINK≠NULL) do
    1. ptr1=ptr1.LINK
3. EndWhile
4. ptr1.LINK=PTR //Insert the node at the end
5. PTR.LINK=NULL //Node inserted is the last node
6. STOP

DELETE_SL_FONT(HEADER)
INPUT: HEADER is the pointer to the header node.
Output: A single linked list eliminating the node at the front.

Data Structure: A single linked list whose address of the starting node is known from HEADER.
Steps:

1. ptr=HEADER.LINK
2. If (ptr = NULL) then
    1. print “The list is empty : No deletion”
    2. EXIT
3. Else
    1. ptr1=ptr.LINK
    2. HEADER.LINK=ptr1
    3. RETURNNODE(ptr)
4. EndIf
5. Stop

2) deletion of node at end

ptr=HEADER

2. If (ptr.LINK = NULL) then
    1. print “The list is empty : No deletion is possible”
    2. EXIT
3. Else
    1. While(ptr.LINK≠NULL) do
    1. ptr1=ptr
    2. ptr=ptr.LINK
    2. EndWhile
    3. ptr1.LINK=NULL
    4. RETURNNODE(ptr)
4. EndIf
5. Stop.

3) Deletion of node from any position

Refer page no:28, summer 08-1(b)

1.(c) What are records? How do they differ from the array? What mechanism is used to access repeated data item from the record? Explain.

Refer page no: 65, summer 05- 1(c)

1.(d) Write an algorithm and draw a neat diagram for deletion of an element at the front and at the end of a linked list.

Refer page no :86, 1(b)

2.(a) Write an algorithm to evaluate an arithmetic expression in postfix notation using stack.

Refer page no: 65 summer 05- 1(c)

2.(b) What is deque? How does it differ from queue? Explain push operation of deque.

Refer page no: 51, summer 06- 2(d)

1.PUSHDQ(ITEM):To inset ITEM at FRONT end of dqueue

2.INJECT(ITEM):To insert ITEM at REAR end of dqueue.

Algorithm PUSHDQ(ITEM)
Input: ITEM to be inserted at the FRONT.
Output: Deque with newly inserted element ITEM if it is not full already.
Data structures: DQ being the circular array representation of deque.

Steps:

```
1. If (FRONT = 1) then                    //If FRONT is at extreme left
   1. ahead = LENGTH
2. Else                                   //If FRONT is at extreme right or deque is empty
   1. If (FRONT = LENGTH) or (FRONT = 0) then
      1. ahead = 1
   2. Else
   1. ahead = FRONT – 1                   //FRONT is at an intermediate position
   3. EndIf
3. If (ahead = REAR) then
   1. Print "Deque is full"
   2. Exit
4. Else
   1. FRONT = ahead                       //Push the ITEM
   2. DQ[FRONT] = ITEM
5. Stop
```

