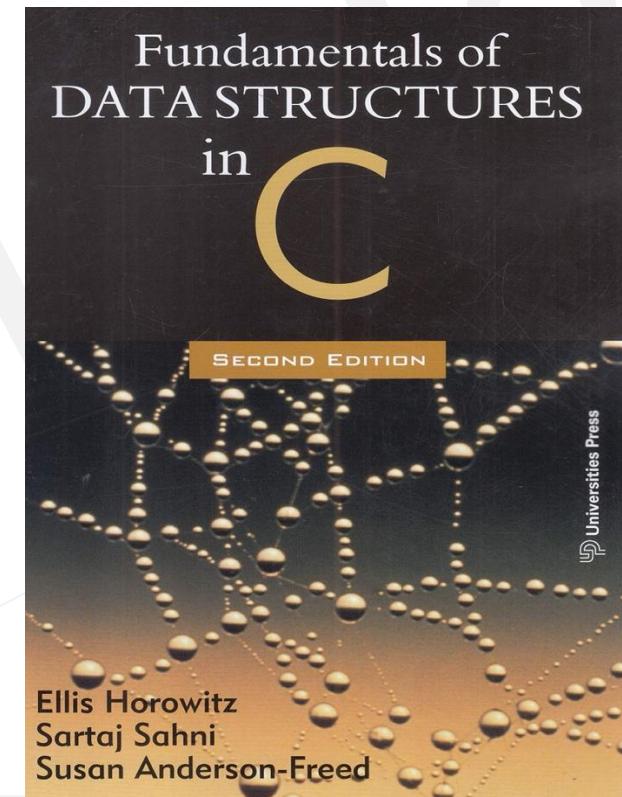
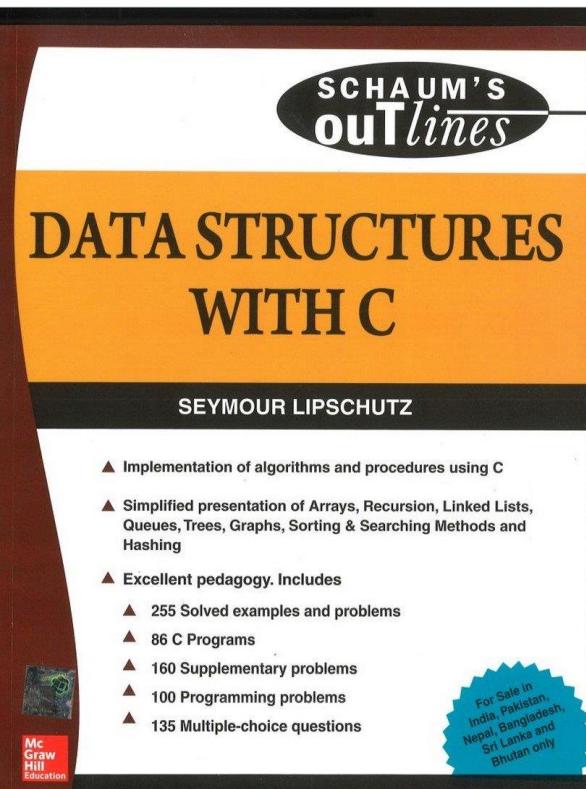


Data Structure

- Core subjects for CS/IT Students
- In GATE 7-8 Marks out of 100 Marks, and 5-6 questions on an average
- In NET 14-15 Marks out of 200 marks and 8-10 questions
- Both parts are important Theory and Numerical
- Needs a little time, good scoring
- Very important in Industry



Syllabus

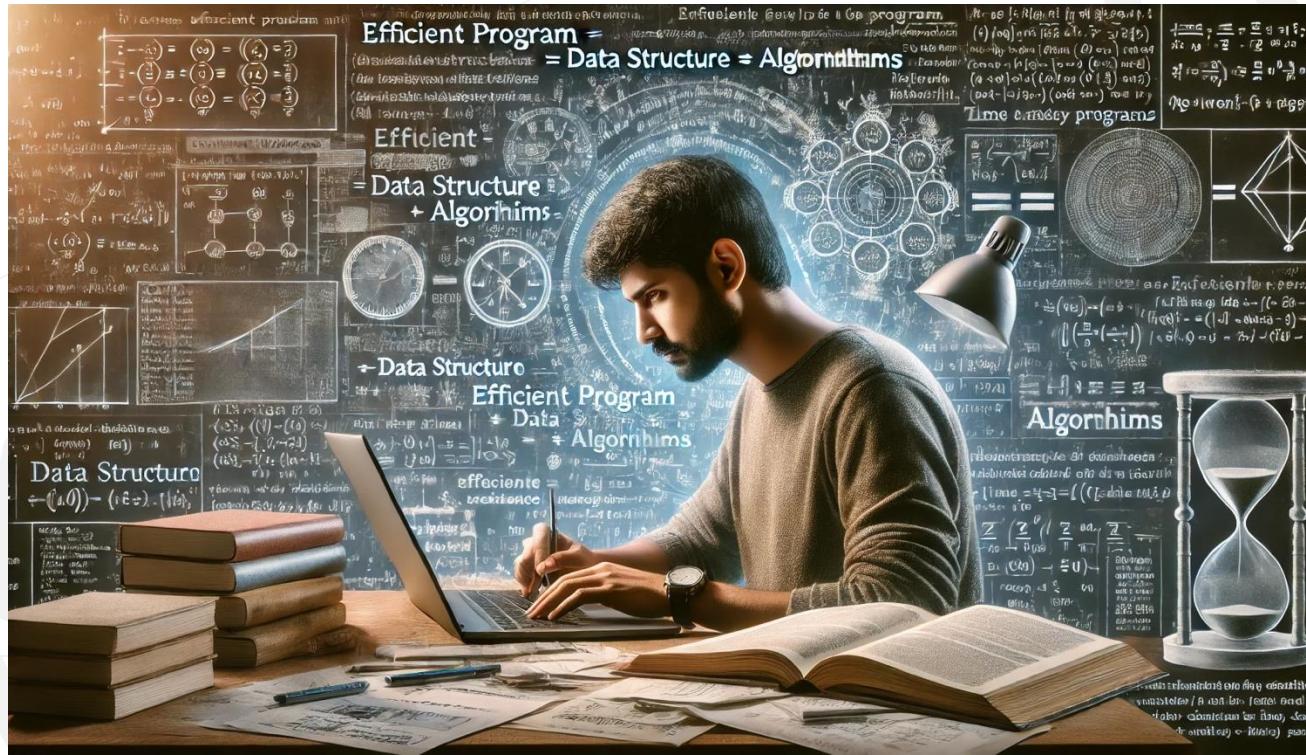
- Arrays, Stacks, Queues, Linked lists, Trees, Binary search trees, Binary heaps, Graphs

Section 4: Programming and Data Structures

Programming in C. Recursion. Arrays, stacks, queues, linked lists, trees, binary search trees, binary heaps, graphs.

