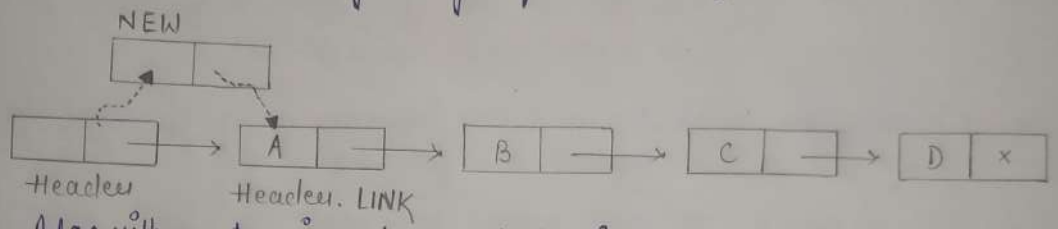
1. Define link list and explain algorithm for insertion at beginning of the linked list.

Linked list :- Linked list is a linear data structure in which the list is arranged logically. It sequially consists of node in which data and a pointer is stored.

Insertion at beginning of the linked list :-



Algorithm to insert a node in the front of the linked list.

Algorithm : INS_NODE_FRONT (LIST, x)

Step 1 :- NEW = Getnode (Node)

Step 2 :- If (NEW = NULL) then
Print " Memory is insufficient : new node cannot be inserted "
Exit.

Step 3 :- Else
i. NEW → DATA = X
ii. NEW → LINK = Header → LINK
iii. Header → LINK = NEW.

Step 4 :- End if

Step 5 :- STOP.

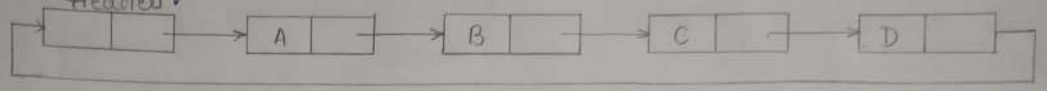
Explanation :-

- In step 1 we call the procedure Getnode and search for the availability of free node of the required size and make NEW point at it.

- In step 2 if NEW points at NULL then a free node of required size is not available and insertion of node is not possible and hence we exit at this point.
- In step 3 if a node of given size is available
  i) Assign the data x to the new node. We then adjust the pointer.
  ii) LINK of header should point at NEW as NEW now becomes the first node of the header list.
  i.e. Header → LINK = NEW
  iii) LINK of NEW should point at header → LINK i.e. at the first node of the linked list.
  i.e. NEW → LINK = Header → LINK
- step 4 marks the end of the if statement
- step 5 ends the algorithm.

2. Explain Concept and algorithm for traversing a circular linked list.

A circular linked list is a single linked list in which the last node is pointing at the first node instead of NULL.



Traversing a circular linked list :-

Algorithm :- Traverse_CSLL (Header)

Step 1 :- PTR = Header → LINK

Step 2 :- While (PTR → LINK != Header) do
process (PTR)
PTR = PTR → LINK
End While

Step 3 :- STOP.

Explanation :-

- In step 2 of the above algorithm we search the linked list till the last node does not point at the header node. The moment the last node point at the Header node we will stop as the searching in all possible nodes is over. This is checked with the help of While loop. We visit each node and process it.

Note :- In a single linked list ~~the~~ searching is done till PTR → LINK does not point at NULL and in Circular linked list searching is done till the last node does not point at the first node that is the Header Node.

