Important Instructions to examiners:

1) The answers should be examined by key words and not as word-to-word as given in the model answer scheme.
2) The model answer and the answer written by candidate may vary but the examiner may try to assess the understanding level of the candidate.
3) The language errors such as grammatical, spelling errors should not be given more

Importance (Not applicable for subject English and Communication Skills).

4) While assessing figures, examiner may give credit for principal components indicated in the figure. The figures drawn by candidate and model answer may vary. The examiner may give credit for any equivalent figure drawn.
5) Credits may be given step wise for numerical problems. In some cases, the assumed constant values may vary and there may be some difference in the candidate's answers and model answer.
6) In case of some questions credit may be given by judgement on part of examiner of relevant answer based on candidate's understanding.
7) For programming language papers, credit may be given to any other program based on equivalent concept.

Q .1]a) i) (Note: -Definition 01 marks and any 4 correct operations 01 marks)

i. Define the data structure and give 4 operation of it.

Ans:-**Data structure:-** A data structure is the branch of computer science. A mathematical or logical model of a particular organization of a data is called a data structure.

Data Structure operations:-

1)Traversing:-Accessing each record exactly once so that certain items in the record may be processed.

2)Searching:-Finding the location of the record with the given key value or finding the locations of all records which satisfy one or more conditions.

3)Inserting: Adding new record to the structure.

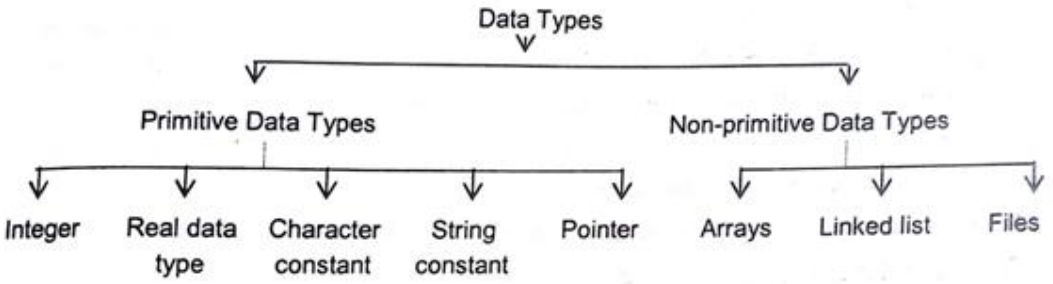
4)Deleting: Removing a record from the structure.

5)Sorting:-Arranging the records in some logical order .

6)merging:-Combining the records in two different sorted files into a single sorted file.

Copying , concatenations are also data structure operations

Q.1a] ii) Give Classification of data structure.(Note: -Accurate classification with diagram 02 marks)



1. **Primitive Data Structure :** These are the basic structures and directly operated upon by the machine instructions. Integer, floating-point numbers, characters constant, string constants, pointers etc. come in this category.

   Every computer has a set of native data types i.e. it is constructed with a mechanism of manipulating bit pattern at a given location as binary numbers. Primitive data types are basic data types of any language that form the basic unit of the data structure defined by the user. A primitive data type defines how the data will be internally represented in, stored and retrieved from the memory. Ex. Integer, character, real / float numbers etc.

2. **Non-Primitive Data Structure :** These are more sophisticated data structures. These are derived from the primitive data structures. The non-primitive data structures emphasize on structuring of a group of homogeneous (same type) or heterogeneous (different types) data items. Arrays, linked list and files are examples.

Q.1a] iii) Describe big'O' notation used in the algorithm.(Note:-Accurate definition of big'O'notation contains 02 marks.)

Big'O' notation:- If f(n) represents the computing time of some algorithm and g(n) represents a known standard function like n,n2,nlogn,etc then to write

F(n) is 0 g(n)

Means that f(n) of n is equal to biggest order of function g(n).

Q.1a] iv) Define stack and give basic operations on it. (Note:-Accurate definition of stack 01 mark and operations 01 mark)

A stack is a list of elements in which an elements may be inserted or deleted only at one end , called the top of the stack.That elements are removed from a stack in the reverse order of that in which they were inserted into the stack.

Basic operations of stack:

a)PUSH:- Insert an element into a stack.

b)POP:-Delete an element from stack.

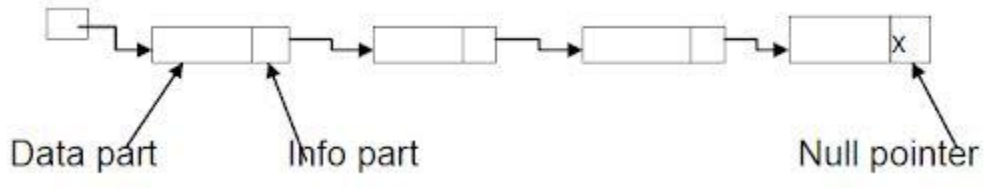
Q.1 a] v) Describe priority queues. (Note: With accurate meaning and any correct 02 points of description 02 marks)

Ans: A **priority queue** is an abstract data type which is like a regular queue of data structure, but where additionally each element has a "priority" associated with it.

In a priority queue, an element with high priority is served before an element with low priority.

If two elements have the same priority, they are served according to their order in the queue.

Q.1 a] vi) Define NULL pointer and Empty List (Note : NULL pointer 01 mark and Empty List 01 mark)



Null Pointer:-

A pointer to the head of the list is used to gain access to the list itself and the end of the list is denoted by NULL POINTER.

NULL POINTER is a pointer containing 0 address.

P=NULL;

The above statement will set the pointer to NULL or 0 address.

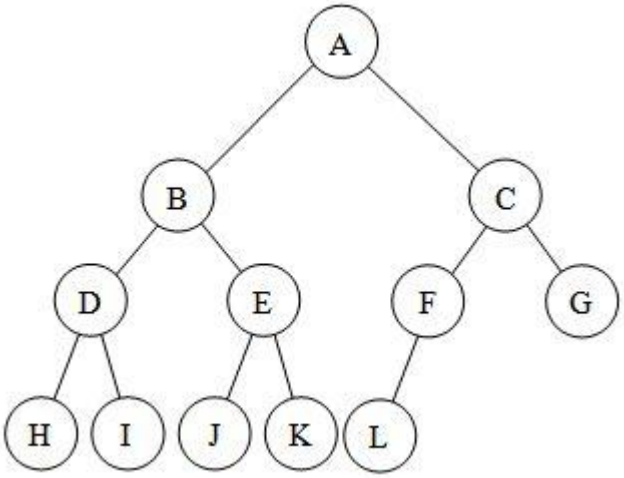
Empty list:

An empty list has no elements in it.

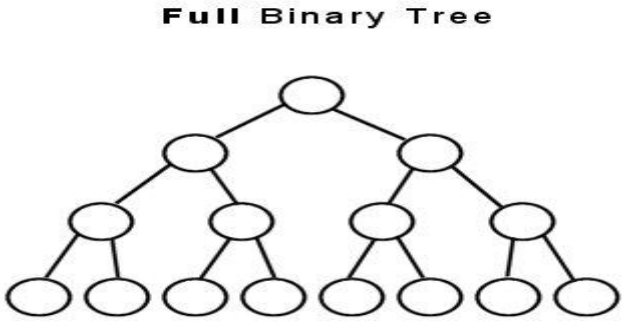
L=(),

Q.1a] vii)Define the term complete binary tree and fully binary tree. (Note:- complete binary tree 01 mark and fully binary tree 01 mark.)

- complete binary tree is a binary tree in which every level, except possibly the last, is completely filled, and all nodes are as far left as possible.



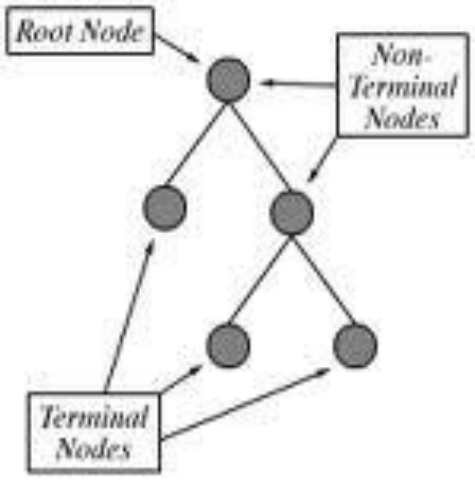
A full binary tree (sometimes proper binary tree or 2-tree) is a tree in which every node other than the leaves has two children.



Q.1a] viii)Define the term tree and forest. (NOTE:- tree o1 mark and and forest 01 mark)

A tree is a non-empty collection of vertices & edges that satisfies certain requirements. A vertex is a simple object (node) that can have a name and carry other associated information. An edge is a connection between two vertices.

A Tree is a finite set of a zero or more vertices such that there is one specially designated vertex called **Root** and the remaining vertices are partitioned into a collection of sub-trees, each of which is also a tree. A node may not have children, such node is known as **Leaf (terminal node)**. The line from parent to a child is called a branch or an edge. Children to same parent are called siblings



Forest:

A forest is defined as a set of trees .

It can be represented by a binary tree.

Q.1B] i) What is data structure? Why do we need it. (Note: Definition -01 mark and Need 03 mark)

A data structure is the branch of computer science. A mathematical or logical model of a particular organization of a data is called a data structure.

Need of a Data Structure

A data structure helps you to understand the relationship of one data element with the other and organize it within the memory. Sometimes the organization might be simple and can be very clearly visioned, for example, list of names of months in a year. We can see that names of months have a linear relationship between them and thus can be stored in a sequential location or in a type of data structure in which each month points to the next month of the year and it is itself pointed by its preceding month. This principle is overruled in case of first and last month's names. Similarly, think of a set of data that represents location of historical places in a country (Fig. 1.1). For each historical place the location is given by country name fo e followed by city name, and th istorical place name. We can see a hierarchical relationship between them and must be represented in the memory using a hierarchical type of data structure.

data.png
Type: PNG Image
Size: 104 KB
Dimension: 800 x 1132 pixels

The above two examples clearly identify the usefulness of a data structure. A data structure helps you to analyze the data, store it and organize in a logical or mathematical manner.