2)Algorithm to push an element from rear

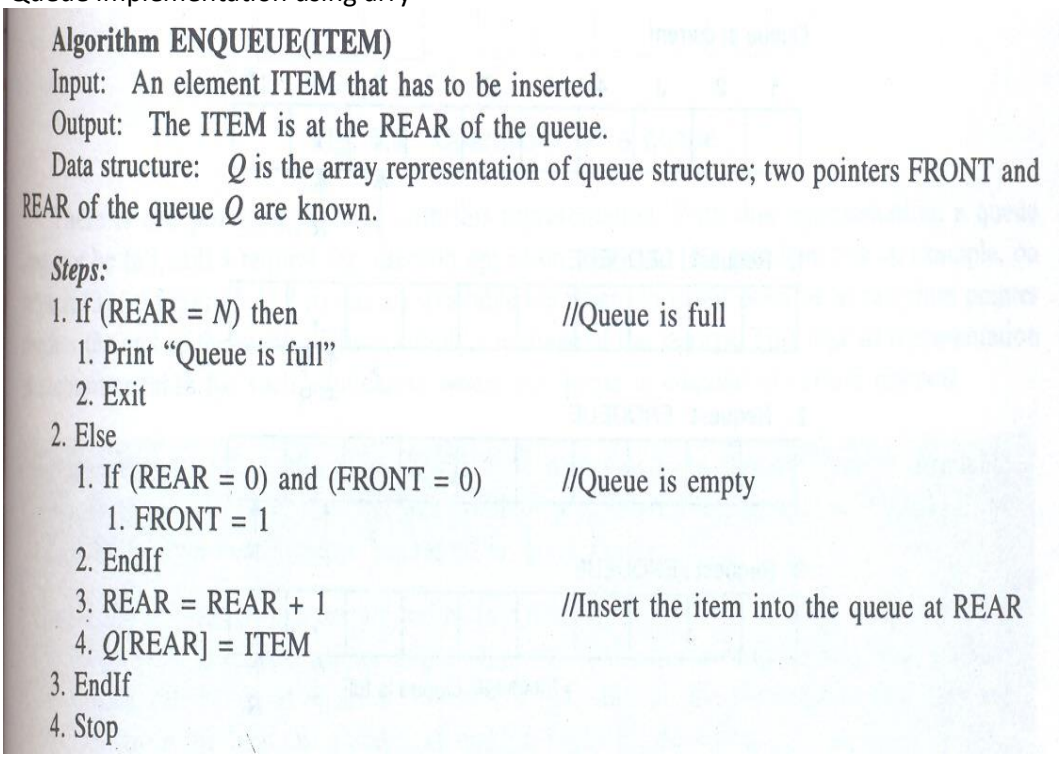
Steps:

```
1. If (FRONT = 0) then                    //When queue is empty
   1. FRONT = 1
   2. REAR = 1
   3. CQ[FRONT] = ITEM
2. Else                                   //Queue is not empty
   1. next = (REAR MOD LENGTH) + 1
   2. If (next ≠ FRONT) then              //If queue is not full
      1. REAR = next
      2. CQ[REAR] = ITEM
   3. ELSE
      1. Print "Queue is full"
   4. EndIf
3. EndIf
4. Stop
```

2.(c) How can stack be used for the implementation of recursion?
Refer page no: 12, 30[2(a)] & 52.

2(d) Discuss implementation of a queue using an array and linked list

Queue implementation using arry



Algorithm ENQUEUE(ITEM)

Input: An element ITEM that has to be inserted.

Output: The ITEM is at the REAR of the queue.

Data structure: Q is the array representation of queue structure; two pointers FRONT and REAR of the queue Q are known.

Steps:

```
1. If (REAR = N) then                          //Queue is full
   1. Print "Queue is full"
   2. Exit
2. Else
   1. If (REAR = 0) and (FRONT = 0)            //Queue is empty
      1. FRONT = 1
   2. EndIf
   3. REAR = REAR + 1                          //Insert the item into the queue at REAR
   4. Q[REAR] = ITEM
3. EndIf
4. Stop
```

Implementation of queue using linked list:

Insert_queue(item, rear)

Where item-is an information to be inserted

Rear- indicate the address of last node and each node consist of two parts

newnode=getnode()

if(newnode=null)

write "overflow"

return

endif

newnode➔info=item

newnode➔link=rear

rear=newnode

return

3.(a) explain and write a recursive algorithm for multiplication of natural numbers

Refer page no: 71, summer 2005- 3(c)

3(b) Let A be an array of integers, present recursive algorithm to compute:

i)the product of the elements of the array

ii) The minimum element of the array

iii) the average of the elements of the array

Recursive algorithm to find product of elements

```
product(a,n)
If (n=0)
 Return(1)
Endif
s=a[n]*product(a,n-1)
Return(s)
```

```
Min(I,j,min)
Where I,j – lower and upper bound of index of array
Min- smallest value in array
La is global array with N element 1<=i<=<=j<=n
If(i=j)
min=la[i]
Return
        Endif
If(i=j-1)
If(la[i]<a[j])
        Min=la[i]
Else
        Min=la[j]
Endif
Endif
Mid=int((i+j)/2))
Min(I,mid,min)
Min(mid+1,j,min1)
        If(min1>min)
                Min=min1
        Endif
Return
```

Recursive algorithm to find average of elements in array

```
Average(a,n,m)
Where a is linear array with n elements
M is integer and ititally contains 0
If(n=0)
Return(1)
Endif
Sum=a[n] +average(a,n-1,m+1)
Return(sum/m)
```

3.(c) Explain the simulation of recursive factorial function.