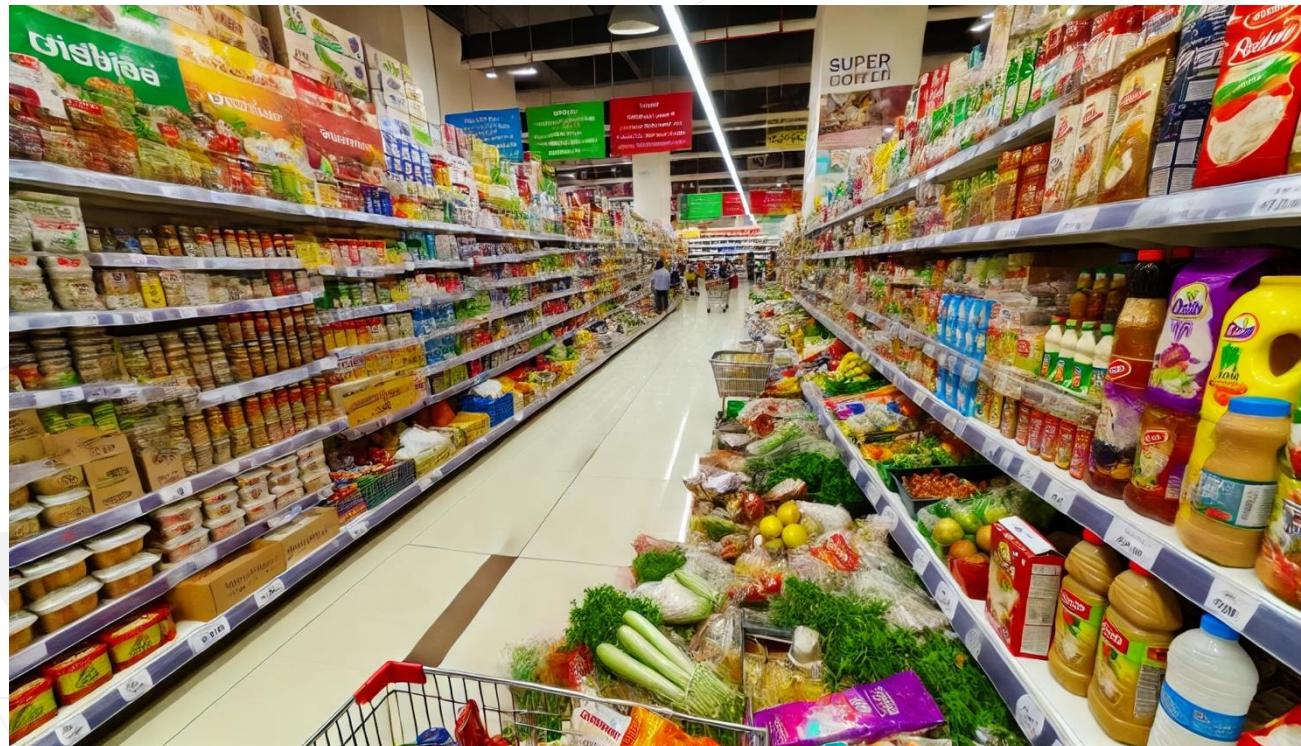
Idea of computer science

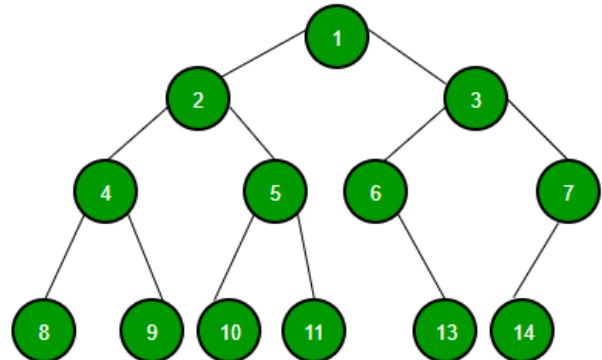
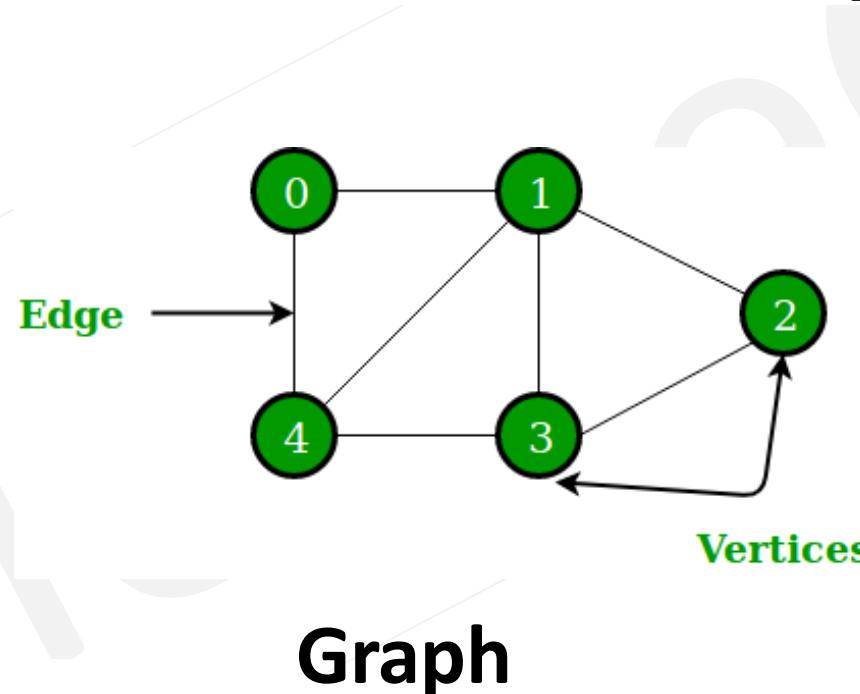
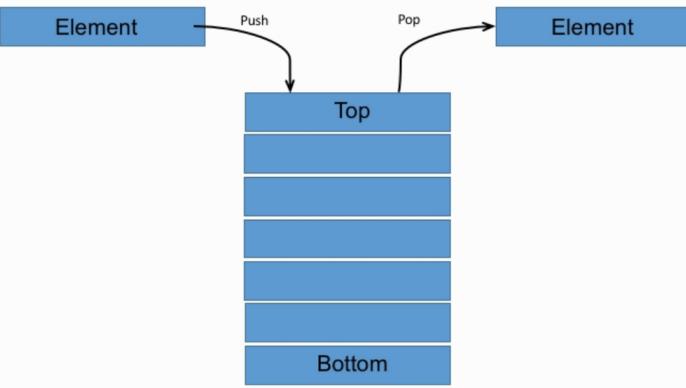
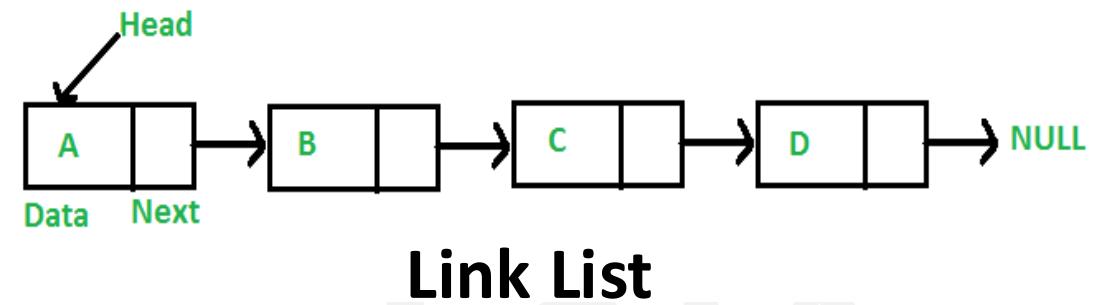
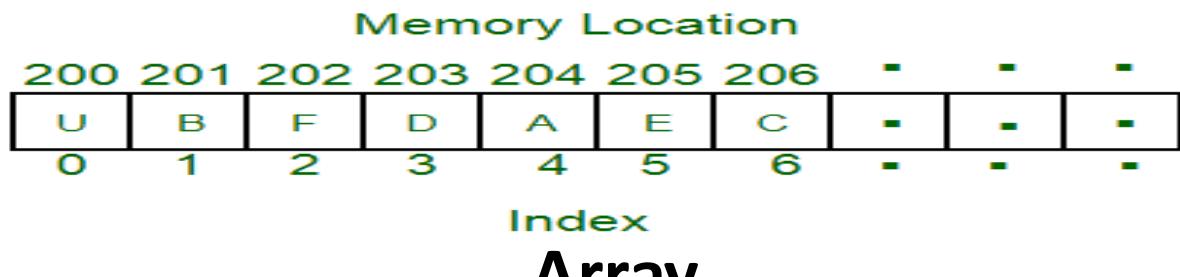
- Computer science is about solving problems by designing **algorithms** that can be converted into efficient programs. The key challenge is optimizing for **time** and **memory**.
- To write an efficient program, you need both:
 - **Data Structures:** The way you organize data.
 - **Algorithms:** The process to solve the problem using organized data.
 - **Efficient Program = Data Structure + Algorithm**



What is data structure

- **Data structure** is a particular way of organizing data in a computer memory, so that Memory can be used efficiently both in terms of time and space. It is a logical relationship existing between individual elements of data, it considers elements stored and also their relationship to each other. Data structure mainly specifies the following four things: -
 - Organization of data, Accessing methods, Degree of association, Processing methods

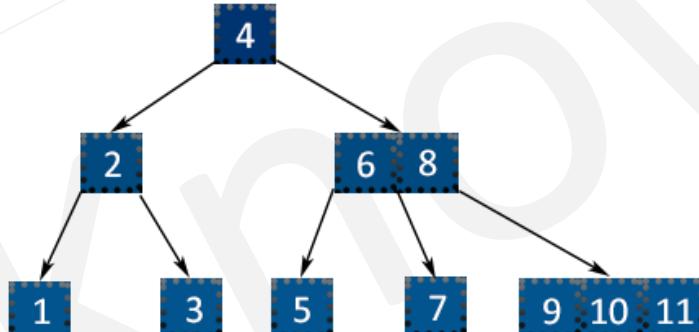




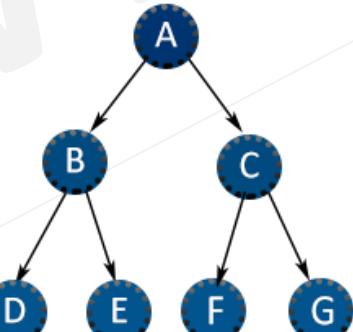
The Impact of Data Structures on Programs

- Data structures affect both the **structural** and **functional** aspects of a program. The choice of data structure plays a crucial role in program design and efficiency.
- **Specialized Data Structures for Different Applications**
 - Relational Databases: Often use **B-tree indexes** for efficient data retrieval.
 - Compiler Implementations: Commonly rely on **hash tables** for fast identifier lookups.
- Data Structures & Algorithm Efficiency: Efficient data structures are essential to designing efficient algorithms. In some programming paradigms, data structures are emphasized over algorithms as the primary organizing factor in software design.
- Implementing Data Structures: The implementation of a data structure typically involves creating a set of **procedures** to instantiate and manipulate the structure, ensuring efficient use and access.

