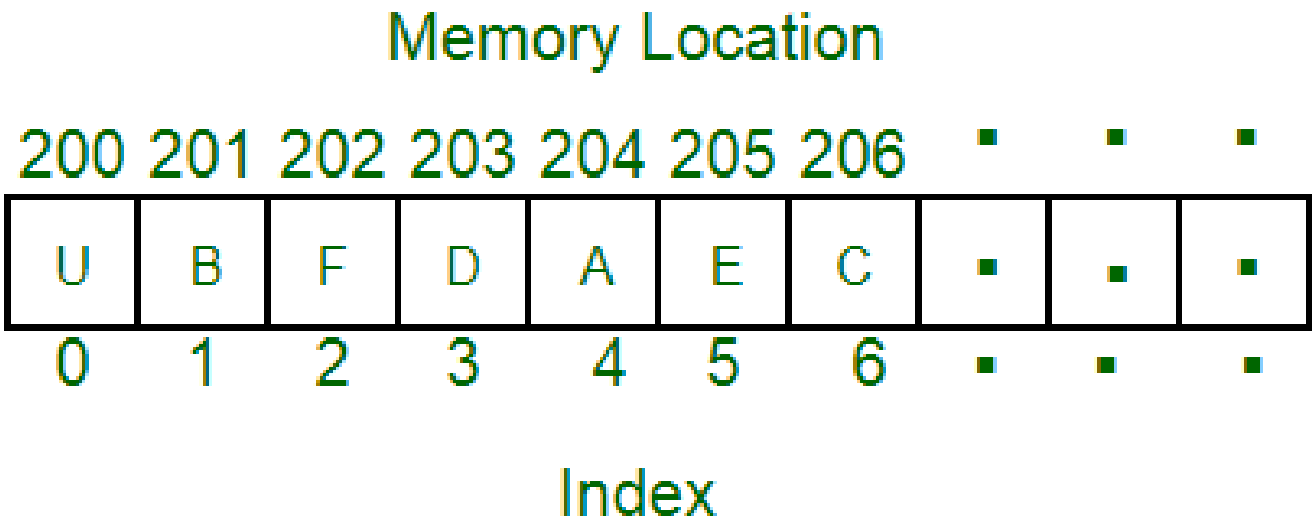
Array

- An array is a collection of items stored at contiguous memory locations. The idea is to store multiple items of the same type together. This makes it easier to calculate the position of each element by simply adding an offset to a base value, i.e., the memory location of the first element of the array (generally denoted by the name of the array).
- Each element can be uniquely identified by their index in the array (in a similar way as you could identify your friends by the step on which they were on in the above example).



- How to declare an array in C
  - datatype array Name[array Size];
  - int data[100];
- How to initialize an array
  - It is possible to initialize an array during declaration. For example,
    - int mark[5] = {19, 10, 8, 17, 9};
  - You can also initialize an array like this.
  - int mark[] = {19, 10, 8, 17, 9};
  - Here, we haven't specified the size. However, the compiler knows its size is 5 as we are initializing it with 5 elements.
- Change Value of Array elements
  - int mark[5] = {19, 10, 8, 17, 9};
  - make the value of the third element to -1
    - mark[2] = -1;
  - make the value of the fifth element to 0

- mark[4] = 0;

- Types of indexing in array:
  - 0 (zero-based indexing): The first element of the array is indexed by subscript of 0
  - 1 (one-based indexing): The first element of the array is indexed by subscript of 1
  - n (n-based indexing): The base index of an array can be freely chosen. Usually programming languages allowing n-based indexing also allow negative index values and other scalar data types like enumerations, or characters may be used as an array index.

- Size of an array
  - Number of elements = (Upper bound – Lower Bound) + 1
  - Size = number of elements * Size of each elements in bytes
    - Lower bound index of the first element of the array
    - Upper bound index of the last element of the array

One Dimensional array

- Address of the element at k^{th} index
  - a[k] = B + W*k
  - a[k] = B + W*(k – Lower bound)
    - B is the base address of the array
    - W is the size of each element
    - K is the index of the element
    - Lower bound index of the first element of the array
    - Upper bound index of the last element of the array

Q Let the base address of the first element of the array is 250 and each element of the array occupies 3 bytes in the memory, then address of the fifth element of a one- dimensional array a[10] ?

Q An array has been declared as follows
A: array [-6--------6] of elements where every element takes 4 bytes, if the base address of the array is 3500 find the address of array[0]?

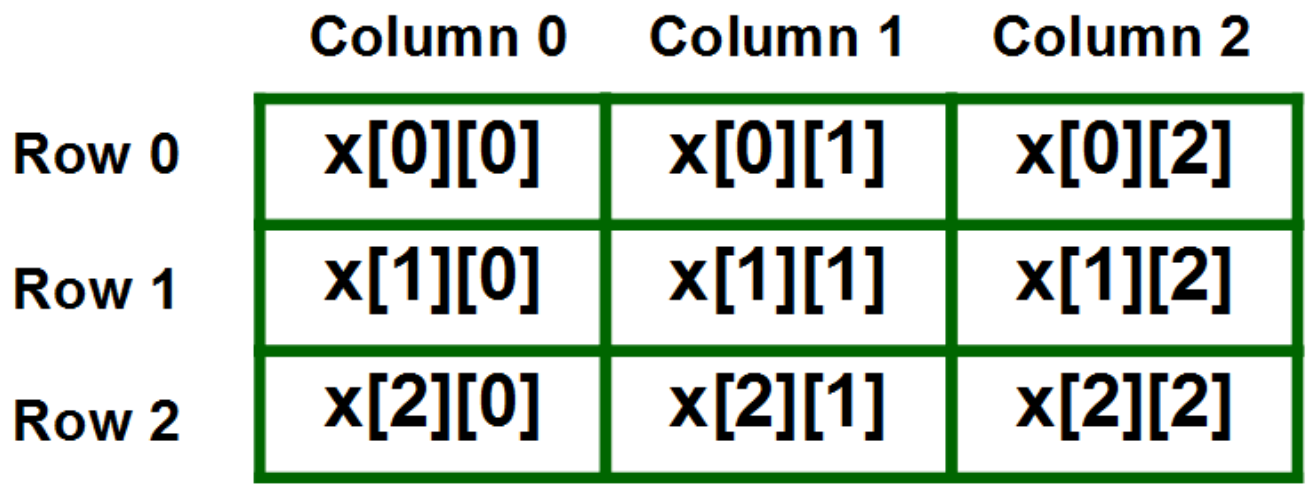
Q A program P reads in 500 integers in the range [0...100] experimenting the scores of 500 students. It then prints the frequency of each score above 50. What would be the best way for P to store the frequencies? (GATE - 2005) (2 Marks)

(A) An array of 50 numbers
(B) An array of 100 numbers
(C) An array of 500 numbers
(D) A dynamically allocated array of 550 numbers

Answer: (A)

Two-Dimensional array

- The two-dimensional array can be defined as an array of arrays. The 2D array is organized as matrices which can be represented as the collection of rows and columns. However, 2D arrays are created to implement a relational database lookalike data structure. It provides ease of holding the bulk of data at once which can be passed to any number of functions wherever required.



- data_type array_name[rows][columns];

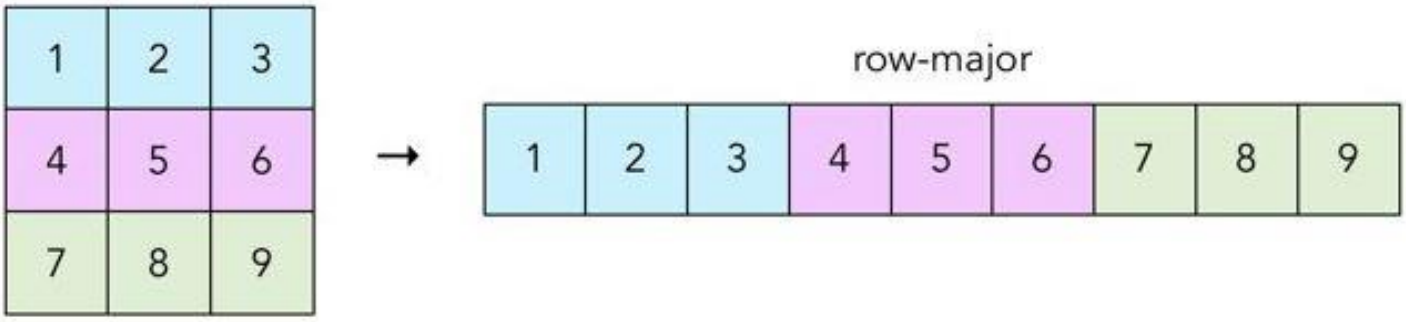
```
int disp[2][4] = {
        {10, 11, 12, 13},
        {14, 15, 16, 17}
                };
```

OR

```
int disp[2][4] = { 10, 11, 12, 13, 14, 15, 16, 17};
```

Row Major implementation of 2D array

- In computing, row-major order and column-major order are methods for storing multidimensional arrays in linear storage such as random access memory.
- The difference between the orders lies in which elements of an array are contiguous in memory.
- In Row major method elements of an array are arranged sequentially row by row. Thus, elements of first row occupies first set of memory locations reserved for the array, elements of second row occupies the next set of memory and so on.



Address of a[i][j] = B + W*[(U_2-L_2+1) $(i-L_1)$ + $(j-L_2)$]

B = Base address

W = Size of each element

L_1 = Lower bound of rows

U_1 = Upper bound of rows

L_2 = Lower bound of columns

U_2 = Upper bound of columns

(U_2-L_2+1) = numbers of columns

$(i-L_1)$ = number of rows before us

$(j-L_2)$ = number of elements before us in current row

22 Let A be a two-dimensional array declared as follows:

A : array [1 ... 10] [1 ... 15] of integer;

Assuming that each integer takes one memory location. The array is stored in row-major order and the first element of the array is stored at location 100, what is the address of the element A[i][j]?

(a) 15i + j + 84 (b) 15j + i + 84

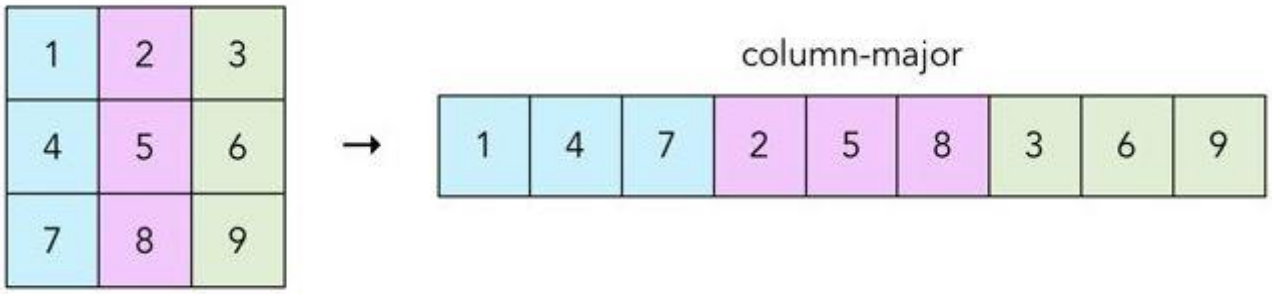
(c) 10i + j + 89 (d) 10j + i + 89

[1998 : 2 Marks]

Ans: a

Column Major implementation of 2D array

- In Column major method elements of an array are arranged sequentially column by column. Thus, elements of first column occupies first set of memory locations reserved for the array, elements of second column occupies the next set of memory and so on.



Address of a[i][j] = B + W*[(U_1-L_1+1) (j-L_2) + (i-L_1)]

B = Base address

W = Size of each element

L_1 = Lower bound of rows

U_1 = Upper bound of rows

L_2 = Lower bound of columns

U_2 = Upper bound of columns

(U_1-L_1+1) = numbers of rows

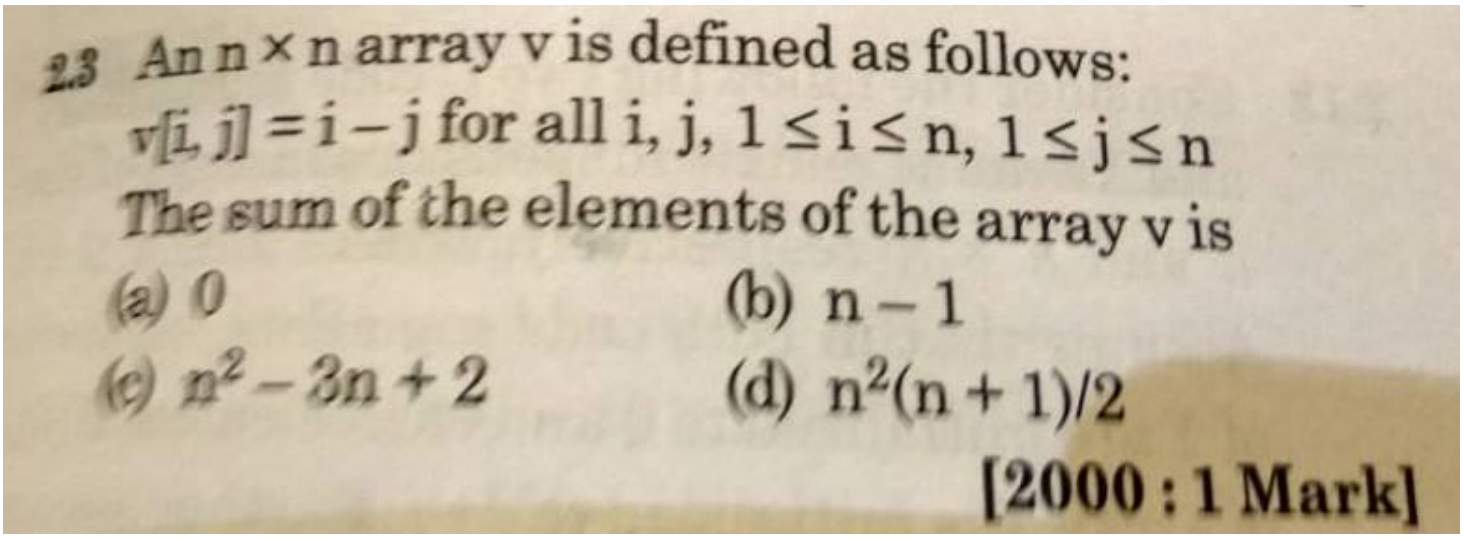
(j-L_2) = number of columns before us

(i-L_1) = number of elements before us in current column

Q An array VAL[1…15][1…10] is stored in the memory with each element requiring 4 bytes of storage. If the base address of the array VAL is 1500, determine the location of VAL[12][9] when the array VAL is stored

(i) Row wise (ii) Column wise.

Q A[5.....15,-8.....8] is A[11][17] and we are supposed to find A[8][5] - Row Major Order Base Address:800, each element occupies 4 memory cells?



2.3 An $n \times n$ array v is defined as follows:

$v[i, j] = i - j$ for all i, j, $1 \le i \le n$, $1 \le j \le n$

The sum of the elements of the array v is

(a) 0 (b) $n - 1$

(c) $n^2 - 3n + 2$ (d) $n^2(n + 1)/2$

[2000 : 1 Mark]

Ans: a

N-Dimensional array

N-Dimensional array

A([L_1]---[U_1]), ([L_2]---[U_2]), ([L_3]---[U_3]), ([L_4]---[U_4])--------([L_N]---[U_N])

Location of A [I, j, k, ----, x] =

B + (i-L_1) (U_2-L_2+1) (U_3-L_3+1) (U_4-L_4+1) ----(U_n-L_n+1)

+ (j-L_2)(U_3-L_3+1) (U_4-L_4+1) ----(U_n-L_n+1)

+(k-L_3)(U_4-L_4+1) ----(U_n-L_n+1)

+

+

+

+(x-L_n)

2.1 In a compact single dimensional array representation for lower triangular matrices (i.e. all the elements above the diagonal are zero) of size $n \times n$, non-zero elements (i.e. elements of the lower triangle) of each row are stored one after another, starting from the first row, the index of the $(i, j)^{th}$ element of the lower triangular matrix in this new representation is

(a) $i + j$

(b) $i + j - 1$

(c) $(j-1) + \frac{i(i-1)}{2}$

(d) $i + \frac{j(j-1)}{2}$

[1994 : 2 Mark

Ans: c

2.4 Suppose you are given an array s[1...n] and a procedure reverse(s, i, j) which reverses the order of elements in between positions i and j (both inclusive). What does the following sequence do, where $1 \leq k \leq n$:

```
reverse (s, 1, k);
reverse (s, k + 1, n);
reverse (s, 1, n);
```

(a) Rotates s left by k positions

(b) Leaves s unchanged

(c) Reverses all elements of s

(d) None of the above

[2000 : 1 Mark

Ans: a

Q Let A be a square matrix of size n x n. Consider the following program. What is the expected output? **(GATE - 2014) (1 Marks)**

```
C = 100
for i = 1 to n do
   for j = 1 to n do
   {
      Temp = A[i][j] + C
      A[i][j] = A[j][i]
      A[j][i] = Temp - C
   }
for i = 1 to n do
   for j = 1 to n do
      Output(A[i][j]);
```

(A) The matrix A itself
(B) Transpose of matrix A
(C) Adding 100 to the upper diagonal elements and subtracting 100 from diagonal elements of A
(D) None of the above
Answer: (A)

Q A Young tableau is a 2D array of integers increasing from left to right and from top to bottom. Any unfilled entries are marked with ∞, and hence there cannot be any entry to the right of, or below a ∞. The following Young tableau consists of unique entries **(GATE - 2015) (2 Marks)**

1	2	5	14
3	4	6	23
10	12	18	25
31	∞	∞	∞

When an element is removed from a Young tableau, other elements should be moved into its place so that the resulting table is still a Young tableau (unfilled
entries may be filled in with a ∞). The minimum number of entries (other than 1) to be shifted, to remove 1 from the given Young tableau is _____

(A) 2 **(B)** 5 **(C)** 6 **(D)** 18
Answer: (B)

Q Consider a n*n square matrix A, such that

A[i][j] = 0 if(i<j)

a) find the space required to store the array in memory

Ans: 1+2+3+4+----+n-1+n

n(n-1)/2

b) derive an address access formula to access this matrix (if stored in row major order)

	0	**1**	**2**	**3**
0	00			
1	10	11		
2	20	21	22	
3	30	31	32	33

0	1	2	3	4	5	6	7	8	9
00	10	11	20	21	22	30	31	32	33

= B + no of element present in (i-L1) + no of element present in the current row before us

= B + $(i-L_1)\ (i-L_1+1)/2 + (j-L_2)$

b) derive an address access formula to access this matrix (if stored in row major order)

	0	**1**	**2**	**3**
0	00			
1	10	11		
2	20	21	22	
3	30	31	32	33

0	1	2	3	4	5	6	7	8	9
00	10	20	30	11	21	31	22	32	33

= B + no of element present in ($j-L_2$) + no of element present in the current column before us

= B + n + n-1 + n-2 + ----- + n-(j-1) + (i-j)

= B + nj – (1+2+3----+j-1) + (i-j)

= B + nj – (j)(j-1)/2 + (i-j)

Q Consider a n*n square matrix A, such that

A[i][j] = A[j][i]

find the space required to store the array in memory

Q Consider a n*n square matrix A, such that

A[i][j] = 0 if |i-j| > 1

find the space required to store the array in memory

Q Consider a n*n square matrix A, such that

A[i][j] = A[i-1][j-1] if i>0, j>0

find the space required to store the array in memory

Q Consider a n*n square matrix A, such that

	0	**1**	**2**	**3**
0	00	01	02	03
1	10	11	12	13
2	20	21	22	23
3	30	31	32	33

Q An n x n array v is defined as follows:
v[i, j] = i-j for all i, j, 1 <= i <= n, 1 <= j <= n
The sum of the elements of the array v is **(Gate-2000) (1 Marks)**
(A) 0 **(B)** n-1 **(C)** n^2 - 3n + 2 **(D)** n^2 (n+1)/2
Answer: (A)

Q Two matrices M1 and M2 are to be stored in arrays A and B respectively. Each array can be stored either in row-major or column-major order in contiguous memory locations. The time complexity of an algorithm to compute M1 × M2 will be **(Gate-2004) (2 Marks)**
(A) best if A is in row-major, and B is in column- major order
(B) best if both are in row-major order
(C) best if both are in column-major order
(D) independent of the storage scheme
Answer: (D)

STACK

- A stack is a non-primitive linear data structure. it is an ordered list in which addition of a new data item and deletion of already existing data item is done from only one end known as top of stack (TOS).
- The element which is added in last will be first to be removed and the element which is inserted first will be removed in last.
- That is why it is called last in first out (LIFO) or first in last out (FILO) type of list.
- Most frequently accessible element in the stack is the top most element, whereas the least accessible element is the bottom of the stack.

Stack Implementation

- Stack is generally implemented in two ways.
- **Static Implementation**: - Here array is used to create stack. it is a simple technique but is not a flexible way of creation, as the size of stack has to be declared during program design, after that size implementation is not efficient with respect to memory utilization.
- **Dynamic implementation**: - It is also called linked list representation and uses pointer to implement the stack type of data structure.

Q Which of the following is true about linked list implementation of stack?
(A) In push operation, if new nodes are inserted at the beginning of linked list, then in pop operation, nodes must be removed from end.
(B) In push operation, if new nodes are inserted at the end, then in pop operation, nodes must be removed from the beginning.
(C) Both of the above
(D) None of the above
Answer: (D)

- **Basics operations on stack**
  - Push
  - Pop

Q Which data structure is used for balancing of symbols?
(A) Stack **(B)** Queue **(C)** Tree **(D)** Graph
Answer: (A)

Q The best data structure to check whether an arithmetic expression has balanced parentheses is a **(GATE - 2004) (1 Marks)**
(A) queue **(B)** stack **(C)** tree **(D)** list
Answer: (B)

Push operation: - The process of adding new element to the top of stack is called push operation. the new element will be inserted at the top after every push operation the top is incremented by one. in the case the array is full and no new element can be accommodated it is called over-flow condition.

```
PUSH (S, N, TOP, x)
{
        if (TOP=N)
                Print stack overflow and exit

        TOP = TOP + 1

        S[TOP] = x

        exit
}
```

Pop: - The process of deleting an element. from the top of stack is called POP operation, after every POP operation the stack is decremented by one if there is no element in the stack and the POP operation is requested then this will result into a stack underflow condition.

```
POP (S, N, TOP)
{
        if (TOP==-1)
                print underflow and exit
        y = S[TOP]
        TOP=TOP-1
        return(y) and exit
}
```

Q A single array A[1...MAXSIZE] is used to implement two stacks. The two stacks grow from opposite ends of the array. Variables top1 and top2 (top 1< top 2) point to the location of the topmost element in each of the stacks. If the space is to be used efficiently, the condition for "stack full" is (GATE - 2004) (2 Marks)

(A) (top1 = MAXSIZE/2) and (top2 = MAXSIZE/2+1)

(B) top1 + top2 = MAXSIZE

(C) (top1= MAXSIZE/2) or (top2 = MAXSIZE)

(D) top1= top2 -1

Answer: (D)

Q Let S be a stack of size n >= 1. Starting with the empty stack, suppose we push the first n natural numbers in sequence, and then perform n pop operations. Assume that Push and Pop operation take X seconds each, and Y seconds elapse between the end of one such stack operation and the start of the next operation. For m >= 1, define the stack-life of m as the time elapsed from the end of Push(m) to the start of the pop operation that removes m from S. The average stack-life of an element of this stack is (GATE - 2003) (2 Marks)

(A) n(X+ Y) **(B)** 3Y + 2X **(C)** n(X + Y)-X **(D)** Y + 2X

Answer: (C)

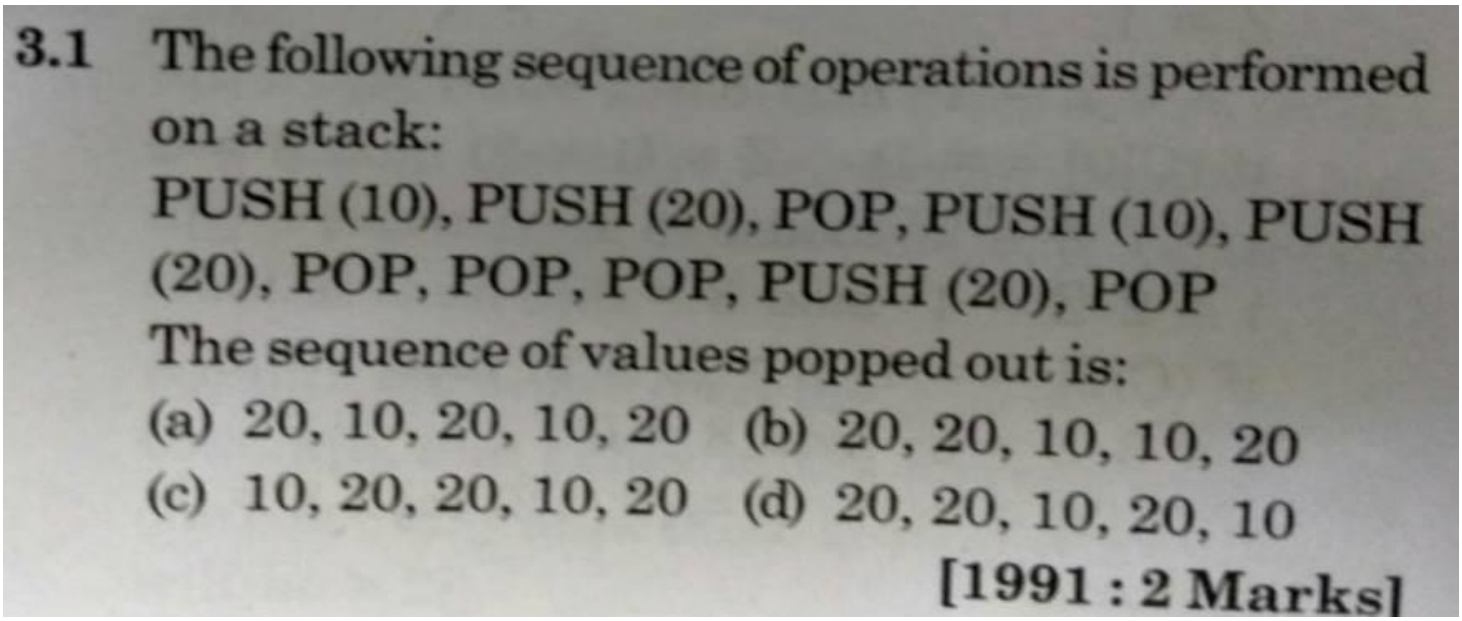
Application of Stack

Q Which one of the following is an application of Stack Data Structure?