Refer page no: 68 summer 05- 2(d)

3.(d) Write an algorithm/program to implement merge sort in a single array using recursion technique.

Refer page no:77, 5(c)

4(a) What is threaded binary tree? Draw amd explain the mechanism of deletion of a node from threaded binary tree.

Refer page no: 56, summer 06- 4(a)

4(b) What is graph traversal? What are different methods of graph traversal? Explain in detail any one of them.

Traversal: To visit all the nodes in a graph exactly once. There are two types of traversal BFS and DFS.

BFS:- refer page no 17,summer 09- 4(a)

Page no- 41, summer 07- 4(d)

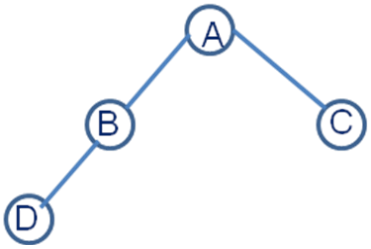
4.(c) What is balance factor? Explain procedure to perform AVL rotation of pivot when new item is inserted in the left subtree of the left child of the pivot node.

This tree was introduced by Adelson-Veiskii and Landis in 1962. This tree is useful to minimize the search time and thereby time of insertion and deletion operation ensure the integrity of balanced tree.

Balance factor is one of the important concepts in AVL tree and calculated by following formula

Balance factor = height(left tree) – height (right tree)

The balance factor shoulf remain within the limit of plus or minus 1. Otherwise left rotation or right rotation or both may be used to make a balanced tree.

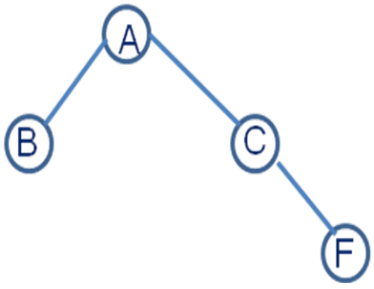


case 1: balance factor = height (left) – height (right)

= 2-1 = 1.

case 2: balance factor = height (left) – height (right)

=1-1 = 0.



case 3: balance factor = height (left) – height (right)

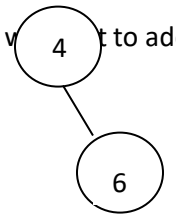
=1-2 = -1.

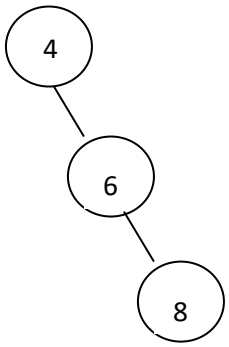
An algorithm to insert a node in AVL tree:

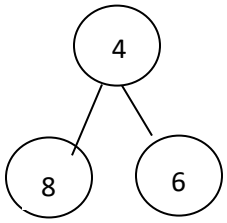
[insert node]
Insert a node by applying properties of bst i.e new node will be inserted to the left sub tree of its value is less than the root otherwise in the rufht sub tree if its value is higher than a root.
[compute factor]
After insertion of a node, the resultant tree must be AVL by checking where the balance factor is within the limit of plus or minus 1.
[Find a pivot node]
If balance factor exceeds the limit, then find the node whose absolute value of node is changed from 1 to 2 called as a rotate node or pivot node.
[balance the tree]
Balance the tree by applying the rotation either towards left or right known as left rotation and right rotation policy or both.
Return

Case 1: for example insertion of a node in the left sub-tree and cause unbalance

Suppose w t to add a new node 8, i.e balance factor exceed the limit 1 and applying left rotation.







4.(d) Define B tree. What are the various operations that can be done on B-tree? Explain any one of them in detail.

Refer page no : 73, Summer 05 – 4(b)

5.(a) Explain with neat diagram the mechanism of shell sort.