B-TREE

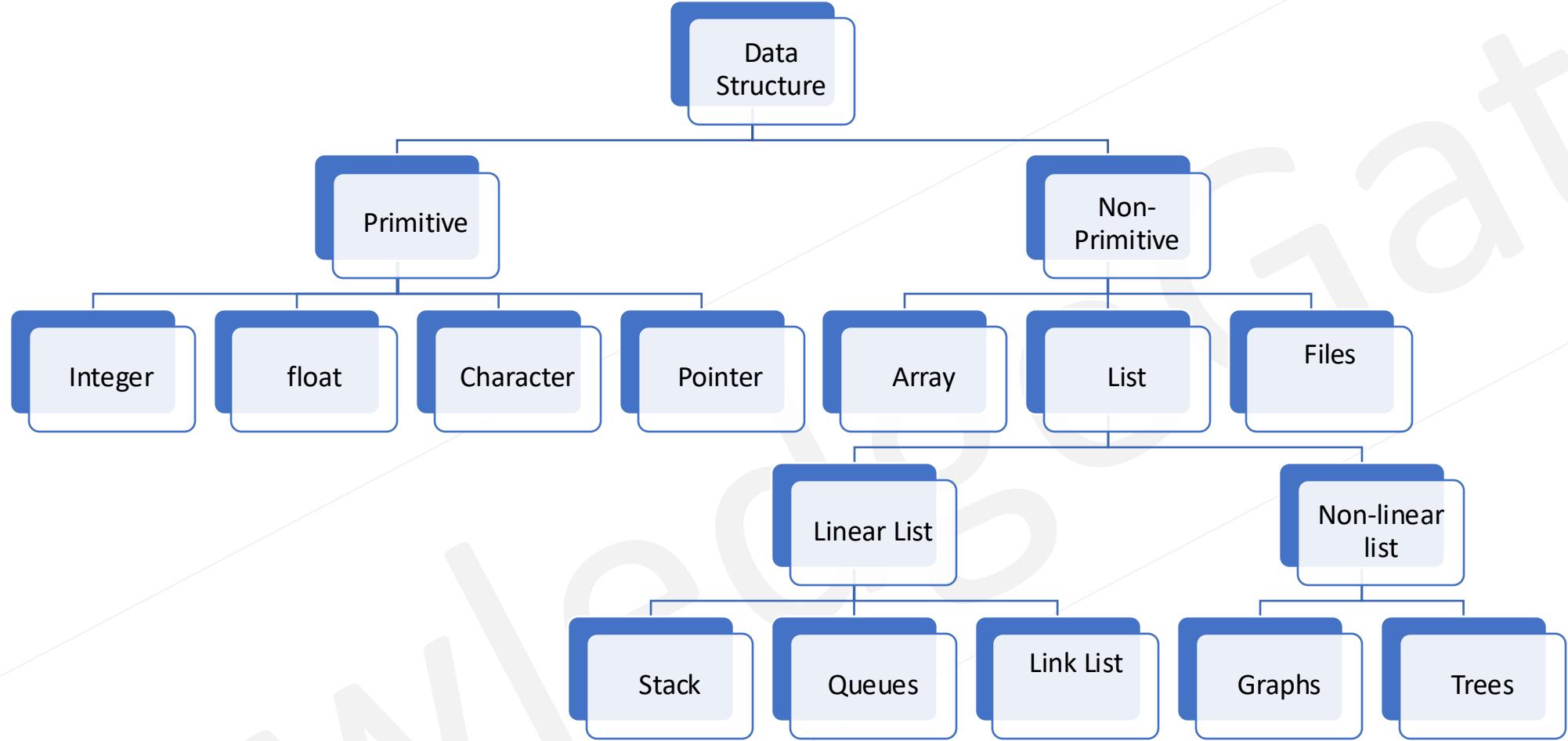


BINARY TREE



Primitive and Non-Primitive Data Structures

- **Primitive Data Structures**
 - **Definition:** Primitive data structures are predefined by the system and are directly operated upon by machine instructions.
 - **Examples:** char, int, float, double.
 - **Operations:** The set of operations (like addition, subtraction) on these data types is predefined and can be directly executed by the system.
- **Non-Primitive Data Structures**
 - When primitive data structures are insufficient, we use **derived** and **user-defined** data structures.
 - **Derived Data Structures**
 - **Definition:** Derived data structures are built using primitive data types (e.g., arrays). They are provided by the system but their structure is more complex.
 - **Examples:** Arrays (e.g., array of int, array of char).
 - **Operations:** Predefined operations, similar to primitives, are available for manipulation.
 - **User-Defined Data Structures**
 - **Definition:** Created by the user using primitive and derived types, with custom structures and operations.
 - **Examples:** Linked Lists, Trees, Stacks, Queues.
 - **Operations:** The user defines the operations to perform on these structures using constructs like structures or classes.



LINEAR DATA STRUCTURE

- In a linear data structure, data elements are arranged in a linear order where each and every element are attached to its previous and next adjacent.
- In linear data structure, single level is involved.
- Its implementation is easy in comparison to non-linear data structure.
- In linear data structure, data elements can be traversed in a single run only.
- Its examples are: array, stack, queue, linked list, etc.

NON-LINEAR DATA STRUCTURE

- In a non-linear data structure, data elements are attached in hierarchical manner.
- Whereas in non-linear data structure, multiple levels are involved.
- While its implementation is complex in comparison to linear data structure.
- While in non-linear data structure, data elements can't be traversed in a single run only.
- While its examples are: trees and graphs.

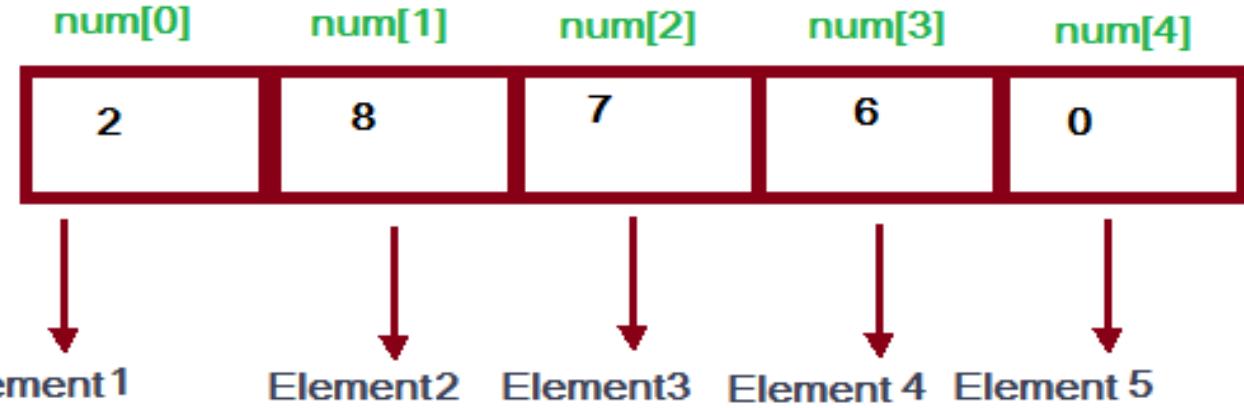
Homogeneous vs. Heterogeneous Data Structures

- **Homogeneous Data**: These data structures store elements of the **same type**, such as integers or floats. A common example is an **array**, where all elements share the same data type.
- **Heterogeneous Data**: These data structures can hold elements of **different types**. For example, a data structure may store integers, floats, and characters together. Examples include **structures** and **unions**.

Array

- An array is a data structure that stores collection of elements of same type stored at contiguous memory locations and can be accessed using an index.

int num[5];



And the award for
Best Answer Ever goes to

INTERVAL - II

Computer programming

1. Define Array.

Ams: * Array:

An Array is used to call a boy or a person who is at a distance far away from us who are visible to our naked eye.

Example: Array Rupesh.

Meet Me

How to declare an array in C

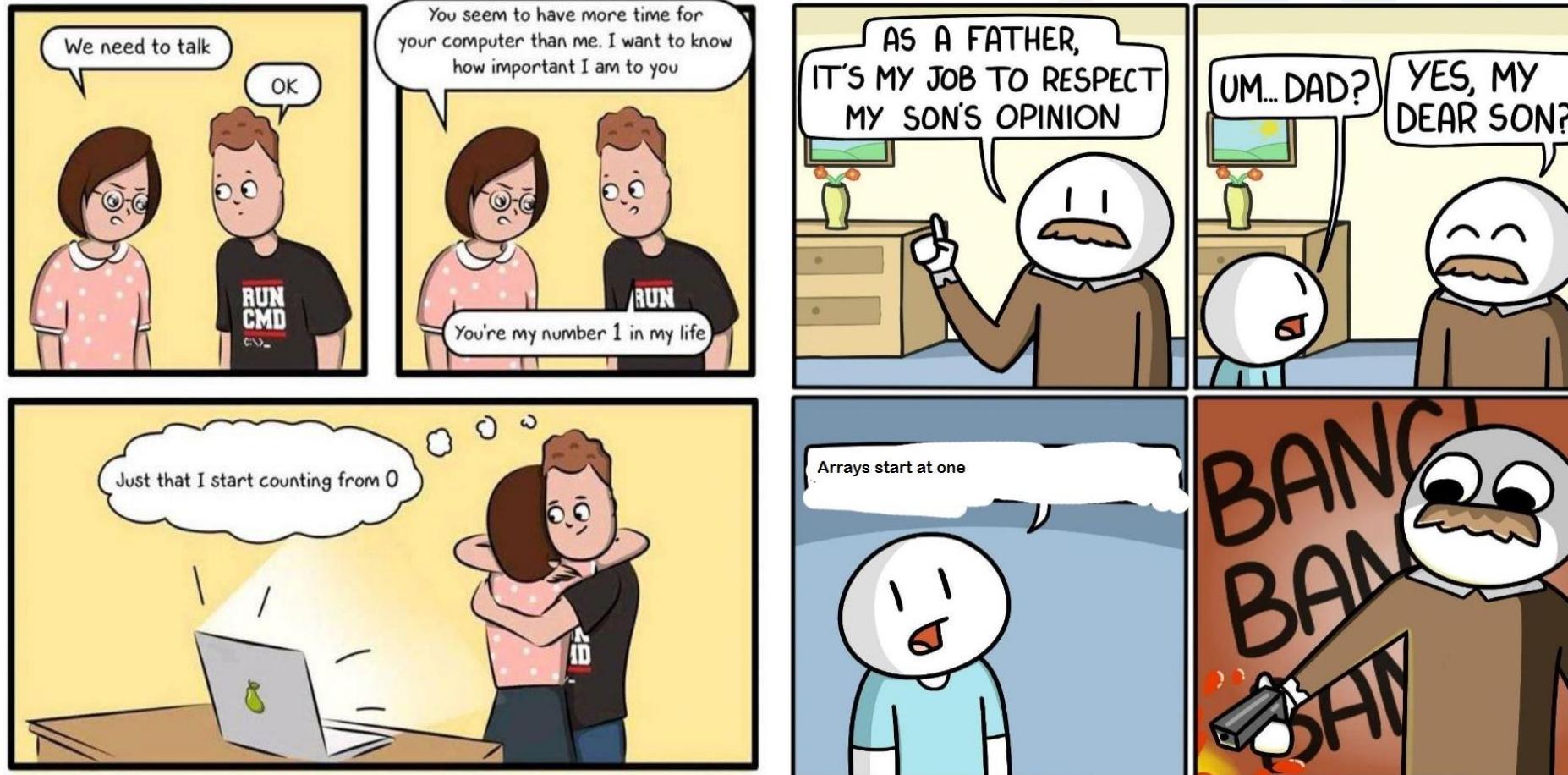
- **datatype arrayName[array Size];**
- **int myarray[5];**
- In C programming, when an array is declared, memory is allocated for its elements, but the initial values of these elements remain **undefined** or contain **garbage** values. This means that until explicitly initialized, the contents of a newly declared array in C are unpredictable.
- **Array Size in C:** It's crucial to note that in C, the size of an array is fixed upon declaration and cannot be altered thereafter. This underscores the importance of planning array usage according to fixed size constraints in C applications.