(A) Managing function call
(B) recursion
(C) Arithmetic expression evaluation
(D) All of the above

Answer: (D)

Stack Permutation



3.1 The following sequence of operations is performed on a stack:
PUSH (10), PUSH (20), POP, PUSH (10), PUSH (20), POP, POP, POP, PUSH (20), POP
The sequence of values popped out is:
(a) 20, 10, 20, 10, 20 (b) 20, 20, 10, 10, 20
(c) 10, 20, 20, 10, 20 (d) 20, 20, 10, 20, 10
[1991 : 2 Marks]

b

Q if the input sequence is 1, 2, 3, 4, 5 then identify the wrong stack permutation (possible pop sequence)?

a) 3, 5, 4, 2, 1 **b)** 2, 4, 3, 5, 1 **c)** 4, 3, 5, 2, 1 **d)** 5, 4, 3, 1, 2

Q if the input sequence is 5, 4, 3, 2, 1 then identify the wrong stack permutation (possible pop sequence)?

a) 4, 2, 1, 3, 5 **b)** 5, 2, 3, 4, 1 **c)** 4, 5, 1, 2, 3 **d)** 3, 4, 5, 2, 1

3.2 Which of the following permutations can be obtained in the output (in the same order) using a stack assuming that the input is the sequence 1, 2, 3, 4, 5 in that order?

(a) 3, 4, 5, 1, 2 (b) 3, 4, 5, 2, 1

(c) 1, 5, 2, 3, 4 (d) 5, 4, 3, 1, 2

[1994 : 2 Marks]

b

Evaluation of arithmetic expression

- An expression is defined as a number of operands or data items combined using several operators. There are basically three of notation to represent an expression.
- Infix notation: the operator is written in between the operands. e.g. A+B. the reason why this notation is called infix is the place of operator in the expression.
- Prefix notation: In which the operator is written before the operands it is also called as polish notation. e.g. +AB
- Postfix: In the postfix notation the operator are written after the operands, so it is called the postfix notation. It is also known as suffix notation or recursive polish notation. AB+
- Postfix notation is type of notation which is most suitable for a computer to calculate any expression. It is universally accepted notation for designing arithmetic and logical unit (ALU) of the CPU.
- Any expression entered into the computer is first converted into postfix notation, stored in stack and then calculated.

Q Consider an expression a + b * c / d ^ e ^ f * d – c + b, convert it into both prefix and post fix notation?

3.3 The postfix expression for the infix expression A + B * (C + D) / F + D * E is

(a) AB + CD + *F/D+ E* (b) ABCD+*F/+DE*+

(c) A *B + CD/F *DE++ (d) A+*BCD/F* DE++

[1995 : 2 Marks]

b

Q Assume that the operators +, -, × are left associative and ^ is right associative. The order of precedence (from highest to lowest) is ^, x, +, -. The postfix expression corresponding to the infix expression a + b × c – d ^ e ^ f is **(GATE - 2004) (2 Marks)**

(A) abc × + def ^ ^ –

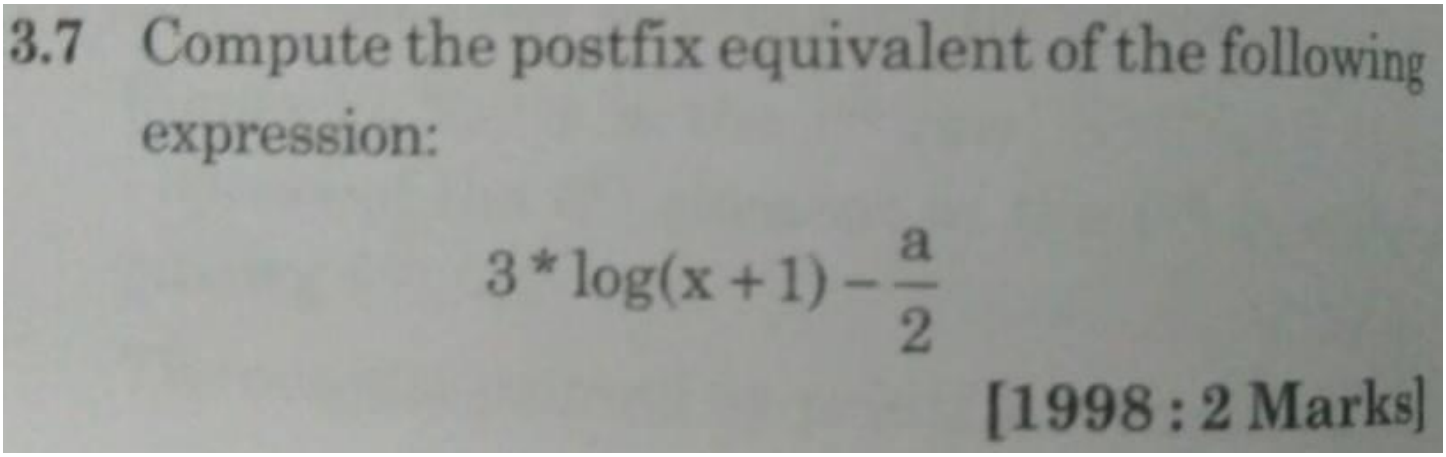
(B) abc × + de ^ f ^ –

(C) ab + c × d – e ^ f ^

(D) – + a × bc ^ ^ def

Answer: (A)

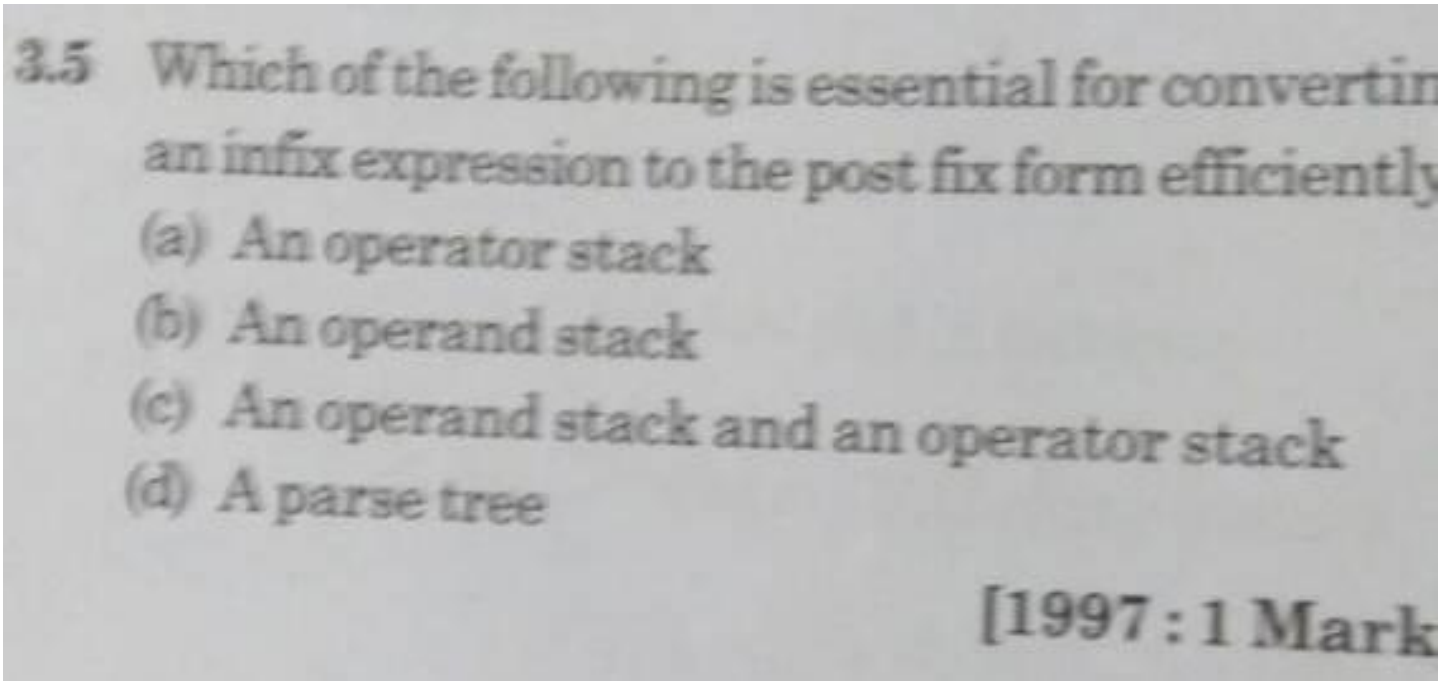
Q Consider an expression log(x!), convert it into both prefix and post fix notation?



3.7 Compute the postfix equivalent of the following expression:

$$3 * \log(x+1) - \frac{a}{2}$$

[1998 : 2 Marks]



3.5 Which of the following is essential for convertin an infix expression to the post fix form efficiently

(a) An operator stack
(b) An operand stack
(c) An operand stack and an operator stack
(d) A parse tree

[1997 : 1 Mark

A

Q The result evaluating the postfix expression 8 2 3 * 1 / + 4 1 * 2 / +

Q The result evaluating the prefix expression + + 8 / * 2 3 1 / * 4 1 2

Q The result evaluating the postfix expression 10 5 + 60 6 / * 8 – is **(GATE - 2015) (1 Marks)**

(A) 284 **(B)** 213 **(C)** 142 **(D)** 71

Ans: c

Q The following postfix expression with single digit operands is evaluated using a stack 8 2 3 ^ / 2 3 * + 5 1 * -
Note that ^ is the exponentiation operator. The top two elements of the stack after the first * is evaluated are: **(GATE - 2007) (2 Marks)**

(A) 6, 1 **(B)** 5, 7 **(C)** 3, 2 **(D)** 1, 5

Answer: (A)

Q To evaluate an expression without any embedded function calls: **(GATE - 2002) (1 Marks)**

(A) One stack is enough

(B) Two stacks are needed

(C) As many stacks as the height of the expression tree are needed

(D) A Turing machine is needed in the general case

Answer: (A)

Q if -, * and $ are used as subtraction, multiplication and exponential.

i) – is highest precedence, then * and the $

ii) all are L to R associative

3 – 2 * 4 $ 1 * 2 $ 3

2 * 2 – 1 $ 1 $ 4 – 2 (all are R to L)

Recursion

- Recursion is one of the most powerful tools in a programming language all it requires is to specify a reasonable condition and instruction that from right logic to solve a problem.
- Recursion is defined as defining anything in terms of itself.
- A function is called recursive if a statement with in the body of a function calls the same function
- Recursion is used to solve problems involving iterations (multiple execution) in reverse order.
- To make a complete function executed repeatedly in terms of itself. Recursion is an alternative to iteration in making a function execute repeatedly.
- Types of recursion: Recursion is of two types depending on weather a function call itself from with in itself than called direct recursion.
- when two function call one another mutually indirect recursion.
- Direct recursion is further of two types tail recursion and head recursion
    - tail when the fun is called in the last, all the other are head recursion

Q Find the output of the following pseudo code?

```
Void main()
{
        fun(4);
}

Void fun(int x)
{
        if (x > 0)
        {
                Printf("%d", x);
                fun(x - 1);
        }
}
```

Q Find the output of the following pseudo code?

```
Void main()
{
        salman(3);
}

Void salman(int x)
{
        if (x > 0)
        {
                Salman(x-1);
                Printf("%d", x);
                Salman(x - 1);
        }
}
```

Q Find the output of the following pseudo code on x = 6?

```
int x(int n)
{
        if (n < 3)
                return 1;
        Else
                return x(n-1) + x(n-1) + 1;
}
```

Q Find the output of the following pseudo code?

```
void Print_array(a, i, j)
{
        if (i = = j)
        {
                printf(''%d', a[i]);
                return;
        }
        Else
        {
                printf(''%d', a[i]);
                print_array(a, i+1, j)
                return x(n-1) + x(n-1) + 1;
        }
}
```

Q Find the output of the following pseudo code?

```
void Print_somthing(a, i, j)
{
        if (i = = j)
        {
                printf(''%d', a[i]);
                return;
        }
        Else
        {
                if(a[i] < a[j])
                        Print_somthing (a, i+1, j);
                Else
                        Print_somthing(a, i, j-1);
        }
}
```

Q Find what this function is doing?

```
 void what (struct Bnode *t)
{
        if (t)
        {
                what(t → LC);
                printf(''%d', t→data);
                what(t → LC);
        }
}
```

Q Find what this function is doing?

```
 void what(struct Bnode *t)
{
        if (t)
        {
                printf(''%d', t→data);
                what(t → LC);
                printf(''%d', t→data);
                what(t → LC);
        }
}
```

Q Find what this function is doing?

```
 void what(struct Bnode *t)
{
        if (t)
        {
                printf(''%d', t→data);
                what(t → LC);
                printf(''%d', t→data);
                what(t → LC);
                printf(''%d', t→data);
        }
}
```

Q Find what this function is doing?

```
void A(struct Bnode *t)
{
        if (t)
        {
                B(t → LC);
                printf(''%d', t→data);
                B(t → LC);
        }
}

void B(struct Bnode *t)
{
        if (t)
        {
                printf(''%d', t→data);
                A(t → LC);
                A(t → LC);
        }
}
```

Excessive recursion: - Does not remember previously estimated value.

fibonacci series

if n==0, then f(n) = 0

if n==1, then f(n) = 1

if n > 1, then f(n-1) + f(n-2)

no of invocation = 2f(n+1) - 1
no of addition = f(n+1) – 1

Tower of hanoi: suppose there are three pegs lablled as A, B and C and suppose on peg A there are placed a finite number n of disks with decreasing size. the object of the game is to move the disks from peg A to peg C using peg B as an auxiliary.

Rules:
only one disk may be moved at a time specially only the top disk on any peg may be involved to any other peg
at no time can a larger disk be placed on smaller disk.

```
Tower(N, B, A, E)
{
        if(n = 1)
        {
                B → E
                return
        }
        tower(n-1, B, E, A)
        B → E
        tower(n-1, A, B, E)
        Return
}
```

total disk moves = 2^n -1
total number of function call = 2^{n+1} -1
how many invocation are required for the first disk to move = n

Ackerman function A(m, n)
if(m = 0), the A(m, n) = n+1
if ((m != 0) && (n=0)), the A(m, n) = A(m-1, 1)
if ((m != 0) && (n != 0)), the A(m, n) = A(m-1, A(m, n-1))

Q Following is C like pseudo code of a function that takes a number as an argument, and uses a stack S to do processing.

```
void fun (int n)
{
```

```
Stack S; // Say it creates an empty stack S
while (n > 0)
{
        // This line pushes the value of n%2 to stack S
        push (&S, n%2);
        n = n/2;
}
// Run while Stack S is not empty
while (!isEmpty(&S))
      printf("%d ", pop(&S)); // pop an element from S and print it

}
```

What does the above function do in general?

(A) Prints binary representation of n in reverse order
(B) Prints binary representation of n
(C) Prints the value of Logn
(D) Prints the value of Logn in reverse order

Answer: (B)

Q What does the following function print for n = 25?

```
void fun(int n)
{

        if (n == 0)
              return;
        printf("%d", n%2);

 fun(n/2);
}
```

(A) 11001 **(B)** 10011 **(C)** 11111 **(D)** 00000

Answer: (B)

Q Consider the following recursive function fun (x, y). What is the value of fun (4, 3)

```
int fun(int x, int y)
{
        if (x == 0)
              return y;
        return fun(x - 1,  x + y);
```

```
}
```

(A) 13 **(B)** 12 **(C)** 9 **(D)** 10

Answer: (A)

Q The output of following program

```
#include <stdio.h>
int fun(int n)
{
        if (n == 4)
                return n;
        else return 2*fun(n+1);
}

 int main()
{
         printf("%d ", fun(2));
         return 0;

}
```

(A) 4 **(B)** 8 **(C)** 16 **(D)** Runtime Error

Answer: (C)

Q What does the following function do?

```
int fun(int x, int y)
{
        if (y == 0)
                return 0;
        return (x + fun(x, y-1));
}
```

(A) x + y **(B)** x + x*y **(C)** x*y **(D)** x^y

Answer: (C)

Q What does the following function do?

```
int fun(unsigned int n)
{
        if (n == 0 || n == 1)
```

```
        return n;
    if (n%3 != 0)
        return 0;
    return fun(n/3);
}
```

(A) It returns 1 when n is a multiple of 3, otherwise returns 0
(B) It returns 1 when n is a power of 3, otherwise returns 0
(C) It returns 0 when n is a multiple of 3, otherwise returns 1
(D) It returns 0 when n is a power of 3, otherwise returns 1
Answer: (B)

Q Output of following program?

```
#include<stdio.h>
void print(int n)
{
    if (n > 4000)
        return;
    printf("%d ", n);
    print(2*n);
    printf("%d ", n);
}

int main()
{
    print(1000);
    getchar();
    return 0;
}
```

(A) 1000 2000 4000
(B) 1000 2000 4000 4000 2000 1000
(C) 1000 2000 4000 2000 1000
(D) 1000 2000 2000 1000
Answer: (B)

Q Predict the output of following program

```
#include <stdio.h>
int f(int n)
{
    if(n <= 1)
        return 1;
```

```
        if(n%2 == 0)
                return f(n/2);
        return f(n/2) + f(n/2+1);
}

 int main()
{
        printf("%d", f(11));
        return 0;
}
```

(A) Stack Overflow **(B)** 3 **(C)** 4 **(D)** 5

Answer: (D)

Q Consider the following recursive C function. If get(6) function is being called in main() then how many times will the get() function be invoked before returning to the main()? (GATE - 2015) (2 Marks)

```
void get (int n)
{

  if (n < 1) return;
  get(n-1);
  get(n-3);
  printf("%d", n);

}
```

(A) 15 **(B)** 25 **(C)** 35 **(D)** 45

Answer: (B)

Q Consider the following recursive C function that takes two arguments

```
unsigned int foo(unsigned int n, unsigned int r)
{
        if (n  > 0)
                return (n%r +  foo (n/r, r ));
        else
                return 0;
        }
```

What is the return value of the function foo when it is called as foo(345, 10) **? (GATE - 2011) (2 Marks)**

(A) 345 **(B)** 12 **(C)** 5 **(D)** 3

Answer: (B)

Q Consider the same recursive C function that takes two arguments

```
unsigned int foo(unsigned int n, unsigned int r)
{
        if (n  > 0)
                return (n%r +  foo (n/r, r ));
        else
                return 0;

}
```

What is the return value of the function foo when it is called as foo(513, 2)? **(GATE - 2011) (2 Marks)**

(A) 9 **(B)** 8 **(C)** 5 **(D)** 2

Answer: (D)

Q What does fun2() do in general?

```
int fun(int x, int y)

{
        if (y == 0)
                return 0;
        return (x + fun(x, y-1));
}

int fun2(int a, int b)
{
        if (b == 0)
                return 1;
        return fun(a, fun2(a, b-1));
}
```

(A) x*y **(B)** x+x*y **(C)** x^y **(D)** y^x

Answer: (C)

3.8 What value would the following function return for the input x = 95?

```
Function fun (x:integer): integer;
Begin
If x > 100 then fun = x – 10
Else fun : = fun(fun (x + 11))
End;
```

(a) 89 (b) 90

(c) 91 (d) 92

[1998 : 2 Marks]

C

Q What is the output of following program?

```
#include <stdio.h>
 void print(int n, int j)
{
        if (j >= n)
                return;
        if (n-j > 0 && n-j >= j)
                printf("%d %d\n", j, n-j);
                print(n, j+1);
}

int main()
{
        int n = 8;
        print(n, 1);

}
```

(A) 1 7 2 6 3 5 4 4 4 4

(B) 1 7 2 6 3 5 4 4

(C) 1 7 2 6 3 5

(D) 1 2 3 4 5 6 7 8

Answer: (B)

Explanation: For a given number n, the program prints all distinct pairs of positive integers with sum equal to n.

Q

```
#include<stdio.h>
void crazy(int n, int a, int b)
{
        if (n <= 0)
                return;
         crazy(n-1, a, b+n);
         printf("%d %d %d\n",n,a,b);
         crazy(n-1, b, a+n);
}

int main()
{
   crazy(3,4,5);
   return 0;
}
```

(A)	(B)	(C)	(D)
1 4 10	3 4 5	1 4 10	3 4 5
2 4 8	1 4 10	2 4 8	1 5 9
1 8 6	2 4 8	1 8 6	2 5 7
3 4 5	1 8 6	3 4 5	1 7 7
1 5 9	1 5 9		
2 5 7	2 5 7		
1 7 7	1 7 7		

Answer: (A)

Q Consider the following C function. **(GATE - 2015) (2 Marks)**

```
int fun (int n)
{
  int x=1, k;
  if (n==1) return x;
  for (k=1; k<n; ++k)
     x = x + fun(k) * fun(n – k);
  return x;
}
```

The return value of fun(5) is _____.

(A) 0 **(B)** 26 **(C)** 51 **(D)** 71

Answer: (C)

Q What is the return value of following function for 484? What does it to in general?

```
bool fun(int n)
{
   int sum = 0;
   for (int odd = 1; n > sum; odd = odd+2)
         sum = sum + odd;
   return (n == sum);

}
```

(A) False, it checks whether a given number is power of 3
(B) False, it checks whether a given number is even or not
(C) False, it checks whether a given number is odd or not
(D) True, it checks whether a given number is perfect square.

Answer: (D)

Explanation: The given function adds all odd numbers 1, 3, 5, 7, 9, 11…. till the sum is smaller than n. If the sum becomes equal to n, then it returns true. This is basically a test for perfect square numbers. All perfect square numbers can be written as sum of odd numbers.

4 = 1 + 3
9 = 1 + 3 + 50
16 = 1 + 3 + 5 + 7
36 = 1 + 3 + 5 + 7 + 9
49 = 1 + 3 + 5 + 7 + 9 + 11

Q Following is an incorrect pseudocode for the algorithm which is supposed to determine whether a sequence of parentheses is balanced:

```
declare a character stack
while (more input is available)
{
   read a character
   if (the character is a '(')
         push it on the stack
   else if (the character is a ')' and the stack is not empty)
         pop a character off the stack
   else
         print "unbalanced" and exit
}
```

print "balanced"

Which of these unbalanced sequences does the above code think is balanced?

(A) ((()) **(B)** ())(() **(C)** (()())) **(D)** ((()))()

Q Which of the following is not a backtracking algorithm?

(A) Knight tour problem
(B) N queen problem
(C) Tower of Hanoi
(D) M coloring problem

Answer: (C)

Queue

- A queue is a linear list of elements in which deletions can take place only at one end called the front, and insertions can take place only at the end called rear.
- Queue is a first in first out types of data structure(FIFO), the terms 'Front' and 'Rear' are used in describing a linear list only when it is implemented as a queue.

- In computer science queue are used in multiple places e.g. in time sharing system program with the same priority from a queue waiting to be executed.
- A queue is a non-primitive linear data structure. it is homogeneous collection of elements.

Representation of Queues

- Mostly each of our queues will be maintained by a linear array QUEUE and two pointer variables: FRONT containing the location of the Front element of the queue and REAR, containing the location of the rear element of the queue.
- The condition FRONT = null will indicate that the queue is empty. Whenever an element is deleted from the queue, the value of FRONT is increased by 1
    - Front = Front + 1
- Whenever an element is added to the queue, the value of REAR is increased by 1
    - REAR = REAR + 1
- This means that after N insertion the rear element of the queue will occupy QUEUE[N] or queue will occupy the last part of the array. This may occur even though the queue itself may not contain many elements.
- Total number of elements in a queue
    - Rear – Front + 1

Insertion

```
Q_insert (QUEUE, N, F, R, ITEM)
{
        if (R == N - 1)
                Write over flow and exit
        if (F = = -1)
                Set F = 0 && R = 0
        Else
                R = R + 1
        Queue[R] = ITEM
}
```

Deletion

```
Q_insert (QUEUE, N, F, R, ITEM)
{


        if (F == - 1)
                Write under flow and exit
        ITEM = QUEUE[F]
        if (F = = R)
                Set F = -1 && R = -1
        Else
                F = F + 1
}
```

Circular Queue

Insertion

```
Q_insert (QUEUE, N, F, R, ITEM)
{
        if ((F==0 && R==N-1) :: (F == R – 1))
                Write over flow and exit
        if (F = = -1)
                Set F = 0 && R = 0
        Else if (R = = N-1)
                Set R = 0
        Else
                R = R + 1
        Queue[R] = ITEM
}
```

Deletion

```
Q_insert (QUEUE, N, F, R, ITEM)
{
        if (F == - 1)
                Write under flow and exit
        ITEM = QUEUE[F]
        if (F = = R)

        Set F = -1 && R = -1
        Else if (F = N-1)
                Set F = 0
        Else
                F = F + 1
        Return item
}
```

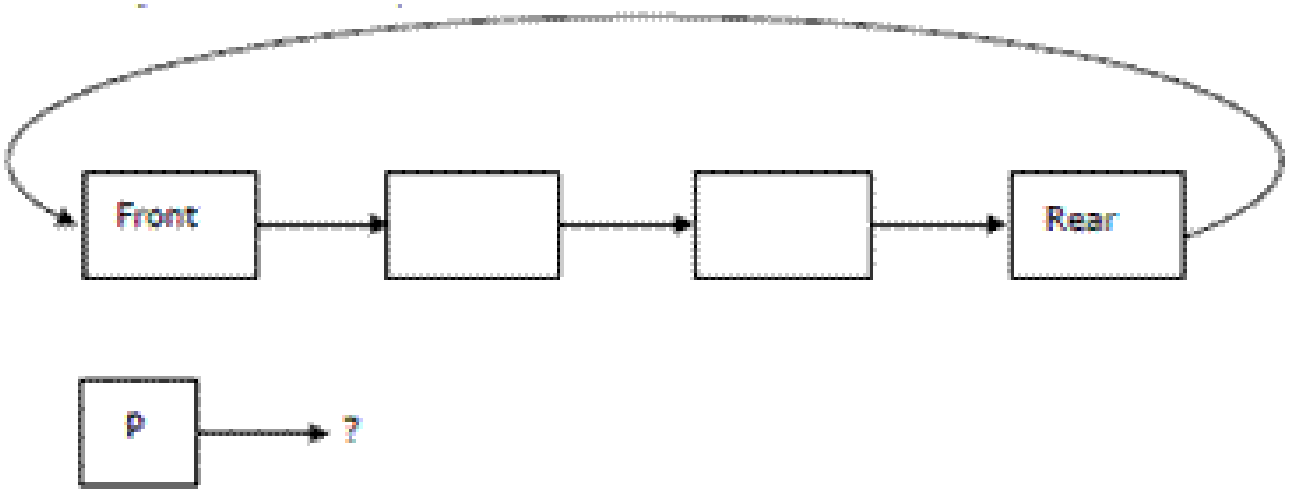
Priority Queue

- A priority queue is a collection of elements such that each element has been assigned a priority and such that the order in which elements are deleted and processed comes from the following rules.
  - An element of higher priority is processed before any element of lower priority
  - Two element with the same priority are processed according to the order in which they were added to the queue.