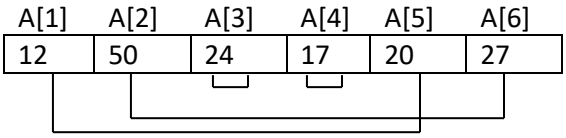
In thid method, instead of sorting the entire list, a list is divided into sub list by means of incremental value say k and this value is continuously decreased by certain step up to 1 and hence it is also called diminishing increment sort. This technique is due ti Donald L Shell.

Suppose an array A contain following elements

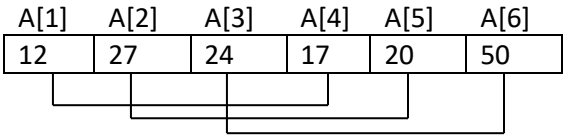
A[1]	A[2]	A[3]	A[4]	A[5]	A[6]

12	50	24	17	20	27

Pass 1, for K=4

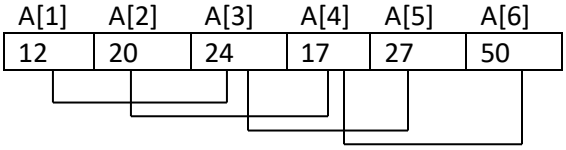


We divide the list into four sub list and after comparing and keeping the sub list elements in sorted order we obtain.



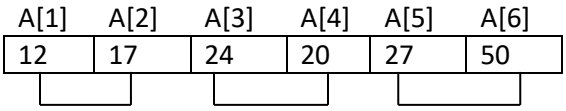
Pass 2, for K= 3

After comparing and keeping the sub list elements in sorted order we obtain



Pass 3, k=2

After comparing and keeping the sub list elements in sorted order we obtain.



Pass 4, k= 1

A[1]	A[2]	A[3]	A[4]	A[5]	A[6]
12	17	20	24	27	50

The array is sorted.

5.(b) Explain the procedure of binary search method. What are the limitations of binary search?
Refer page no: 20, summer 08 – 5(b)
Limitations of binary search: elements of array should ne sorted.
5.(c) Explain sequential search. Comment on the efficiency or sequential search.
Linear array DATA with N elements and a specific ITEM of information are given. This algorithm finds the location LOC of ITEM in the array of DATA or sets LOC=0.
[Initialize] Set k:=1 and Loc:=0
Repeat steps 3 and 4 while LOC = 0 and K<=N.
If ITEM = DATA[K], then set LOC := K.
Set K:=K+1 [increments counter]

[successful?]
If LOC = 0 then
    Write: ITEM is not in the array DATA.
Else
    Write: LOC is the location of ITEM.
[End of If structure]
Exit
Efficiency :
Avg. Case :

$$f(n)=1*1/n+2*1/n+\ldots\ldots+n*1/n$$
$$=(1+2+3+\ldots.+n)1/n$$
$$=\frac{n(n+1)}{2}-\frac{1}{n}$$
$$=\frac{n+1}{2}$$

worst Case :

$$f(n)=n$$

5.(d) Write an algorithm for the implementation of merge sort.
Refer page no: 77, summer 05 – 5(c)

KSD/W4/2262
SECOND SEMESTER MASTER IN COMPUTER APPLICATION
DATA STRUCTURES
Paper- 2CSA1

1.(a) Explain abstract data type specification and the sequences of value definition.
Refer page no : 85, winter 05 – 1(a)

1.(b) Write an algorithm to insert new element after the given location LOC in singly linked list.
Refer page no :86, winter 05 – 1(b)

1.(c) Write an algorithm to print the linked list in reverse order.
Refer page no: 64, Summer 05 – 1(b)

1.(d) Write the ADT specification for varying length data string.
Refer page no : 80, winter 06 – 1(b)

2.(a) Discuss the implementation of stack as an array and linked list.
Refer page no : 5 & 30, summer 08 & 07 – 2(a)

2.(b) What is priority queue? Explain the array representation of priority queue in memory.
Refer page no: 51, summer 06 – 2(d)

2.(c) Write an algorithm to translate the infix expression into postfix form.

Refer page no: 7 summer 08- 2(c)

2.(d) Write an algorithm to copy one queue to another when the queue is implemented as linked list.
For a given list we can copy it into another list by duplicating the content of each node into newly allocated node.