How to initialize and Change values in an array in C

- `dataType arrayName[arraySize] = {value1, value2, ..., valueN};`
- `int myarray[5] = {1, 2, 3, 4, 5};`
- `int myarray[] = {1, 2, 3, 4, 5};`
- make the value of the third element to -1
 - `myarray[2] = -1;`
- make the value of the fifth element to 0
 - `myarray[4] = 0;`

- **Advantages of Arrays:**
 - **Efficient Storage and Access:** Arrays store elements in contiguous memory locations, allowing for fast retrieval using indices, making them suitable for handling large amounts of data.
 - **Random Access:** Arrays provide constant-time access to any element using its index, making operations like reading or updating an element extremely efficient.
 - **Ease of Use:** Arrays are simple to understand and use, with common sorting and searching algorithms like binary search applicable, making them an accessible data structure for beginners and experts alike.
- **Disadvantages of Arrays:**
 - **Fixed Size:** Once created, the size of an array cannot be changed, which can be restrictive when working with dynamic data where the size of the data set may increase or decrease.
 - **Lack of Flexibility for Insertion/Deletion:** Arrays do not natively support efficient insertion or deletion of elements, as these operations may require shifting elements, which can be time-consuming.
 - **Homogeneous Elements:** Arrays can only store elements of the same data type, limiting their flexibility in scenarios that require handling multiple types of data.

- 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
 - Lower bound index of the first element of the array
 - Upper bound index of the last element of the array
- Size = number of elements * Size of each elements in bytes



One Dimensional array

- Address of the element at k^{th} index
 - $a[k] =$

One Dimensional array

- Address of the element at k^{th} index
 - $a[k] = B + W*k$
 - $a[k] = B + W*(k - \text{Lower bound})$
 - 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)**
[Asked in Cognizant 2016]

- (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

Two-Dimensional array

- A **two-dimensional (2D) array** is essentially an array of arrays. It is organized in a matrix-like structure, consisting of rows and columns. This makes it ideal for representing data in a tabular format.
- 2D arrays are often used to simulate relational databases or other grid-based structures. They allow for efficient storage and retrieval of large amounts of data, which can easily be passed to multiple functions as needed, simplifying data management.

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

	Column 0	Column 1	Column 2
Row 0	x[0][0]	x[0][1]	x[0][2]
Row 1	x[1][0]	x[1][1]	x[1][2]
Row 2	x[2][0]	x[2][1]	x[2][2]

OR

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

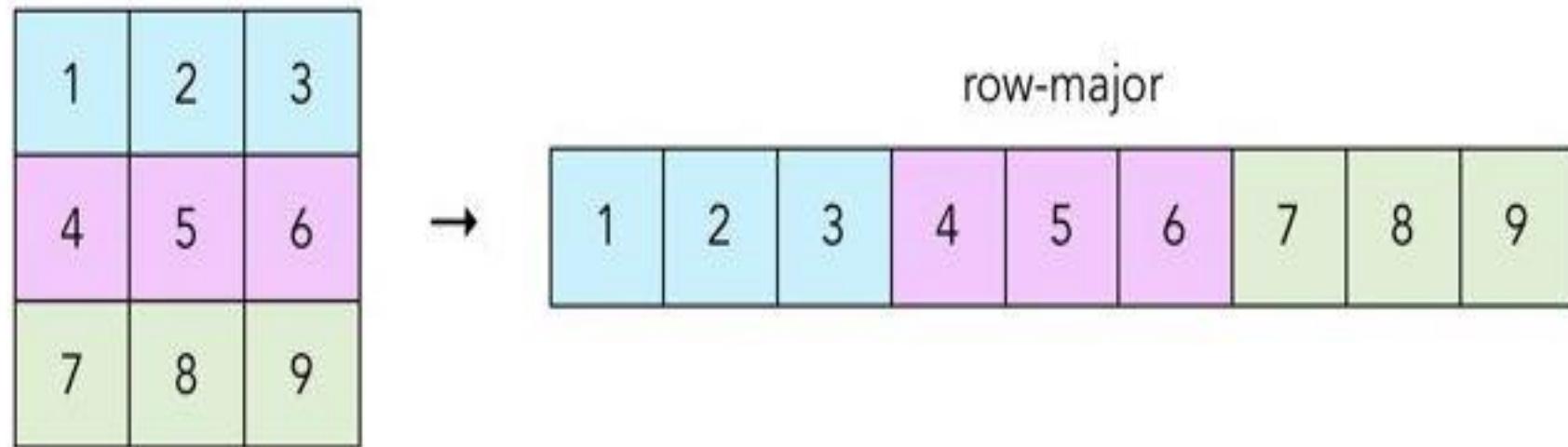
Implementation of 2D array

- 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.

	Column 0	Column 1	Column 2
Row 0	x[0][0]	x[0][1]	x[0][2]
Row 1	x[1][0]	x[1][1]	x[1][2]
Row 2	x[2][0]	x[2][1]	x[2][2]

Row Major implementation of 2D array

- 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.



Row Major implementation of 2D array

Address of $a[i][j] =$

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

Row Major implementation of 2D array

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

Q 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 $a[i][j]$?

(Gate-1998) (2 Marks)

(A) $15i + j + 84$

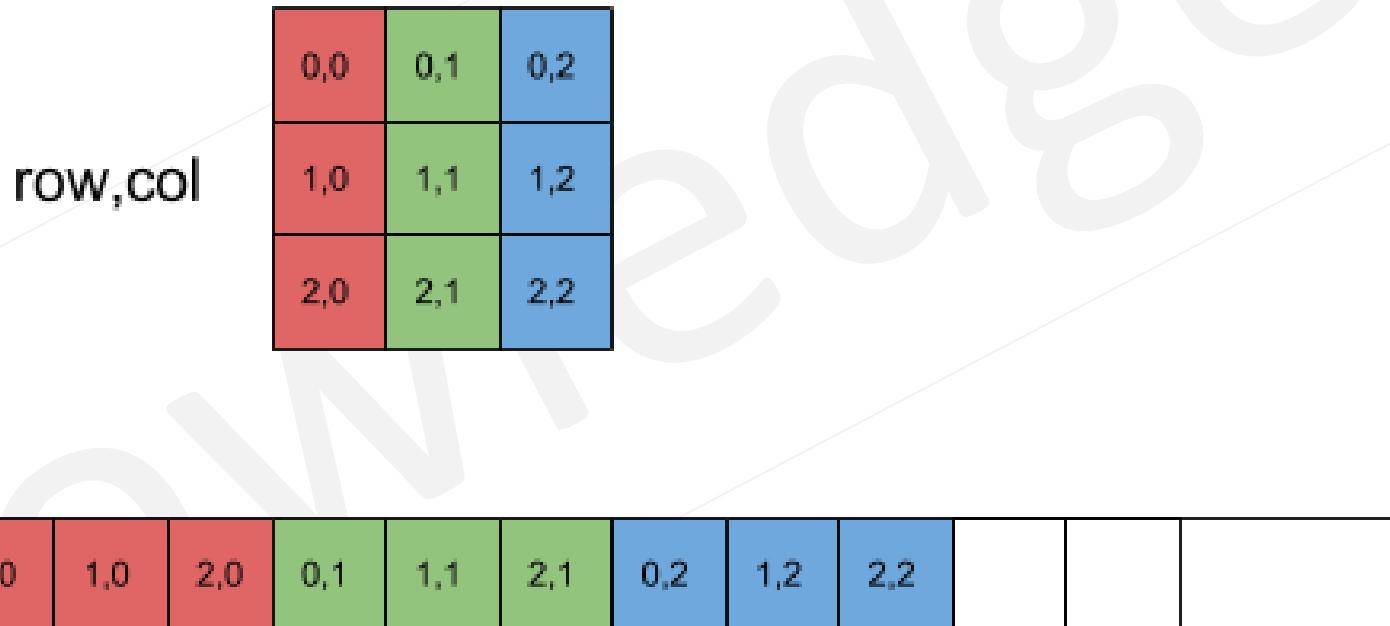
(B) $15j + i + 84$

(C) $10i + j + 89$

(D) $10j + i + 89$

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.



Column Major implementation of 2D array

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 $VAL[1\dots15][1\dots10]$ 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?

Q Two matrices M_1 and M_2 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 $M_1 \times M_2$ 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

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

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

The sum of the elements of the array v is (Gate-2000) (1 Marks)

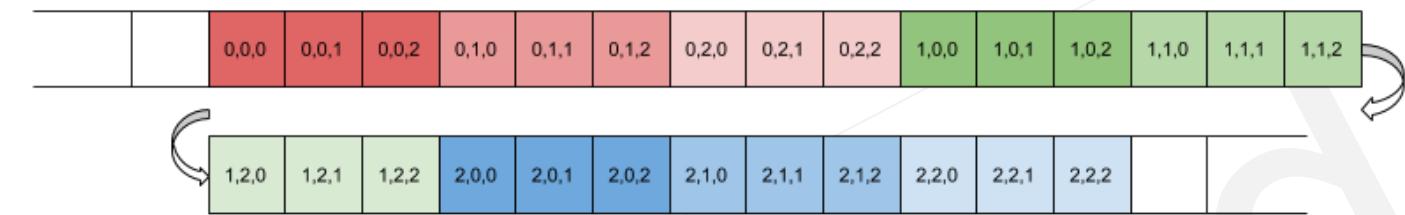
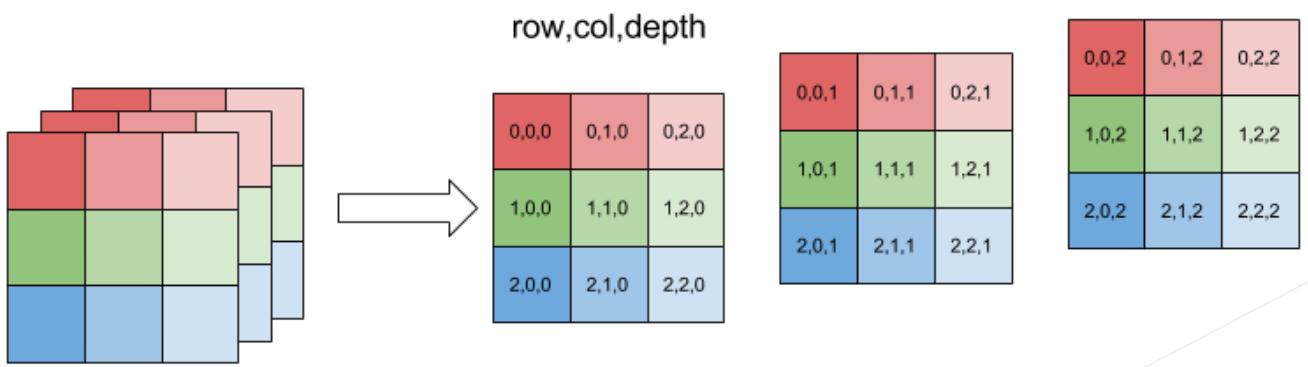
(A) 0