Q Which one of the following is an application of Queue Data Structure?
(A) When a resource is shared among multiple consumers.
(B) When data is transferred asynchronously (data not necessarily received at same rate as sent) between two processes
(C) Load Balancing
(D) All of the above
Answer: (D)

Q Which of the following is true about linked list implementation of queue?
(A) In push operation, if new nodes are inserted at the beginning of linked list, then in pop operation, nodes must be removed from end.
(B) In push operation, if new nodes are inserted at the end, then in pop operation, nodes must be removed from the beginning.
(C) Both of the above
(D) None of the above
Answer: (C)

Q A circularly linked list is used to represent a Queue. A single variable p is used to access the Queue. To which node should p point such that both the operations enQueue and deQueue can be performed in constant time? **(GATE - 2004) (2 Marks)**



(A) rear node
(B) front node
(C) not possible with a single pointer
(D) node next to front
Answer: (A)

Q Suppose a circular queue of capacity (n – 1) elements is implemented with an array of n elements. Assume that the insertion and deletion operation are carried out using REAR and FRONT as array index variables, respectively. Initially, REAR = FRONT = 0. The conditions to detect queue full and queue empty are **(GATE-2012) (2 Marks)**

a) Full: (REAR+1) mod n == FRONT, empty: REAR == FRONT
b) Full: (REAR+1) mod n == FRONT, empty: (FRONT+1) mod n == REAR
c) Full: REAR == FRONT, empty: (REAR+1) mod n == FRONT
d) Full: (FRONT+1) mod n == REAR, empty: REAR == FRONT
ANSWER A

Q Following is C like pseudo code of a function that takes a Queue as an argument, and uses a stack S to do processing.

```
void fun(Queue *Q)
{
        Stack S;  // Say it creates an empty stack S
        // Run while Q is not empty
        while (!isEmpty(Q))
        {
                // deQueue an item from Q and push the dequeued item to S
                push(&S, deQueue(Q));
```

```
        }
        // Run while Stack S is not empty
        while (!isEmpty(&S))
        {

                // Pop an item from S and enqueue the poppped item to Q
                enQueue(Q, pop(&S));

        }
}
```

What does the above function do in general?

(A) Removes the last from Q
(B) Keeps the Q same as it was before the call
(C) Makes Q empty
(D) Reverses the Q

Answer: (D)

Q Suppose you are given an implementation of a queue of integers. The operations that can be performed on the queue are:
i. isEmpty (Q) — returns true if the queue is empty, false otherwise.
ii. delete (Q) — deletes the element at the front of the queue and returns its value.
iii. insert (Q, i) — inserts the integer i at the rear of the queue.

Consider the following function:

```
void f (queue Q)
{
        int i ;
        if (!isEmpty(Q))
        {

                i = delete(Q);
                f(Q);
                insert(Q, i);

        }
}
```

What operation is performed by the above function f? **(GATE - 2007) (2 Marks)**

(A) Leaves the queue Q unchanged

(B) Reverses the order of the elements in the queue Q

(C) Deletes the element at the front of the queue Q and inserts it at the rear keeping the other elements in the same order

(D) Empties the queue Q

Answer: (B)

Q Suppose implementation supports an instruction REVERSE, which reverses the order of elements on the stack, in addition to the PUSH and POP instructions. Which one of the following statements is TRUE with respect to this modified stack? **(GATE - 2014) (2 Marks)**

a) A queue cannot be implemented using this stack.

b) A queue can be implemented where ENQUEUE takes a single instruction and DEQUEUE takes a sequence of two instructions.

c) A queue can be implemented where ENQUEUE takes a sequence of three instructions and DEQUEUE takes a single instruction.

d) A queue can be implemented where both ENQUEUE and DEQUEUE take a single instruction each.

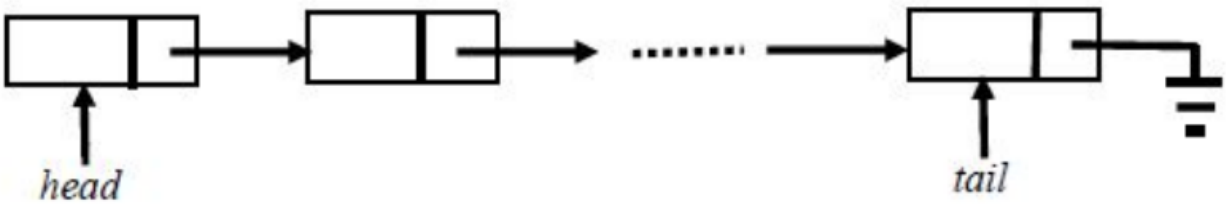
Answer c

Q How many stacks are needed to implement a queue. Consider the situation where no other data structure like arrays, linked list is available to you.

(A) 1 **(B)** 2 **(C)** 3 **(D)** 4

Answer: (B)

Q A queue is implemented using a non-circular singly linked list. The queue has a head pointer and a tail pointer, as shown in the figure. Let n denote the number of nodes in the queue. Let 'enqueue' be implemented by inserting a new node at the head, and 'dequeue' be implemented by deletion of a node from the tail.



Which one of the following is the time complexity of the most time-efficient implementation of 'enqueue' and 'dequeue, respectively, for this data structure? **(GATE - 2018) (2 Marks)**

a) Θ(1), Θ(1) **b)** Θ(1), Θ(n) **c)** Θ(n), Θ(1) **d)** Θ(n), Θ(n)

ANSWER-B

Q A Circular queue has been implemented using singly linked list where each node consists of a value and a pointer to next node. We maintain exactly two pointers FRONT and REAR pointing to the front node and rear node of queue. Which of the following statements is/are correct for circular queue so that insertion and deletion operations can be performed in O(1) i.e. constant time. **(GATE - 2017) (2 Marks)**

I. Next pointer of front node points to the rear node.

II. Next pointer of rear node points to the front node.

a) I only **b)** II only **c)** Both I and II **d)** Neither I nor II

ANSWER-B

Q A queue is implemented using an array such that ENQUEUE and DEQUEUE operations are performed efficiently. Which one of the following statements is CORRECT (n refers to the number of items in the queue)? **(GATE - 2016) (1 Marks)**

a) Both operations can be performed in O(1) time

b) At most one operation can be performed in O(1) time but the worst case time for the other operation will be Ω(n)

c) The worst case time complexity for both operations will be Ω(n)

d) Worst case time complexity for both operations will be Ω(log n)

ANSWER-A

Q Let Q denote a queue containing sixteen numbers and S be an empty stack. Head(Q) returns the element at the head of the queue Q without removing it from Q. Similarly, Top(S) returns

the element at the top of S without removing it from S. Consider the algorithm given below. **(GATE - 2016) (2 Marks)**

```
while Q is not Empty do
    if S is Empty OR Top(S) ≤ Head(Q) then
        x := Dequeue(Q);
        Push(S,x);
    else
        x := Pop(S);
        Enqueue(Q,x);
    end
end
```

The maximum possible number of iterations of the while loop in the algorithm is
Ans: 256

Q An implementation of a queue Q, using two stacks S1 and S2, is given below:

```
void insert(Q, x)
{
        push (S1, x);
}
void delete(Q)
{
        if(stack-empty(S2)) then
        if(stack-empty(S1)) then
        {
                print("Q is empty");
                return;
        }
        else while (!(stack-empty(S1)))
        {
```

```
            x=pop(S1);
            push(S2,x);

        }
        x=pop(S2);

}
```

Let n insert and m (<=n) delete operations be performed in an arbitrary order on an empty queue Let x and y be the number of push and pop operations performed respectively in the process. Which one of the following is true for all m and n? **(GATE-2006) (1 Marks)**

(A) n+m <= x < 2n and 2m <= y <= n+m

(B) n+m <= x < 2n and 2m<= y <= 2n

(C) 2m <= x < 2n and 2m <= y <= n+m

(D) 2m <= x <2n and 2m <= y <= 2n

Answer: (A)

Q A priority queue Q is used to implement a stack S that stores characters. PUSH(C) is implemented as INSERT(Q, C, K) where K is an appropriate integer key chosen by the implementation. POP is implemented as DELETEMIN(Q). For a sequence of operations, the keys chosen are in **(GATE - 1997) (2 Marks)**

(A) Non-increasing order

(B) Non-decreasing order

(C) strictly increasing order

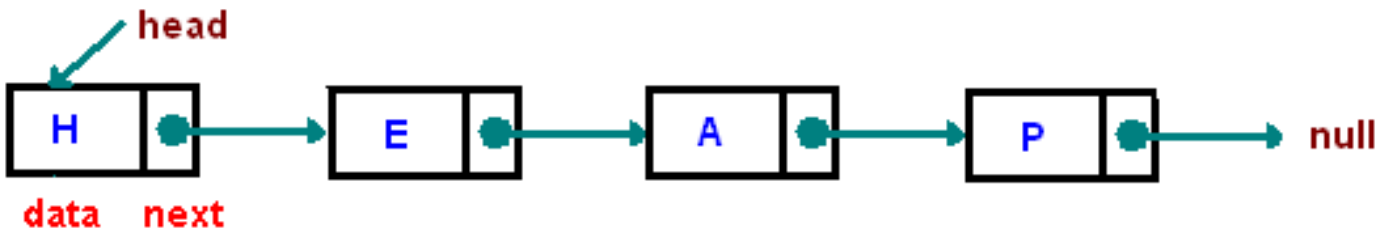
(D) strictly decreasing order

Answer: (D)

- Problem with array / stack / link list
    - One disadvantage of using arrays to store data is that, arrays are static structures and therefore cannot be easily extended or reduced to fit the data set (suffer from both internal and external fragmentation).
- Arrays are also expensive to maintain new insertions and deletions.

- Solution is link list

    - A linked list is a linear collection of data elements called nodes, where the linear order is given by the means of pointers.
    - Each node is divided into two parts the first part is the information part of the node, which contains data.
    - The second part called linked field or next pointer field, contains the address of the next node of the list.
    - The pointer of the last node contains a null pointer, which is an invalid address (0 or negative value).
    - The linked also contains a list pointer variable called start/first/head which contain the address of the first node in the list.
    - A special case is the list that has no nodes, such a list is called null list or empty list and is denoted by a null pointer in the variable start/first/head.



Implementation of link list

```
struct node {

   int data;

   struct node *next;

};
```

Advantage of link list

- Do not suffer from internal and external fragmentation.
- A linked list is a dynamic data structure. The number of nodes in a list is not fixed and can grow and shrink on demand. Can easily support insertion or deletion.

Disadvantage of link list

- Slow access: - It does not allow direct search, supports only sequential search even if link list is sorted. i.e. If we want to access a particular node then we have to start from the first node.
- A good amount of space is wasted in storing pointers. For e.g. 4 bytes (on 32-bit CPU) to store a reference to the next node.

Operation on link list

Traversing a link list means starting from the first node and read or process each node os the list exactly once

Traversing a link list iteratively

```
Void main()
{
        Traverse(head )
}

void traverse (Node* head)
{
        Node *ptr = head;
        while(ptr != NULL)
        {
                Printf("%d", ptr-> data):
                ptr = ptr->next;
        }
}
```

Traversing a link list recursively

```
Void main()
{
        Traverse(head )
}
void traverse (node *head)
{
         Node *ptr = head;
         Printf("%d", ptr-> data):
        traverse(ptr->next);
}
```

Searching a data item in a link list iterative

```
Void main()
{
        Traverse(*head, key)
}

void traverse (Node *head, key)
{
        Node *ptr = head;
        while(ptr != NULL)
        {
                If(ptr-> data = = key)
                {
                        Return ptr;
                }
                else
                        ptr = ptr->next;
        }
}
```

Searching a data item in a sorted link list(increasing order) iterative

```
Void main()
{
        Traverse(*head, key)
}

void traverse (Node *head, int key)
{
        Node *ptr = head;
        while(ptr != NULL)
        {
                If(ptr-> data = = key)
                {
                        Return ptr;
                }
                Else if(ptr-> data < key)
                {
                                ptr = ptr->next;
                }
                Else
                {
                        Return null;
                }
        }
}
```

Searching a data item in a link list recursive

```
Void main()
{
        Traverse(*head, key )
}

void traverse (node *head, int key)
{
         Node *ptr = head;
         If(ptr-> data = = key)
        {
                Return ptr;
        }
        Else
        {
                traverse(ptr->next, key)
        }
}
```

Searching a data item in a sorted link list (increasing order)recursive

```
Void main()

{
        Traverse(*head, key )
}

void traverse (node *head, int key)
{
         Node *ptr = head;
         If(ptr-> data = = key)
        {
                Return ptr;
        }
        Else If(ptr-> data < key)
        {
                traverse(ptr->next, key)
        }

        Else
        {
                Return null;
        }
}
```

Insertion in the starting of link list

```
Void main()
{
        Insert(*head, int key)
}

Void insert(node *head, int key)
{
        node *Ptr = Create a node();
        ptr->next = head->next;
        head =ptr:
}
```

Insertion after a location in the link list

```
Void main()
{
        Insert(*head, int key, location)
}

Void insert(node *head, int key)
{
        node *Ptr = Create a node();
        ptr->next = location->next;
        location->next = ptr;
}
```

Deletion first node of the link list

```
Void main()
{
        delete(*head)
}

int delete(node *head)
{
        If(head == null)
        {
                Prinf("underflow");
        }
        Item = head->data;
        head = head->next;
        return item;
}
```

Deletion a node after a location in the link list

```
Void main()
{
        delete(*location);
}

int delete(node *location)
{
        If(location == null)
        {
                Prinf("underflow");
        }
        Item = location->next->data;
        Location->next = location->next->next;
        return item;
}
```

Reversal of link list iterative

```
Void main()
{
        reverse(struct node* head)
}

Void reverse(struct node* head)
{
        Struct node *p = head;
        Struct node *q = null;
        Struct node *r;
        While(p !=null)
        {
                r=q;
                q=r;
                p=p->next;
                q->next = r;
        }
        head = q;
}
```

Reversal of link list recursive

```
Void main()
{
        reverse(null, head)
}

Void reverse(struct node* previous, current)
{
        If(current != null)
        {
                Reverse(current, current->next);
                Current->link = previous;
        }
        Else
        {
                Head = previous;
        }
}
```

Q The following C function takes a single-linked list of integers as a parameter and rearranges the elements of the list. The function is called with the list containing the integers 1, 2, 3, 4, 5, 6, 7 in the given order. What will be the contents of the list after the function completes execution? **(GATE-2008) (2 Marks)**

```
struct node
{
  int value;
  struct node *next;
};
void rearrange(struct node *list)
{
  struct node *p, * q;
  int temp;
  if ((!list) || !list->next)
     return;
  p = list;
  q = list->next;
  while(q)
  {
    temp = p->value;
    p->value = q->value;
    q->value = temp;
    p = q->next;
    q = p?p->next:0;
  }

}
```

(A) 1,2,3,4,5,6,7 **(B)** 2,1,4,3,6,5,7 **(C)** 1,3,2,5,4,7,6 **(D)** 2,3,4,5,6,7,1

Answer: (B)

Q The following C function takes a simply-linked list as input argument. It modifies the list by moving the last element to the front of the list and returns the modified list. Some part of the code is left blank. Choose the correct alternative to replace the blank line. (GATE-2010) (2 Marks)

```
typedef struct node
{
  int value;
  struct node *next;
}Node;
```

```
Node *move_to_front(Node *head)
{
  Node *p, *q;
  if ((head == NULL: || (head->next == NULL))
    return head;
  q = NULL; p = head;
  while (p-> next !=NULL)
  {
    q = p;
    p = p->next;
  }
  _____
  return head;

}
```

(A) q = NULL; p->next = head; head = p; **(B)** q->next = NULL; head = p; p->next = head;
(C) head = p; p->next = q; q->next = NULL; **(D)** q->next = NULL; p->next = head; head = p;
Answer: (D)

Q Consider the function f defined below. (GATE-2003) (2 Marks)

```
struct item

{
  int data;
  struct item * next;
};

int f(struct item *p)
{
  return (
        (p == NULL) ||
        (p->next == NULL) ||
        (( P->data <= p->next->data) && f(p->next))
       );

}
```

For a given linked list p, the function f returns 1 if and only if
(A) the list is empty or has exactly one element
(B) the elements in the list are sorted in non-decreasing order of data value

(C) the elements in the list are sorted in non-increasing order of data value
(D) not all elements in the list have the same data value.
Answer: (B)

Q Consider the following function that takes reference to head of a Doubly Linked List as parameter. Assume that a node of doubly linked list has previous pointer as *prev* and next pointer as *next*.

```
void fun(struct node **head_ref)
{
   struct node *temp = NULL;
   struct node *current = *head_ref;

   while (current !=  NULL)
   {
      temp = current->prev;
      current->prev = current->next;
      current->next = temp;
      current = current->prev;
   }

   if(temp != NULL )
      *head_ref = temp->prev;
}
```

Assume that reference of head of following doubly linked list is passed to above function

1 <--> 2 <--> 3 <--> 4 <--> 5 <-->6.

What should be the modified linked list after the function call?

(A) 2 <--> 1 <--> 4 <--> 3 <--> 6 <-->5
(B) 5 <--> 4 <--> 3 <--> 2 <--> 1 <-->6.
(C) 6 <--> 5 <--> 4 <--> 3 <--> 2 <--> 1.
(D) 6 <--> 5 <--> 4 <--> 3 <--> 1 <--> 2
Answer: (C)

Q Suppose each set is represented as a linked list with elements in arbitrary order. Which of the operations among union, intersection, membership, cardinality will be the slowest? (Gate-2004) (2 Marks)

(A) union only
(B) intersection, membership
(C) membership, cardinality
(D) union, intersection

Answer: (D)

Q Consider the C code fragment given below. **(GATE-2017) (1 Marks)**

```
typedef struct node
{
        int data;
        node* next;
} node;

void join(node* m, node* n)
{
      node* p = n;
      while (p->next != NULL)
      {
            p = p->next;
      }
            p–>next = m;
}
```

Assuming that m and n point to valid NULL- terminated linked lists, invocation of join will

a) append list m to the end of list n for all inputs
b) either cause a null pointer dereference or append list m to the end of list n
c) cause a null pointer dereference for all inputs.
d) append list n to the end of list m for all inputs.

ANSWER- D

Q N items are stored in a sorted doubly linked list. For a delete operation, a pointer is provided to the record to be deleted. For a decrease-key operation, a pointer is provided to the record on which the operation is to be performed. An algorithm performs the following operations on the list in this order: Θ(N) delete, O(log N) insert, O(log N) find, and Θ(N) decrease-key What is the time complexity of all these operations put together (Gate-2016) (2 Marks)

ANSWER- C

Q Consider the following statements: **(GATE - 1996) (2 Marks)**

i) First-in-first out types of computations are efficiently supported by STACKS.

ii) Implementing LISTS on linked lists is more efficient than implementing LISTS on an array for almost all the basic LIST operations.

iii) Implementing QUEUES on a circular array is more efficient than implementing QUEUES on a linear array with two indices.

iv) Last-in-first-out type of computations are efficiently supported by QUEUES.

a) (ii) and (iii) are true

b) (i) and (ii) are true

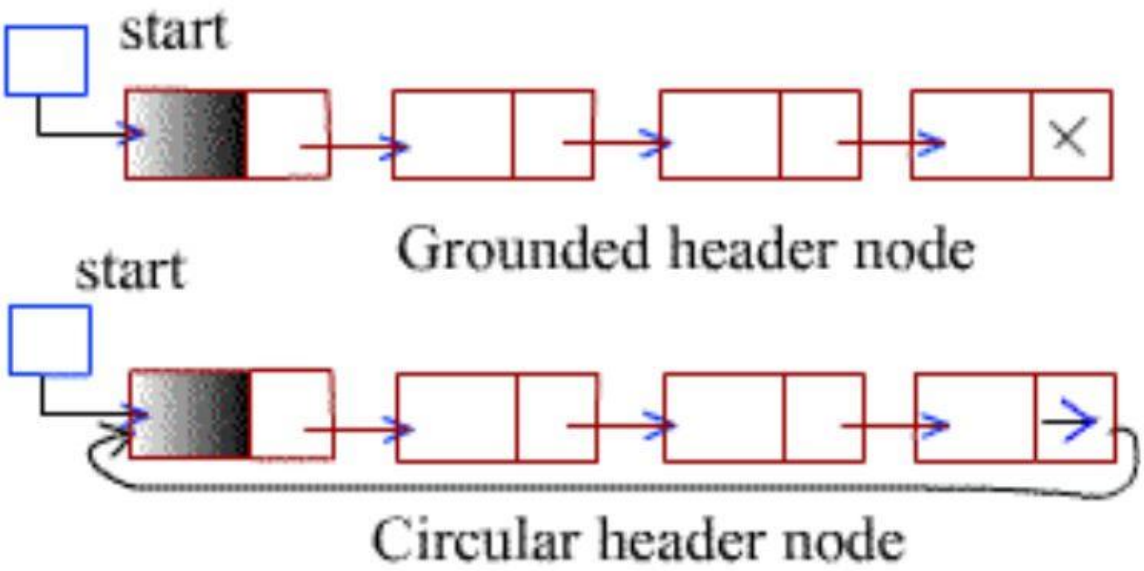
c) (iii) and (iv) are true

d) (ii) and (iv) are true

Ans: a

Header link list

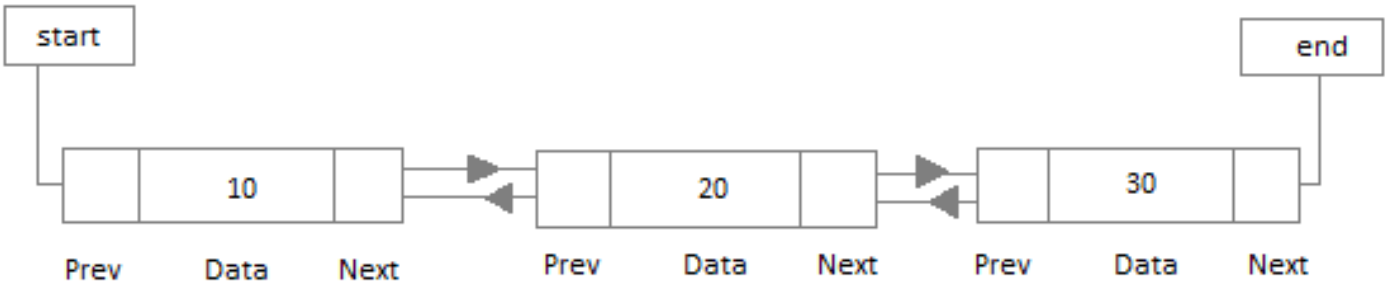
- A header linked list is a linked list which always contains a special node, called head node at the beginning of the list. This special node in general used to store number of nodes present in the linked list or some other meta data



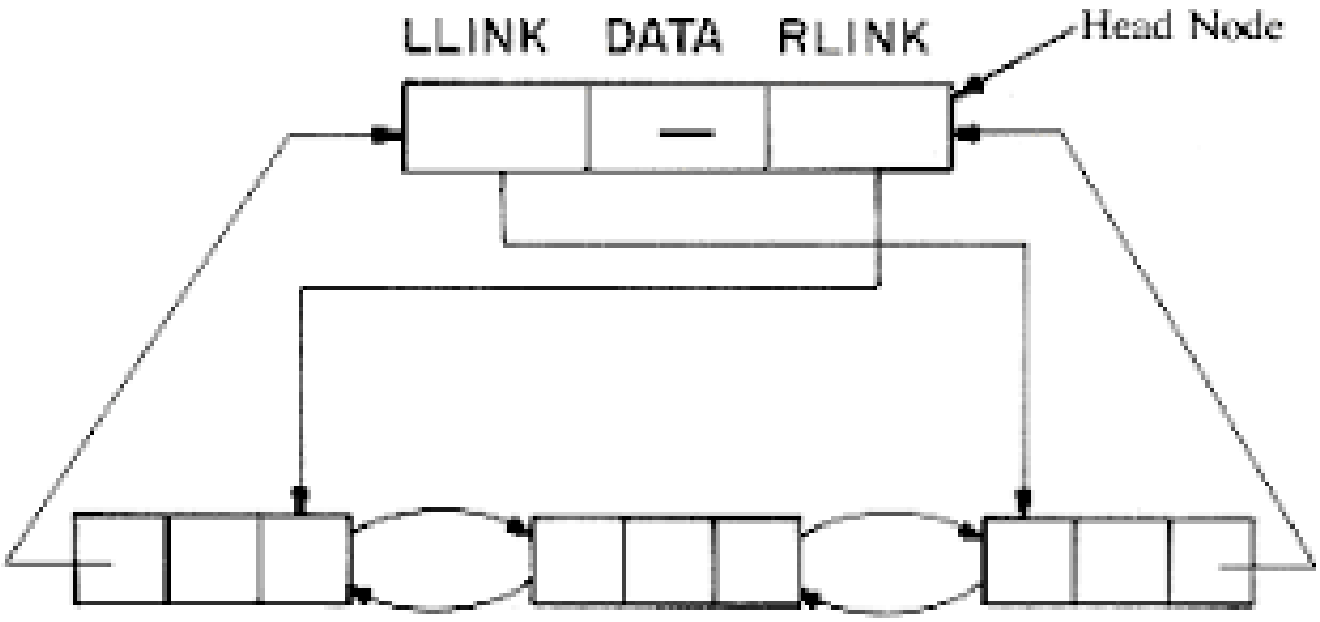
- Header circular link list are frequently used instead of ordinary linked list, because many operations are much easier to state and implement using header list.
- Null pointer is not used and hence all pointer contains valid address.
    - loop condition while(ptr != start)

Doubly link list

- The list which can be traversed in two directions: in the unusual forward direction from the beginning of the list to the ends or in the end direction from the end of the list to the beginning of the list.
- Furthermore given the location LOC of a node N in the list, one now has immediate access to both the next node and the preceding node of the list. This means in particular that one is able to delete N from the list without traversing any part of the list.