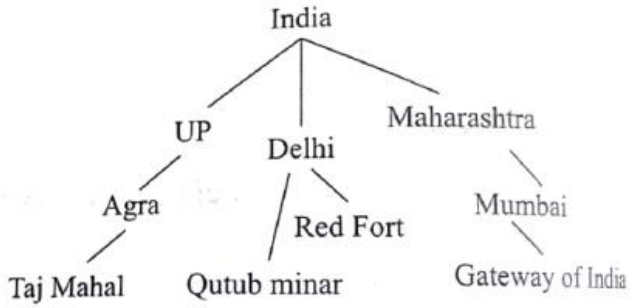


Fig. 1.1

Q.1b] ii)Explain time and space complexity related to algorithms. (Note:-Time complexity with example-02 marks)(Space complexity with example 02 marks)

Time Complexity:

The time complexity of a program or algorithm is the amount of computer time that it needs to run to completion.

Algorithm A: a=a+1; Frequency count of algorithm A is 1

Algorithm B: For x=1 to n step 1

a=a+1

loop

Frequency count of algorithm B is n

Space complexity:

The space complexity of an algorithm or program is the amount of memory that it needs to run to completion .

Eg. Algorithm

```
void main()
{
int I ,n,sum ,x;
sum=0;
printf("\n enter no of data to be added");
scanf("%d",&n);
for(i=1;i<=n;i++)
{
scanf( "%d",&x);
sum=s  um+x;
}
printf("\n  sum =%d", sum);
}
```

Space required to store the variables I,n,sum and x=2+2+2+2=8 (int requires 2 bytes of memory space)

Q.1b] iii) Write a procedure to push on element of stack. Also give meaning of stack overflow.

(Note:- procedure to push on element of stack 02 marks ; meaning of stack overflow-02 marks)

Ans: **Procedure for the PUSH:**

PUSH (Stack_ pointer, Maximum, Data)

PROCEDURE PUSH (Stack_ pointer, Maximum, Data)

(*Check if the stack is already full*)

IF Stack_ pointer=Maximum THEN

PRINT No room on the stack

EXIT PROCEDURE

ELSE

(*Push data item on the stack*)

SET Stack_ pointer= Stack_pointer+1

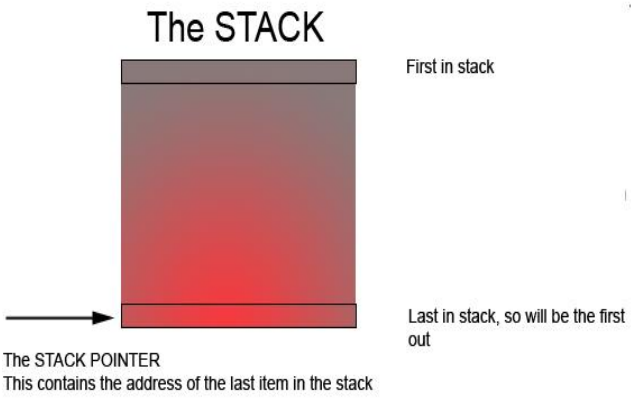
SET Stack_ pointer= Data

END IF

END PROCEDURE

Stack Overflow:-

A stack cannot grow indefinitely, because there is always a limit to memory.



When it comes to its limit and yet another operation tries to PUSH data onto the stack then a '**stack overflow**' occurs

Q.2] a) Convert the following Infix into postfix expression using stack and also show the details of stack content after each step.

Expression: ((A+B)*D) T (E-F)

Note:

In this expression student can take T as operand and can solve in that way.

Also student can take T as any operator such as (+,-,/,*,^)also

All the ways of solution are correct and contains full marks

Consideration of steps is essential with details of stack.

(Each 02 correct steps with proper sequence contains 01 marks.)

WAY 1:

Assume T as an operand.Solution :

Sr. no	Stack	Input	Output(postfix)
1	Empty	((A+B)*D)T(E-F)	-
2	(	(A+B)*D)T(E-F)	-
3	((	A+B)*D)T(E-F)	-
4	((	+B)*D)T(E-F)	A
5	((+	B)*D)T(E-F)	A
6	((+	)*D)T(E-F)	AB
7	(	*D)T(E-F)	AB+
8	(*	D)T(E-F)	AB+
9	(*	)T(E-F)	AB+D
10	Empty	T(E-F)	AB+D*
11	Empty	(E-F)	AB+D*T
12	(	E-F)	AB+D*T
13	(	-F)	AB+D*TE
14	(-	F)	AB+D*TE
15	(-	)	AB+D*TEF
16	Empty	END	AB+D*TEF-
17			
18			
19			

WAY 2:

Assume T as any operator such as (+,-,/,*,^)

Here assumed T as ^

Sr. no	Stack	Input	Output(postfix)
1	Empty	((A+B)*D)^(E-F)	-
2	(	(A+B)*D)^(E-F)	-
3	((	A+B)*D)^(E-F)	-
4	((	+B)*D)^(E-F)	A
5	((+	B)*D)^(E-F)	A
6	((+	)*D)^(E-F)	AB
7	(	*D)^T(E-F)	AB+
8	(*	D)^(E-F)	AB+
9	(*	)^(E-F)	AB+D
10	Empty	^(E-F)	AB+D*
11	^	(E-F)	AB+D*
12	^(	E-F)	AB+D*E
13	^(	-F)	AB+D*E
14	^(-	F)	AB+D*E
15	^(-	)	AB+D*EF
16	^	END	AB+D*EF-
17	Empty	END	AB+D*EF-^
18			
19			

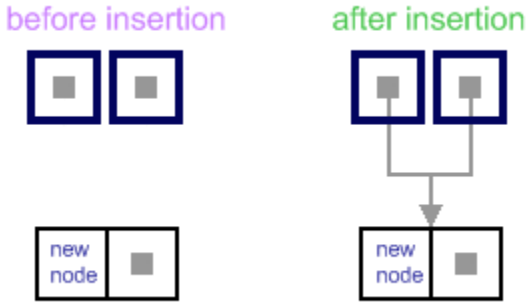
(At any ways of solutions upto 10th steps solutions are same hence contain '04' marks

Remaining steps contains 04 marks (according to students assumption for' T')

Examiner Please consider the steps of solution.)

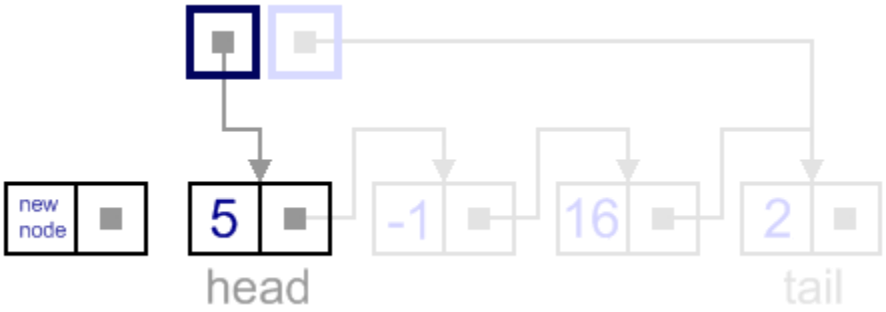
Q.2] b) Write an algorithm to insert a new node at the beginning at the middle position and at the end of the single linked list. (Note: Each algorithm contains 02 marks; Accuracy with diagrams will add 02 marks)

When list is empty, which is indicated by (head == NULL)condition. Algorithm sets both head and tail to point to the new node.



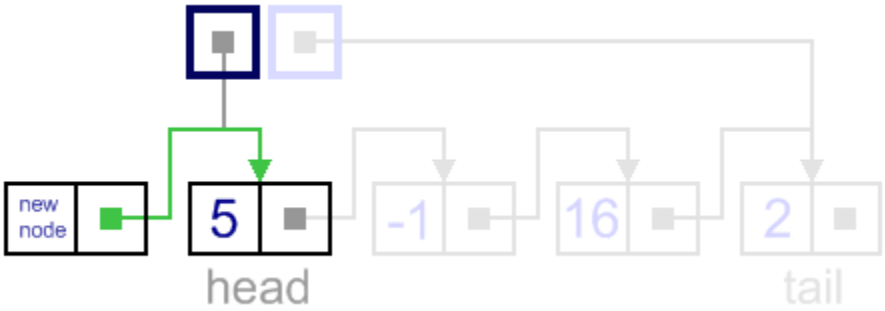
Add first

In this case, new node is inserted right before the current head node.

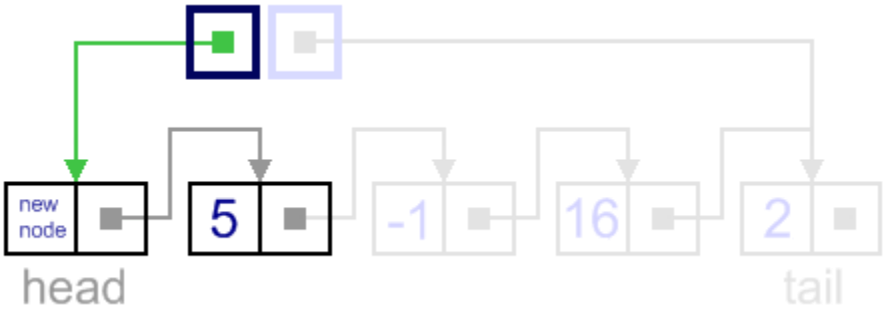


It can be done in two steps:

1. Update the next link of a new node, to point to the current head node.

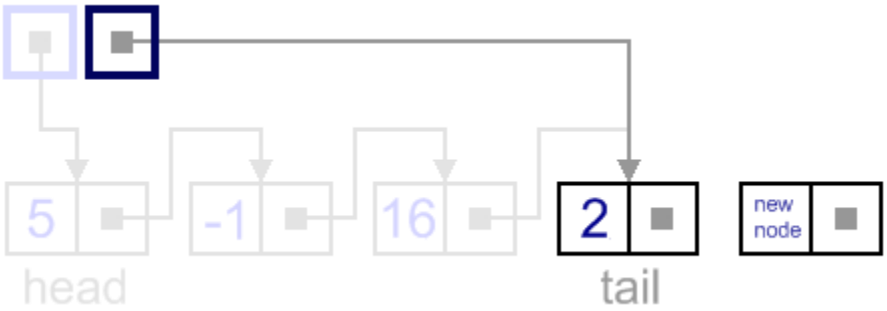


2. Update head link to point to the new node.



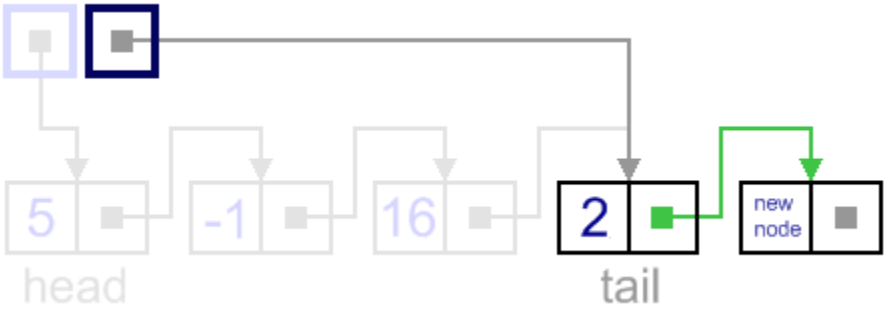
Add last

In this case, new node is inserted right after the current tail node.