3. Write short note :-

i. Overflow :- Suppose we want to insert an element in data structure and there is no node of required size available in the memory bank. This condition is known as overflow condition. It is always necessary to check this condition before inserting a node. If a free node of required size is not available then insertion is not possible in the data structure. Overflow cannot occurs as long as there is at least one chunk of allocable memory large enough to hold one element.

ii. Underflow :- Consider a data structure with no elements i.e the data structure is empty and we are trying to delete an element from it. Such a condition is known as underflow condition and should be checked before performing the deletion Operation.

iii. Garbage Collection :- The maintenance of data structure in memory ensures the possibility of inserting new nodes into the data structure and hence requires some mechanism which provides unused memory space for the new nodes. Some mechanism is required where the memory space the deleted node becomes available for future use. Collection of nodes that are no longer in use putting them into a pool of free nodes for reuse is known as Garbage Collection. Garbage Collection can be done in two ways.

i) After a node is deleted, it is directly put into the memory bank.

ii) Periodically all the deleted spaces can be collected and can be put in the free memory pool. This can even be done when CPU is free

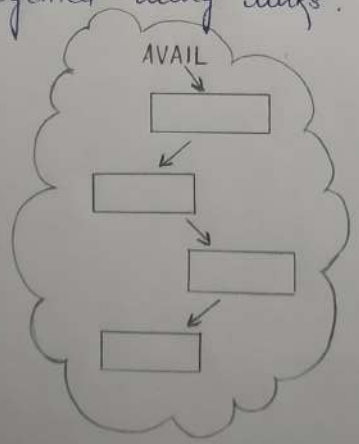
Garbage Collection is done in two steps.

In the first pass the nodes that are no longer required are identified and marked.

In the second pass all these nodes are collected and put in the pool of free nodes for further use.

Garbage collection is often done automatically and is not visible to the programmer.

Memory Bank :- Memory bank is nothing but a pool of free memory bank which consists of free nodes. These memory chuncks can be of different size. When a linked list is to be created or a new node is to be inserted in the linked list a memory chunck of the necessary size will be picked up from this pool. If it is available, it can be inserted in the linked list. Memory bank can be imagined as a linked list of free nodes in which the pointer AVAIL point at the first node. The remaining free nodes are linked together using links.



5. Explain algorithm for deletion of the linked list.

Deletion in a linked list is possible by merely adjusting a pointer. The deleted node is returned to the memory bank.

The deleted node can be returned to the memory bank by using the following algorithm. It is convenient to insert the node in front of the AVAIL LIST in the memory bank.

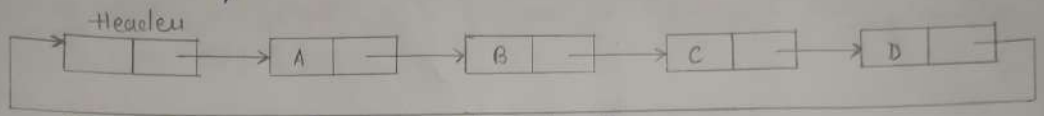
Algorithm :- Return_node (PTR)

This algorithm will return the deleted node to the memory bank and it will be added in the front of the AVAIL LIST.

PTR1 = AVAIL → LINK
AVAIL = PTR
PTR → LINK = PTR1
STOP.

6. Explain Circular linked list in detail.

A Circular linked list is a single linked list in which the last node is pointing at the first node instead of NULL.



Traversing a Circular linked list.

This algorithm will traverse the Circular linked list. It is assumed that the Circular linked list is already present in the memory. Header is pointing at the first node. PTR is pointing at the current node. Process is applied to the data of the current node.

Step 1 :- PTR = Header → LINK
Step 2 :- While (PTR → LINK != Header) do
    process (PTR)
    PTR = PTR → LINK
  End While
Step 3 :- STOP.

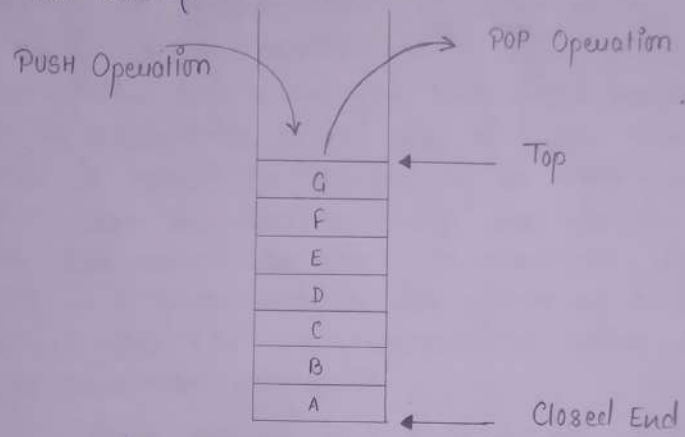
Inserting a node in a Circular linked list.