Header circular doubly link list



Q Which of the following points is/are true about Linked List data structure when it is compared with array
(A) Arrays have better cache locality that can make them better in terms of performance.
(B) It is easy to insert and delete elements in Linked List
(C) Random access is not allowed in a typical implementation of Linked Lists
(D) All of the above
Answer: (d)

Q In the worst case, the number of comparisons needed to search a singly linked list of length n for a given element is
(A) $\log_2 n$ **(B)** $n/2$ **(C)** $\log_2 n - 1$ **(D)** n
Answer: (D)

Q What does the following function do for a given Linked List with first node as *head*?

```
void fun1(struct node* head)
{
 if(head == NULL)
   return;

 fun1(head->next)
 printf("%d  ", head->data);

}
```

(A) Prints all nodes of linked lists **(B)** Prints all nodes of linked list in reverse order
(C) Prints alternate nodes of Linked List **(D)** Prints alternate nodes in reverse order
Answer: (B)

Q What is the output of following function for start pointing to first node of following linked list?

1->2->3->4->5->6

```
void fun(struct node* start)
{
 if(start == NULL)
   return;
 printf("%d  ", start->data);
```

```
  if(start->next != NULL )
    fun(start->next->next);
  printf("%d  ", start->data);

}
```

(A) 1 4 6 6 4 1 **(B)** 1 3 5 1 3 5 **(C)** 1 2 3 5 **(D)** 1 3 5 5 3 1

Answer: (D)

Q The following function reverse() is supposed to reverse a singly linked list. There is one line missing at the end of the function.

```
/* Link list node */
struct node
{
   int data;
   struct node* next;
};

/* head_ref is a double pointer which points to head (or start) pointer
  of linked list */
static void reverse(struct node** head_ref)
{
   struct node* prev   = NULL;
   struct node* current = *head_ref;
   struct node* next;
   while (current != NULL)
   {
      next  = current->next;
      current->next = prev;
      prev = current;
      current = next;
   }
   /*ADD A STATEMENT HERE*/

}
```

What should be added in place of "/*ADD A STATEMENT HERE*/", so that the function correctly reverses a linked list.

(A) *head_ref = prev; **(B)** *head_ref = current;

(C) *head_ref = next; **(D)** *head_ref = NULL;

Answer: (A)

Q What are the time complexities of finding 8th element from beginning and 8th element from end in a singly linked list?

Let n be the number of nodes in linked list, you may assume that n > 8.
(A) O(1) and O(n) **(B)** O(1) and O(1) **(C)** O(n) and O(1) **(D)** O(n) and O(n)
Answer: (A)

Q Given pointer to a node X in a singly linked list. Only one pointer is given, pointer to head node is not given, can we delete the node X from given linked list?
(A) Possible if X is not last node. Use following two steps (a) Copy the data of next of X to X. (b) Delete next of X.
(B) Possible if size of linked list is even.
(C) Possible if size of linked list is odd
(D) Possible if X is not first node. Use following two steps (a) Copy the data of next of X to X. (b) Delete next of X.
Answer: (A)

Q You are given pointers to first and last nodes of a singly linked list, which of the following operations are dependent on the length of the linked list?
(A) Delete the first element **(B)** Insert a new element as a first element
(C) Delete the last element of the list **(D)** Add a new element at the end of the list
Answer: (C)

Q Consider the following function to traverse a linked list.

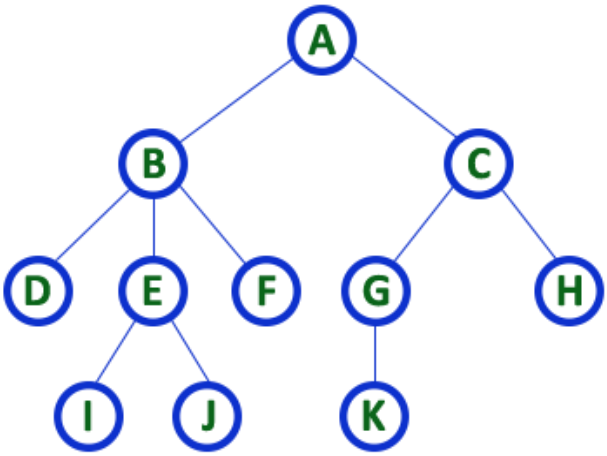
```
void traverse(struct Node *head)
{
  while (head->next != NULL)
  {
     printf("%d  ", head->data);
     head = head->next;
  }

}
```

Which of the following is **FALSE** about above function?
(A) The function may crash when the linked list is empty
(B) The function doesn't print the last node when the linked list is not empty
(C) The function is implemented incorrectly because it changes head
(D) none
Answer: (C)

Tree

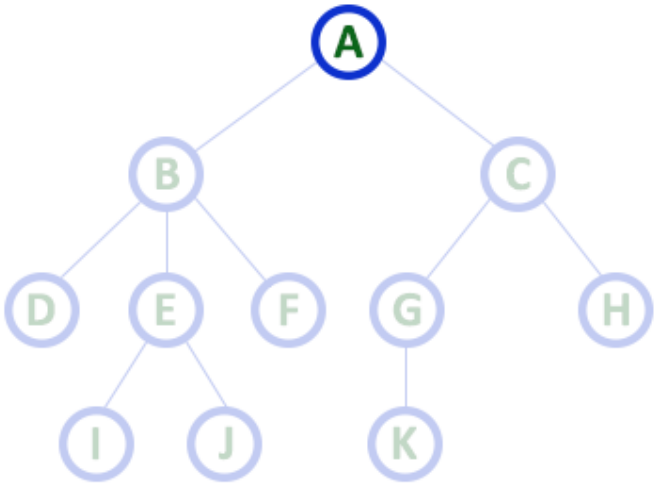
- The tree is one of the most powerful, flexible, versatile and nonlinear advanced data structures, it represents hierarchical relationship existing between several data items. it is used in wide range of applications.

- A tree is a finite set of one or more data items(nodes) such that
    - There is a special data item called root of the tree
    - And its remaining data items are partitioned into number of mutually exclusive (disjoint) subsets, each of which is itself a tree and they are called subtree. i.e. Every node (exclude a root) is connected by a directed edge *from* exactly one other node; A direction is: *parent -> children*



Root

- The first/Top most node is called as Root Node. We always have exactly one root node in every tree. We can say that root node is the origin of tree data structure.

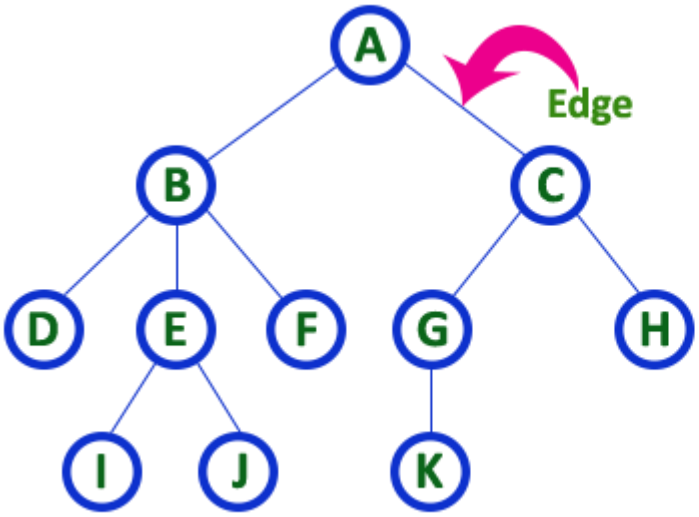


Here 'A' is the 'root' node

- In any tree the first node is called as ROOT node

Edge

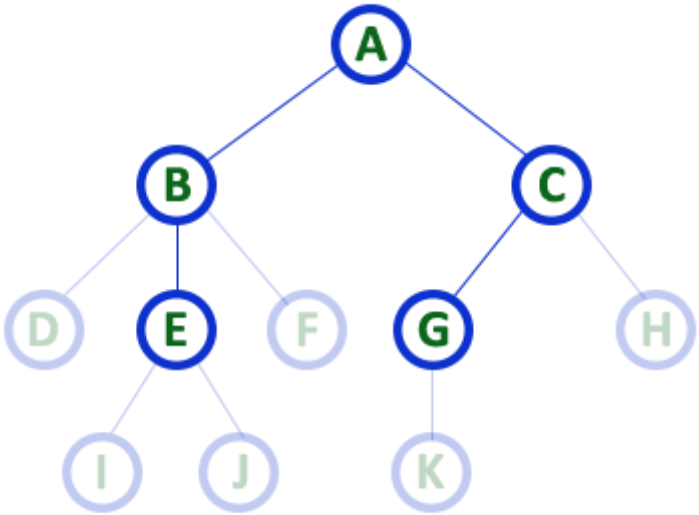
- In a tree data structure, the connecting link between any two nodes is called as EDGE. In a tree with 'N' number of nodes there will be exactly of 'N-1' number of edges.



- In any tree, 'Edge' is a connecting link between two nodes.

Parent / Ascendant / Predecessor / Internal / Non-leaf

- In a tree data structure, the node which is predecessor of any node is called as PARENT NODE. In simple words, the node which has branch from it to any other node is called as parent node. Parent node can also be defined as "The node which has child / children".

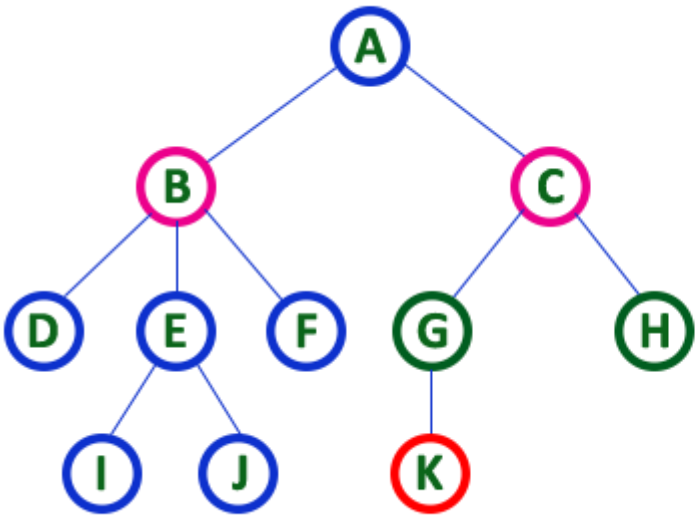


Here A, B, C, E & G are Parent nodes

- In any tree the node which has child / children is called 'Parent'
- A node which is predecessor of any other node is called 'Parent'

Child

- In a tree data structure, the node which is descendant of any node is called as CHILD Node. In simple words, the node which has a link from its parent node is called as child node. In a tree, any parent node can have any number of child nodes. In a tree, all the nodes except root are child nodes.

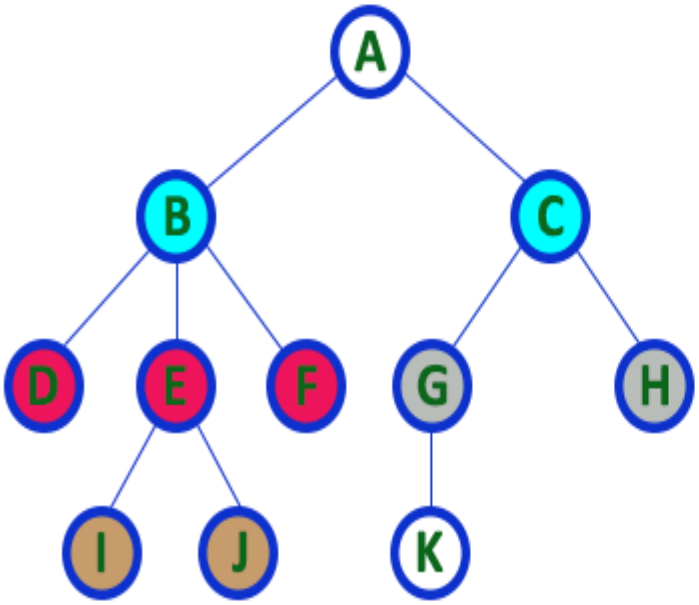


Here B & C are Children of A
Here G & H are Children of C
Here K is Child of G

- descendant of any node is called as CHILD Node

Siblings

- In a tree data structure, nodes which belong to same Parent are called as SIBLINGS. In simple words, the nodes with same parent are called as Sibling nodes.

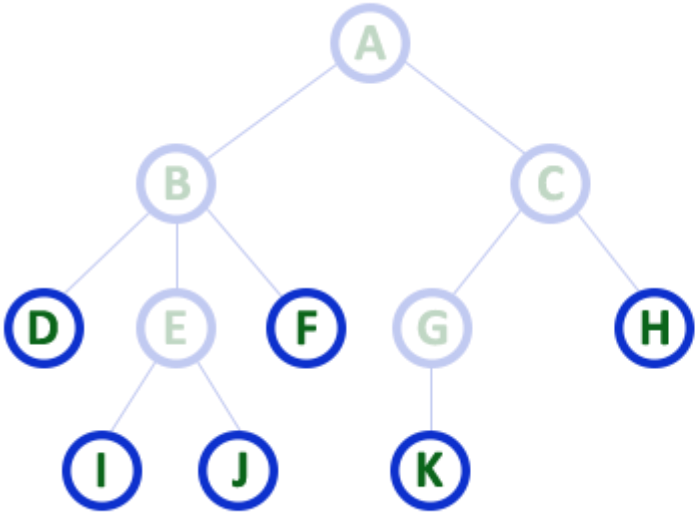


Here B & C are Siblings
Here D E & F are Siblings
Here G & H are Siblings
Here I & J are Siblings

- In any tree the nodes which has same Parent are called 'Siblings'
- The children of a Parent are called 'Siblings'

Leaf

- In a tree data structure, the node which does not have a child is called as LEAF Node. In simple words, a leaf is a node with no child.
- In a tree data structure, the leaf nodes are also called as External Nodes. External node is also a node with no child. In a tree, leaf node is also called as 'Terminal' node.

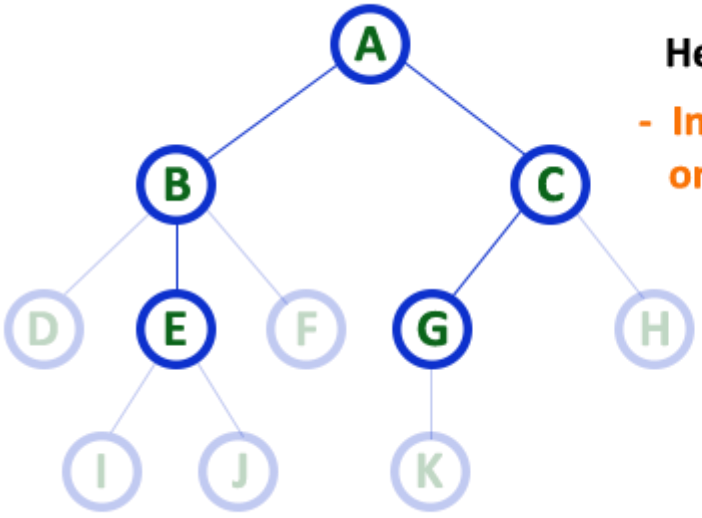


Here D, I, J, F, K & H are Leaf nodes

- **In any tree the node which does not have children is called 'Leaf'**
- **A node without successors is called a 'leaf' node**

Internal Nodes

- In a tree data structure, the node which has at least one child is called as INTERNAL Node. In simple words, an internal node is a node with at least one child.
- In a tree data structure, nodes other than leaf nodes are called as Internal Nodes. The root node is also said to be Internal Node if the tree has more than one node. Internal nodes are also called as 'Non-Terminal' nodes.

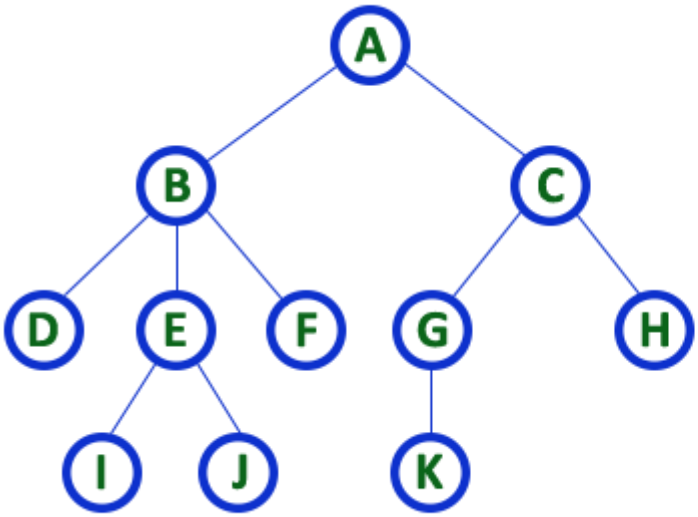


Here A, B, C, E & G are Internal nodes

- **In any tree the node which has atleast one child is called 'Internal' node**
- **Every non-leaf node is called as 'Internal' node**

Degree

- In a tree data structure, the total number of children of a node is called as DEGREE of that Node. In simple words, the Degree of a node is total number of children it has. The highest degree of a node among all the nodes in a tree is called as 'Degree of Tree'



Here Degree of B is 3

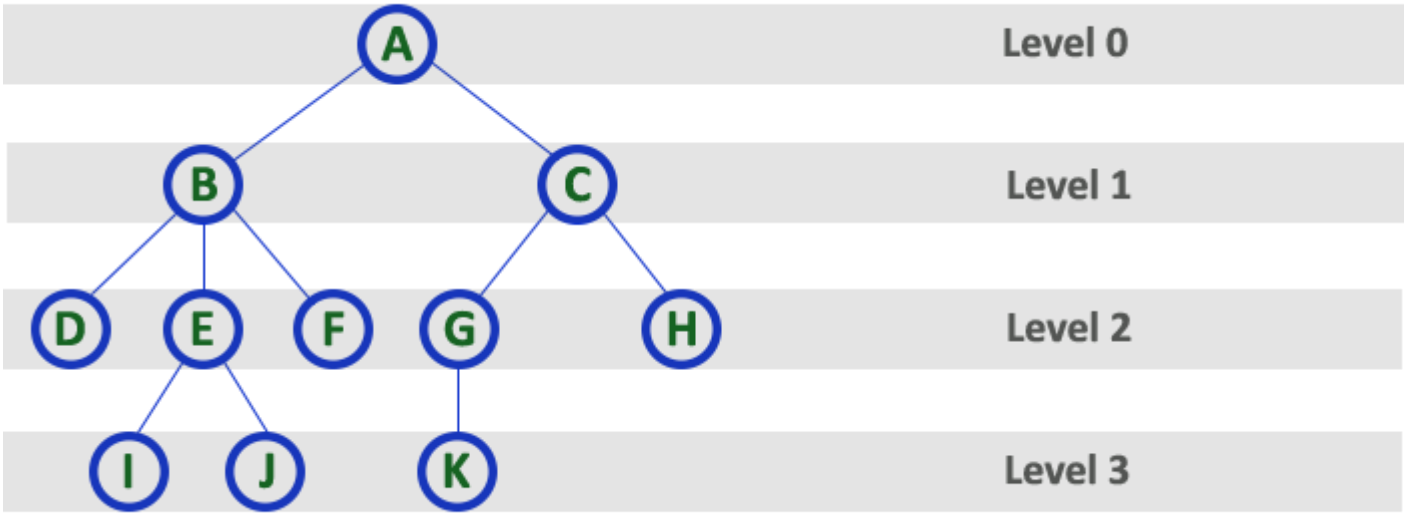
Here Degree of A is 2

Here Degree of F is 0

- In any tree, 'Degree' a node is total number of children it has.

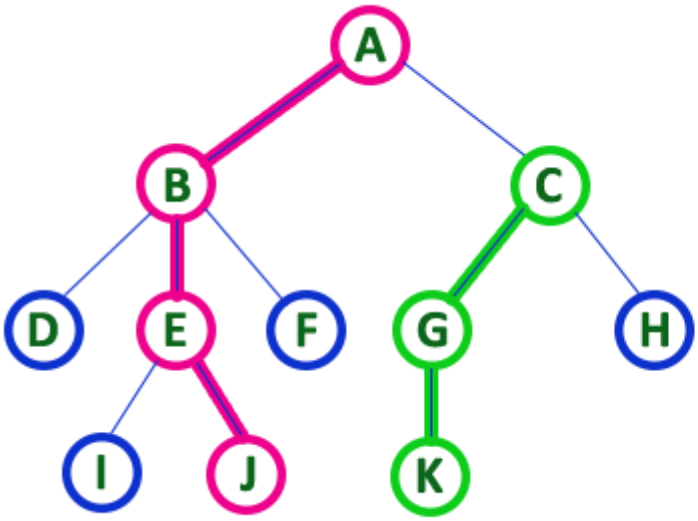
Level / Depth / Height

- In a tree data structure, the root node is said to be at Level 0 and the children of root node are at Level 1 and the children of the nodes which are at Level 1 will be at Level 2 and so on... In simple words, in a tree each step from top to bottom is called as a Level and the Level count starts with '0' and incremented by one at each level (Step).



Path

- In a tree data structure, the sequence of Nodes and Edges from one node to another node is called as PATH between that two Nodes. Length of a Path is total number of nodes in that path. In below example the path A - B - E - J has length 4.



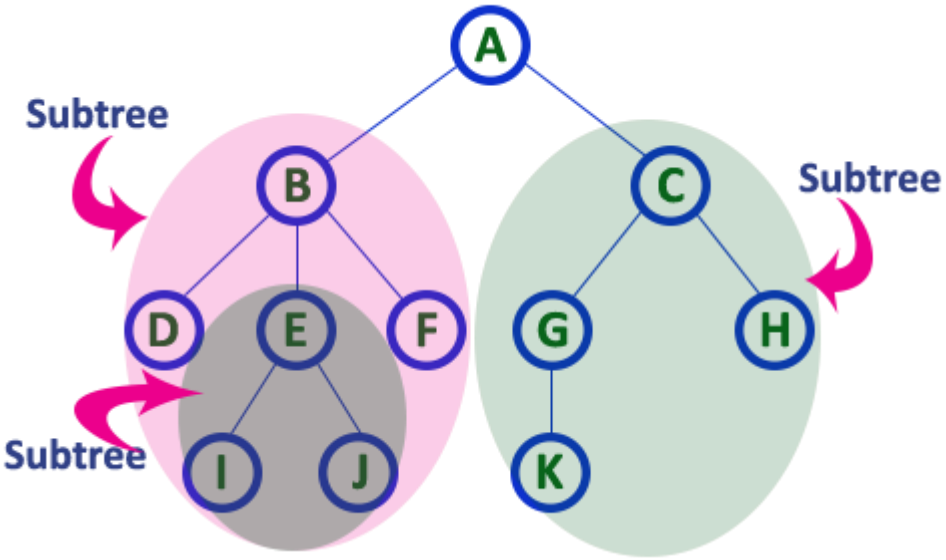
- In any tree, 'Path' is a sequence of nodes and edges between two nodes.

Here, 'Path' between A & J is
A - B - E - J

Here, 'Path' between C & K is
C - G - K

Sub Tree

- In a tree data structure, each child from a node forms a subtree recursively. Every child node will form a subtree on its parent node.



Note- in data structures tree are not natural trees they will always grow from root to leaves downwards.