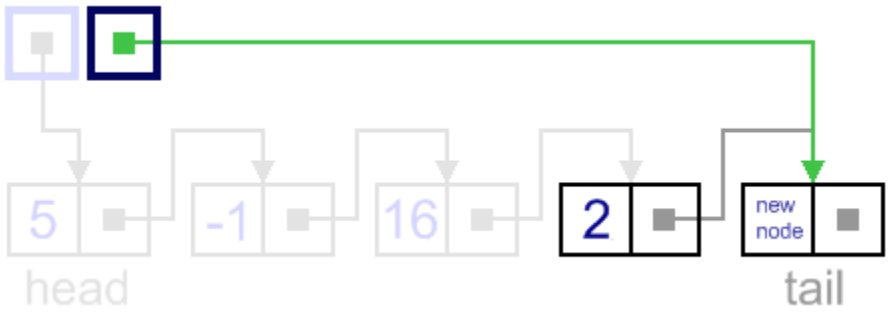


It can be done in two steps:

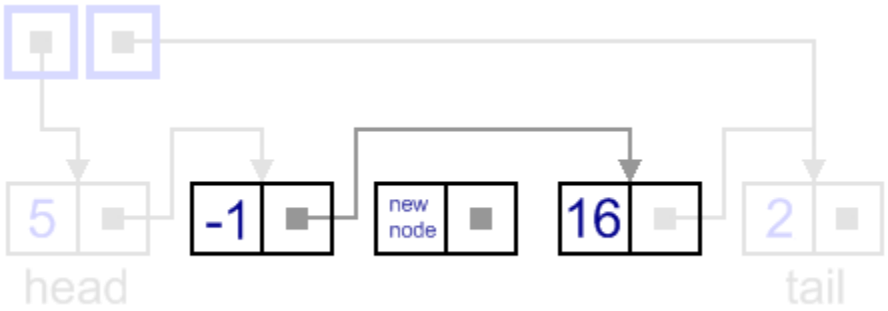
1. Update the next link of the current tail node, to point to the new node.



2. Update tail link to point to the new node.

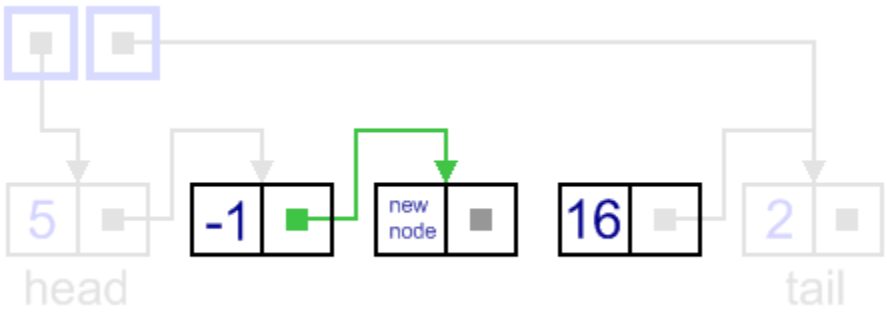


In general case, new node is **always inserted between** two nodes, which are already in the list. Head and tail links are not updated in this case.

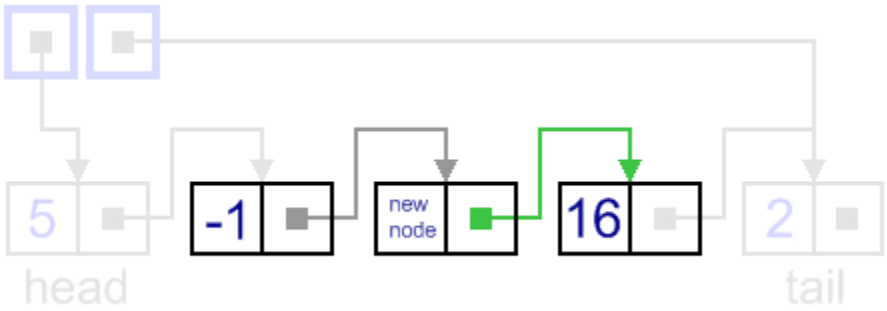


Such an insert can be done in two steps:

1. Update link of the "previous" node, to point to the new node.



2. Update link of the new node, to point to the "next" node.



Each node of linked list contains a data field and a pointer field 'next' to store the address of next node.

Algorithm for inserting an element at the end of the list.

Append (node **head, item)
{
create a node an assign the starting address of that into newnode;
Assign item into newnode−>data.
Assign NULL into newnode−>next.
If (list is empty) update *head by newnode;
Else traverse the list up to the end and attach the newnode at the end of the list.
}

2) Algorithm for inserting an element at the beginning of the list.

Addbeg (node **head, item)
{
Create a node an assign the starting address of that into newnode;
Assign item into newnode−>data.
Assign *head into newnode−>next.
Update *head by newnode.
}

Algorithm for inserting an element at the middle in the list.

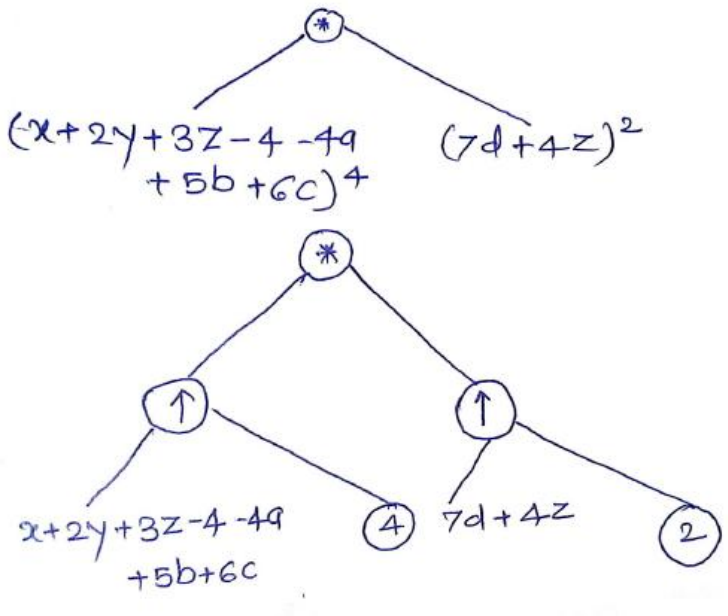
Insert (node *head, key, item)
{
Create a node an assign the starting address of that into newnode;

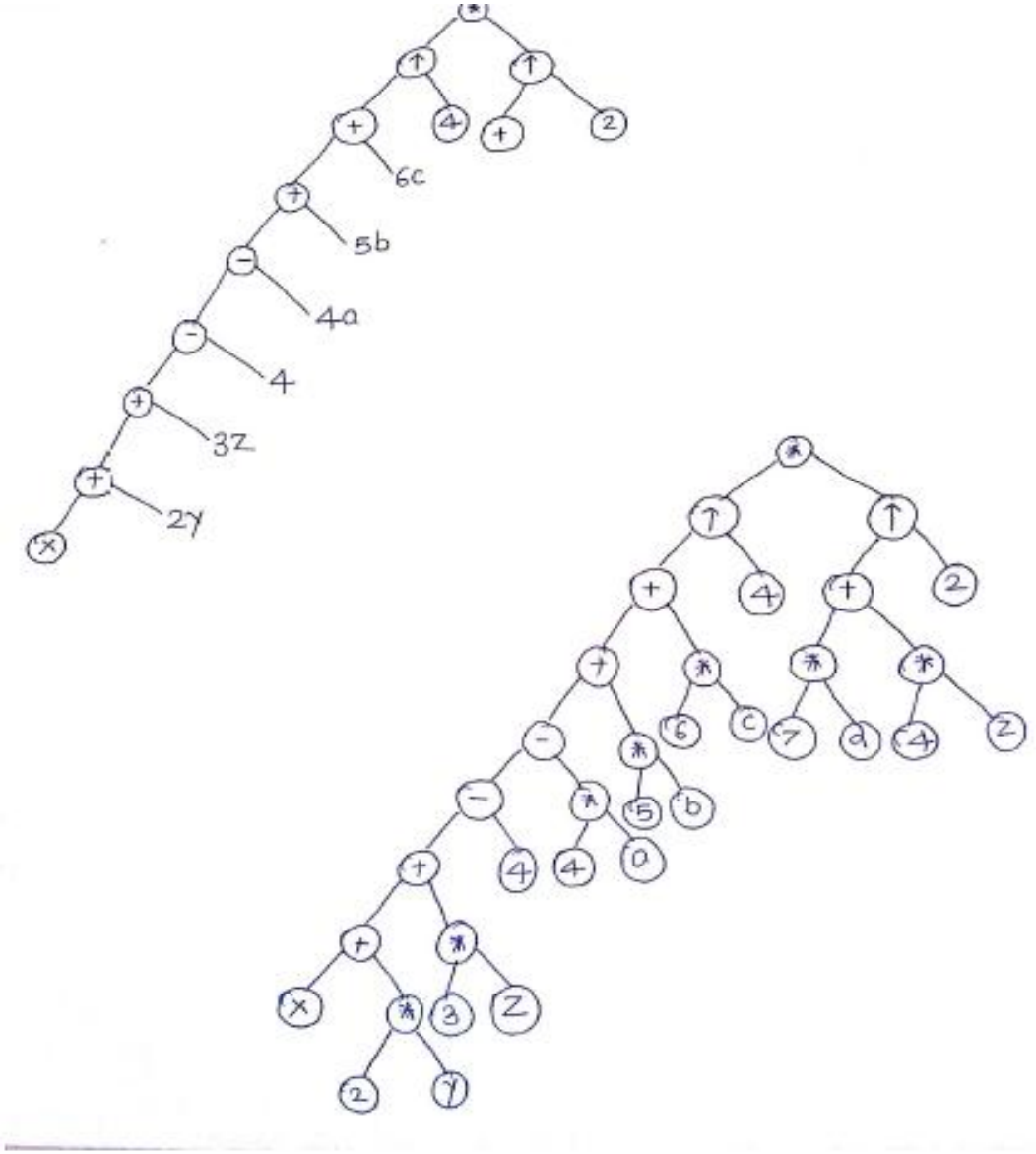
Assign item into newnode−>data.
Call search (head, key) and store the result into previous;
Assign previous−>next into Newnode−>next.
Update parent−>next by newnode;
}

(NOTE:
Examiner please consider the logic behind the algorithm .
It may vary according to books, knowledge.)