If a free node of required size is available in the memory bank then only insertion is possible otherwise a node cannot be inserted. A node can be inserted in 3 ways.

i) Insertion at the beginning of the Circular linked list.
ii) Insertion at the end of the Circular linked list.
iii) Insertion after a given key value in the Circular linked list.

4 Define stack and explain implementation of stack.

Stack :- A stack is a list of elements in which elements may be inserted or deleted only at one end called the Top of the stack. Insertion and deletion in stack are known as PUSH and POP operations in stack.



The diagram of a stack.

Representation of stack :-

A stack can be showed in an array or a linked list so there are two representations of a stack.

i) Linear array representation

ii) Linked list representation.

• Linear array representation of stack :-

A stack can be implemented by a linear array. Whenever a stack of fixed size is required then linear array implementation can be used. Since a stack is last in first out data structure, insertion and deletion takes place from one end called TOP. The pointer TOP points to the topmost element of stack and MAXSTACK denotes the size of the stack.

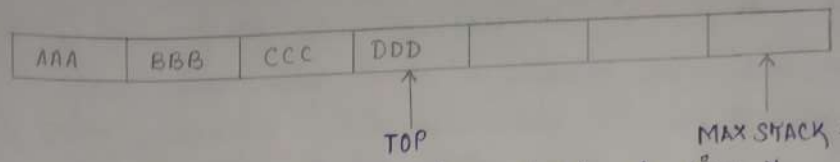
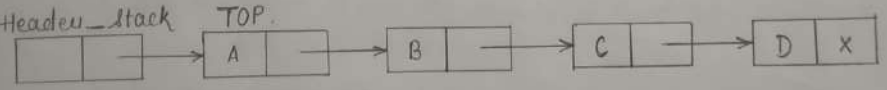


Diagram shows the stack of size 7.

Linked List Representation of Stack :-

The major problem with array representation of stack is, it will work for only fixed number of data i.e. there is restriction on the size of Queue. Once the size is decided it cannot be changed. If an additional element greater than the maximum size has to be added to the stack this cannot be done. To overcome this problem a stack is represented in the form of a linked list. An additional node can be represented added to a linked list whenever required.

Stack_Header is a node which contains the information about the stack. Pointer TOP points to the topmost element of the stack.



Explain Quick Sort algorithm in detail.

Quick Sort Algorithm is divided into two parts

Quick procedure :- This procedure actually performs the Quick Sort Operation on the array.

Quicksort algorithm :- This algorithm considers the array on which quick sort is to be applied. It partitions the array into sublist and calls the procedure QUICK to sort the Sublist.

Procedure : QUICK (A, N, BEG, END, LOC)

This procedure performs the Quick Sort operations on the array A with size N. Parameters BEG and END contain the boundary value of the sub list of A to which this procedure applies. LOC keeps tracks of the position of the first elements A[BEG] of the Sub list during the procedure. The variable LEFT and RIGHT contain the boundary values of the list of elements that have not been scanned.

Step 1 :- Set LEFT = BEG, RIGHT = END and LOC = BEG

Step 2 :- [Scan from right to left]

a) Repeat while A[LOC] <= A[RIGHT] and LOC ≠ RIGHT

RIGHT = RIGHT - 1

b) if LOC = RIGHT then Return

c) if A[LOC] > A[RIGHT] then

i) [Interchange A[LOC] and A[RIGHT]

TEMP = A[LOC], A[LOC] = A[RIGHT]

A[RIGHT] = TEMP

ii) Set LOC = RIGHT

iii) Go to Step 3.