Binary tree

- A binary tree T is defined as a finite set of elements called nodes such that,
  - T is empty (null tree)
  - T contain a distinguished node R, called the root of T, and the remaining nodes of T form an ordered pair of disjoint binary tree T_1 and T_2
- Direct: - A tree T in which any node can have maximum two children (left and right)
- Representation of tree in memory
  - Sequential representation – using an array info and left child and right child
  - Linked Representation – using self-referential structure node

```
struct node {
   int data;
   struct node* left;
   struct node* right;
}
```

Q if number of leaves in a tree is not a power of 2, then the tree is not a binary tree? **(GATE-1987) (1 Marks)**
(f)

Q The height of a binary tree is the maximum number of edges in any root to leaf path. The maximum number of nodes in a binary tree of height h is: **(GATE-2007) (1 Marks)**
a) 2^h-1 b) $2^{h-1}-1$ c) $2^{h+1}-1$ d) 2^{h+1}
ANSWER C

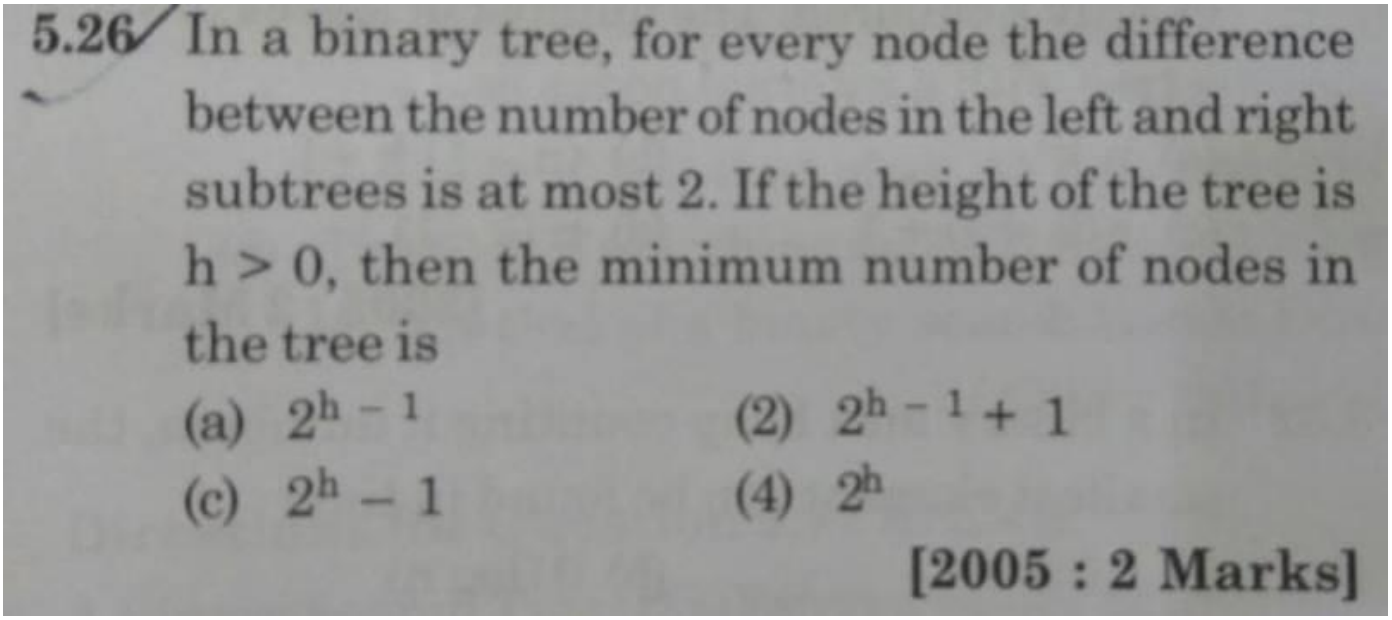
Q Level of a node is distance from root to that node. For example, level of root is 1 and levels of left and right children of root is 2. The maximum number of nodes on level i of a binary tree is. In the following answers, the operator '^' indicates power.
(A) $2^{(i-1)}$ **(B)** 2^i **(C)** $2^{(i+1)}$ **(D)** $2^{[(i+1)/2]}$
Answer: (A)

Q The height of a binary tree is the maximum number of edges in any root to leaf path. The maximum number of nodes in a binary tree of height h is: **(GATE - 2007) (1 Marks)**

(A) 2^h-1 **(B)** $2^{h-1}-1$ **(C)** $2^{h+1}-1$ **(D)** 2^{h+1}
Answer: (C)

Q In a binary tree with n nodes, every node has an odd number of descendants. Every node is considered to be its own descendant. What is the number of nodes in the tree that have exactly one child? **(GATE - 2010) (1 Marks)**
(A) 0 **(B)** 1 **(C)** (n-1)/2 **(D)** n-1
Answer: (A)

Q The height of a tree is the length of the longest root-to-leaf path in it. The maximum and minimum number of nodes in a binary tree of height 5 are **(GATE - 2015) (1 Marks)**
(A) 63 and 6, respectively **(B)** 64 and 5, respectively
(C) 32 and 6, respectively **(D)** 31 and 5, respectively
Answer: (A)


5.26 In a binary tree, for every node the difference between the number of nodes in the left and right subtrees is at most 2. If the height of the tree is $h > 0$, then the minimum number of nodes in the tree is

(a) 2^{h-1} (2) $2^{h-1}+1$

(c) 2^h-1 (4) 2^h

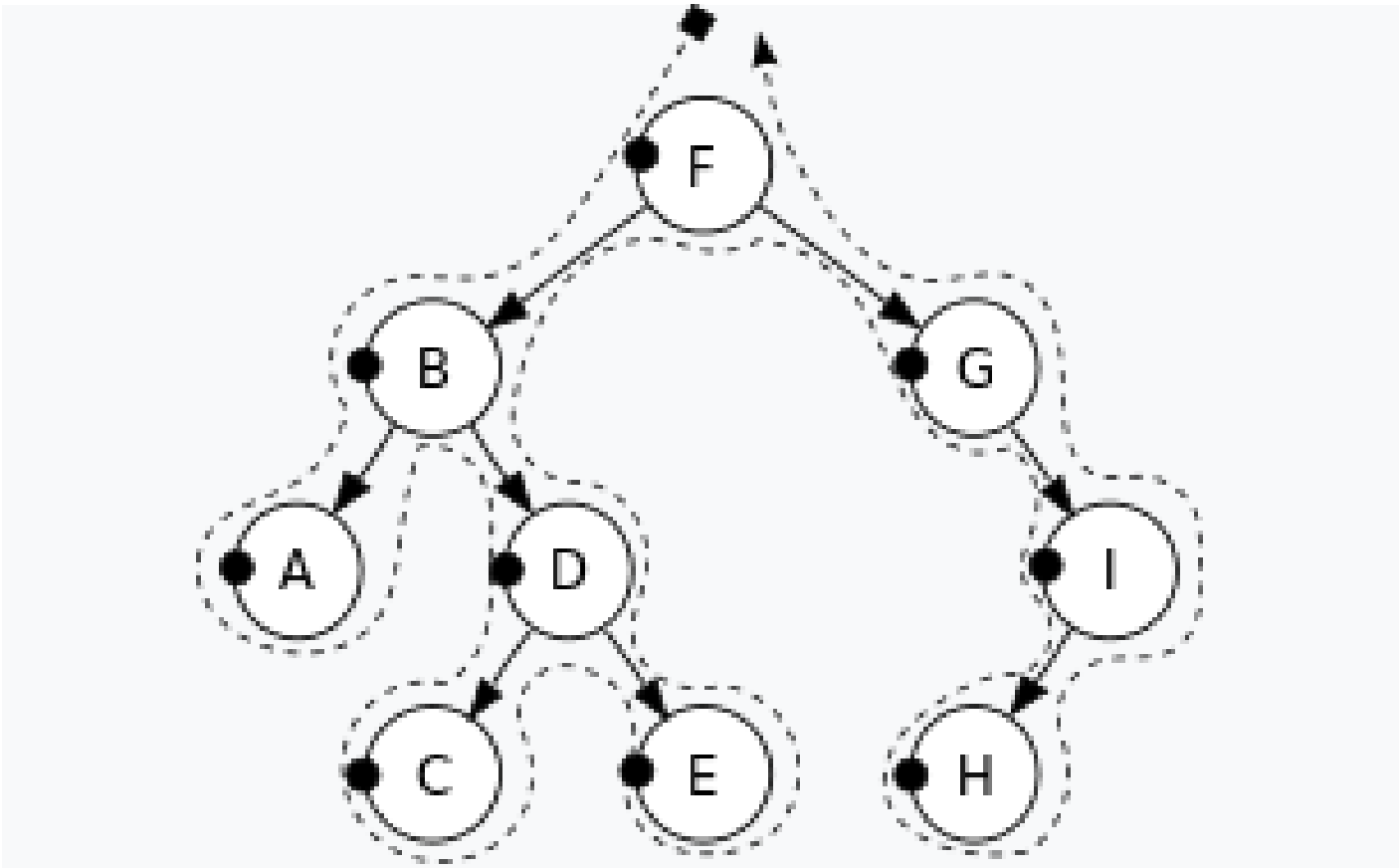
[2005 : 2 Marks]

Q Let T be a binary search tree with 15 nodes. The minimum and maximum possible heights of T are: _____ **(GATE-2017) (1 Marks)**
(A) 4 and 15 respectively **(B)** 3 and 14 respectively
(C) 4 and 14 respectively **(D)** 3 and 15 respectively
Answer: (B)

Traversal of binary tree

- The process of visiting (checking and/or updating) each node in a tree data structure, exactly once in called tree traversal. Such traversals are classified by the order in which the nodes are visited.
- Unlike linked lists, one-dimensional arrays and other linear data structures, which are canonically traversed in linear order, trees may be traversed in multiple ways. They may be traversed in depth-first or breadth-first order. There are three common ways to traverse them in depth-first order: in-order, pre-order and post-order. Beyond these basic traversals, various more complex or hybrid schemes are possible, such as depth-limited searches like iterative deepening depth-first search.
- Some applications do not require that the nodes be visited in any particular order as long as each node is visited precisely once. For other applications, nodes must be visited in an order that preserves some relationship.
- These steps can be done *in any order*. If (L) is done before (R), the process is called left-to-right traversal, otherwise it is called right-to-left traversal. The following methods show left-to-right traversal:

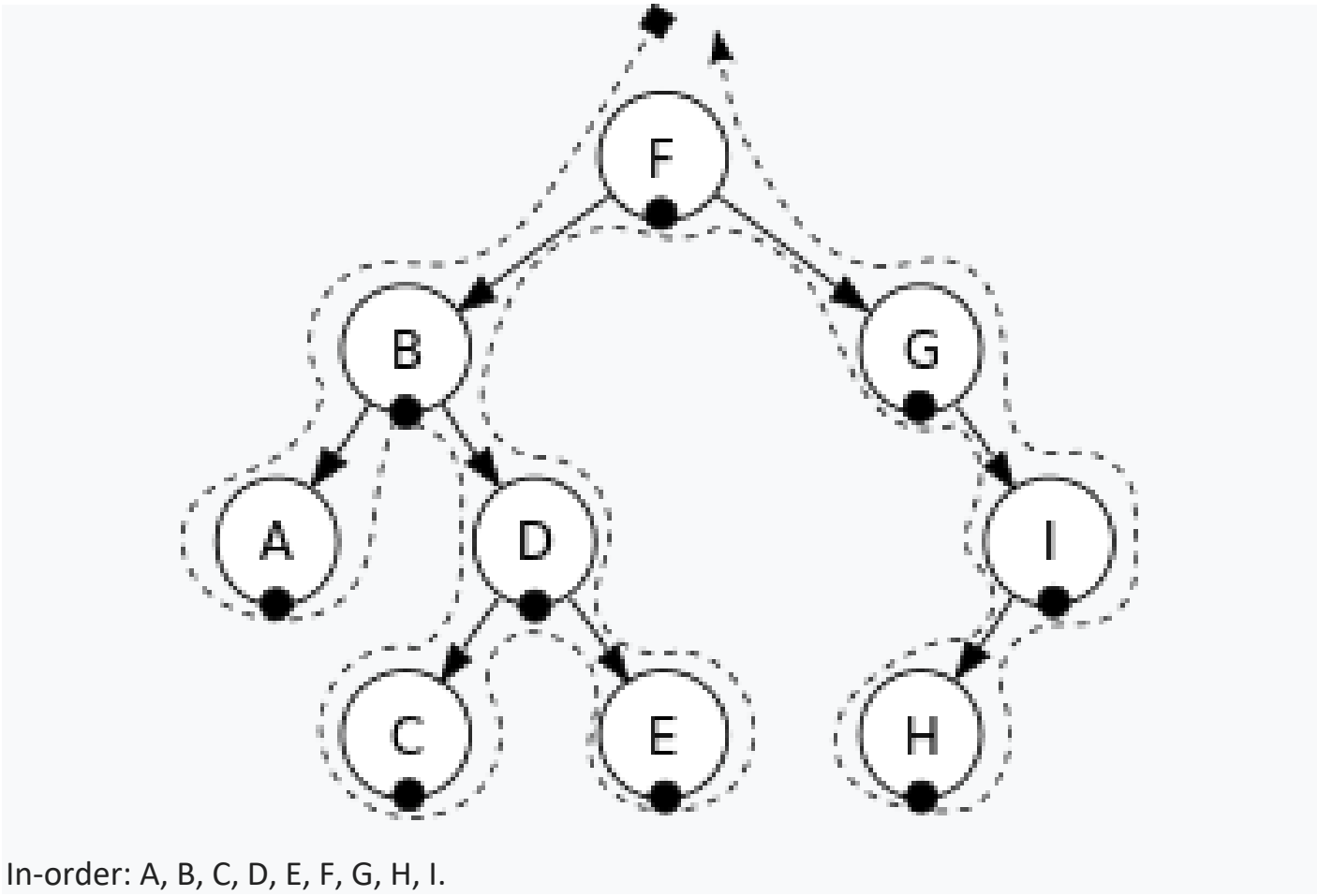
Pre-order (NLR)



Pre-order: F, B, A, D, C, E, G, I, H.

1. Check if the current node is empty or null.
2. Display the data part of the root (or current node).
3. Traverse the left subtree by recursively calling the pre-order function.
4. Traverse the right subtree by recursively calling the pre-order function.

In-order (LNR)

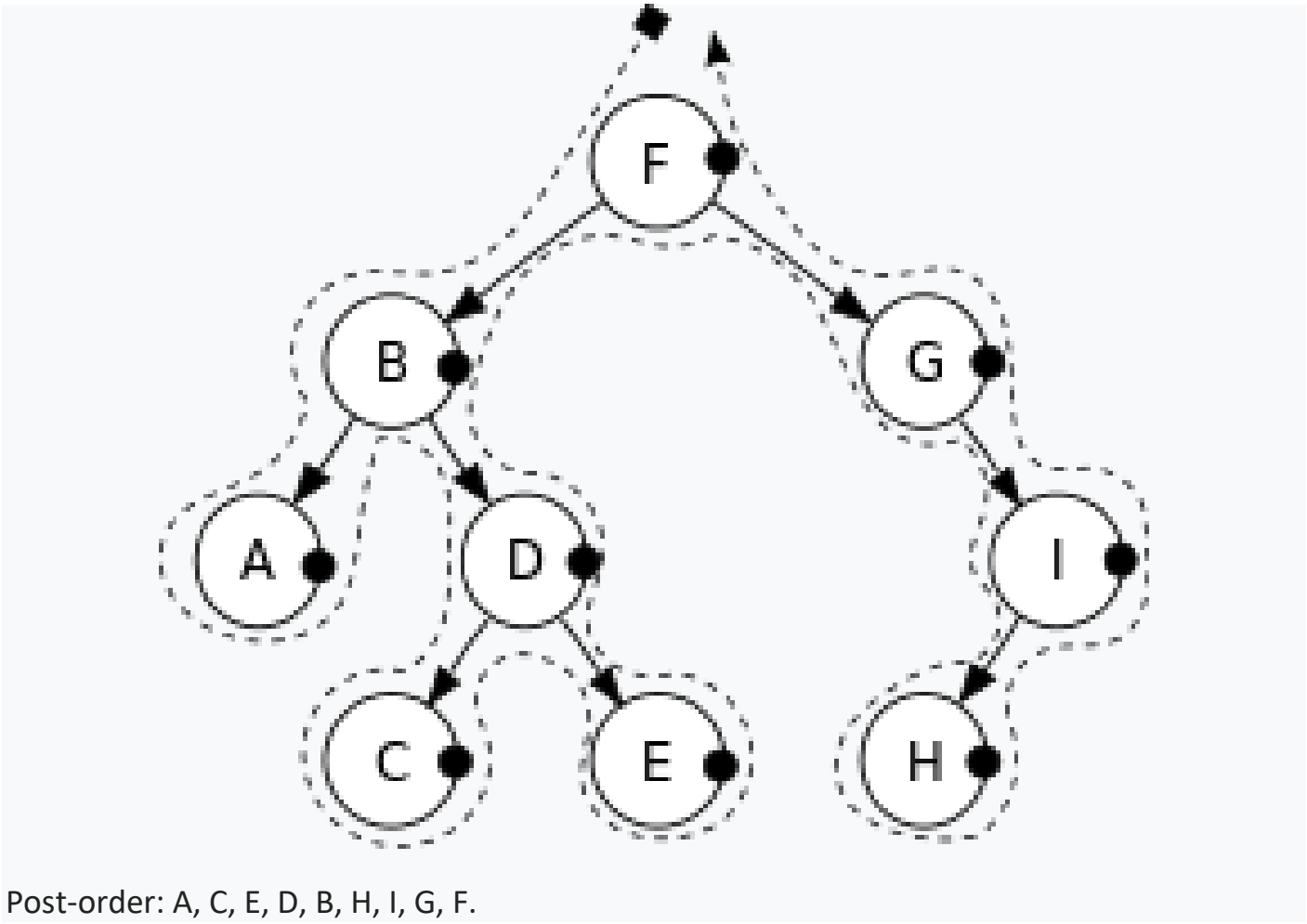


In-order: A, B, C, D, E, F, G, H, I.

1. Check if the current node is empty or null.
2. Traverse the left subtree by recursively calling the in-order function.
3. Display the data part of the root (or current node).
4. Traverse the right subtree by recursively calling the in-order function.

In a binary search tree, in-order traversal retrieves data in sorted order.

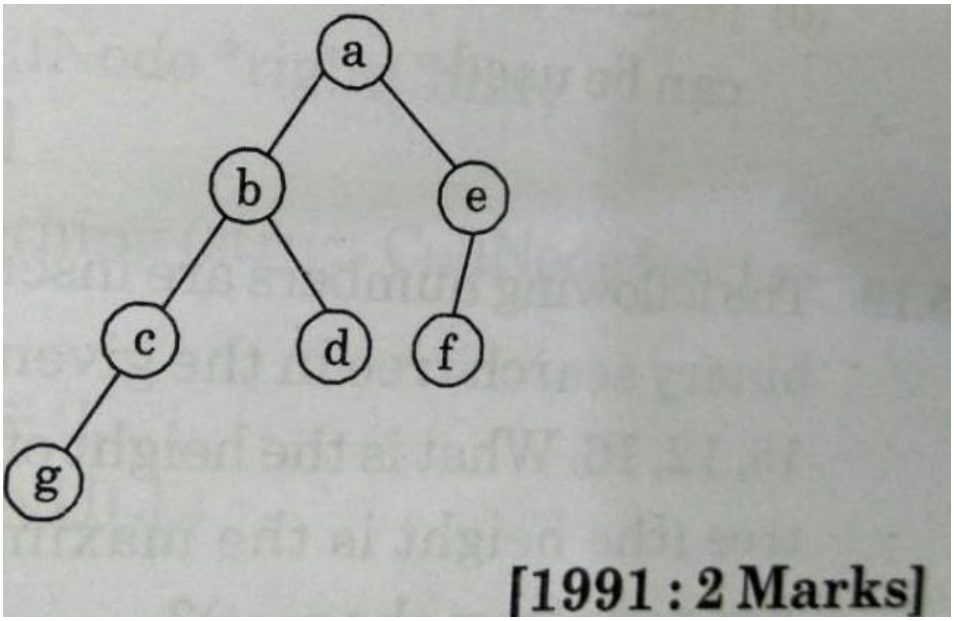
Post-order (LRN)[edit]



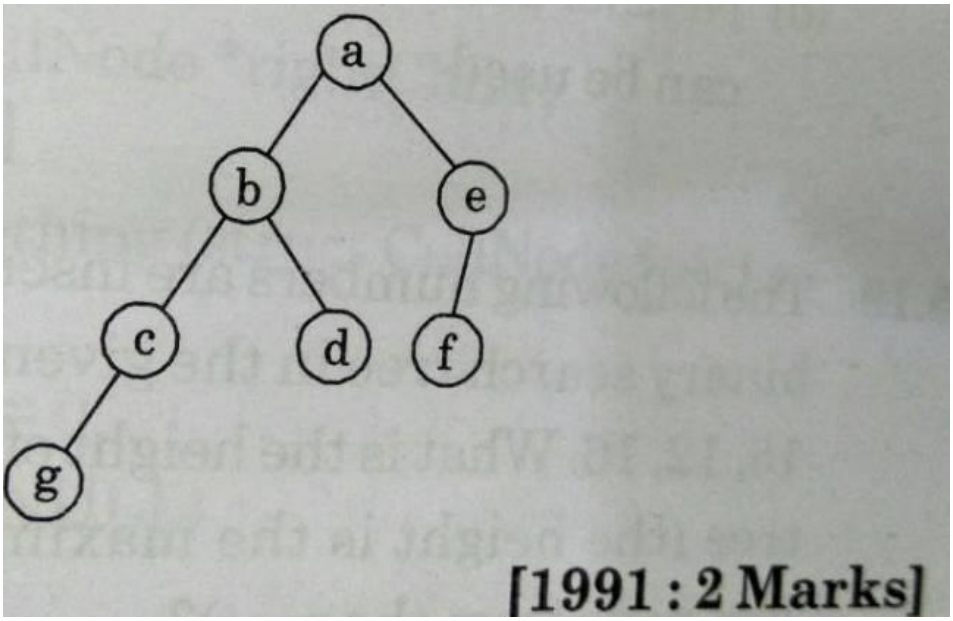
Post-order: A, C, E, D, B, H, I, G, F.

1. Check if the current node is empty or null.
2. Traverse the left subtree by recursively calling the post-order function.
3. Traverse the right subtree by recursively calling the post-order function.
4. Display the data part of the root (or current node).

Q if the binary tree in fig is traversed in inorder, then the order in which the nodes will be visited is……. **(GATE-1991) (1 Marks)**

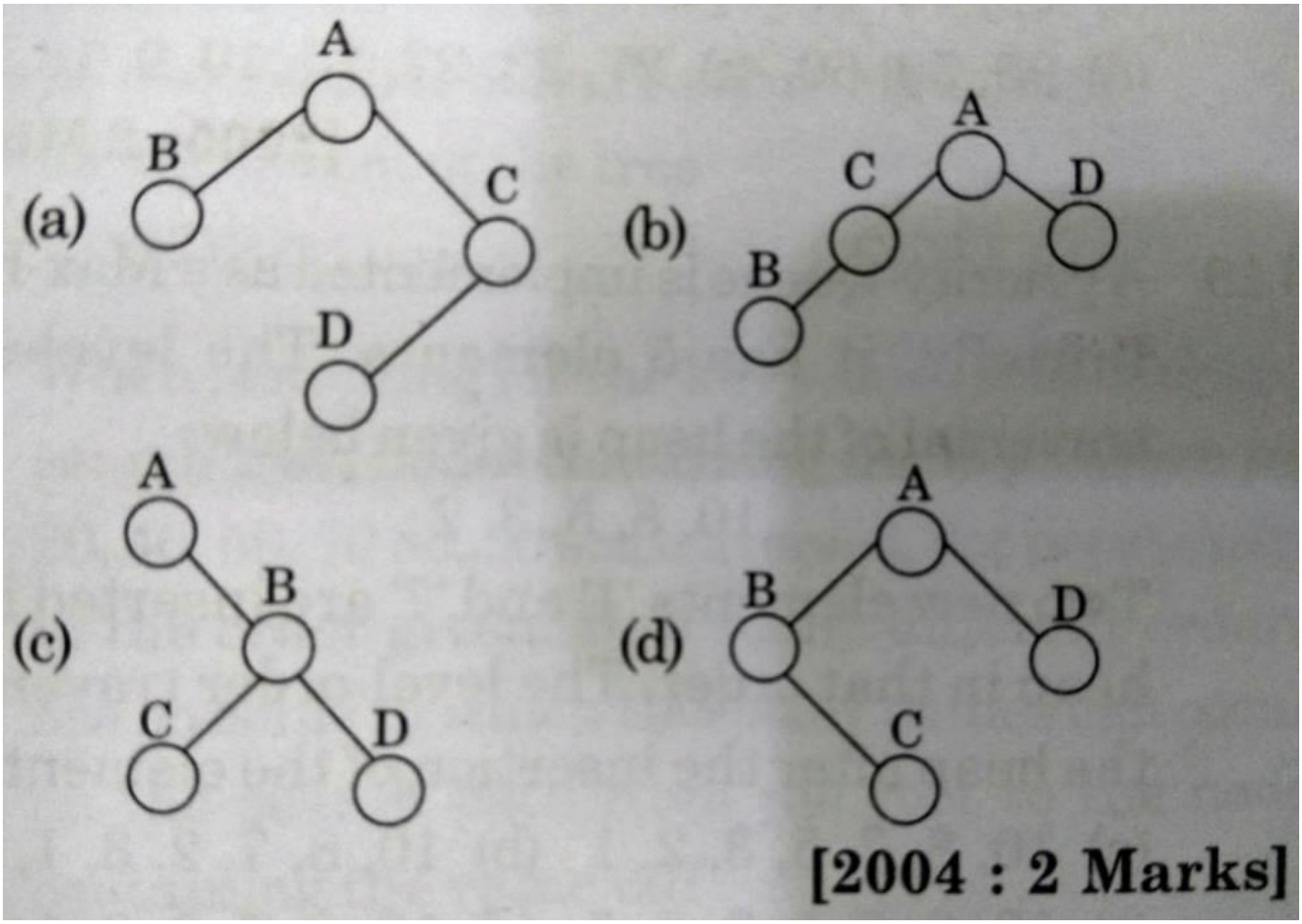


Q which of the following is post order traversal of the above tree **(GATE-1996) (1 Marks)**



a) fegcbdba **b)** gcbdafe **c)** gcdbfea **d)** fedgcba

Q which of the following binary trees has its inorder and preorder traversal as BCAD and ABCD, respectively? **(GATE-2004) (1 Marks)**



Q

The inorder and preorder traversal of a binary tree are

d b e a f c g and a b d e c f g, respectively.