Algorithm:COPY_SL(HEADER,HEADER1)
Input: HEADER is the pointer to the header node of the list
Output: HEADER1 is the pointer to the duplicate list.
Data Structure: Single Linked List.
Steps:

1. ptr=HEADER
2. HEADER1=GETNODE(NODE)
3. ptr1=HEADER1
4. ptr1.DATA=NULL
5. While(ptr ≠ NULL)do
    1. new=GETNODE(NODE)
    2. new.DATA=ptr.DATA
    3. ptr1.LINK=new
    4. new.LINK=NULL
    5. ptr1=new
    6. ptr=ptr.LINK
6. EndWhile
7. STOP

3.(a) Explain the simulation of recursive factorial function.
Refer page no: 68 summer 05- 2(d)

3.(b) Write a non-recursive algorithm for the tower of Hanoi problem using stack.
Tower(n,beg,aux,end)
This is non recursive solution to the tower of Hanoi problem for n disks which is obtained by translating the recursive solution. Stacks stn, stbeg, staux, stend and stadd will correspond, respectively, to the variables n, beg, aux, end and add.

Set top=null
If n=1 then
Write beg➔ebd
Go to step 5
[end of if structure]
[translation of "call tower(n-1,beg,end,aux"]
[push current values and new return address onto stackes]
Set top = top+1
Set stn[top] = n, stbeg[top]=beg,
Staux[top]=aux, stend[top]=end,
Stadd[top]=3.
[reset parameters]
Get n= n+1, beg= beg, aux= end, end =aux
Go to step 1
Write beg ➔end

[translation of “call tower(n-1,aux,beg,end”)]
[push current values and new return address onto stacks]
Set top=top+1
Set stn[top]=n, stbeg[top]=beg,
Staux[top]=aux, stend[top]=end,
Stadd[top]=5
[reset parameters]
Set n= n-1, beg=aux, aux=beg, end=end.
Go to step 1
[translation of return]
If top=null then return
[restore top values on stacks]
Set n= stn[top], beg = stbeg[top],
Aux=staux[top], stend[top],
Add = stadd[top].
Set top=top-1
Go to step add.

3.(c) Explain and write a recursive algorithm for the multiplication of natural numbers.

Refer page no : 71, summer 05- 3(c)

3(d) Let n denote a positive integer. Suppose a function L is defined recursively as follows

$$L(n) = \begin{cases} 0 \text{ if } n=1 \\ L(|n/2|)+1 \text{ if } n>1 \end{cases}$$

(Note: |k| denote the greatest integer which does not exceed k)
Find L(25)
What does this function do?

L(25) = L(|25/2|) + 1
= L(12)+1
= (L(|12/2|) +1) +1 as 12>1
= L(6) +2
= (L(|6/2|)+1) + 2 as 6>1
= L(3)+3
= (L(|3/2|)+1) +3 as 3>1
= (L(1)+4
= 0+4 as n=1
= 4.

4.(a) What is balance factor? Explain procedure to perform an AVL notation of pivot when new item is inserted in the left subtree of the left of the pivot node.

Refer Page no : 92 winter 05- 4(c)

4.(b) write an algorithm to find the minimum spanning tree of the weighted graph G.

Refer Page no:37, summer 07- 4(b)

4.(c) Write an algorithm for deleting anode from lexically ordered tree.

Refer Page no: 73, summer 05- 4(b)

4.(d) write an algorithm to calculate the shortest distance from start node using BFS strategy. Assume that the two queue handling procedure qinsert and qdelete are already available

Refer Page no: 17, summer 05-4(d)

5.(a) explain the logic of merge sort method with suitable example

Refer Page no: 77, summer 05- 5(c)

5.(b) Explain the procedure for binary search method. What are the limitations of binary search?

Refer Page no: 95, winter 05- 5(b)

5.(c) write an insertion sort algorithm to sort an array A of N elements

Refer Page no: 23 summer 08-5 (d)

5.(d) what is heap? Write an algorithm to insert an element into heap.

Refer page no: 75 summer 05- 5(a)