(B) $n - 1$

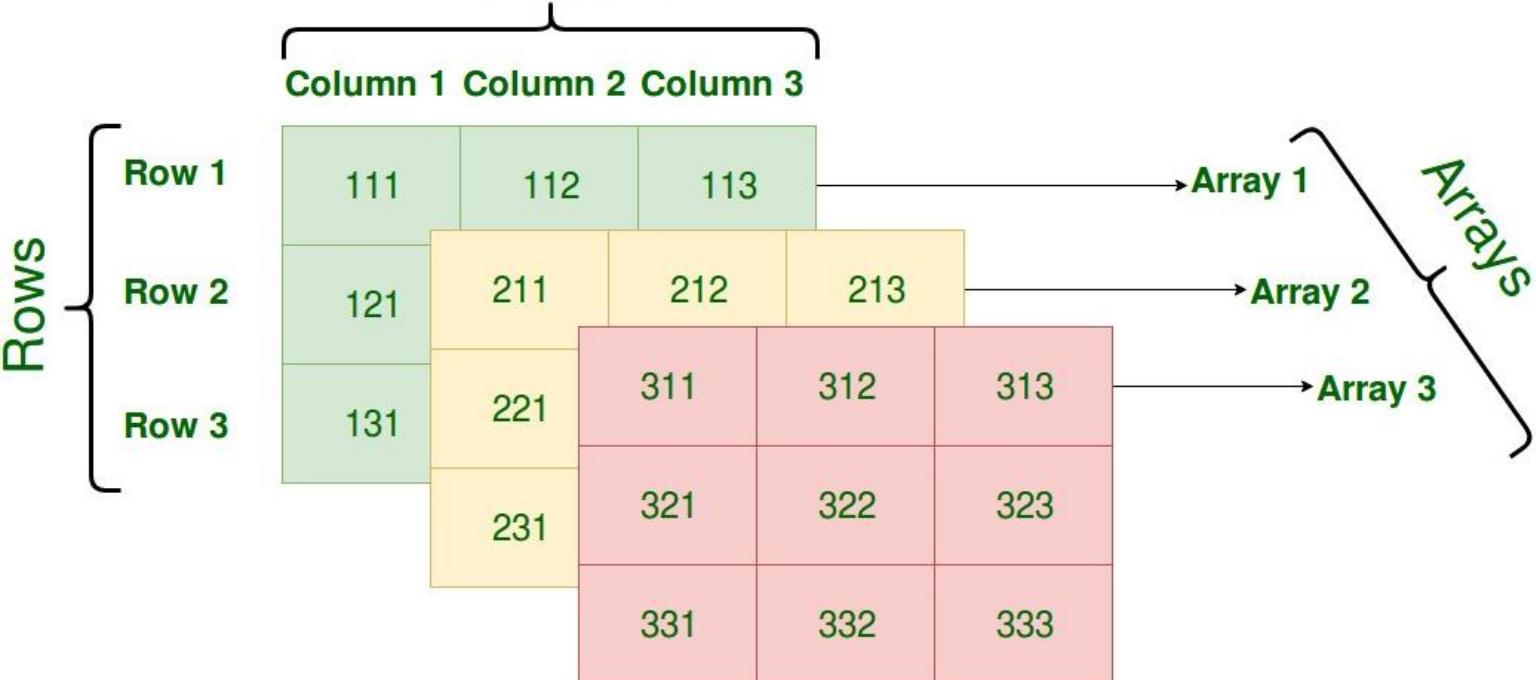
(C) $n^2 - 3n + 2$

(D) $n^2(n+1)/2$

3-Dimensional array



Columns



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 [l, j, k, ----, x] =

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 $A [i, j, k, ----, x] =$

$$\begin{aligned} & B + (i-L_1) (U_2-L_2+1) (U_3-L_3+1) (U_4-L_4+1) \cdots (U_n-L_n+1) \\ & + (j-L_2)(U_3-L_3+1) (U_4-L_4+1) \cdots (U_n-L_n+1) \\ & + (k-L_3)(U_4-L_4+1) \cdots (U_n-L_n+1) \\ & + \\ & + \\ & + \\ & + (x-L_n) \end{aligned}$$

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

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) [Asked in Hexaware 2017] [Asked in Accenture]

```
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

Q Suppose you are given an array $s[1..n]$ and a procedure reverse (s, i, j) which reverses the order of elements in a between positions i and j (both inclusive). What does the following sequence do, **(GATE - 2014) (1 Marks)**

where $1 \leq k < n$:

```
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

Q Consider a $n \times n$ square matrix A, such that

$$A[i][j] = 0 \quad \text{if } (i < j)$$

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

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

Q In a compact one dimensional array representation for lower triangular matrix (all elements above diagonal are zero) of size $n \times n$, non zero elements of each row are stored one after another, (i.e. elements of lower triangle) of each row are stored one after another, starting from first row, the index of $(i, j)^{\text{th}}$ element of the lower triangular matrix in this new representation is **(GATE - 1994) (2 Marks)**

(A) $i+j$

(B) $i + j-1$

(C) $j-1 + i(i-1)/2$

(D) $i+j(j-1)/2$

Q Consider a $n \times n$ square matrix A, such that

$$A[i][j] = 0 \quad \text{if } i < j$$

a) derive an address access formula to access this matrix (if stored in column major order)

Q Consider a $n \times n$ square matrix A, such that

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

find the space required to store the array in memory

Q Consider a $n \times n$ square matrix A, such that

$$A[i][j] = 0 \quad \text{if } |i-j| > 1$$

find the space required to store the array in memory

Q Consider a $n \times n$ square matrix A, such that

$A[i][j] = A[i-1][j-1]$ if $i > 0 \text{ \&\& } j > 0, 0 \leq i, j \leq n-1$

find the space required to store the array in memory

Q Consider a $n \times n$ square matrix A, such that

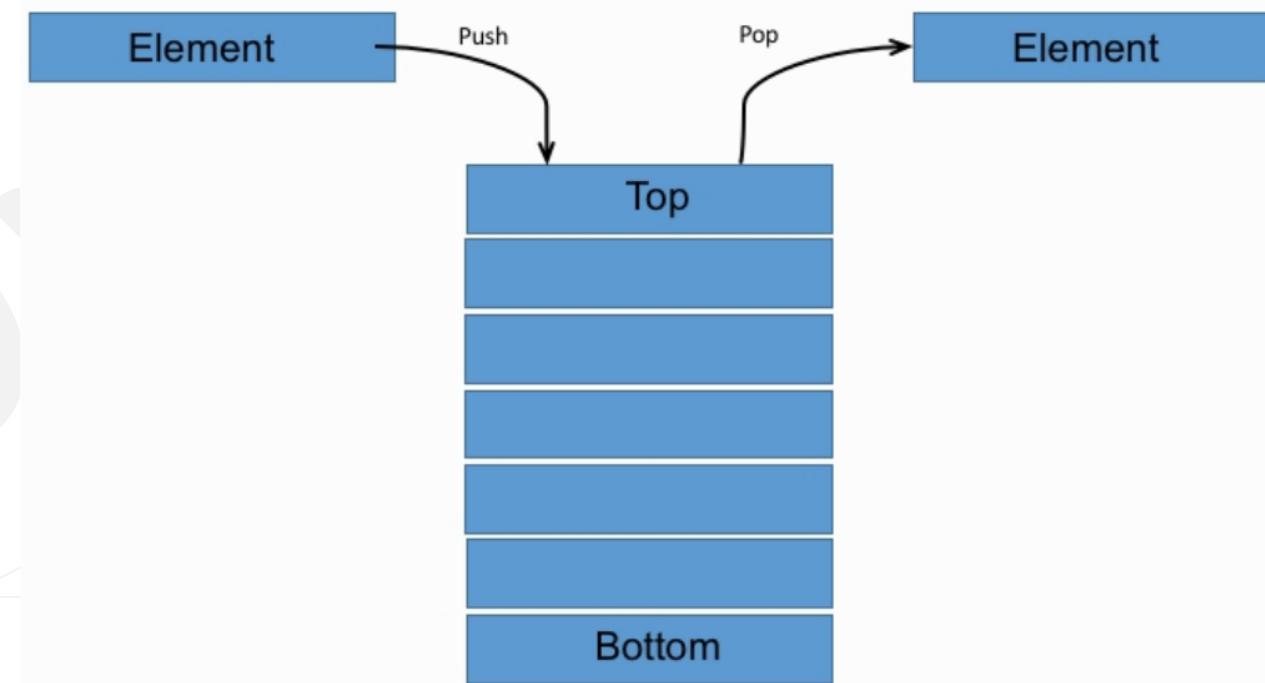
	0	1	2	3
0	00	01	02	03
1	10	11	12	13
2	20	21	22	23
3	30	31	32	33

Basics of Stack



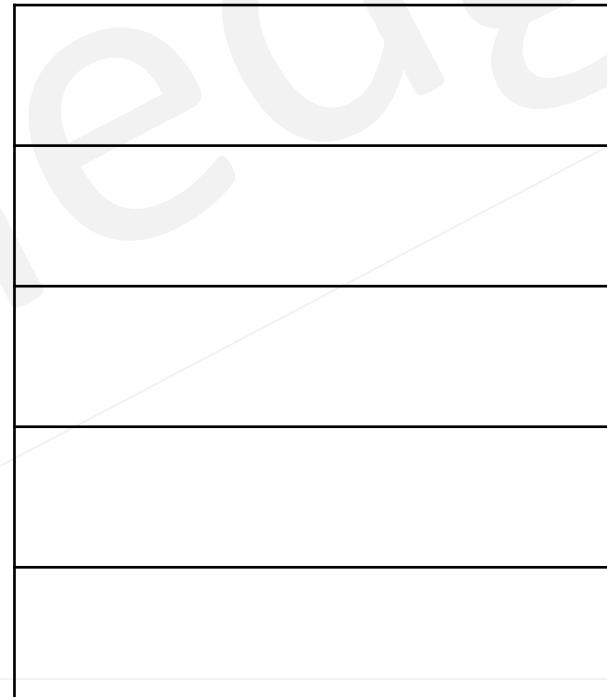
STACK

- A **stack** is a linear data structure that follows the **Last In, First Out (LIFO)** or **First In, Last Out (FILO)** principle. In a stack, both the addition (push) and removal (pop) of elements are performed from one end, known as the **Top of Stack (TOS)**.
 - The last element added is the first to be removed.
 - The first element added is the last to be removed.
- The **top element** is the most frequently accessible, while the element at the **bottom** is the least accessible in the stack. This structure is widely used for tasks like function call management, expression evaluation, and undo operations in software.