The postorder traversal of the binary tree is:

a) **e b f g c a**
b) **e d b g f c a**
c) **e d b f g c a**
d) **d e f g b c a**

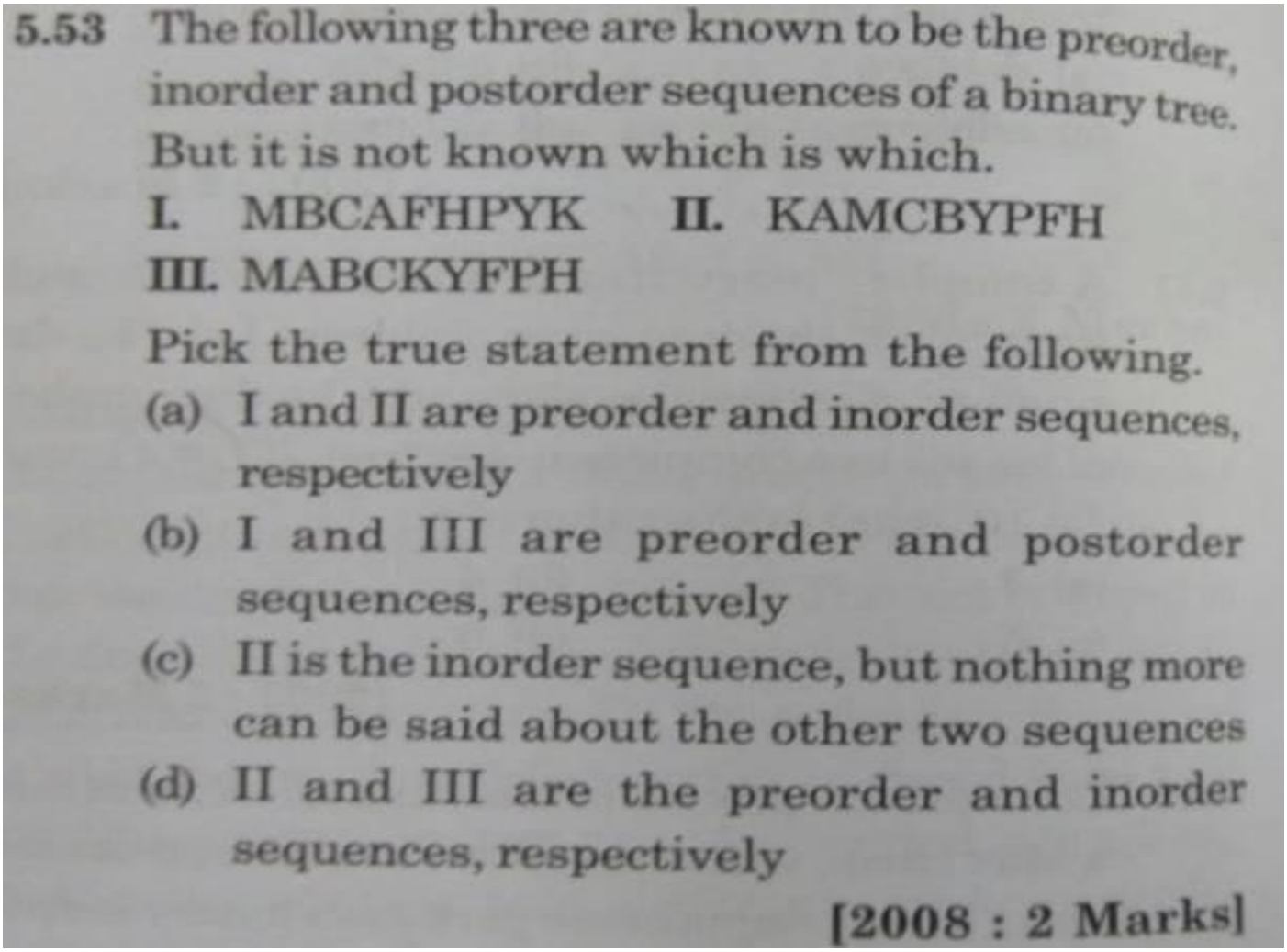
ANSWER a

Q is it possible to construct a binary tree uniquely whose post order and pre order traversal are given? **(GATE-1987) (1 Marks)**
(f)

Q Which of the following pairs of traversals is not sufficient to build a binary tree from the given traversals?
(A) Preorder and Inorder
(B) Preorder and Post order
(C) Inorder and Post order
(D) level order and post order
Answer: (B)

Q What is common in three different types of traversals (Inorder, Preorder and Post order)?
(A) Root is visited before right subtree
(B) Left subtree is always visited before right subtree
(C) Root is visited after left subtree
(D) All of the above
Answer: (B)

Q The inorder and preorder traversal of a binary tree are d b e a f c g and a b d e c f g, respectively. The post order traversal of the binary tree is: **(GATE-2007) (2 Marks)**
(A) d e b f g c a **(B)** e d b g f c a **(C)** e d b f g c a **(D)** d e f g b c a
Answer: (A)



5.53 The following three are known to be the preorder, inorder and postorder sequences of a binary tree. But it is not known which is which.

I. MBCAFHPYK II. KAMCBYPFH
III. MABCKYFPH

Pick the true statement from the following.

(a) I and II are preorder and inorder sequences, respectively
(b) I and III are preorder and postorder sequences, respectively
(c) II is the inorder sequence, but nothing more can be said about the other two sequences
(d) II and III are the preorder and inorder sequences, respectively

[2008 : 2 Marks]

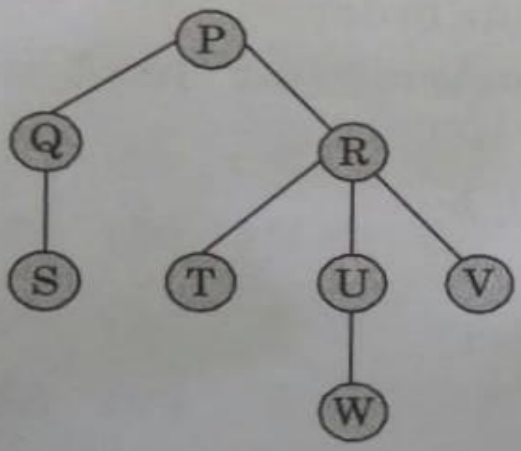
Q The post order traversal of a binary tree is 8, 9, 6, 7, 4, 5, 2, 3, 1. The inorder traversal of the same tree is 8, 6, 9, 4, 7, 2, 5, 1, 3. The height of a tree is the length of the longest path from the root to any leaf. The height of the binary tree above is _____. **(GATE-2018) (2 Marks)**
(ANSWER 4)

Q level order traversal of a rooted tree can be done by starting from the root and performing **(GATE-2004) (2 Marks)**

a) preorder traversal
b) inorder traversal
c) dfs
d) bfs

Ans: d

5.63 Consider the following rooted tree with the vertex labeled P as the root:



The order in which the nodes are visited during an in-order traversal of the tree is

(a) SQPTRWUV
(b) SQPTUWRV
(c) SQPTWUVR
(d) SQPTRUWV

[2014 (Set-3) : 1 Mark]

Ans: a

Q Consider the following New-order strategy for traversing a binary tree: Visit the root; Visit the right subtree using New-order Visit the left subtree using New-order The New-order traversal of the expression tree corresponding to the reverse polish expression 3 4 * 5 - 2 ^ 6 7 * 1 + - is given by: **(GATE-2016) (1 Marks)**

a) + - 1 6 7 * 2 ^ 5 - 3 4 *
b) - + 1 * 6 7 ^ 2 - 5 * 3 4
c) - + 1 * 7 6 ^ 2 - 5 * 4 3
d) 1 7 6 * + 2 5 4 3 * - ^ -

ANSWER - c

Q What does the following function do for a given binary tree?

```
int fun (struct node *root)
{
  if (root == NULL)
```

```
    return 0;
  if (root->left == NULL && root->right == NULL)
    return 0;
  return 1 + fun(root->left) + fun(root->right);
}
```

(A) Counts leaf nodes
(B) Counts internal nodes
(C) Returns height where height is defined as number of edges on the path from root to deepest node
(D) Return diameter where diameter is number of edges on the longest path between any two nodes.
Answer: (B)

Q Consider the following C program segment

```
struct CellNode
{
 struct CellNode *leftchild;
 int element;
 struct CellNode *rightChild;
}

int Dosomething (struct CellNode *ptr)
{
   int value = 0;
   if (ptr != NULL)
   {
    if (ptr->leftChild != NULL)
      value = 1 + DoSomething(ptr->leftChild);
    if (ptr->rightChild != NULL)
      value = max (value, 1 + DoSomething(ptr->rightChild));
   }
   return (value);
}
```

The value returned by the function DoSomething when a pointer to the root of a non-empty tree is passed as argument is **(GATE - 2004) (2 Marks)**
(A) The number of leaf nodes in the tree
(B) The number of nodes in the tree
(C) The number of internal nodes in the tree
(D) The height of the tree(consider root at level 1)
Answer: (D)

Q Following function is supposed to calculate the maximum depth or height of a Binary tree — the number of edges along the longest path from the root node down to the farthest leaf node. (meaning root is at level 1)

```
int maxDepth (struct node* node)
{
  if (node==NULL)
     return 0;
  else
  {
     /* compute the depth of each subtree */
     int lDepth = maxDepth(node->left);
     int rDepth = maxDepth(node->right);

     /* use the larger one */
     if (lDepth > rDepth)
        return X;
     else return Y;
  }

}
```

What should be the values of X and Y so that the function works correctly?

(A) X = lDepth, Y = rDepth
(B) X = lDepth + 1, Y = rDepth + 1
(C) X = lDepth – 1, Y = rDepth -1
(D) None of the above

Answer: (B)

Q The height of a tree is defined as the number of edges on the longest path in the tree. The function shown in the pseudocode below is invoked as height (root) to compute the height of a binary tree rooted at the tree pointer root. **(GATE-2012) (2 Marks)**

```
int height (treeptr n)
{ if (n== NULL) return -1;
  if (n → left == NULL)
        if (n → right ==NULL) return 0;
        else return [B1];                                  // Box 1
else {h1 = height (n → left);
        if (n → right == NULL) return (1 + h1);
        else {h2 = height (n → right);
                return [B2] ;                              // Box 2
                }
        }
}
```

The appropriate expression for the two boxes B1 and B2 are
(A) B1 : (1 + height(n->right)), B2 : (1 + max(h1,h2))
(B) B1 : (height(n->right)), B2 : (1 + max(h1,h2))
(C) B1 : height(n->right), B2 : max(h1,h2)
(D) B1 : (1 + height(n->right)), B2 : max(h1,h2)
Ans : a

Binary search tree / ordered tree / sorted binary tree

- A binary search tree (BST) is a binary tree in which left subtree of a node contains a key less than the node's key and right subtree of a node contains only the nodes with key greater than the node's key. Left and right sub tree must each also be a binary search tree.
- A binary search tree is a rooted binary tree, whose internal nodes each store a key (and optionally, an associated value) and each have two distinguished sub-trees, commonly denoted *left* and *right*. The tree additionally satisfies the binary search property, which states that the key in each node must be greater than or equal to any key stored in the left sub-tree, and less than or equal to any key stored in the right sub-tree

Algorithm	Average	Worst case
Space	O(*n*)	O(*n*)
Search	O(log *n*)	O(*n*)
Insert	O(log *n*)	O(*n*)
Delete	O(log *n*)	O(*n*)

- The major advantage of binary search trees over other data structures is that the related sorting algorithms and search algorithm such as in-order traversal can be very efficient; they are also easy to code

Operations

Binary search trees support three main operations: insertion of elements, deletion of elements, and lookup (checking whether a key is present).

Searching

We begin by examining the root node. If the tree is *null*, the key we are searching for does not exist in the tree. Otherwise, if the key equals that of the root, the search is successful and we return the node. If the key is less than that of the root, we search the left subtree. Similarly, if the key is greater than that of the root, we search the right subtree. This process is repeated until the key is found or the remaining subtree is *null*. If the searched key is not found after a *null* subtree is reached, then the key is not present in the tree.

Insertion

Insertion begins as a search would begin; if the key is not equal to that of the root, we search the left or right subtrees as before. Eventually, we will reach an external node and add the new key-value pair (here encoded as a record 'newNode') as its right or left child, depending on the node's key. In other words, we examine the root and recursively insert the new node to the left subtree if its key is less than that of the root, or the right subtree if its key is greater than or equal to the root.

Deletion

- Deleting a node with no children: simply remove the node from the tree.
- Deleting a node with one child: remove the node and replace it with its child.
- Deleting a node with two children: call the node to be deleted *D*. Do not delete *D*. Instead, choose either its in-order predecessor node or its in-order successor node as replacement node *E* (s. figure). Copy the user values of *E* to *D*.[note 2] If *E* does not have a child simply remove *E* from its previous parent *G*. If *E* has a child, say *F*, it is a right child. Replace *E* with *F* at *E*'s parent.

5.7 A binary search tree is generated by inserting in order of the following integers: 50, 15, 62, 5, 20, 58, 91, 3, 8, 37, 60, 24. The number of nodes in the left subtree and right subtree of the root respectively is

(a) (4, 7) (b) (7, 4)

(c) (8, 3) (d) (3, 8)

[1996 : 2 Marks]

5.19 The following numbers are inserted into an empty binary search tree in the given order: 10, 1, 3, 5, 15, 12, 16. What is the height of the binary search tree (the height is the maximum distance of a leaf node from the root)?

(a) 2 (b) 3

(c) 4 (d) 6

[2004 : 1 Mark]

Q The following numbers are inserted into an empty binary search tree in the given order: 10, 1, 3, 5, 15, 12, 16. What is the height of the binary search tree (the height is the maximum distance of a leaf node from the root)(means root at level 0)?

(A) 2 **(B)** 3 **(C)** 4 **(D)** 6

Answer: (B)

Q While inserting the elements 71, 65, 84, 69, 67, 83 in an empty binary search tree (BST) in the sequence shown, the element in the lowest level is (GATE-2015) (1 Marks)

(A) 65 **(B)** 67 **(C)** 69 **(D)** 83

Answer: (B)

Q Suppose the numbers 7, 5, 1, 8, 3, 6, 0, 9, 4, 2 are inserted in that order into an initially empty binary search tree. The binary search tree uses the usual ordering on natural numbers. What is the in-order traversal sequence of the resultant tree? (Gate-2003) (2 Marks)

(A) 7 5 1 0 3 2 4 6 8 9
(B) 0 2 4 3 1 6 5 9 8 7
(C) 0 1 2 3 4 5 6 7 8 9
(D) 9 8 6 4 2 3 0 1 5 7

Answer: (C)

Q Which of the following is/are correct inorder traversal sequence(s) of binary search tree(s)? **(GATE-2015) (1 Marks)**

1. 3, 5, 7, 8, 15, 19, 25
2. 5, 8, 9, 12, 10, 15, 25
3. 2, 7, 10, 8, 14, 16, 20
4. 4, 6, 7, 9, 18, 20, 25

a) 1 and 4 only **b)** 2 and 3 only **c)** 2 and 4 only **d)** 2 only

ANSWER A

Q The pre-order traversal of a binary search tree is given by 12, 8, 6, 2, 7, 9, 10, 16, 15, 19, 17, 20. Then the post-order traversal of this tree is: **(GATE-2017) (2 Marks)**

a) 2, 6, 7, 8, 9, 10, 12, 15, 16, 17, 19, 20
b) 2, 7, 6, 10, 9, 8, 15, 17, 20, 19, 16, 12
c) 7, 2, 6, 8, 9, 10, 20, 17, 19, 15, 16, 12
d) 7, 6, 2, 10, 9, 8, 15, 16, 17, 20, 19, 12

ANSWER- B

Q Post order traversal of a given binary search tree, T produces the following sequence of keys 10, 9, 23, 22, 27, 25, 15, 50, 95, 60, 40, 29 Which one of the following sequences of keys can be the result of an in-order traversal of the tree T? (GATE - 2005) (1 Marks)

(A) 9, 10, 15, 22, 23, 25, 27, 29, 40, 50, 60, 95
(B) 9, 10, 15, 22, 40, 50, 60, 95, 23, 25, 27, 29
(C) 29, 15, 9, 10, 25, 22, 23, 27, 40, 60, 50, 95
(D) 95, 50, 60, 40, 27, 23, 22, 25, 10, 9, 15, 29

Answer: (A)

Q The preorder traversal sequence of a binary search tree is 30, 20, 10, 15, 25, 23, 39, 35, 42. Which one of the following is the post order traversal sequence of the same tree? **(GATE-2013) (1 Marks)**

a) 10, 20, 15, 23, 25, 35, 42, 39, 30
b) 15, 10, 25, 23, 20, 42, 35, 39, 30
c) 15, 20, 10, 23, 25, 42, 35, 39, 30
d) 15, 10, 23, 25, 20, 35, 42, 39, 30

ANSWER D

5.9 A binary search tree contains the values 1, 2, 3, 4, 5, 6, 7, 8. The tree is traversed in pre-order and the values are printed out. Which of the following sequences is a valid output?

(a) 53124786 (b) 53126487

(c) 53241678 (d) None of these

[1997 : 2 Marks]

5.27 A binary search tree contains the numbers 1, 2, 3, 4, 5, 6, 7, 8. When the tree is traversed in preorder and the values in each node printed out, the sequence of values obtained is 5, 3, 1, 2, 4, 6, 8, 7. If the tree is traversed in postorder, the sequence obtained would be

(a) 8, 7, 6, 5, 4, 3, 2, 1

(b) 1, 2, 3, 4, 8, 7, 6, 5

(c) 2, 1, 4, 3, 6, 7, 8, 5

(d) 2, 1, 4, 3, 7, 8, 6, 5

[2005 : 2 Marks]

Q The pre-order traversal of a binary search tree is given by 12, 8, 6, 2, 7, 9, 10, 16, 15, 19, 17, 20. Then the post-order traversal of this tree is: **(GATE - 2017) (2 Marks)**

a) 2, 6, 7, 8, 9, 10, 12, 15, 16, 17, 19, 20

b) 2, 7, 6, 10, 9, 8, 15, 17, 20, 19, 16, 12

c) 7, 2, 6, 8, 9, 10, 20, 17, 19, 15, 16, 12

d) 7, 6, 2, 10, 9, 8, 15, 16, 17, 20, 19, 12

Q The preorder traversal sequence of a binary search tree is 30, 20, 10, 15, 25, 23, 39, 35, 42. Which one of the following is the post order traversal sequence of the same tree? **(GATE - 2013) (2 Marks)**

(A) 10, 20, 15, 23, 25, 35, 42, 39, 30
(B) 15, 10, 25, 23, 20, 42, 35, 39, 30
(C) 15, 20, 10, 23, 25, 42, 35, 39, 30
(D) 15, 10, 23, 25, 20, 35, 42, 39, 30

Answer: (D)

Q The number of ways in which the numbers 1, 2, 3, 4, 5, 6, 7 can be inserted in an empty binary search tree, such that the resulting tree has height 6, is _____ Note: The height of a tree with a single node is 0. **(GATE-2016) (1 Marks)**

ANSWER - 64

5.8 A binary search tree is used to locate the number 43. Which of the following probe sequences are possible and which are not? Explain.

(a) 61 52 14 17 40 43
(b) 2 3 50 40 60 43
(c) 10 65 31 48 37 43
(d) 81 61 52 14 41 43
(e) 17 77 27 66 18 43

[1996 : 2 Marks]

5.35 Suppose that we have numbers between 1 and 100 in a binary search tree and want to search for the number 55. Which of the following sequences CANNOT be the sequence of nodes examined?

(a) {10, 75, 64, 43, 60, 57, 55}

(b) {90, 12, 68, 34, 62, 45, 55}

(c) {9, 85, 47, 68, 43, 57, 55}

(d) {79, 14, 72, 56, 16, 53, 55}

[2006 : 2 Marks]

5.44 When searching for the key value 60 in a binary search tree, nodes containing the key values 10, 20, 40, 50, 70 80, 90 are traversed, not necessarily in the order given. How many different orders are possible in which these key values can occur on the search path from the root to the node containing the value 60?

(a) 35 (b) 64

(c) 128 (d) 5040

[2007 : 2 Marks]

A Binary Search Tree (BST) stores values in the range 37 to 573. Consider the following sequence of keys.

I. 81, 537, 102, 439, 285, 376, 305
II. 52, 97, 121, 195, 242, 381, 472
III. 142, 248, 520, 386, 345, 270, 307
IV. 550, 149, 507, 395, 463, 402, 270

5.46 Suppose the BST has been unsuccessfully searched for key 273. Which all of the above sequences list nodes in the order in which we could have encountered them in the search?

(a) II and III only (b) I and III only
(c) III and IV only (d) III only

[2008 : 2 Marks]

5.47 Which of the following statements is TRUE?

(a) I, II and IV are inorder sequences of three different BSTs
(b) I is a preorder sequence of some BST with 439 as the root
(c) II is an inorder sequence of some BST where 121 is the root and 52 is a leaf
(d) IV is a postorder sequence of some BST with 149 as the root

[2008 : 2 Marks]

5.25 The numbers 1, 2,..., n are inserted in a binary search tree in some order. In the resulting tree, the right subtree of the root contains p nodes. The first number to be inserted in the tree must be

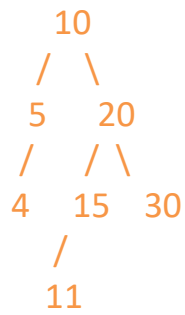
(a) p (b) p + 1

(c) n – p (d) n – p + 1

[2005 : 1 Mark]

Ans c

Q Consider the following Binary Search Tree



If we randomly search one of the keys present in above BST, what would be the expected number of comparisons?

(A) 2.75 **(B)** 2.25 **(C)** 2.57 **(D)** 3.25

Answer: (C)

Explanation: Expected number of comparisons = (1*1 + 2*2 + 3*3 + 4*1)/7 = 18/7 = 2.57

Q What is the worst case time complexity for search, insert and delete operations in a general Binary Search Tree?

(A) O(n) for all

(B) O(Logn) for all

(C) O(Logn) for search and insert, and O(n) for delete

(D) O(Logn) for search, and O(n) for insert and delete

Answer: (A)

Q In delete operation of BST, we need inorder successor (or predecessor) of a node when the node to be deleted has both left and right child as non-empty. Which of the following is true about inorder successor needed in delete operation?
(A) Inorder Successor is always a leaf node
(B) Inorder successor is always either a leaf node or a node with empty left child
(C) Inorder successor may be an ancestor of the node
(D) Inorder successor is always either a leaf node or a node with empty right child
Answer: (B)

Q You are given the post order traversal, P, of a binary search tree on the n elements 1, 2, ..., n. You have to determine the unique binary search tree that has P as its post order traversal. What is the time complexity of the most efficient algorithm for doing this? **(GATE-2008) (1 Marks)**
a) θ(log n) b) θ(n)
c) θ(nlog n) d) None of the above, as the tree cannot be uniquely determined
ANSWER B

Q Which one of the following is the tightest upper bound that represents the time complexity of inserting an object into a binary search tree of n nodes? **(GATE-2013) (1 Marks)**
a) O(1) **b)** O(Logn) **c)** O(n) **d)** O(nLogn)
ANSWER C

Q The worst-case running time to search for an element in a balanced in a binary search tree with n2^n elements is **(GATE-2013) (1 Marks)**
(A) $\Theta(nlogn)$ **(B)** $\Theta(n2^n)$ **(C)** $\Theta(n)$ **(D)** $\Theta(logn)$
ANSWER C

Q Suppose we have a balanced binary search tree T holding n numbers. We are given two numbers L and H and wish to sum up all the numbers in T that lie between L and H. Suppose there are m such numbers in T. If the tightest upper bound on the time to compute the sum is O(nalogb n + mc logd n), the value of a + 10b + 100c + 1000d is ____. **(GATE-2014) (1 Marks)**
ANSWER 100

Q What are the worst-case complexities of insertion and deletion of a key in a binary search tree? **(GATE-2015) (1 Marks)**
a) Θ(logn) for both insertion and deletion
b) Θ(n) for both insertion and deletion
c) Θ(n) for insertion and Θ(logn) for deletion
d) Θ(logn) for insertion and Θ(n) for deletion

ANSWER B

Q Suppose we have a balanced binary search tree T holding n numbers. We are given two numbers L and H and wish to sum up all the numbers in T that lie between L and H. Suppose there are m such numbers in T. If the tightest upper bound on the time to compute the sum is $O(n^a \log^b n + m^c \log^d n)$, the value of a + 10b + 100c + 1000d is ____. **(GATE - 2014) (1 Marks)**

(A) 60 **(B)** 110 **(C)** 210 **(D)** 50

Answer: (B)

Q You are given the post order traversal, P, of a binary search tree on the n elements 1, 2, …, n. You have to determine the unique binary search tree that has P as its post order traversal. What is the time complexity of the most efficient algorithm for doing this? (GATE - 2008)

(A) O(Logn)

(B) O(n)

(C) O(nLogn)

(D) none of the above, as the tree cannot be uniquely determined.

Answer: (B)

AVL tree

- In computer science, an **AVL tree** (named after inventors **A**delson-**V**elsky and **L**andis) is a self-balancing binary search tree. It was the first such data structure to be invented.
- In an AVL tree, the heights of the two child subtrees of any node differ by at most one; if at any time they differ by more than one, rebalancing is done to restore this property.
- Lookup, insertion, and deletion all take O(log *n*) time in both the average and worst cases, where *n* is the number of nodes in the tree prior to the operation.
- Insertions and deletions may require the tree to be rebalanced by one or more tree rotations.