Q.2] c) Draw the tree structure for the following expression.
(x+2y-3z-4-4a+5b+6c)4*(7d=4z)^2

(Note: Each steps with accuracy contains 02 marks. 08-marks can also be allotted for the correct expression tree even if steps are not mentioned.)




6c
5b
4a
4
3z
2y
x
6
c
7
d
4
z
5
b
0
x
3
z
2
y

Q.3] a) (2-marks for explanation, 2-marks for example)

Explain Merge Sort with an example

Merging is process of combining two or more sorted files into a third sorted file.

1. **Divide Step** If a given array *A* has zero or one element, simply return; it is already sorted. Otherwise, split *A*[*p* .. *r*] into two sub arrays *A*[*p* .. *q*] and *A*[*q* + 1 .. *r*], each containing about half of the elements of *A*[*p* .. *r*]. That is, *q* is the halfway point of *A*[*p* .. *r*].

2. **Conquer Step** Conquer by recursively sorting the two sub arrays *A*[*p* .. *q*] and *A*[*q* + 1 .. *r*].

3. **Combine Step** Combine the elements back in *A*[*p* .. *r*] by merging the two sorted subarrays *A*[*p* .. *q*] and *A*[*q* + 1 .. *r*] into a sorted sequence. To accomplish this step, we will define a procedure MERGE (*A*, *p*, *q*, *r*). An example of a routine two sorted arrays a & b of n1 & n2 elements respectively, and merges them into third array c containing n3 elements is as following :

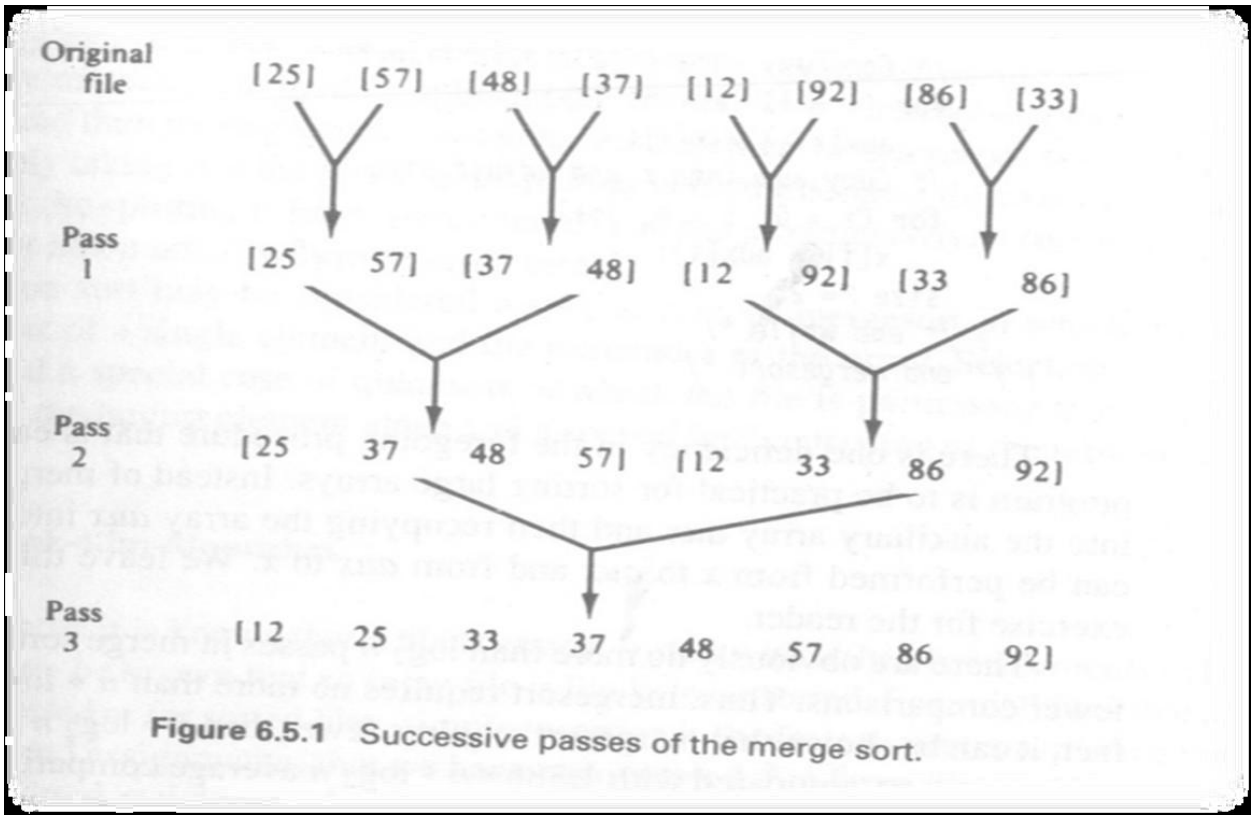


Figure 6.5.1 Successive passes of the merge sort.

Q.3] b) (2-marks for logic, 2-marks for main program)

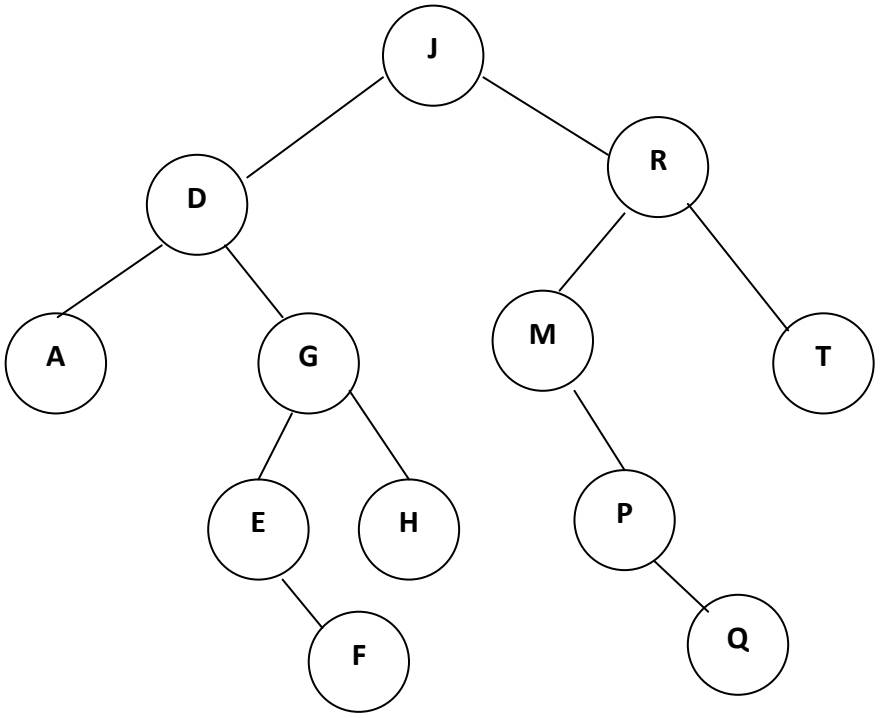
```
/* BUBBLE SORT */

#include<stdio.h>
#include<conio.h>
int a[10],i,j,temp;
int ex=0,c=0,e;
void main()
{clrscr();
printf("\n Bubble sort ");
printf("\n Enter the elements of array:");
for(i=0;i<10;i++)
{
scanf("\n%d",&a[i]);
}
c=0;
e=0;
ex=0;
printf("\n*****BUBBLE SORT*****\n");
for(j=0;j<=(n-1);j++)
{
        e=0;
        for(i=0;i<(n-1)-j;i++)
        {
                if(a[i]>a[i+1])
                {
                temp=a[i];
                a[i]=a[i+1];
```

```
                a[i+1]=temp;
                e++;
                ex++;
                }
                c++;
        }
                if(e==0)
        {
        break;
        }
}
printf("\n The sorted array=");
for(i=0;i<p;i++)
{
printf("\t %d",a[i]);
}
printf("\nThe no of exchanges:%d",ex);
printf("\nThe no of comparisons:%d",c);
printf("\nthe no of passes:%d",(j+1));
getch();
}
```

Q.3] c) (1-mark for logic, 3-marks for actual tree representation)

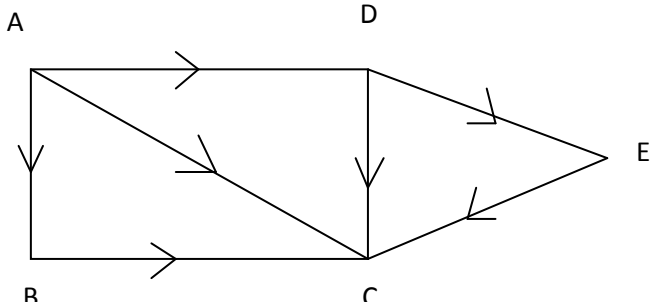
Construct Binary Search tree using the letters, J, R, D, G, T, E, M, H, P, A, F, Q.

Binary Search tree has the property that all the elements in the left sub tree of the root node are less than the content of root node and all elements in the right subtree of root node are greater than or equal to content of root node. Consider position of given letter in alphabet. J is the fist node, hence it is the root node. Next letter R which is greater than the root node will be the left child (position of R is 18, whereas J is 10). Next letter is D which is less than the root node hence will be the right child (position of D is 4 whereas J is 10).