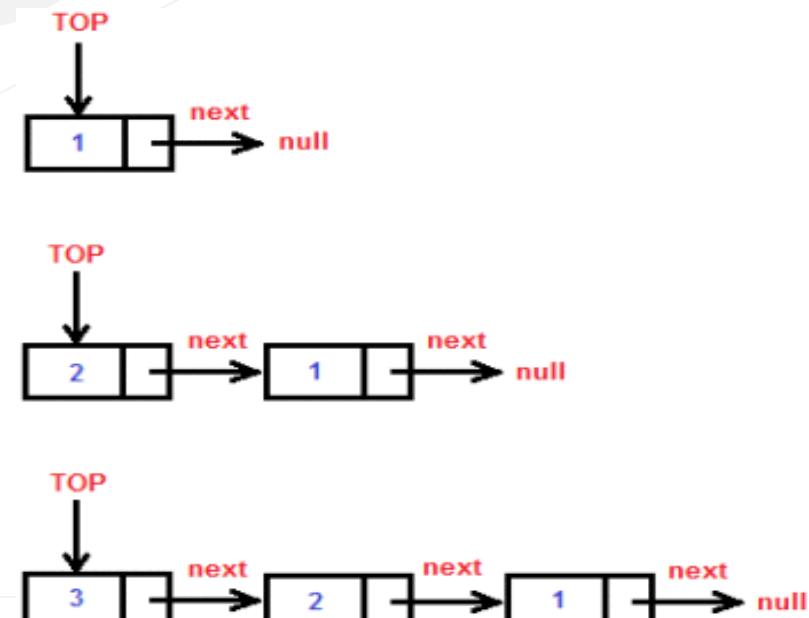
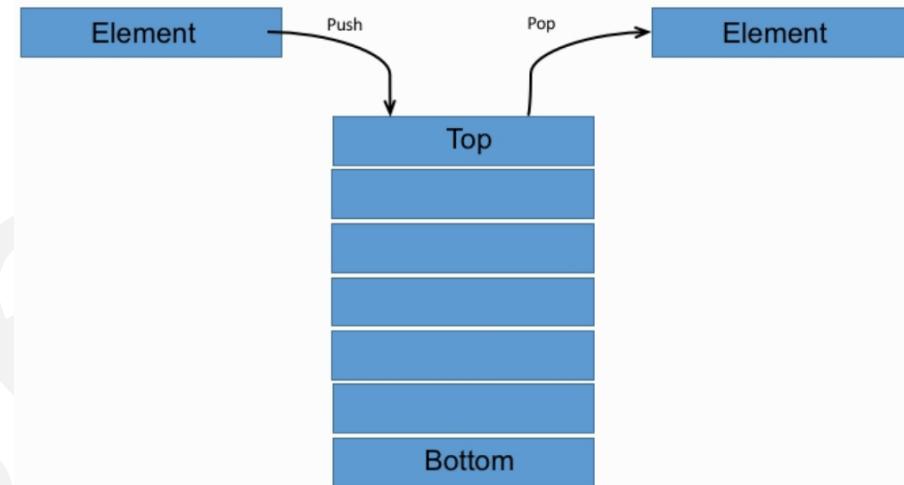
Q Choose the correct alternatives (more than one may be correct) and write the corresponding letters only: The following sequence of operations is performed on a stack: PUSH (10), PUSH (20), POP, PUSH (10), PUSH (20), POP, POP, POP, PUSH (20), POP The sequence of values popped out is ? **(GATE - 1991) (2 Marks)**

- a) 20,10,20,10,20
- b) 20,20,10,10,20
- c) 10,20,20,10,20
- d) 20,20,10,20,10



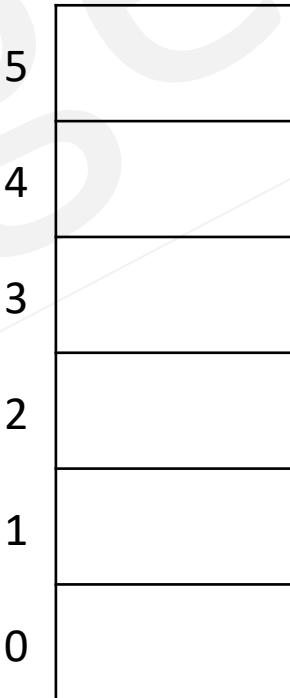
Stack Implementation

- Stack is generally implemented in two ways.
- **Static Implementation:**
 - Uses an **array** to create the stack.
 - It is a simple and straightforward method but lacks flexibility.
 - The size of the stack must be defined during design, which can lead to inefficient memory utilization if the size needs to change later.
- **Dynamic Implementation:**
 - Also known as **linked list implementation**.
 - This method uses **pointers** to dynamically allocate memory, making it more flexible and efficient in terms of memory usage.



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-1)
        Print stack overflow and exit
    TOP = TOP + 1
    S[TOP] = x
    exit
}
```

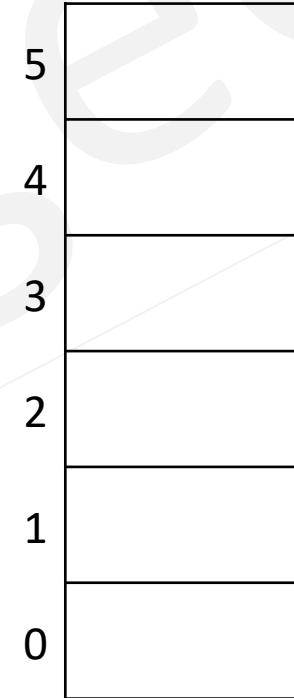


Q Which of the following is true about linked list implementation of stack?

- (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

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 $A[1\dots\text{MAXSIZE}]$ is used to implement two stacks. The two stacks grow from opposite ends of the array. Variables top_1 and top_2 ($\text{top}_1 < \text{top}_2$) 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)** [Asked in Goldman Sachs 2018]

- (A)** $(\text{top}_1 = \text{MAXSIZE}/2)$ and $(\text{top}_2 = \text{MAXSIZE}/2+1)$
- (B)** $\text{top}_1 + \text{top}_2 = \text{MAXSIZE}$
- (C)** $(\text{top}_1 = \text{MAXSIZE}/2)$ or $(\text{top}_2 = \text{MAXSIZE})$
- (D)** $\text{top}_1 = \text{top}_2 - 1$

Q Let S be a stack of size $n \geq 1$. 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 $m \geq 1$, 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) $Y + 2X$

Q 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? **(GATE - 1994) (2 Marks)**

- a) 3, 4, 5, 1, 2
- b) 3, 4, 5, 2, 1
- c) 1, 5, 2, 3, 4
- 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

Arithmetic expression

- An expression consists of operands combined with operators. There are three common notations for representing expressions:
- **Infix Notation**: The operator is placed between operands (e.g., A + B). This is the most familiar form used in everyday math.
- **Prefix Notation (Polish Notation)**: The operator is placed before the operands (e.g., +AB). This notation was introduced by the Polish logician Jan Łukasiewicz in 1924.
- **Postfix Notation (Reverse Polish Notation)**: The operator is placed after the operands (e.g., AB+). It is widely used in computer systems because it's well-suited for computational processes, particularly in the design of Arithmetic and Logic Units (ALUs) within CPUs.
- Expressions entered into a computer are usually converted to postfix notation, stored in a stack, and then evaluated.



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

$a + b \times c - d \wedge e \wedge f$

- (A) $abc \times + def \wedge \wedge -$
- (B) $abc \times + de \wedge f \wedge -$
- (C) $ab + c \times d - e \wedge f \wedge$
- (D) $- + a \times bc \wedge \wedge def$

Q The postfix expression for the infix expression
A+B*(C+D)/F+D*E is: **(GATE - 1995) (2 Marks)**

$$A + B * (C + D) / F + D * E$$

a) AB+CD+*F/D+E*

b) ABCD+*F/DE*++

c) A*B+CD/F*DE++

d) A+*BCD/F*DE++

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

**Q Compute the post fix equivalent of the following expression
(GATE - 1998) (2 Marks)**

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

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

10 5 + 60 6 / * 8 -

(A) 284

(B) 213

(C) 142

(D) 71

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)**
[Asked in Hexaware 2017]

$8\ 2\ 3\ ^\ / \ 2\ 3\ * \ + \ 5\ 1\ * \ -$

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

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 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

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

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

ii) all are R to L associative

2 * 2 – 1 \$ 1 \$ 4 – 2

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

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

Recursion

- Recursion is a powerful concept in programming, where a function calls itself to solve a problem. It requires two essential elements:
 - A base condition to stop the recursion.
 - Logic to solve the problem step-by-step.
- **Definition:** A function is recursive if it invokes itself within its own definition. It is particularly useful for solving problems involving repetitive tasks or iterations in reverse order.
- **Types of Recursion:**
 - **Direct Recursion:** When a function calls itself directly. It can be:
 - **Tail Recursion:** The recursive call is the last statement in the function.
 - **Head Recursion:** The recursive call occurs before other operations.
 - **Indirect Recursion:** When two or more functions call each other in a cycle.
- Recursion is an alternative to iteration, and both techniques are used to repeatedly execute a function.

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()

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

Void fun(int x)

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

Q Find the output of the following pseudo code?

Void main()

```
{  
    fun(3);  
}
```

Void fun(int x)

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

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

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

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

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 (513, 2)? **(GATE - 2011) (2 Marks)**

(A) 9

(B) 8

(C) 5

(D) 2

Q Consider the following ANSI C function: int SomeFunction (int x, int y) **(GATE - 2021) (2 Marks)**