3. [Scan from left to right]

a] Repeat while A[LEFT] <= A[LOC] and LEFT ≠ LOC

LEFT = LEFT + 1

b] IF LOC = LEFT then Return

c] if A[LEFT] > A[LOC] then

i) [Interchange A[LEFT] and A[LOC]

TEMP = A[LOC], A[LOC] = A[LEFT]

A[LEFT] = TEMP

ii) Set LOC = LEFT

iii) Goto Step 2.

Algorithm : QUICKSORT.

This algorithm sort an array A with N elements.

Step 1 :- TOP = NULL

Step 2 :- if N>1 then TOP = TOP + 1, LOWER[1] = 1, UPPER[1] = N

Step 3 :- Repeat Steps 4 to 7 while TOP ≠ NULL

Step 4 :- [Pop sub list from stack]

Set BEG = LOWER[TOP], END = UPPER[TOP], TOP = TOP - 1

Step 5 :- Call QUICK (A, N, BEG, END, LOC)

Step 6 :- [PUSH left sub list onto stacks when it has 2 or more elements]

if BEG < LOC - 1 then

TOP = TOP + 1, LOWER[TOP] = BEG

UPPER[TOP] = LOC-1

Step 7 :- [Push right sub list onto stacks when it has 2 or more elements]

TOP = TOP + 1, LOWER[TOP] = LOC + 1,

UPPER[TOP] = END

[END of Step 3 Loop]

Step 8 :- Exit

1. Explain the concept of Tower of Hanoi.

Tower of Hanoi is an interesting problem that can be considered as an application of stack.

A recursive solution can be given to this problem. The problem can be stated as follows.

Suppose there are three pegs labeled A, B and C and suppose on peg A there are n disks arranged in decreasing order of its size.

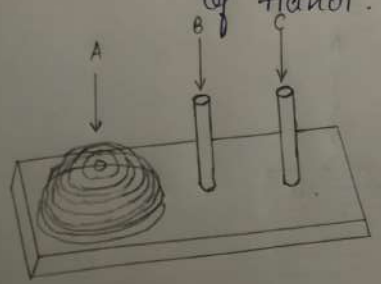
- The problem of moving the disks from peg A to Peg C using an intermediate or auxiliary peg B subject to the constraint that only one disk can be moved at a time and a larger disk cannot be placed on the smaller disk is known as Tower of Hanoi problem.

The rules that should be followed while solving the problem are :-

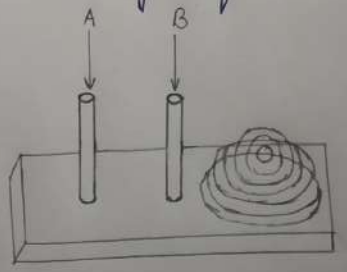
i) Only one top most disk can be moved at a time.

ii) While placing the disk on peg, smaller disk can only be placed on the larger disk.

Below shows the initial and final stage of Tower of Hanoi.



Initial Stage.



Final Stage.

1. Explain implementation of queue using linked list.

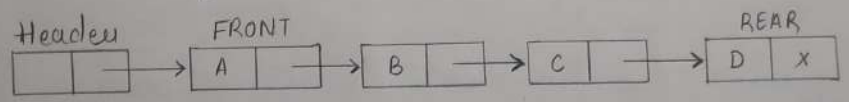
The major problem with array implementation of Queue is it will work for only fixed number of data i.e there is a restriction on the size of queue. Once the size is decided it cannot be changed. If an additional element greater than the maximum size has to be added to the queue this cannot be done.

Queue using array cannot be used when the size of data is not known. To overcome this problem a queue is implemented in the form of a linked list. Two pointers FRONT and REAR are used to point to the front and rear end of the Queue respectively.

1. FRONT :- Front pointer points to the first element (node) of the Queue.

2. REAR :- Rear pointer points to the last element of the Queue.

Diagrammatic implementation of a Queue using Single linked list.