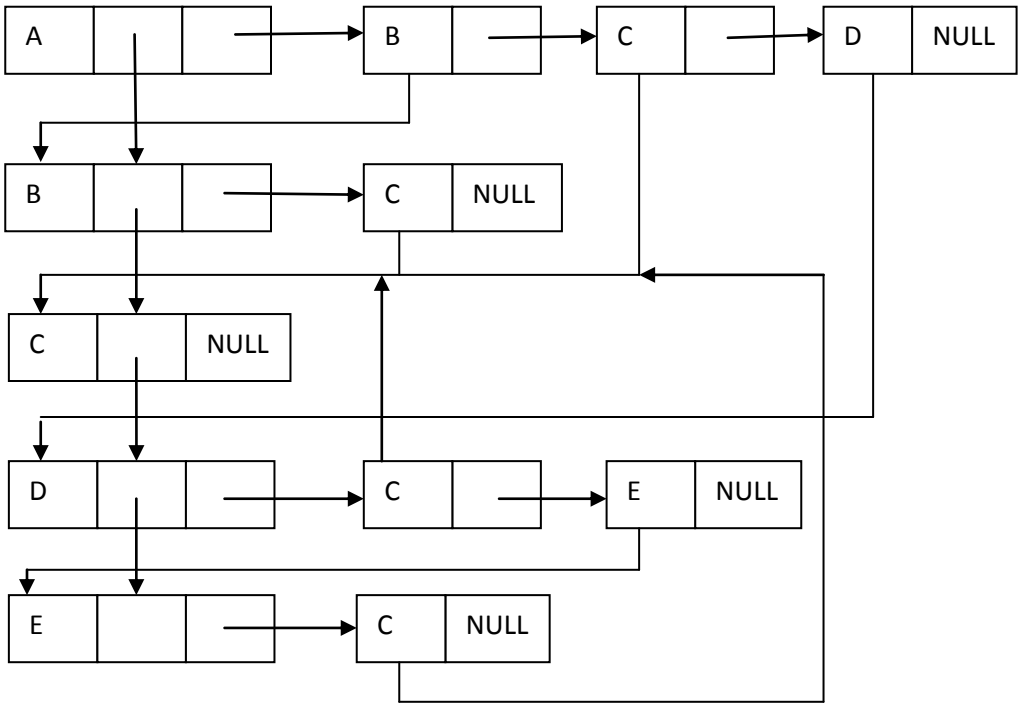


Q.3] d) (4-marks for any example of graph & its link representation should be considered)

Explain the link representation of graph using suitable example.



Vertex	Adjacency list
A	B, C, D
B	C
C	-
D	C, E
E	C

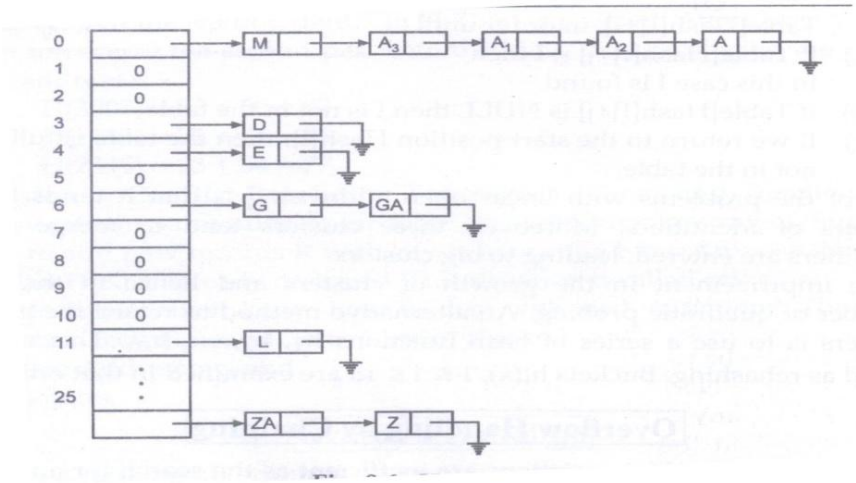


Q.3] e) (2-marks for Rehashing, 2-marks for Chaining)

Rehashing: It involves using a secondary hash function on the hash key of the item. The rehash function is applied successively until an empty position is found where the item can be inserted. If

the hash position of the item is found to be occupied during a search the rehash function is again used to locate the item.

Chaining : It builds a linked list of all items whose key has the same value.



Q.4] a) (4-marks for Correct Program, 4-marks for Example)

```
/* SELECTION SORT */
#include<stdio.h>
#include<conio.h>
int p,a[10],i,j,t,n,temp,type,k;
int ex=0,c=0,e,min=0;
void main()
{
clrscr();
printf("Selection sort\n ");
printf("\n Enter the no of elements:");
scanf("\n%d",&n);
p=n;
```

```
printf("\n Enter the elements of array:");
for(i=0; i<n; i++)
{
scanf("\n%d",&a[i]);
}
c=0;
e=0;
p=0;
printf("\n*****SELECTION SORT*****\n");
j=0;
min=a[0];
for(j=0; j<n; j++)
{
        for(i=j; i<n; i++)
        {
                c++;
                if(a[i]<=min)
                {
                min=a[i];
                p=i;
                }
                }
                if(a[j]= =min)
                {
                a[j]=min;
                }
                else
                {
```

```
            e++;
            t=a[j];
            a[j]=min;
            a[p]=t;
            }
            min=a[j+1];
            }
printf("\n The sorted array=");
for(j=0;j<n;j++)
{
printf("\t %d",a[j]);
}
printf("\n Number Of Exchanges Are : %d",e);
printf("\n Number Of Comparisons Are : %d",c);
printf("\n Number Of Passes Are : %d ",n-1);
getch();
}
```

Example:

16 23 13 9 7 5

- Select position 0 and element a[0]
- Consider a[0]=min
- From remaining array (a[1] to a[5]), find out the smallest element.
- If the smallest element is less then min, then perform exchange, (Smallest element with min).

Pass 1

min = a[0] = 16

smallest element = 5

as smallest element < min, perform exchange.

Output of Pass 1

5	23	13	9	7	16

Element at 0^{th} position gets its proper position.

Pass 2

min = a[1] = 23

smallest element = 9

as smallest element < min, perform exchange.

Output of Pass 2

5	7	13	9	23	16

Pass 3

min = a[2] = 13

smallest element = 9

as smallest element < min, perform exchange.

Output of Pass 3

5	7	9	13	23	16

Pass 4

Min=a[3] = 13

smallest element = 16

as smallest element > min, no exchange.

Output of Pass4

5	7	9	13	23	16

Pass 5

Min=a[4] = 23

smallest element = 16

as smallest element < min, perform exchange.

Output of Pass 5

5	7	9	13	16	23

Array is sorted using selection sort technique.th final sorted array is

5 7 9 13 16 23.

Q.4] b) (2-marks for each function (insert, delete, display), 2-marks for main program)

```
// Implementation Of CIRCULAR Queue
#include<stdio.h>
#include<conio.h>
#include<process.h>
# define QSIZE 5
int q[QSIZE];
int f=0,r=-1,t;
void dis();
void insert();
void del();
void insert()
{
int x,j;
if(f= =(r+1)%QSIZE && (r>-1))
{
printf("\nQueue is full");
}
else
{
r=(r+1)%QSIZE;
printf("\nEnter the element:");
scanf("%d",&q[r]);
   printf("%d",q[r]);
}
```

```
}

void del()
{
if(r<=-1)
{
printf("\nQueue is empty:");
}
else
{
t=q[f];
printf("\nThe deleted element is:%d",t);
if(f!=r)
 {
f=(f+1)%QSIZE;
}
else
{
f=0;
r=-1;
}
}
}
void dis()
{
if(r<=-1)
{
printf("\nQueue is empty");
```

```
}

else
{
int i;
for(i=f;i!=r;i=(i+1)%QSIZE)
{
printf("\t%d",q[i]);
}
printf("\t%d",q[i]);
}
}
void main()
{
int ch;
clrscr();
do
{
printf("\n....Queue Operations....");
printf("\n1.Insert \t2.Delete \t3.Display");
printf("\t 4.Exit");
printf("\nEnter your choice:");
scanf("%d", &ch);
switch(ch)
{
case 1:
insert();
break;
```

```
case 2:
del();
break;
case 3:
dis();
break;
case 4:
break;
default:
printf("\n\n!!!!!Invalid Option!!!!!");
}
}while (ch!=4);
getch();
}
```

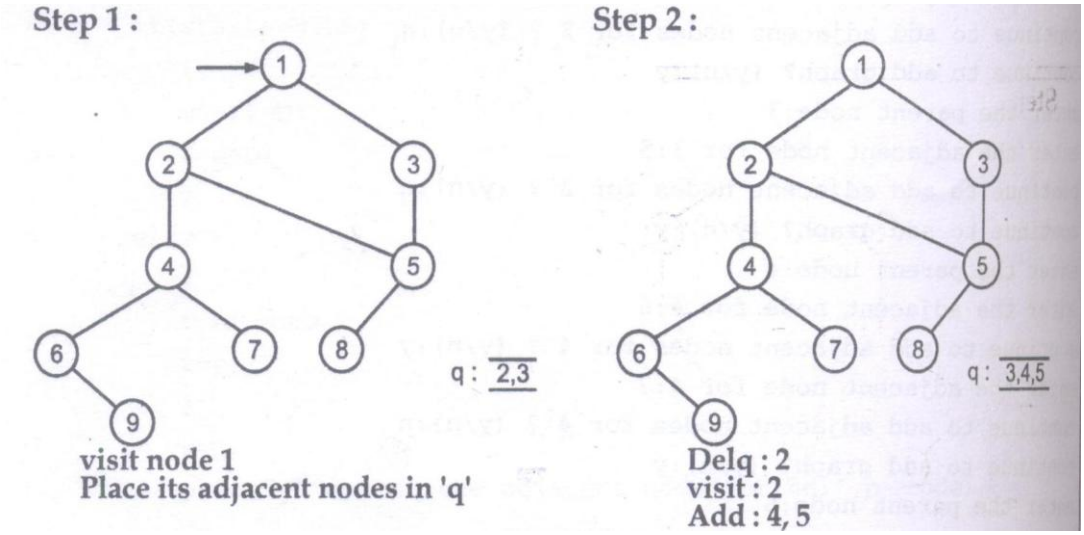
Q.4] c) Describe Breadth Search technique with suitable example.