```
int SomeFunction (int x, int y)
```

```
{
```

```
    if ((x==1) || (y==1))
```

```
        return 1;
```

```
    if (x==y)
```

```
        return x;
```

```
    if (x > y)
```

```
        return SomeFunction(x-y, y);
```

```
    if (y > x)
```

```
        return SomeFunction (x, y-x);
```

```
}
```

The value returned by SomeFunction(15, 255) is _____

Q Consider the following ANSI C program (GATE - 2021) (2 Marks)

```
#include <stdio.h>
int foo(int x, int y, int q)
{
    if ((x<=0) && (y<=0))
        return q;
    if (x<=0)
        return foo(x, y-q, q);
    if (y<=0)
        return foo(x-q, y, q);
    return foo(x, y-q, q) + foo(x-q, y, q);
}
int main( )
{
    int r = foo(15, 15, 10);
    printf("%d", r);
    return 0;
}
```

The output of the program upon execution is

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

Q 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

Q. A function f defined on stacks of integers satisfies the following properties.
 $f(\emptyset) = 0$ and $f(\text{push } (S, i)) = \max(f(S), 0) + i$ for all stacks S and integers i .
If a stack S contains the integers 2, -3, 2, -1, 2 in order from bottom to top, what is $f(S)$? [Asked in AMCAT 2016] (GATE-2005)(1 Marks)

- a) 6
- b) 4
- c) 3
- d) 2

Q Consider the following C function. (Gate-2015) (1 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 _____.

- In mathematics, the **Fibonacci numbers**, commonly denoted F_n , form a **sequence**, called the **Fibonacci sequence**, such that each number is the sum of the two preceding ones, starting from 0 and 1.
 - if $n==0$, then $f(n) = 0$
 - if $n==1$, then $f(n) = 1$
 - if $n > 1$, then $f(n-1) + f(n-2)$

n	0	1	2	3	4	5	6	7	8	9	10	11	12
f(n)													
No of invocation													
No of addition													

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

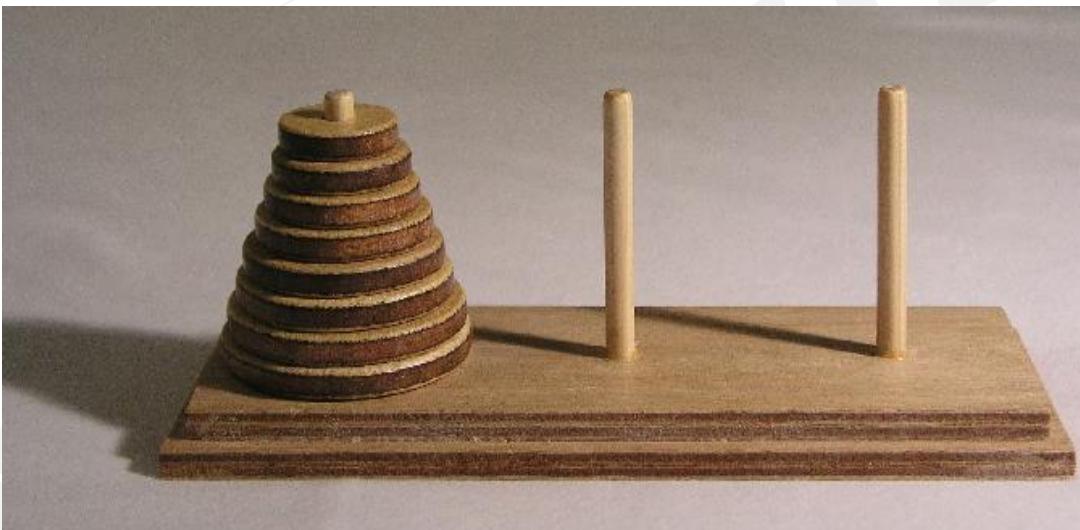
no of addition = $f(n+1) - 1$

- Fibonacci numbers are named after Italian mathematician Leonardo of Pisa, also known as Fibonacci. He introduced the sequence in his 1202 book, *Liber Abaci*, which brought it to the attention of Western European mathematics. However, the sequence was earlier described in Indian mathematics around 200 BC by Acharya Pingala, who explored patterns in Sanskrit poetry.
- **Mathematical Significance:** Fibonacci numbers frequently appear in various mathematical areas, which has even led to the establishment of a dedicated journal, *Fibonacci Quarterly*.
- **Applications:**
 - **Computer Science:** The Fibonacci sequence is applied in algorithms like the Fibonacci search technique, Fibonacci heaps, and Fibonacci cubes (used for parallel and distributed system interconnections).
 - **Biology:** Fibonacci numbers are observed in natural patterns such as tree branching, leaf arrangement on stems, the sprouting of pineapples, the flowering of artichokes, the uncurling of ferns, and the structure of pine cones.
- **Historical Indian Contributions:** The Fibonacci sequence ties back to Indian mathematics and concepts such as the **binary numeral system**, **binomial theorem**, **Pascal's triangle**, and **zero**.

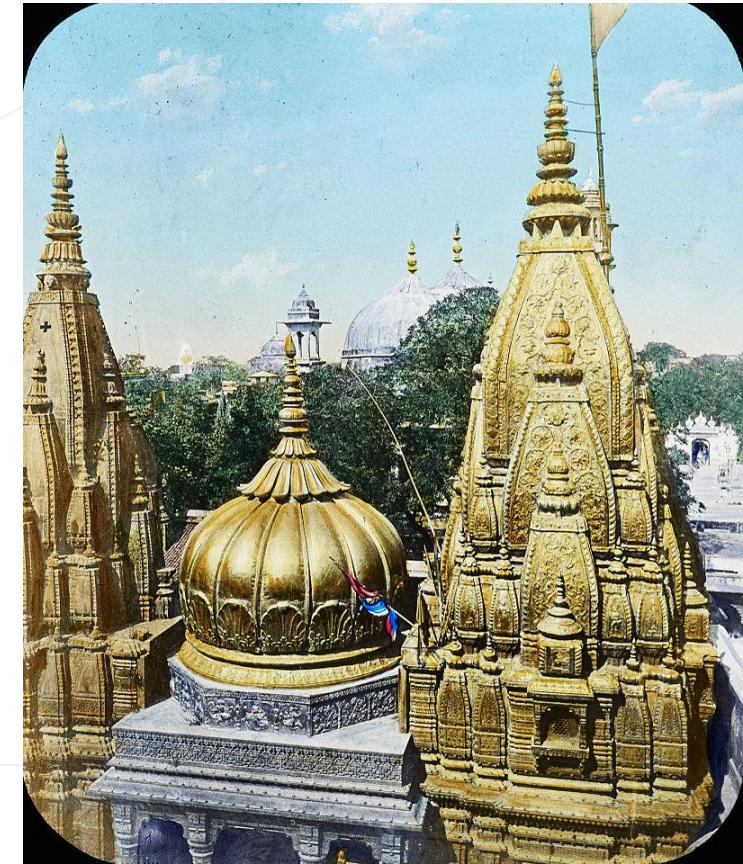


Tower of hanoi

- The **Tower of Hanoi** (also known as the **Tower of Brahma**) is a classic mathematical puzzle consisting of three rods and a set of disks of varying sizes. The puzzle starts with the disks stacked on one rod in ascending order, forming a conical shape.
- **Objective:** The goal is to move the entire stack of disks to another rod while following these rules:
 - Only one disk can be moved at a time.
 - A disk can only be placed on an empty rod or on a larger disk.
 - No larger disk can be placed on top of a smaller disk.



- **Story & Legend**
 - According to a story, an ancient temple in **Kashi Vishwanath** contains three posts with 64 golden disks. **Brahmin priests**, following an ancient prophecy, have been transferring the disks according to the rules of Brahma since ancient times. This is why the puzzle is sometimes referred to as the **Tower of Brahma**.



```
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 ($(m \neq 0) \&\& (n=0)$), the $A(m, n) = A(m-1, 1)$

if ($(m \neq 0) \&\& (n \neq 0)$), the $A(m, n) = A(m-1, A(m, n-1))$

Values of $A(m, n)$

$m \setminus n$	0	1	2	3	4	n
0	1	2	3	4	5	$n + 1$
1	2	3	4	5	6	$n + 2 = 2 + (n + 3) - 3$
2	3	5	7	9	11	$2n + 3 = 2 \cdot (n + 3) - 3$
3	5	13	29	61	125	$2^{(n+3)} - 3$
4	13	$2^{65536} - 3$	$2^{2^{65536}} - 3$	$2^{2^{2^{65536}}} - 3$	$2^{2^{2^{2^{65536}}}} - 3$	$\underbrace{2^{2^{\dots^2}}} - 3$ $n + 3$
	$= 2^{2^2} - 3$	$= 2^{2^{2^2}} - 3$	$= 2^{2^{2^{2^2}}} - 3$	$= 2^{2^{2^{2^{2^2}}}} - 3$	$= 2^{2^{2^{2^{2^{2^2}}}}} - 3$	

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)
    }
}
```

Q Find the output of the following pseudo code?

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

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 → RC);  
    }  
}
```

Q Find what this function is doing?

```
void what(struct Bnode *t)
{
    if (t)
    {
        printf("%d", t → data);
        what(t → LC);
        what(t → RC);
    }
}
```

Q Find what this function is doing?