11. Explain Priority Queue in detail.

Priority Queue :- Sometimes it is necessary to assign priorities to the elements in a Queue, such type of queue is known as Priority Queue. The priority is assigned in form of a number. In such a case element having highest priority is processed first.

To maintain Priority Queue in memory, three single dimensional arrays are required.

i) DATA :- Contains the data to be stored

ii) PRIORITY :- Contains the priority for the corresponding data.

iii) LINK :- Contains the link to the next element in array

START = 5

	DATA	PRIORITY	LINK
1	BB	2	6
2			7
3	DD	4	4
4	EE	4	9
5	AA	1	1
6	CC	2	3
7			10
8	GG	5	0
9	FF	4	8
10			11
11			12
12			0

Priority Queue Maintained in Memory.

Operations on Priority Queue.

i) Insertion

ii) Deletion.

2. Explain Merge Sort with Suitable Example.

- Merge Sort is Sorting method which initially divides and list into sublists of size 1. Pairs of these sublists are formed.
- It must be noted that the last sublist may remain without a pair.
- These sublists of size 1 are trivially sorted. These pairs of sublists are merged to get a sorted sublists of size 2.
- These pairs are again merged to get a sorted sublists of size 4. Possibly the last may remain without a pair or may have less number of elements. This procedure is repeated till all the elements in the list are sorted.

Example :- Consider the following list with seven elements
50, 30, 20, 10, 25, 35, 60.

First sublists of size 1 are formed for the given list. Each sublists ~~is know~~ trivially sorted as it contains single elements.

50 , 30 . 20, 10 25, 35 60

These pairs are merged to get the Sorted Sublists.

30, 50 10, 20 25, 35 60

The above sublists are merged to get two sublists of size 4.

30, 50 10, 20 20, 25 25, 35 60

10, 20, 30, 50 . 25, 35, 60

Then two sub lists are merged to get the final sorted list.

10, 20, 25, 30, 35, 50, 60

From the above example it is observed that

- After K passes the array list is partitioned into sorted sub arrays where each sub array except possibly the last will contain $L = 2^K$
- The Merge sort algorithm requires at the most log n passes to sort an array list with N elements.
- Each sub array consists of L elements excepts the last sub array may have fewer than L elements
- If we divide the total number N of the elements in an array by 2*L then the quotient obtained gives us the information about the pairs of L elements sorted sub arrays.

$$Q = INT(N/(2*L))$$

- The total number of elements S of sub arrays are

$$S = 2*L*Q$$

- The number of remaining elements are

$$R = L - S$$

13. Explain representation of binary tree in memory.

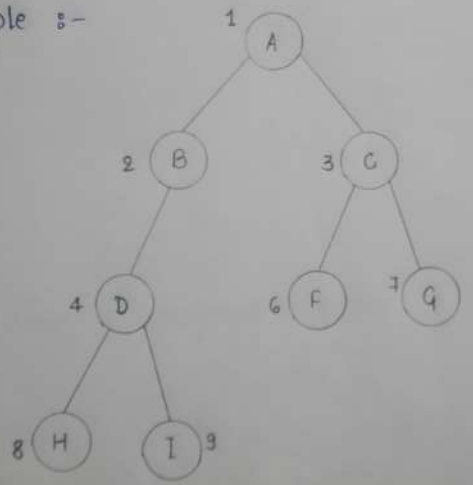
A binary can be represented in two ways.

1. Sequential or linear representation
2. Linked representation.

- Sequential or linear representation of binary tree :-

This representation requires a single linear array to store a binary tree. if a binary tree is complete then this is the most efficient way of storing a binary tree in the memory. In this representation if the parent is stored at the Kth location then its left child will be stored at $(2k)^{th}$ location and the right child is stored at $(2k+1)^{th}$ location.