Algorithm	Average	Worst case
Space	O(*n*)	O(*n*)
Search	O(log *n*)	O(log *n*)
Insert	O(log *n*)	O(log *n*)
Delete	O(log *n*)	O(log *n*)

Balance factor

- In a binary tree the *balance factor* of a node N is defined to be the height difference Balance Factor(N): = Height (LeftSubtree(N)) – Height (RightSubtree(N)) of its two child subtrees.
- A binary tree is defined to be an *AVL tree* if the invariant Balance Factor(N) ∈ {–1, 0, +1} holds for every node N in the tree.
- A node N with Balance Factor(N) > 0 is called "left-heavy"
- One with Balance Factor(N) < 0 is called "right-heavy"
- One with Balance Factor(N) = 0 is sometimes simply called "balanced".

Insertion in an AVL tree

- Insert a node similarly as we do in binary search tree.
- After insertion start checking the balancing factor of each node in a bottom up fashion that is from newly inserted node towards the root.
- Stop on the first node whose balancing factor is violated and go two steps towards the newly inserted nodes. watch the movement, which is identified as the problem.

Problem	Solution
LL	R
RR	L
LR	LR
RL	RL

- After every insertion at most two rotations are sufficient to balance the AVL tree

Q Consider an empty AVL tree and insert the following nodes in sequence 21, 26, 30, 9, 4, 14, 28, 18, 15, 10, 2, 3, 7?

Q Consider an empty AVL tree and insert the following nodes in sequence a, z, x, i, d, n, m, r, s, j, b, c, g?

Q Create an AVL tree with 70,60,80,50,65 and 68 how many leaves are there in the resultant tree?

a) 2 **b)** 3 **c)** 4 **D)** 5

Q What is the worst-case possible height of AVL tree?

(A) 2 Log_2 n **(B)** 1.44 log_2 n

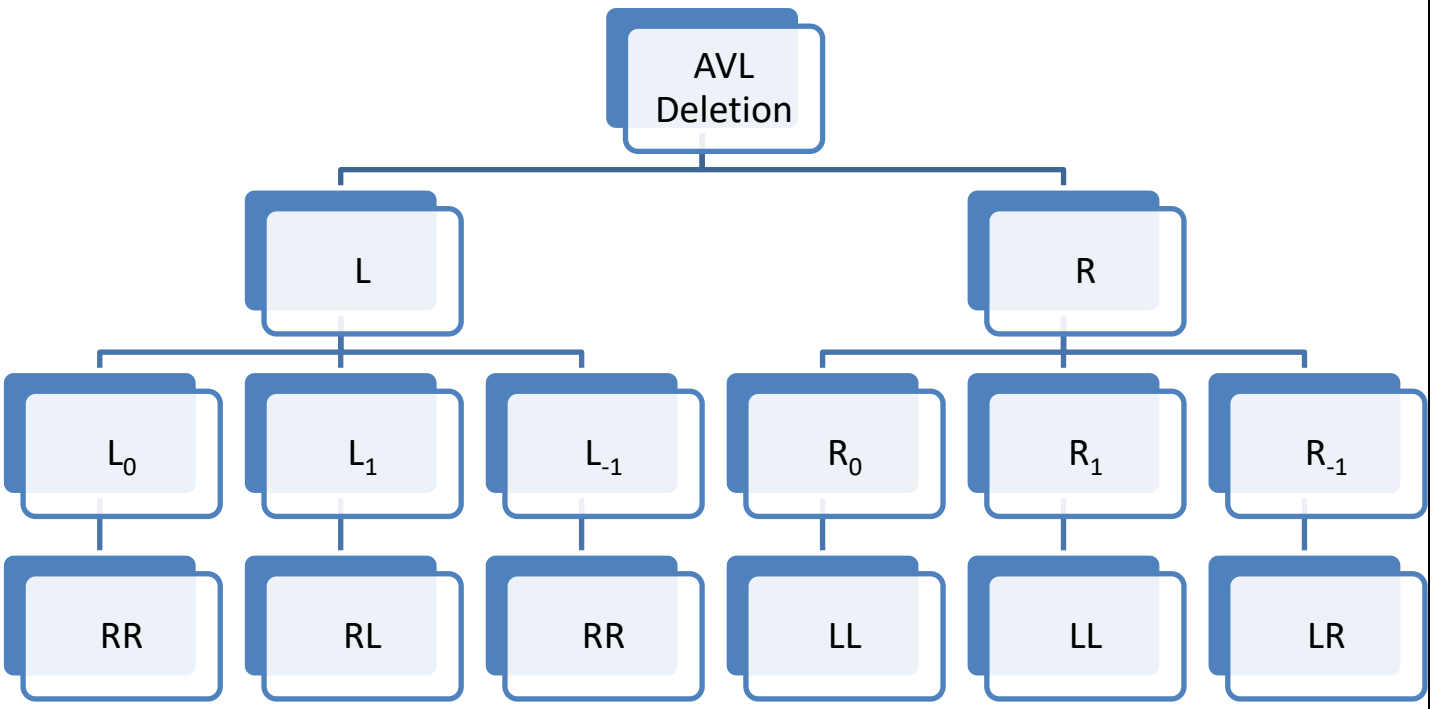
(C) Depends upon implementation **(D)** Theta(n)

Answer: (B)

Q What is the maximum height of any AVL-tree with 7 nodes? Assume that the height of a tree with a single node is 0? **(GATE-2009) (1 Marks)**

ANSWER 3

Deletion in an AVL tree



Q Delete the following nodes in sequence 2, 3, 10, 18, 4, 9, 14, 7, 15**?**

Q Insert the following keys in the order to build AVL tree: - A, Z, B, Y, C, X, D, W, E, V, F, the root of the resultant tree is

a) C **b) D** **c) E** **d) V**

Q from the above tree if A, Z, B, Y, C are deleted, then what will be the new root

a) C **b) D** **c) E** **d) V**

5.45 Which of the following is TRUE?

(a) The cost of searching an AVL tree is $\Theta(\log n)$ but that of a binary search tree is $O(n)$

(b) The cost of searching an AVL tree is $\Theta(\log n)$ but that of a complete binary tree is $\Theta(n \log n)$

(c) The cost of searching a binary search tree is $O(\log n)$ but that of an AVL tree is $\Theta(n)$

(d) The cost of searching an AVL tree is $\Theta(n \log n)$ but that of a binary search tree is $O(n)$

[2008 : 1 Mark]

Ans: a

Complete Binary Tree

- Consider a binary tree T, the maximum number of nodes at height h is 2^h nodes.
- The binary tree T is said to be complete binary tree, if all its level except possibly the last, have the maximum number of nodes and if all the nodes at the last level appear as far left as possible.
- One can easily determine the children and parent of a node k in any complete tree T
- Specially the left and right children of the node K are 2*k, 2*k + 1 and the parent of k is the node lower bound(k/2)

Q Let LASTPOST, LASTIN and LASTPRE denote the last vertex visited in a post order, inorder and preorder traversal, respectively, of a complete binary tree. Which of the following is always true? (GATE - 2000) (1 Marks)

(A) LASTIN = LASTPOST **(B)** LASTIN = LASTPRE
(C) LASTPRE = LASTPOST **(D)** None of the above
Answer: (D)

Q A scheme for storing binary trees in an array X is as follows. Indexing of X starts at 1 instead of 0. the root is stored at X[1]. For a node stored at X[i], the left child, if any, is stored in X[2i] and the right child, if any, in X[2i+1]. To be able to store any binary tree on n vertices the minimum size of X should be. (GATE - 2006) (2 Marks)

(A) $\log_2 n$ **(B)** n **(C)** 2n + 1 **(D)** $2^n - 1$
Answer: (D)

Heap

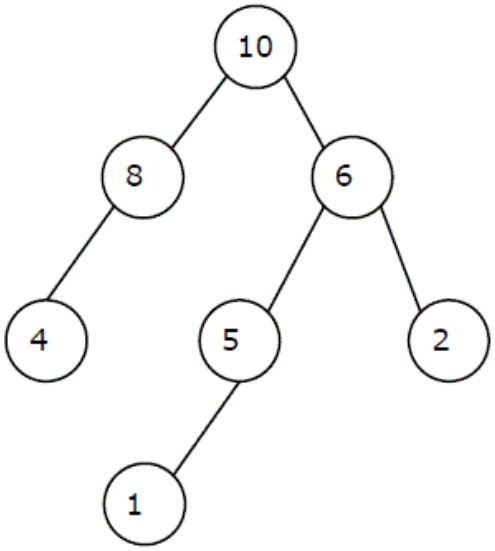
- Suppose H is a complete binary tree with n elements, H is called a Heap, if each node N of H has following properties:
  - The value of N is greater than to the value at each of the children of N then it is called Max heap.
  - A min heap is defined as the value at N is less than the value at any of the children of N.

Q Consider any array representation of an n element binary heap where the elements are stored from index 1 to index n of the array. For the element stored at index i of the array (i <= n), the index of the parent is (GATE - 2001) (1 Marks)

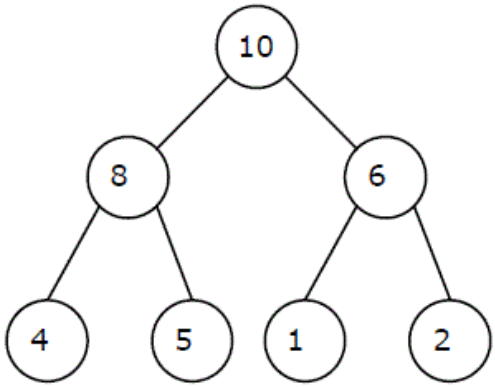
(A) i – 1 **(B)** floor(i/2) **(C)** ceiling(i/2) **(D)** (i+1)/2
Answer: (B)

Q A max-heap is a heap where the value of each parent is greater than or equal to the values of its children. Which of the following is a max-heap? (GATE - 2011) (2 Marks)

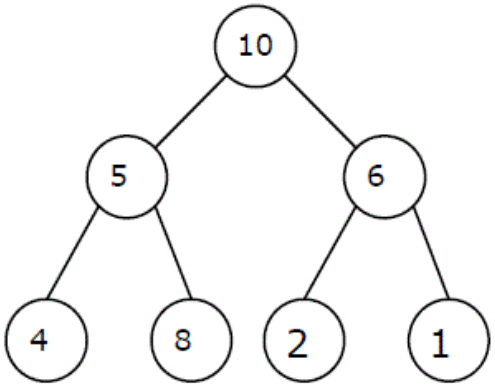
(A)



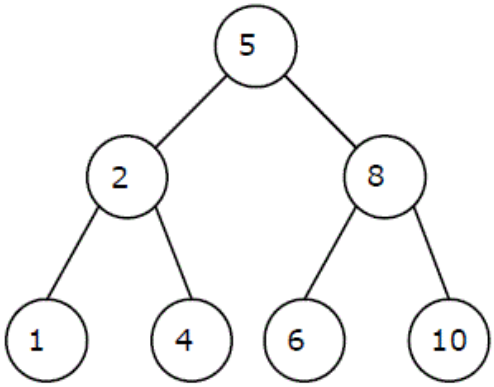
(B)



(C)



(D)



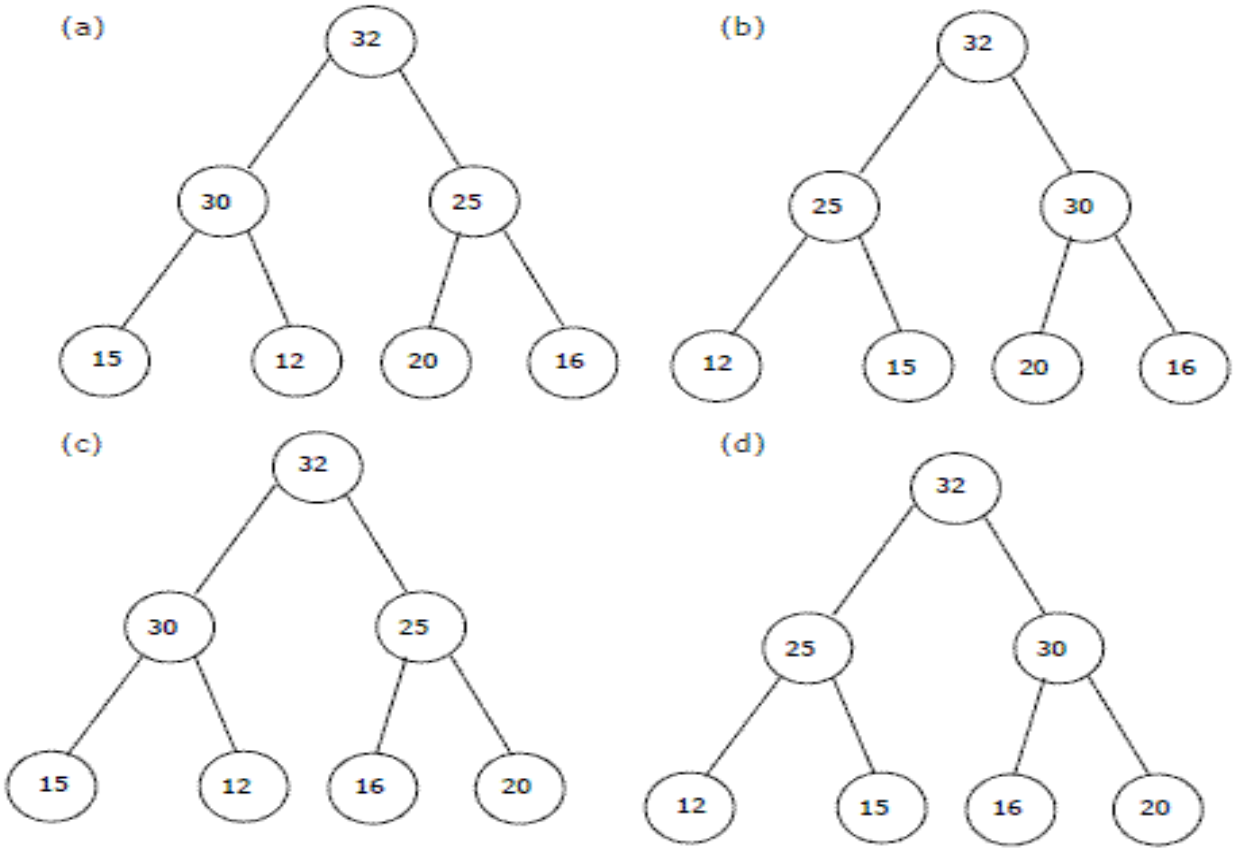
Answer: (B)

Q Consider a binary max-heap implemented using an array. Which one of the following arrays represents a binary max-heap? (GATE - 2006) (Marks)

(A) 23,17,14,6,13,10,1,12,7,5
(B) 23,17,14,6,13,10,1,5,7,12
(C) 23,17,14,7,13,10,1,5,6,12
(D) 23,17,14,7,13,10,1,12,5,7

Answer: (C)

Q The elements 32, 15, 20, 30, 12, 25, 16 are inserted one by one in the given order into a Max Heap. The resultant Max Heap is. (GATE - 2004) (2 Marks)



Answer: (A)

Q Consider the following array of elements. ⟨89, 19, 50, 17, 12, 15, 2, 5, 7, 11, 6, 9, 100⟩. The minimum number of interchanges needed to convert it into a max-heap is (GATE - 2015) (2 Marks)

(A) 4 **(B)** 5 **(C)** 2 **(D)** 3

Answer: (D)

Q A priority queue is implemented as a Max-Heap. Initially, it has 5 elements. The level-order traversal of the heap is: 10, 8, 5, 3, 2. Two new elements 1 and 7 are inserted into the heap in that order. The level-order traversal of the heap after the insertion of the elements is: (GATE - 2014) (2 Marks)

(A) 10, 8, 7, 3, 2, 1, 5 **(B)** 10, 8, 7, 2, 3, 1, 5

(C) 10, 8, 7, 1, 2, 3, 5 **(D)** 10, 8, 7, 5, 3, 2, 1

Answer: (A)

Q Consider a max heap, represented by the array: 40, 30, 20, 10, 15, 16, 17, 8, 4. Now consider that a value 35 is inserted into this heap. After insertion, the new heap is (Gate-2015) (2 Marks)

a) 40, 30, 20, 10, 15, 16, 17, 8, 4, 35 **b)** 40, 35, 20, 10, 30, 16, 17, 8, 4, 15

c) 40, 30, 20, 10, 35, 16, 17, 8, 4, 15 **d)** 40, 35, 20, 10, 15, 16, 17, 8, 4, 30

ANSWER B

Q Consider a binary max-heap implemented using an array. Which one of the following arrays represents a binary max-heap? (GATE - 2009) (2 Marks)

(A) 25,12,16,13,10,8,14 **(B)** 25,12,16,13,10,8,14

(C) 25,14,16,13,10,8,12 **(D)** 25,14,12,13,10,8,16

Answer: (C)

Q What is the content of the array after two delete operations on the correct answer to the previous question? (GATE - 2009) (2 Marks)

(A) 14,13,12,10,8 **(B)** 14,12,13,8,10

(C) 14,13,8,12,10 **(D)** 14,13,12,8,10

Answer: (D)

Q We have a binary heap on n elements and wish to insert n more elements (not necessarily one after another) into this heap. The total time required for this is (GATE - 2008) (1 Marks)

(A) Q(logn) (B) Q(n) (C) Q(nlogn) (D) Q(n^2)

Answer: (B)

Q In a heap with n elements with the smallest element at the root, the 7th smallest element can be found in time (GATE - 2008) (1 Marks)

a) (n log n) **b)** (n) **c)** (log n) **d)** (1)

Answer: (D)

Explanation: The 7th smallest element must be in first 7 levels.

Q In a binary max heap containing n numbers, the smallest element can be found in time (GATE - 2006) (1 Marks)

(A) 0(n) **(B)** O(logn) **(C)** 0(loglogn) **(D)** 0(1)

Answer: (A)

Q Given two max heaps of size n each, what is the minimum possible time complexity to make a one max-heap of size from elements of two max heaps?
(A) O(nLogn) **(B)** O(nLogLogn) **(C)** O(n) **(D)** O(nLogn)
Answer: (C)
Explanation: We can build a heap of 2n elements in O(n) time. Following are the steps. Create an array of size 2n and copy elements of both heaps to this array. Call build heap for the array of size 2n. Build heap operation takes O(n) time.

Q A complete binary min-heap is made by including each integer in [1, 1023] exactly once. The depth of a node in the heap is the length of the path from the root of the heap to that node. Thus, the root is at depth 0. The maximum depth at which integer 9 can appear is _____ (Gate-2016) (1 Marks)
ANSWER – 8

Q The number of possible min-heaps containing each value from {1, 2, 3, 4, 5, 6, 7} exactly once is _____. (Gate-2018) (1 Marks)
(ANSWER-80)

Q Which of the following Binary Min Heap operation has the highest time complexity?
(A) Inserting an item under the assumption that the heap has capacity to accommodate one more item
(B) Merging with another heap under the assumption that the heap has capacity to accommodate items of other heap
(C) Deleting an item from heap
(D) Decreasing value of a key
Answer: (B)
Explanation: The merge operation takes O(n) time, all other operations given in question take O(Logn) time.

Q in the following array representation of binary heap, how many nodes are in the right subtree of the root of the heap? 11,23,19,31,27,26,31,46,45,38,35,37,63,82,71,95?
a) 5 **b)** 6 **c)** 7 **d)** 8

Tree in general

Q How many distinct binary search trees can be created out of 4 distinct keys?
(A) 4 **(B)** 14 **(C)** 24 **(D)** 42
Answer: (B)

Q how many distinct BST can be constructed with 3 distinct keys?
A) 4 b) 5 c) 6 d) 9

Q The maximum number of binary trees that can be formed with three unlabelled nodes is:
(GATE-2007) (1 Marks)
a) 1 **b)** 5 **c)** 4 **d)** 3
ANSWER 5

Q We are given a set of n distinct elements and an unlabeled binary tree with n nodes. In how many ways can we populate the tree with the given set so that it becomes a binary search tree? (GATE - 2011) (2 Marks)
(A) 0 **(B)** 1 **(C)** n! **(D)** (1/(n+1)).2nCn
Answer: (B)

Q A complete n-ary tree is a tree in which each node has n children or no children. Let I be the number of internal nodes and L be the number of leaves in a complete n-ary tree. If L = 41, and I = 10, what is the value of n? (GATE - 2007) (2 Marks)
(A) 3 **(B)** 4 **(C)** 5 **(D)** 6
Answer: (C)

Q In a complete k-ary tree, every internal node has exactly k children or no child. The number of leaves in such a tree with an internal node is:
(A) nk **(B)** (n – 1) k+ 1 **(C)** n (k – 1) + 1 **(D)** n (k – 1)
Answer: (C)

5.11 A complete n-ary tree is one in which every node has 0 or n sons. If x is the number of internal nodes of a complete n-ary tree, the number of leaves in it is given by

(a) $x(n-1)+1$ (b) $xn-1$

(c) $xn+1$ (d) $x(n+1)$

[1998 : 2 Marks]

5.14 The number of leaf nodes in a rooted tree of n nodes, with each node having 0 or 3 children is

(a) $n/2$ (b) $(n-1)/2$

(c) $(n-1)/2$ (d) $\lceil (2n+1)/3 \rceil$

[2002 : 2 Marks]

5.69 A binary tree T has 20 leaves. The number of nodes in T having two children is _____.

[2015 (Set-2) : 1 Mark]

5.33 In a binary tree, the number of internal nodes of degree 1 is 5, and the number of internal nodes of degree 2 is 10. The number of leaf nodes in the binary tree is

(a) 10 (b) 11

(c) 12 (d) 15

[2006 : 1 Mark]

5.12 Consider the following nested representation of binary trees: (X Y Z) indicates Y and Z are the left and right subtrees, respectively, of node X. Note that Y and Z may be NULL, or further nested. Which of the following represents a valid binary tree?

(a) (1 2 (4 5 6 7)) (b) (1 ((234) 56)7)

(c) (1(234) (567)) (d) (1(23NULL) (45))

[2000 : 1 Mark]

5.10 Which of the following statements is false?

(a) A tree with n nodes has $(n - 1)$ edges

(b) A labeled rooted binary tree can be uniquely constructed given its postorder and preorder traversal results.

(c) A complete binary tree with n internal nodes has $(n + 1)$ leaves

(d) The maximum number of nodes in a binary tree of height h is $(2^{h+1} - 1)$

[1998 : 1 Mark]

5.18 A data structure is required for storing a set of integers such that each of the following operations can be done in O(log n) time, where n is the number of elements in the set.

1. Deletion of the smallest element.
2. Insertion of an element if it is not already present in the set.

Which of the following data structures can be used for this purpose?

(a) A heap can be used but not a balanced binary search tree

(b) A balanced binary search tree can be used but not a heap

(c) Both balanced binary search tree and heap can be used

(d) Neither balanced binary search tree nor heap can be used

[2003 : 2 Marks]

Q What are the main applications of tree data structure?

a) Manipulate hierarchical data

b) Make information easy to search (see tree traversal).

c) Form of a multi-stage decision-making, like Chess Game

d) all

Ans: d

Q Let T(n) be the number of different binary search trees on n distinct elements. Then (GATE - 2003) (1 Marks)

$$T(n)=\sum_{k=1}^{n} T(k-1)T(x)$$

where x is

(A) n-k+1 **(B)** n-k **(C)** n-k-1 **(D)** n-k-2

Answer: (B)

Explanation: The idea is to make a key root, put (k-1) keys in one subtree and remaining n-k keys in another subtree.

Q Which data structure is most efficient to find the top 10 largest items out of 1 million items stored in file?

(A) Min heap **(B)** Max heap **(C)** BST **(D)** Sorted array

Answer: (A)

Q A data structure is required for storing a set of integers such that each of the following operations can be done in (log n) time, where n is the number of elements in the set. **(GATE - 2003) (2 Marks)**

- Deletion of the smallest element
- Insertion of an element if it is not already present in the set

Which of the following data structures can be used for this purpose?

(A) A heap can be used but not a balanced binary search tree

(B) A balanced binary search tree can be used but not a heap

(C) Both balanced binary search tree and heap can be used

(D) Neither balanced binary search tree nor heap can be used

Answer: (B)

Let T be a full binary tree with 8 leaves. (A full binary tree has every level full.) Suppose two leaves a and b of T are chosen uniformly and independently at random. The expected value of the distance between a and b in T (i.e., the number of edges in the unique path between a and b) is (rounded off to 2 decimal places) _____.

(GATE-2019) (2 Marks)

Q Consider a rooted Binary tree represented using pointers. The best upper bound on the time required to determine the number of subtrees having exactly 4 nodes O(na Logn b). Then the value of a + 10b is _____ **(GATE-2015) (1 Marks)**

Answer 1

5.4 A binary tree T has n leaf nodes. The number of nodes of degree 2 in T is

(a) $\log_2 n$ (b) $n - 1$

(c) n (d) 2^n

[1995 : 1 Mark]

Linked Questions 5.49 & 5.50:

A binary tree with $n > 1$ nodes has n_1, n_2 and n_3 nodes of degree one, two and three respectively. The degree of a node is defined as the number of its neighbours.

5.49 n_3 can be expressed as:

(a) $n_1 + n_2 - 1$ (b) $n_1 - 2$

(c) $[((n_1 + n_2)/2)]$ (d) $n_2 - 1$

[2008 : 2 Marks]

5.50 Starting with the above tree, while there remains a node v of degree two in the tree, add an edge between the two neighbours of v and then remove v from the tree.

How many edges will remain at the end of the process?