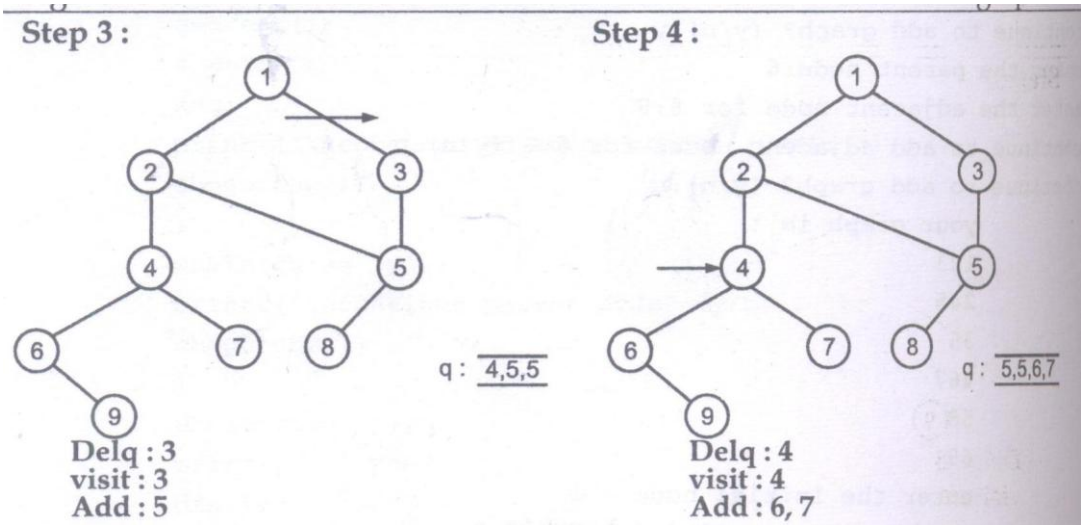
(4-marks for explanation, 4-marks for example)

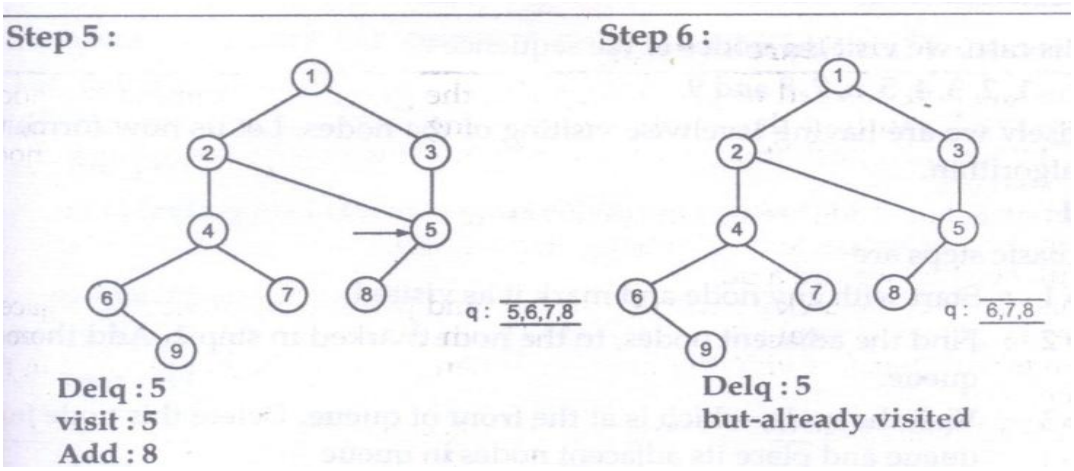
Breadth First Search (BFS)

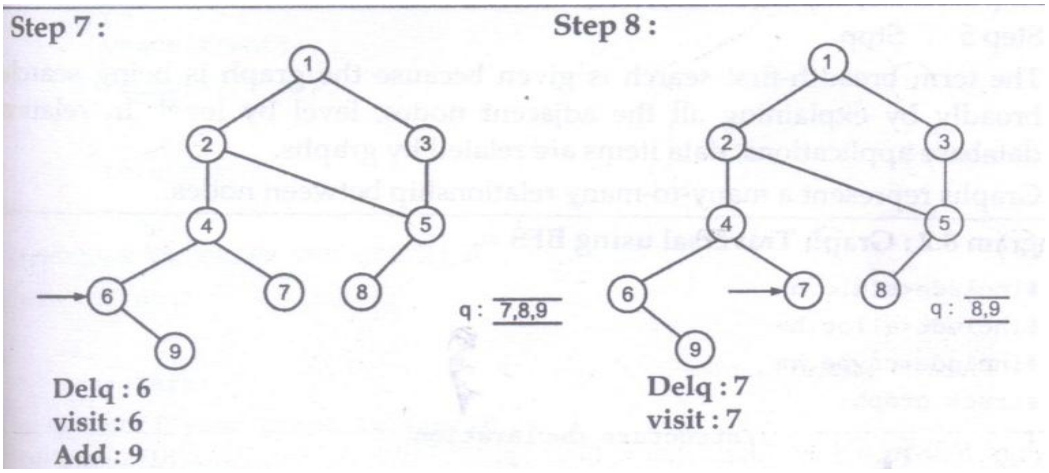
- In this m, instead of going deep into the graph, we examine the nodes across the breadth of the graph, before going to the next level. In DFS, we visit a node then push it on stack. The next node we visit is an adjacent node, unvisited one.
- If no such has node exist, we backtrack to the most recently visited node which has unprocessed neighbor. Hence the stack structure, LIFO was useful. For BFS we use a queue.
- In this method, we pick a node to visit first and place its unprocessed adjacent nodes in queue. Repeatedly take a node from the front of queue. If this node is unvisited, then we visit the node and then place all its neighbours in the queue.

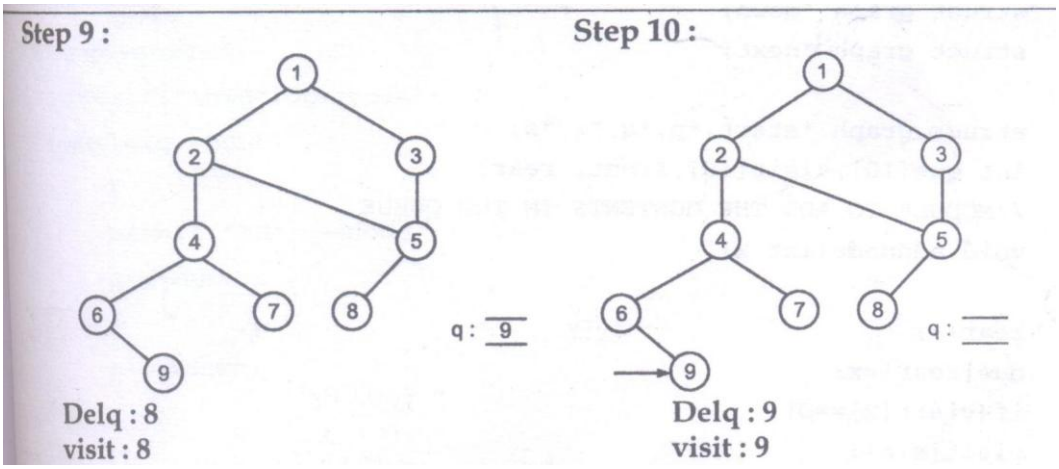
- Consider the following example:











In this case, we visit the nodes in the sequence:

1,2,3,4,5,6,7,8 and 9

Precisely we are having level wise visiting of the nodes. Let us now formalize the algorithm.

The basic steps are:

Step 1: start with any node and mark it as visited.

Step 2: Find the adjacent nodes, to the node marked in step1. Add them in queue.

Step 3: Visit the node, which is at the front of queue. Delete this node from queue and place its adjacent nodes in queue.

Step 4: Repeat step 3, till the queue is not empty.

Step 5: Stop.

Q.5] a) Compare quick sort & radix sort with respect to working principle & time complexity. (working principle-02 marks and time complexity-02 marks)

Ans: W**orking principle:** Quick sort is based on divide & conquer rule. Select a pivot element and divide an array such that element in the left side is < pivot and element in the right side is > pivot. Again repeat the process till all elements are sorted.

Radix sort is used generally when we intend to sort a large list of names alphabetically. Radix in this can be 26, as there are 26 alphabets.

The radix sort is based on the values of the actual digits in the positional representation. For example, the number 235 in decimal notation is written with a 2 in a hundred's position, 3 in ten's position and 5 in the unit's position. The sorter used above uses decimal numbers where the radix is 10 and hence uses only the first 10 pockets of the sorter. Passes required are no. of digits in the largest number in the list.

Time complexity:

Sorting	Quick sort	Radix sort
Best case	O(nlogn)	O(nlogn)
Average case	O(nlogn)	O(nlogn)
Worst case	O(n)2	O(n)2

Q.5] b) What is searching? Explain linear search with example.(Definitation-01 mark, principle and explanation=02 marks and example=01 mark)

Ans: **SEARCHING:** Let A be a collection of data elements of size n, and suppose ITEM is given to be searched. **Searching refers to the operation of finding the location LOC of ITEM in A.**

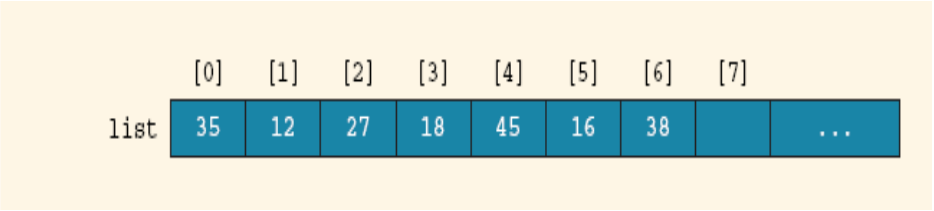
There are 2 types of algorithm used to search an element in array:

- ➔ Linear search.
- ➔ Binary search.

Algorithm is selected based on arrangement of list A.

Linear Search

Suppose arr is a linear array with n elements. Given no other information about arr, the simplest way to search for a given item in arr is to compare the item with each element in arr on by one. That is, first we test arr[1] = item, and then we test arr[2] = item, and so on. This method which traverses the data sequentially is called linear or sequential search.



- **Suppose that you want to determine whether 27 is in the list**
- **First compare 27 with list[0]; that is, compare 27 with 35**
- **Because list[0] ≠ 27, you then compare 27 with list[1]**
- **Because list[1] ≠ 27, you compare 27 with the next element in the list**
- **Because list[2] = 27, the search stops**
- **This search is successful!**

Complexity of linear search algorithm:

Worst Case

Clearly the worst case occurs when ITEM is the last element in the array arr.

f(n)=n

Average Case

f(n)= **n/2**

Best case:

Best case occurs when ITEM is the first element of array arr.

f(n)=1

Q.5] c) Explain queue as abstract data type. (definitation-01 mark, ADT-03 marks)

Ans: Def: A **Queue** is an ordered collection of items from which items may be deleted at one end (called the ***front*** of the queue) and into which items may be inserted at the other end (the ***rear*** of the queue).