For example :-

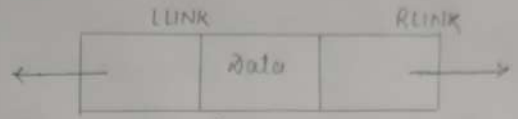


This binary tree can be stored in the memory using sequential representation as follows.

1	2	3	4	5	6	7	8	9	10	11	12	13	14
A	B	C	D		F	G	H	I					

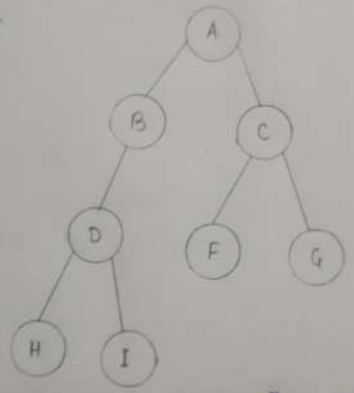
Linked representation of binary tree :-

Linked representation of a binary tree consist of nodes and link. The structure of the node are as follows :-

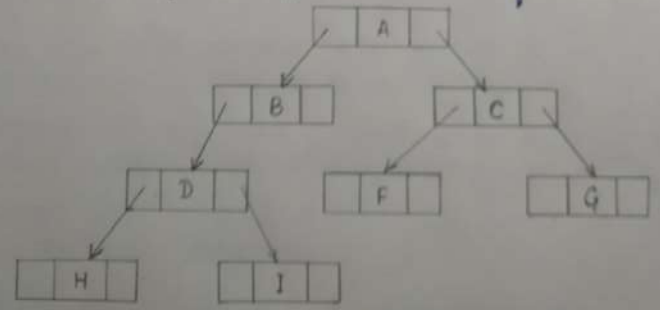


A node is divided into three parts. Data is stored in one of the parts and the remaining two parts will store two links. The node in a binary tree has two pointers LLINK and RLINK. LLINK will taker to left node in the binary tree and RLINK will taker to right node in the binary tree.

For Example :-



The linked representation of above binary tree.

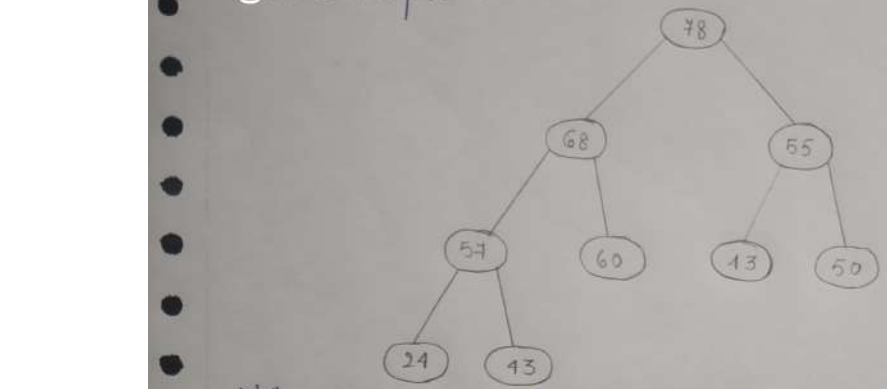


if the location of the root is known to us then the binary tree can be easily traversed.

Explain Sequential representation of heap.

Heap can be represented with the help of one dimensional array. In this representation if the parent is stored at the Kth location then its left child will be stored at $(2k)^{th}$ location and the right child is stored at $(2k+1)^{th}$ location.

An example :-



This heap can be stored in the memory using Sequential representation. as follows :

1	2	3	4	5	6	7	8	9	10	11	12	13	14
78	68	55	57	60	13	50	24	43					

Formation of Heap :— Step to form a Heap.

i) Make the first element the root of the Heap.

ii) Take the next element and insert it in the tree to form a complete binary tree.

iii) Compare it with the parent node and if the data in the parent node is smaller interchange it with the child.

iv) Repeat the process 3 till the node is placed in its proper location.

v) Repeat step 2, 3 and 4 till the entire heap is formed.