(a) $2 * n_1 - 3$ (b) $n_2 + 2 * n_1 - 2$

(c) $n_3 - n_2$ (d) $n_2 + n_1 - 2$

[2008 : 2 Marks]

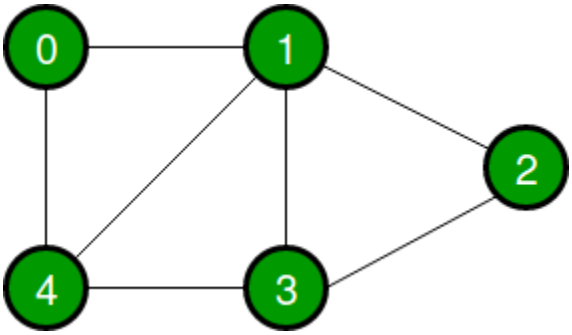
Graph

- Graph is a data structure that consists of following two components:
  - A finite set of vertices also called as nodes.
  - A finite set of ordered pair of the form (u, v) called as edge. The pair is ordered because (u, v) is not same as (v, u) in case of a directed graph(di-graph). The pair of the form (u, v) indicates that there is an edge from vertex u to vertex v. The edges may contain weight/value/cost.
- Graphs are used to represent many real-life applications: Graphs are used to represent networks. The networks may include paths in a city or telephone network or circuit network.
- Graphs are also used in social networks like LinkedIn, Facebook. For example, in Facebook, each person is represented with a vertex (or node). Each node is a structure and contains information like person id, name, gender and locale.

Representation of Graph in Memory

- Following two are the most commonly used representations of a graph.
  - Adjacency Matrix
  - Adjacency List
- There are other representations also like, Incidence Matrix and Incidence List. The choice of the graph representation is situation specific. It totally depends on the type of operations to be performed and ease of use.

- **Adjacency Matrix:** Adjacency Matrix is a 2D array of size V x V where V is the number of vertices in a graph. Let the 2D array be adj[][], a slot adj[i][j] = 1 indicates that there is an edge from vertex i to vertex j.
- Adjacency matrix for undirected graph is always symmetric.
- Adjacency Matrix is also used to represent weighted graphs. If adj[i][j] = w, then there is an edge from vertex i to vertex j with weight w.

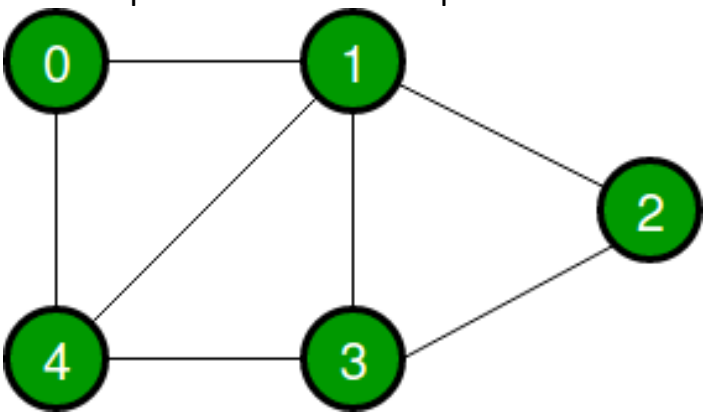


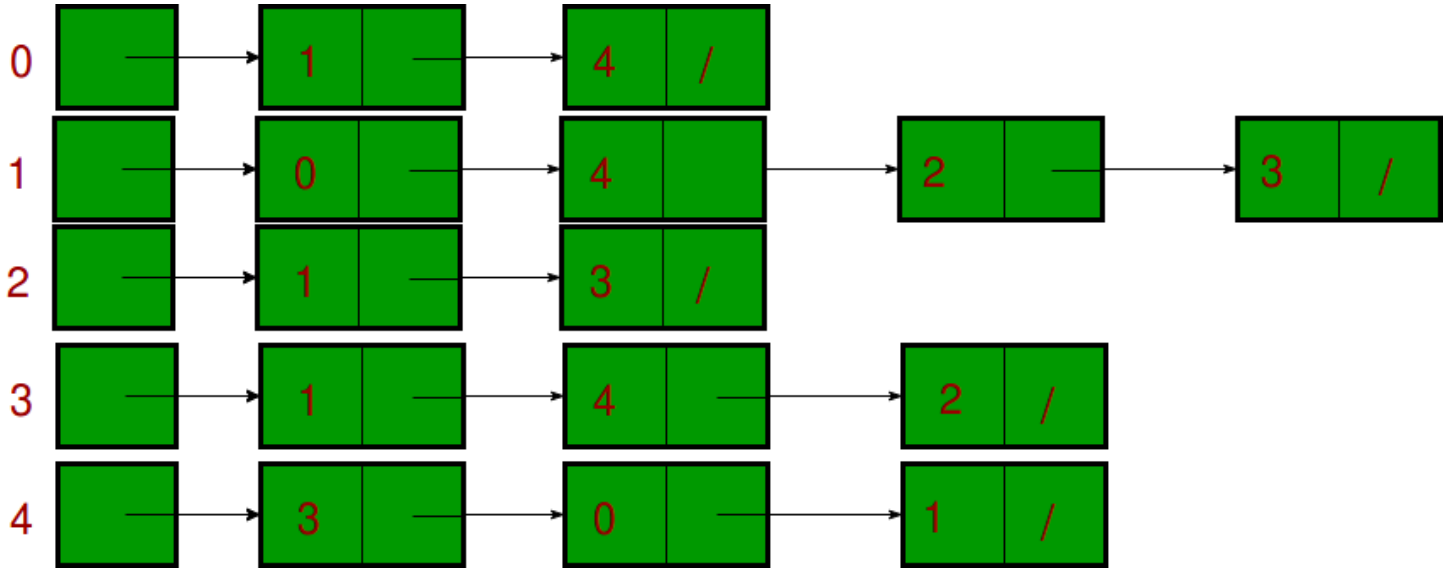
	0	1	2	3	4
0	0	1	0	0	1
1	1	0	1	1	1
2	0	1	0	1	0
3	0	1	1	0	1
4	1	1	0	1	0

- *Pros:* Representation is easier to implement and follow. Removing an edge takes O(1) time. Queries like whether there is an edge from vertex ‘u’ to vertex ‘v’ are efficient and can be done O(1).

- *Cons:* Consumes more space O(V^2). Even if the graph is sparse(contains less number of edges), it consumes the same space. Adding a vertex is O(V^2) time.
  Please see this for a sample Python implementation of adjacency matrix.

- **Adjacency List:**
  An array of lists is used. Size of the array is equal to the number of vertices. Let the array be array[]. An entry array[i] represents the list of vertices adjacent to the ***i***th vertex. This representation can also be used to represent a weighted graph. The weights of edges can be represented as lists of pairs.





Graph Traversal

- Traversal means visiting all the nodes of a graph.
- Depth First Traversal (or Search) for a graph is similar to Depth First Traversal of a tree. The only catch here is, unlike trees, graphs may contain cycles, so we may come to the same node again. To avoid processing a node more than once, we use a Boolean visited array.

- A standard DFS implementation puts each vertex of the graph into one of two categories:
  - Visited
  - Not Visited
- The purpose of the algorithm is to mark each vertex as visited while avoiding cycles.

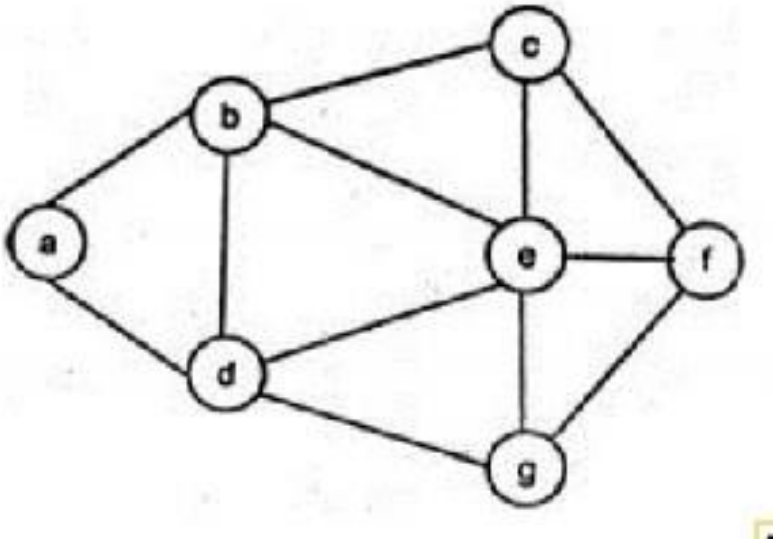
- The DFS algorithm works as follows:
  - Start by putting any one of the graph's vertices on top of a stack.
  - Take the top item of the stack and add it to the visited list.
  - Create a list of that vertex's adjacent nodes. Add the ones which aren't in the visited list to the top of stack.
  - Keep repeating steps 2 and 3 until the stack is empty.

```
DFS(G, v)
{
        visited(v) = 1
        For all x adjacent to v
        {
                if (x is not visited)
                        DFS(v)
        }
}


DFS-iterative (G, s):                          //Where G is graph and s is source vertex
{
        let S be stack
        S.push( s )          //Inserting s in stack
        mark s as visited
        while ( S is not empty)
        {
                v = S.top( )
                S.pop( ) //Pop a vertex from stack to visit next
                for all neighbours w of v in Graph G:
                {
                        if w is not visited:
                        S.push( w )       //Push all the neighbors of v in stack that are not visited
                        mark w as visited

                }
        }
}
```

Q Consider the following sequence of nodes for the undirected graph given below.

1) a b e f d g c **2)** a b e f c g d **3)** a d g e b c f **4)** a d b c g e f

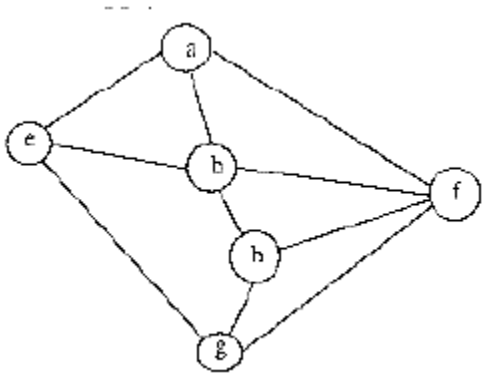
A Depth First Search (DFS) is started at node a. The nodes are listed in the order they are first visited. Which all of the above is (are) possible output(s)? **(Gate-2008) (2 Marks)**



(A) 1 and 3 only **(B)** 2 and 3 only **(C)** 2, 3 and 4 only **(D)** 1, 2, and 3

Answer: (B)

Q Consider the following graph



Among the following sequences

I) a b e g h f **II)** a b f e h g **III)** a b f h g e **IV)** a f g h b e

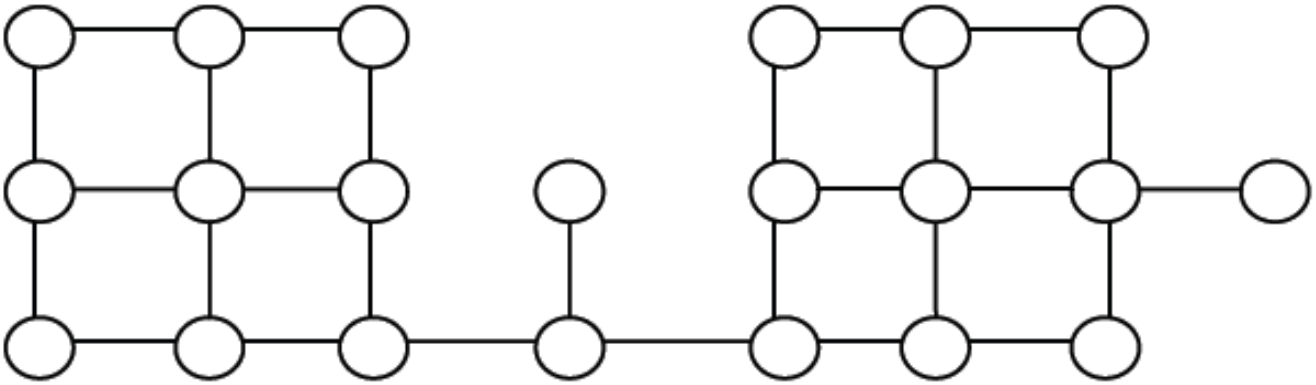
Which are depth first traversals of the above graph? **(GATE-2003) (1 Marks)**

(A) I, II and IV only **(B)** I and IV only

(C) II, III and IV only **(D)** I, III and IV only

Answer: (D)

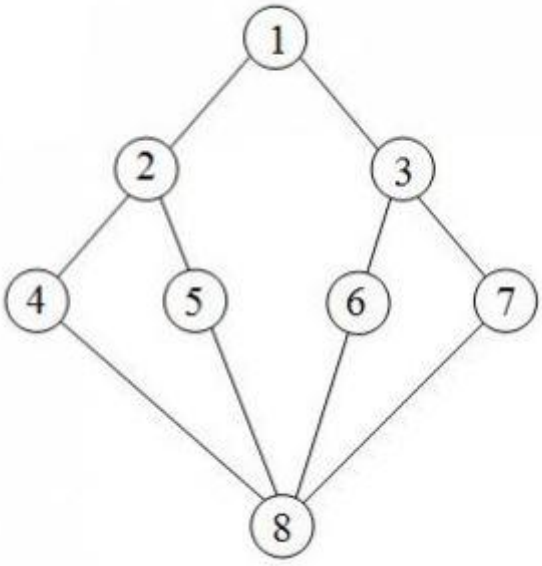
Q Suppose depth first search is executed on the graph below starting at some unknown vertex. Assume that a recursive call to visit a vertex is made only after first checking that the vertex has not been visited earlier. Then the maximum possible recursion depth (including the initial call) is _____. **(Gate-2014) (2 Marks)**



(A) 17 **(B)** 18 **(C)** 19 **(D)** 20

Answer: (C)

Q Which of the following are valid and invalid DFS traversal sequence



a) 1, 3, 7, 8, 5, 2, 4, 6 b) 1, 2, 5, 8, 6, 3, 7, 4

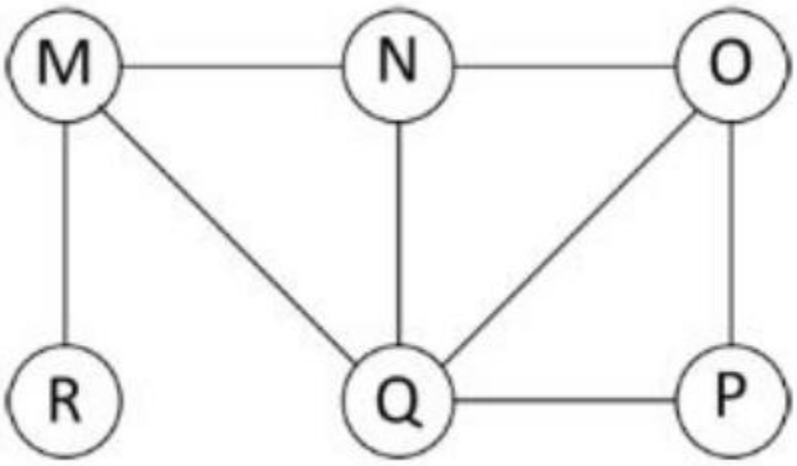
c) 1, 3, 6, 7, 8, 5, 2, 4 d) 1, 2, 4, 5, 8, 6, 7, 3

Breadth First Traversal (or Search)

Breadth First Traversal (or Search) for a graph is similar to Breadth First Traversal of a tree. The only catch here is, unlike trees, graphs may contain cycles, so we may come to the same node again. To avoid processing a node more than once, we use a Boolean visited array. For simplicity, it is assumed that all vertices are reachable from the starting vertex, i.e. the graph is connected

```
BFS(v)
{
        visited(v) = 1
        insert[V,Q]
        While(Q != Phi)
        {
                u = Delete(Q);
                for all x adjacent to u
                {
                        if (x is not visited)
                        {
                                visited(x) = 1
                                insert(x,Q)
                        }
                }
        }
}
```

Q Breath First Search (BFS) has been implemented using queue data structure.



Which one of the following is a possible order of visiting the nodes in the graph above? **(Gate-2017) (1 Marks)**

a) MNOPQR **b)** NQMPOR **c)** QMNROP **d**) POQNMR

Ans: d

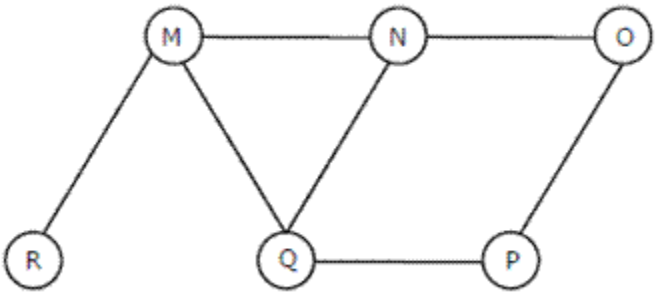
Q Level order traversal of a rooted tree can be done by starting from the root and performing **(Gate-2004) (1 Marks)**

(A) preorder traversal **(B)** inorder traversal

(C) depth first search **(D)** breadth first search

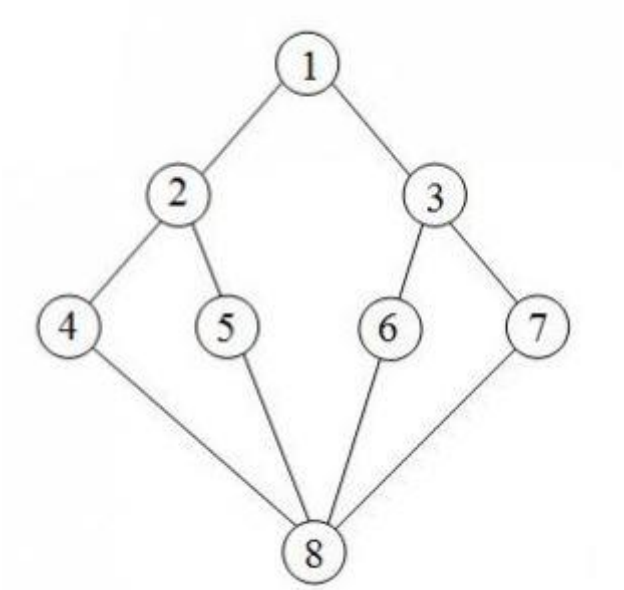
Answer: (D)

Q The Breadth First Search algorithm has been implemented using the queue data structure. One possible order of visiting the nodes of the following graph is (Gate-2008) (1 Marks)



(A) MNOPQR **(B)** NQMPOR **(C)** QMNPRO **(D)** QMNPOR
Answer: (C)

Q Which of the following are valid and invalid BFS traversal sequence



a) 1, 3, 2, 5, 4, 7, 6, 8
b) 1, 3, 2, 7, 6, 4, 5, 8
c) 1, 2, 3, 5, 4, 7, 6, 8
d) 1, 2, 3, 7, 5, 6, 4, 8

Q Let G be an undirected graph. Consider a depth-first traversal of G, and let T be the resulting depth-first search tree. Let u be a vertex in G and let v be the first new (unvisited) vertex visited after visiting u in **the traversal. Which of the following statements is always true? (GATE-2000) (2 Marks)**
(A) {u,v} must be an edge in G, and u is a descendant of v in T
(B) {u,v} must be an edge in G, and v is a descendant of u in T
(C) If {u,v} is not an edge in G then u is a leaf in T
(D) If {u,v} is not an edge in G then u and v must have the same parent in T
Answer: (C)

Q Let G be a graph with n vertices and m edges. What is the tightest upper bound on the running time on Depth First Search of G? Assume that the graph is represented using adjacency matrix.**(Gate-2014) (1 Marks)**

(A) O(n) **(B)** O(m+n) **(C)** $O(n^2)$ **(D)** O(mn)
Answer: (C)

Q In an adjacency list representation of an undirected simple graph G = (V, E), each edge (u, v) has two adjacency list entries: [v] in the adjacency list of u, and [u] in the adjacency list of v. These are called twins of each other. A twin pointer is a pointer from an adjacency list entry to its twin. If |E| = m and |V | = n, and the memory size is not a constraint, what is the time complexity of the most efficient algorithm to set the twin pointer in each entry in each adjacency list?(Gate-2016) (2 Marks)
(A) $\Theta(n^2)$ **(B)** $\Theta(m+n)$ **(C)** $\Theta(m^2)$ **(D)** $\Theta(n^4)$
Answer: (B) .

Q Consider the tree arcs of a BFS traversal from a source node W in an unweighted, connected, undirected graph. The tree T formed by the tree arcs is a data structure for computing. **(Gate-2014) (1 Marks)**
(A) the shortest path between every pair of vertices.
(B) the shortest path from W to every vertex in the graph.
(C) the shortest paths from W to only those nodes that are leaves of T.
(D) the longest path in the graph
Answer: (B)

Q Let T be a depth first search tree in an undirected graph G. Vertices u and n are leaves of this tree T. The degrees of both u and n in G are at least 2. which one of the following statements is true? **(Gate-2006) (2 Marks)**
(A) There must exist a vertex w adjacent to both u and n in G
(B) There must exist a vertex w whose removal disconnects u and n in G
(C) There must exist a cycle in G containing u and n
(D) There must exist a cycle in G containing u and all its neighbours in G.
Answer: (B)

Q The most efficient algorithm for finding the number of connected components in an undirected graph on n vertices and m edges has time complexity. (Gate-2008)(1 Marks)

(A) (n) **(B)** (m) **(C)** (m + n) **(D)** (mn)

Answer: (C) **Explanation:**

Q Consider an undirected unweighted graph G. Let a breadth-first traversal of G be done starting from a node r. Let d(r, u) and d(r, v) be the lengths of the shortest paths from r to u and v respectively, in G. lf u is visited before v during the breadth-first traversal, which of the following statements is correct? **(GATE-2001) (2 Marks)**

(A) d(r, u) < d (r, v) **(B)** d(r, u) > d(r, v) **(C)** d(r, u) <= d (r, v) **(D)** None of the above

Answer: (C)

Q How many undirected graphs (not necessarily connected) can be constructed out of a given set V= {V 1, V 2,…V n} of n vertices ?

(A) n(n-l)/2 **(B)** 2^n **(C)** n! **(D)** 2^(n(n-1)/2)

Answer: (D)

Q Which of the following is an advantage of adjacency list representation over adjacency matrix representation of a graph?

(A) In adjacency list representation, space is saved for sparse graphs.

(B) DFS and BSF can be done in O(V + E) time for adjacency list representation. These operations take O(V^2) time in adjacency list representation. Here is V and E are number of vertices and edges respectively.

(C) Adding a vertex in adjacency list representation is easier than adjacency matrix representation.

(D) All of the above

Answer: (D)

Q Traversal of a graph is different from tree because

(A) There can be a loop in graph so we must maintain a visited flag for every vertex

(B) DFS of a graph uses stack, but inorrder traversal of a tree is recursive

(C) BFS of a graph uses queue, but a time efficient BFS of a tree is recursive.

(D) All of the above
Answer: (A)

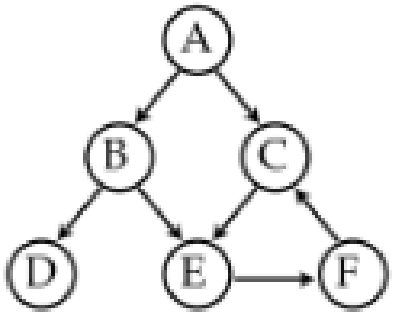
Q Given two vertices in a graph s and t, which of the two traversals (BFS and DFS) can be used to find if there is path from s to t?
(A) Only BFS **(B)** Only DFS **(C)** Both BFS and DFS **(D)** Neither BFS nor DFS
Answer: (C)

Q Let G be a simple undirected graph. Let TD be a depth first search tree of G. Let TB be a breadth first search tree of G. Consider the following statements. (I) No edge of G is a cross edge with respect to TD. (A cross edge in G is between two nodes neither of which is an ancestor of the other in TD). (II) For every edge (u, v) of G, if u is at depth i and v is at depth j in TB, then |i − j| = 1. Which of the statements above must necessarily be true? (Gate-2018) (2 Marks)
a) I only **b)** II only **c)** Both I and II **d)** Neither I nor II
(ANSWER-A)

Q Consider the tree arcs of a BFS traversal from a source node W in an unweighted, connected, undirected graph. The tree T formed by the tree arcs is a data structure for computing. **(Gate-2014) (2 Marks)**
a) the shortest path between every pair of vertices.
b) the shortest path from W to every vertex in the graph.
c) the shortest paths from W to only those nodes that are leaves of T.
d) the longest path in the graph
ANSWER B

Q (UGC - June – 2007)

Depth ion travels of the following directed graph is :



(A) A B C D E F
(B) A B D E F C
(C) A C E B D F
(D) None of the above

Q Which of the following statements is false? **(UGC - Jan – 2017)**
(A) Optimal binary search tree construction can be performed efficiently using dynamic programming.
(B) Breadth-first search cannot be used to find connected components of a graph.
(C) Given the prefix and postfix walks of a binary tree, the tree cannot be re-constructed uniquely.
(D) Depth-first-search can be used to find the connected components of a graph.

Q Breadth First Search (BFS) is started on a binary tree beginning from the root vertex. There is a vertex t at a distance four from the root. If t is the n-th vertex in this BFS traversal, then the maximum possible value of n is _____ **(Gate-2016) (2 Marks)**
Ans: 31

Q Let G = (V, E) be a simple undirected graph, and s be a particular vertex in it called the source. For x ∈ V, let d(x) denote the shortest distance in G from s to x. A breadth first search (BFS) is performed starting at s. Let T be the resultant BFS tree. If (u, v) is an edge of G that is not in T, then which one of the following CANNOT be the value of d(u) – d(v)? **(Gate-2015) (2 Marks)**
a) -1 **b) 0** **c) 1** **d) 2**
Ans: d