Queue ADT:

```
typedef struct Q
{
        int R,F;
        int data[MAX];
}
```

Intialize(): rear=-1, front=0

isFullQ(*queue, max_queue_size*) ::=
if(rear == *max_queue_size*)
return *TRUE*
else return *FALSE*

isempty():

```
if(rear==-1)
        Return true
        Else return false
```

Queue **Enqueue**(*queue, item*) ::=
if (IsFullQ(*queue)) queue_full*
else

```
rear++;
queue[rear]=item;
```

[insert *item* at rear of *queue* and return *queue]*

Element **dequeue(*queue*) ::=**
if (IsEmptyQ(*queue*)) **return**
else

[remove and return the *item* at front of queue]

```
x=queue[front]
If(front==rear) then
Initialize();
Else
front++;
```

Q.5] d) Explain the operation of searching a desired node in the singly linked list. (searching defn-01 mark and explanation-03 marks)

SEARCHING A LINKED LIST

Suppose the data in LIST are not necessarily sorted. Then one searches for ITEM in LIST by traversing through the list using a pointer variable PTR and comparing ITEM with the INFO[PTR] of each node, one by one, of LIST. Before we update the pointer PTR by

PTR := LINK[PTR]

We require two tests. First we have to check to see whether we have reached the end of the list; first we check to see whether

PTR := NULL

If not, then we check to see whether

INFO[PTR]:= ITEM

If element not found search end in failure.

Algorithm SEARCH (NFO, LINK, START, ITEM, LOC)

LIST is a linked list in memory. This algorithm finds the location LOC of the node where ITEM first appears in LIST or sets LOC = NULL

1. Set PTR := START.
2. Repeat Step 3 while PTR = NULL:
3. If ITEM= INFO[PTR], then:

Set LOC := PTR. and Exit

Else:

Set PTR:= LINK[PTR]. [PTR now points to the next node]

[End of If structure]

[End of Step 2 loop]

4. [Search i s unsuccessful.] set LOC:=NULL.

5. Exit.

Complexity:

Worst case : f(n)=n

Average case: f(n)=n/2

Q.5] e) Give stepwise procedure for deletion of node in a binary tree. ([each case =01 mark)

Ans: In order to delete a node, we must find a node to be deleted. The node to be deleted may be:

a. A leaf node

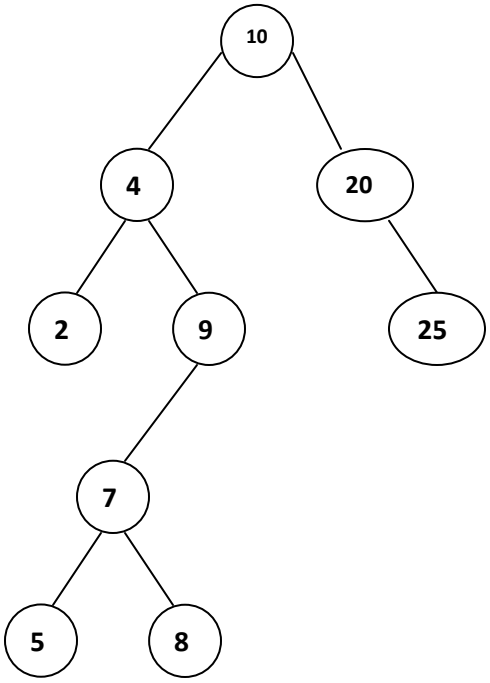
b. A node having one child.

c. A node having two child.

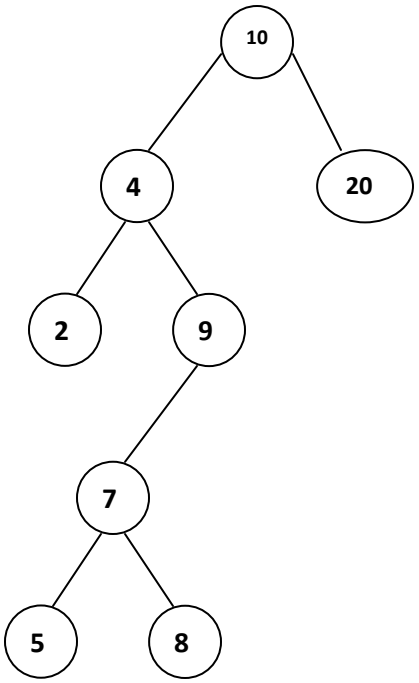
d. A node not present in a tree.

a. if node to be deleted is leaf node then it can be deleted by immediately by setting the parent pointer to null.

Ex: suppose node to be deleted is 25, then set Right of 20 as NULL.

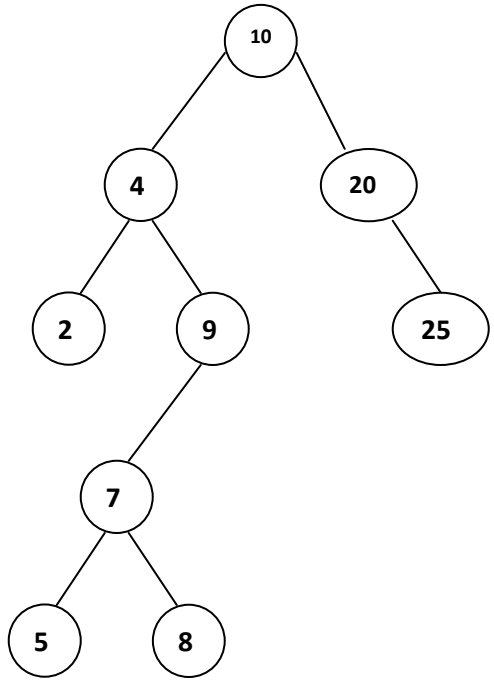


Before deletion

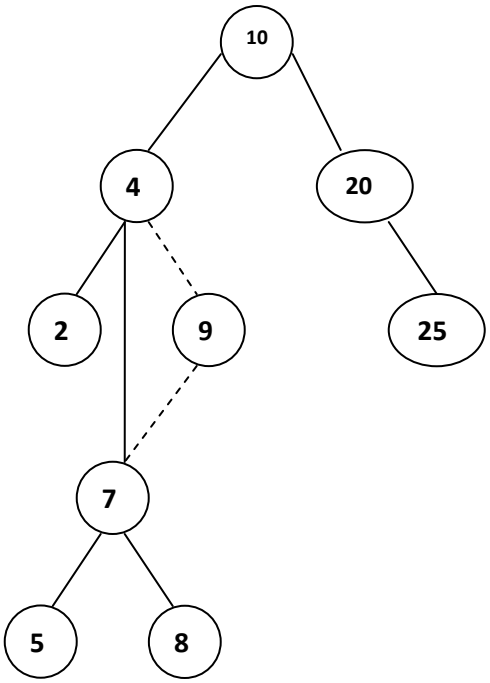


After deletion

b. If node to be deleted having one child, it can be deleted easily by assigning address of its child node to its parent node. Suppose q is to be deleted and if q is right child of p parent, then its only child will become the right child of p.



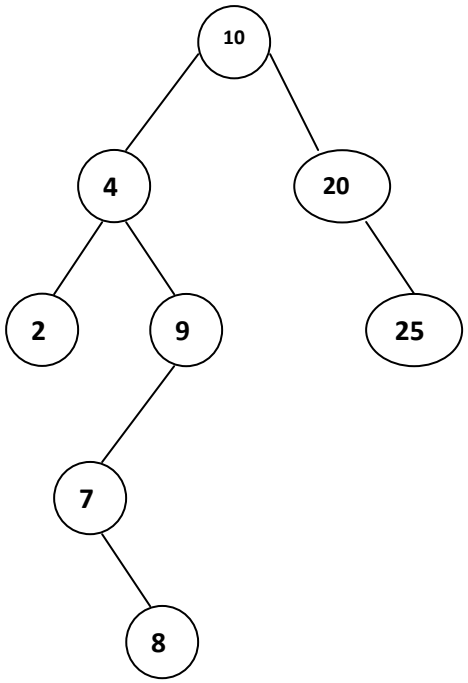
Before deletion

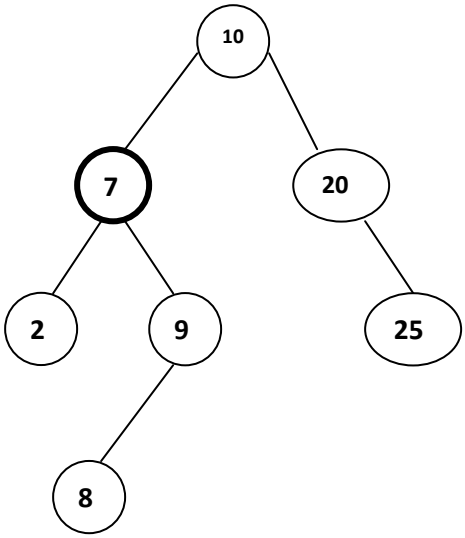


After deletion

Suppose 9 is to be deleted, as 9 is right child of 4, then only child subtree of node 9 , with node 7 will become right subtree of node 4 after deletion.

c. If node is to be deleted having two child, then we find in order successor of that node and replace with its in order successor and then it can be easily deleted by using case of deletion of node having 1 child.



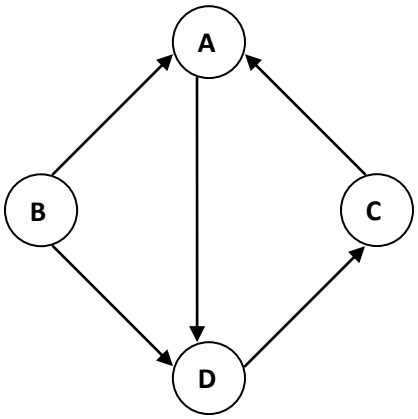


Before deletion After deletion

Suppose 4 is to bee deleted, 7 is in order successor of 4, then replace 4 with 7 by copying 7 into node 4 and node 4 is deleted by adjusting its parent and child to node 7.

d. If node to be deleted is not present in the tree, then deletion is not possible.

Q.5] f) Explain in degree & out degree of node with example.(in degree with ex=02 marks, out degree with ex=02 marks)



The total number of edges linked to the vertex is called as **degree.**

The **in degree** of vertex is the total no. of edges coming to that node.

The **out degree** of vertex is the total no. of edges going out from that node.

A vertex which has only outgoing edges and no incoming edges is called as **source**.

A vertex which has only incoming edges and no outgoing edges is called as **sink**.

When degree of vertex is 0, then it is **isolated** vertex.

Q.6] a) Write a C program for recursive binary search using function. (program with function = 04 marks)

Ans: /* binary search using recursive*/

```
#include <stdio.h>
#include <conio.h>
void main()
{
int a[10],i,n,m,c,l,u;
print("\n enter the size of an array");
scanf("%d",&n);
printf("\n enter the elements of an array");
for(i=0;i<n;i++)
scanf("%d",&a[i]);
printf("\n enter the element to be searched");
scanf("%d",&m);
l=0,u=n-1;
c=binary(a,n,m,l,u);
if(c==0)
printf("\n no is not found");
else
```

```
    printf("\n no is found");
   getch();
 }
    int binary(int a[], int n, int m, int l, int u)
    {
    int mid,c=0;
    if(l<=u)
    {
    mid=(l+u)/2;
    if(m==a[mid])
    c=1;
    else if(m<a[mid])
          return binary(a,n,m,l,mid-1);
    else
         return binary(a,n,m,mid+1,u);
    }
    return c;
    }
```

Output:

enter the size of an array: 5

enter the elements of an array: 8 9 10 11 12

enter the element to be searched: 8

no. is foun

Q.6] b) Write a C program for implementation of stack using array. (program with 2 functions= 4 marks)

Ans: /* simulation of stack using array */

```
#include <stdio.h>
#include <conio.h>
#define MAX 6
typedef struct stack
{
  int data[MAX];
  int top;
}stack;
void init(stack *);
int empty(stack *);
int full(stack *);
int pop(stack *);
void push(stack *,int);
void print(stack *);
void main()
{
  Stack s;
  int x,op;
  init(&s);
  clrscr();
  do
  {
  printf("\n 1. push \n 2.pop. \n print");
```

```
    printf("\n enter ur choice");
    scanf("%d",&op);
    switch(op)
    {
     case 1:
       printf("\n enter a element:");
       scanf("%d",&x);
       if (!full(&s))
       push(&s,x);
       else
      printf("\n stack is full….");
       break;
     case  2: if(!empty(&s))
            {
                  x=pop(&s);
                  printf("\n popped value= %d", x);
            }
          else
                  printf("\n stack is empty….");
     break;
case 3:
          print(&s);
  break;
}
}while(op!=4);
```

```
}
void init(stack *s)
{
  s->top=-1;
}
int empty(stack *s)
 {
   if(s->top==-1)
     return(1);
     return(0);
 }
int full(stack *s)
{
 if(s->top==MAX-1)
    return(1);
     return(0);
}
void push(stack *s,int x)
{
  s->top=s->top+1;
s->data[s->top]=x;
}
    int spop(stack *s)
    {
    int x;
```

```
x=s->data[s->top];
s->top=s->top-1;
return(x);
}
void print(stack *s)
{
int i;
printf("\n");
for(i=s->top;i>=0;i--)
printf("%d",s->data[i]);
}
}
getch();
}
```

Q.6] c) Explain any 2 application of queue. (each application=02 marks)

Ans : 1. Various features of operating system are implemented using a queue.

a. Scheduling of processes(round robin algorithm)
b. Spooling – to maintain queue of jobs to be printed.
c. A queue of client processes waiting to receive the service from the server process.

2. Various application software using non-linear data structure tree or graph require queue for breadth first traversal.

3. Airport simulation – maintain queue of flights which going to fly or land

4. Simulation of real life problem- waiting time of person at railway reservation counter can be found if arrival rate, service time and no. of service counters are known.

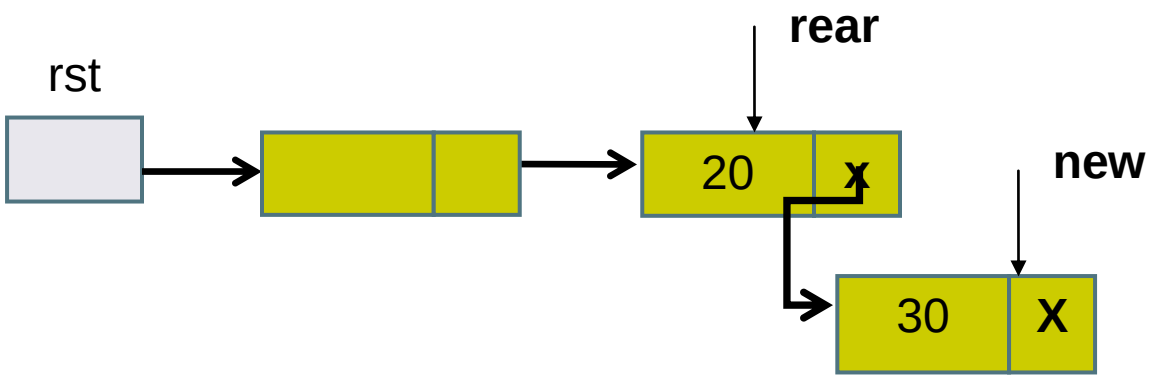
Q.6] d) Explain implementation of queue using linked list.(definition=01 mark, implementation steps for insert & delete = 03 marks)

Ans: Def: A **Queue** is an ordered collection of items from which items may be deleted at one end (called the ***front*** of the queue) and into which items may be inserted at the other end (the ***rear*** of

the queue). Queue is based on fifo concept. As insertion is performed at end so node is added at end and deletion in queue is performed at beginning so node is deleted at beginning.

Operations:

- insert
- delete



Insertion:

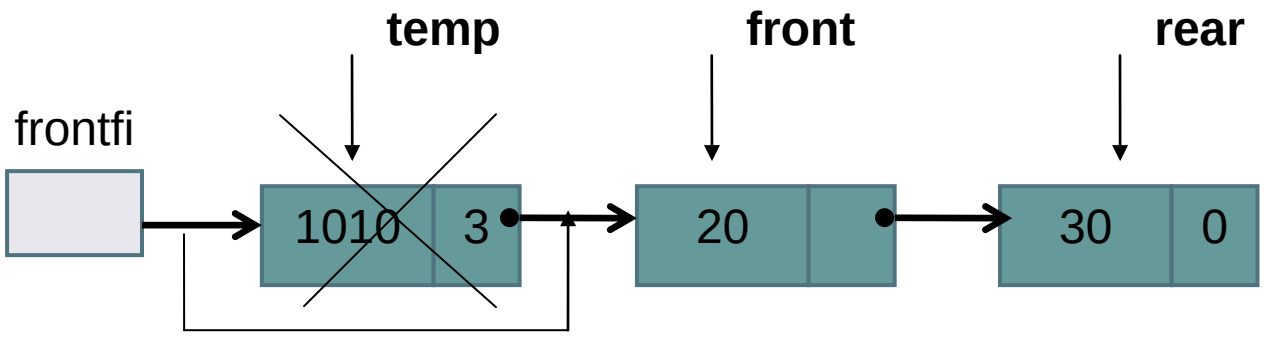
1> Create a new node and add new item in new node.
New->ptr=NULL

New->info=item.

2> add new node at the end.
Rear->ptr=new

Rear=new

Special case: check whether queue is full or not(overflow)



Deletion:

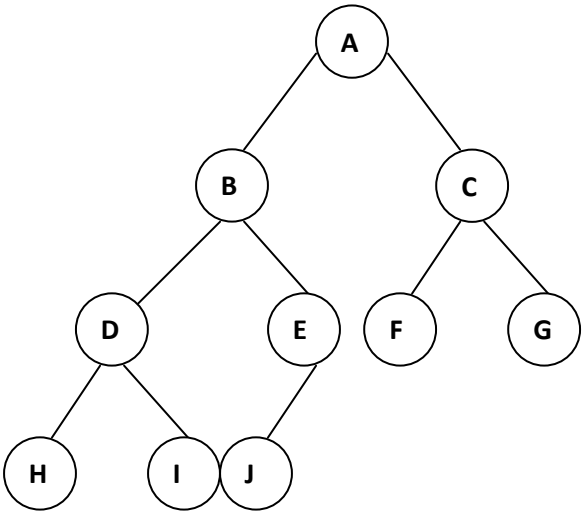
1. assign temp to first node
temp=front

2. move front to next node
front=front->next

3. temp->next=NULL
4. delete the node from beginning
free(temp)

Special case: check whether queue is empty or not(underflow)

Q.6] e) Explain the term with help of the diagram: siblings, leaf node, level of the tree, depth of the tree. (each term=01 mark)



Siblings: Nodes with same parent are sibling. D and E are sibling as they are children of same parent B.

Leaf node: A node of degree 0 is known as a leaf node. A leaf node is terminal node and has no children. In fig H,I,J,F,G are leaf nodes.

Level of the tree: root is having 0 level. Level of any other node in the tree is one more than the level of its father.

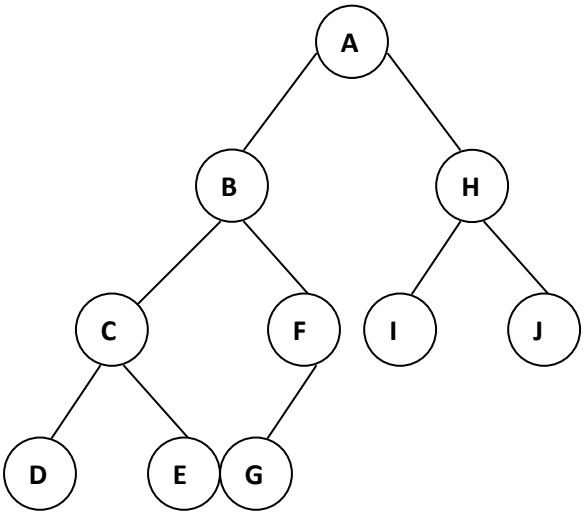
A- level 0
B,C- level 1

D,E,F,G- level 2

H,I,J - level 3

Depth of the tree: depth of tree is the maximum level of any node in the tree. This is equal to the longest path from the root to any leaf node. The depth of the tree in above figure is 4.

Q.6] f) Perform in order, post order & pre order traversal of the binary tree ref. figure No.1. (in order=01 mark, pre order= 1 ½ mark and post order= 1 ½ mark)



In order traversal (left-root-right): D C E B G F A I H J

Pre order traversal (root-left-right): A B C D E F G H I J

Post order traversal (left-right-root): D E C G F B I J H A