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

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 → RC);  
        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 → RC);  
    }  
}
```

Void B(struct Bnode *t)

```
{  
    if (t)  
    {  
        printf("%d", t → data);  
        A(t → LC);  
        A(t → RC);  
    }  
}
```

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

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

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

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

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

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

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

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
- (C) 4

- (B) 3
- (D) 5

Q What does fun2() do in general?

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

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

- (A) x^y
- (C) x^y

- (B) $x+x^y$
- (D) y^x

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

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

Queue

- A **queue** is a linear data structure where elements are added at the **rear** and removed from the **front**, following a **First In, First Out (FIFO)** principle. The first element inserted is the first one to be removed, making it ideal for tasks that require orderly processing.
- As a **non-primitive** and **homogeneous** data structure, a queue ensures that all elements are of the same type and are processed sequentially. This makes it useful for managing tasks like **printer spooling**, **process scheduling**, and **network traffic handling**.



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.
- Whenever an element is added to the queue, the value of REAR is increased by 1
 - $\text{REAR} = \text{REAR} + 1$
- Whenever an element is deleted from the queue, the value of FRONT is increased by 1
 - $\text{Front} = \text{Front} + 1$
- Total number of elements in a queue
 - $\text{Rear} - \text{Front} + 1$



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

Insertion



Enqueue (QUEUE, N, F, R, ITEM)

{

 if (R == N - 1)

 Write over flow and exit

 if (F == -1)

 Set F = 0 && R = 0

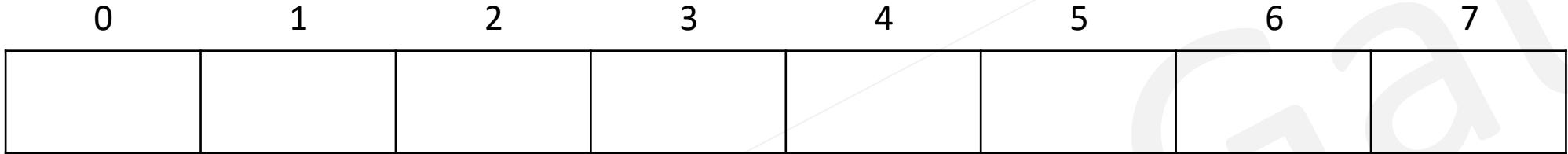
 Else

 R = R + 1

 Queue[R] = ITEM

}

Deletion



Dequeue (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

$F = F + 1$

 Return item

}

Q. Consider the following sequence of operations on an empty stack.

Push(54);push(52);pop();push(55);push(62);s=pop();

Consider the following sequence of operations on an empty queue.

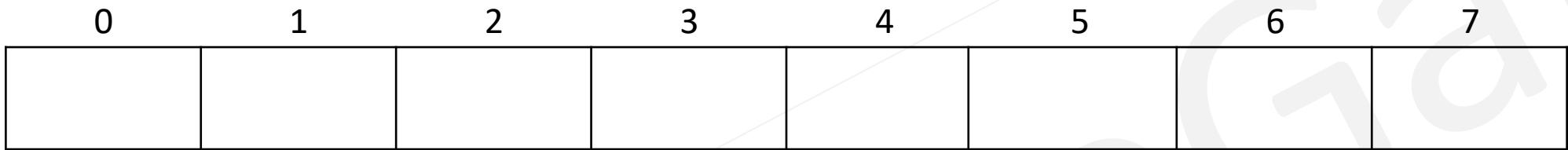
enqueue(21);enqueue(24);dequeue();enqueue(28);enqueue(32);q=dequeue();

The value of $s+q$ is _____.

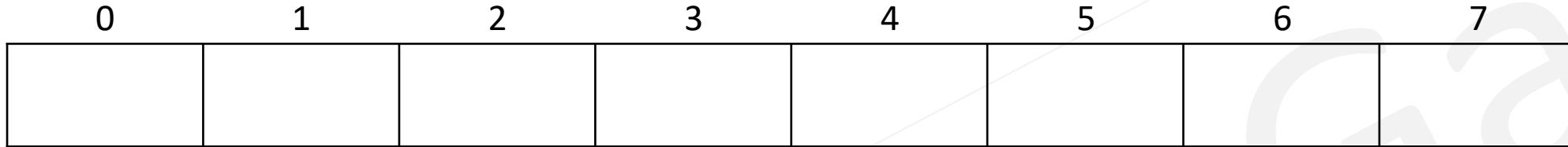
(GATE-2021)[Asked in E-litmus 2019]

- (a)** 86
- (b)** 68
- (c)** 24
- (d)** 94

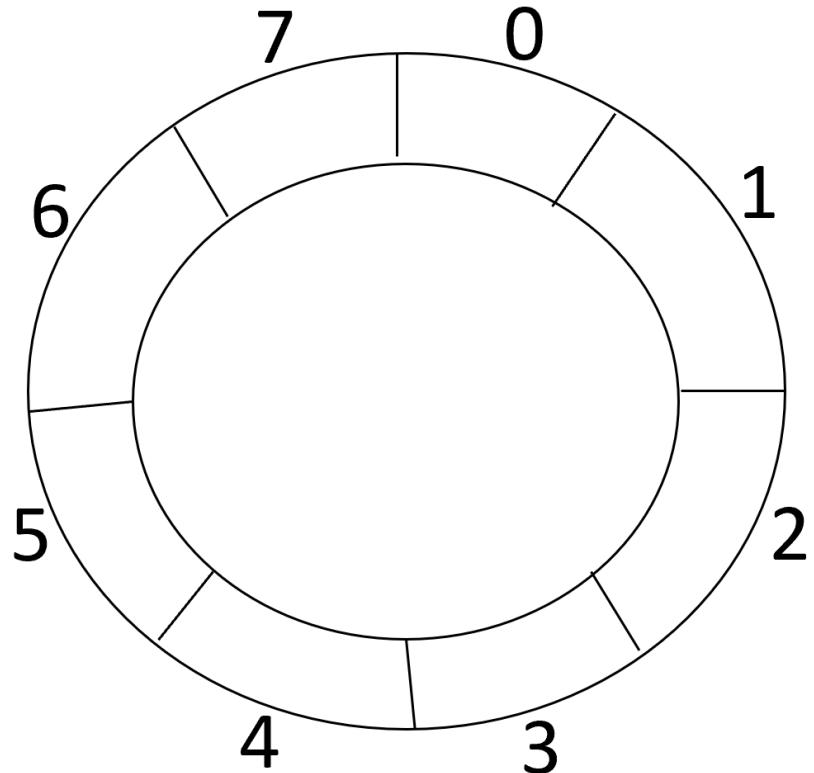
Analysis



Circular Queue



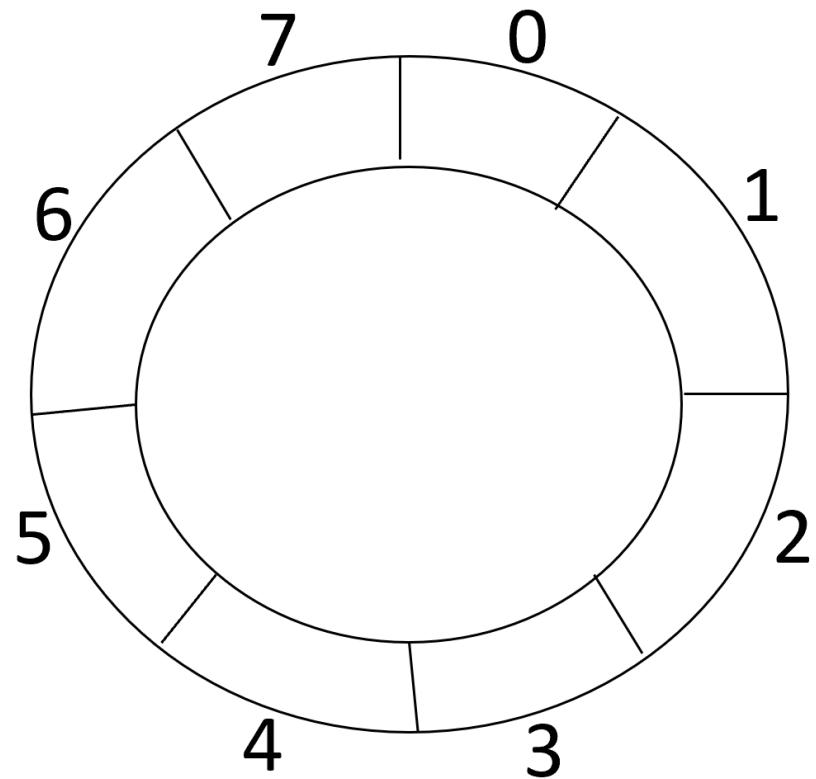
1. EnQ a, b, c
2. DeQ 1 element
3. EnQ d, e, f
4. EnQ g, h, I
5. DeQ 4 element
6. EnQ j, k, l, m, n



Circular Queue

Insertion

```
Enqueue(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
        R = (R + 1)%N
    Queue[R] = ITEM
}
```



Circular Queue

Deletion



Dequeue(QUEUE, N, F, R, ITEM)

{

 if ($F == -1$)
 Write under flow and exit

 ITEM = QUEUE[F]

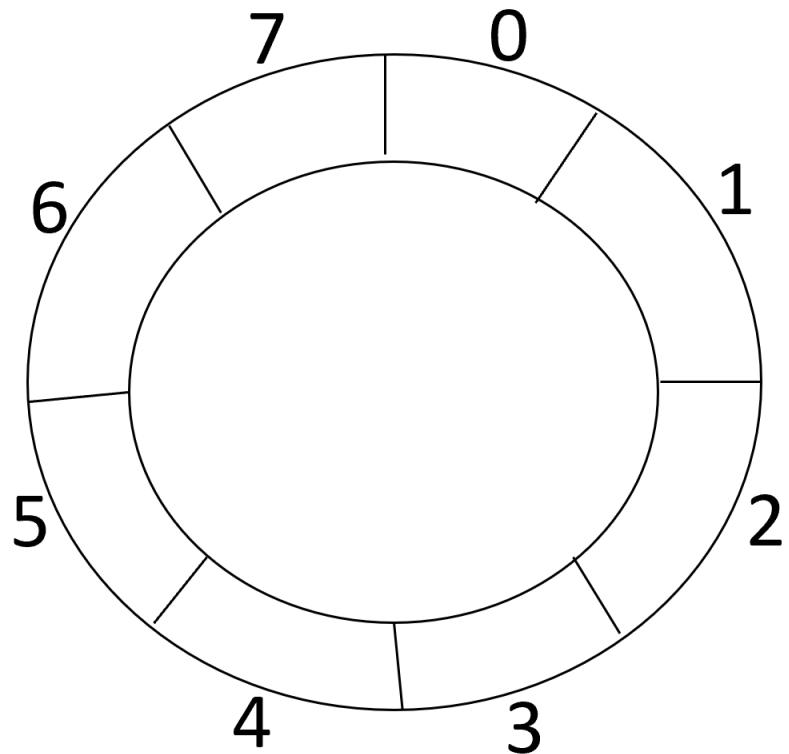
 if ($F == R$)
 Set $F = -1$ & $R = -1$

 Else

$F = (F + 1) \% N$

 Return item

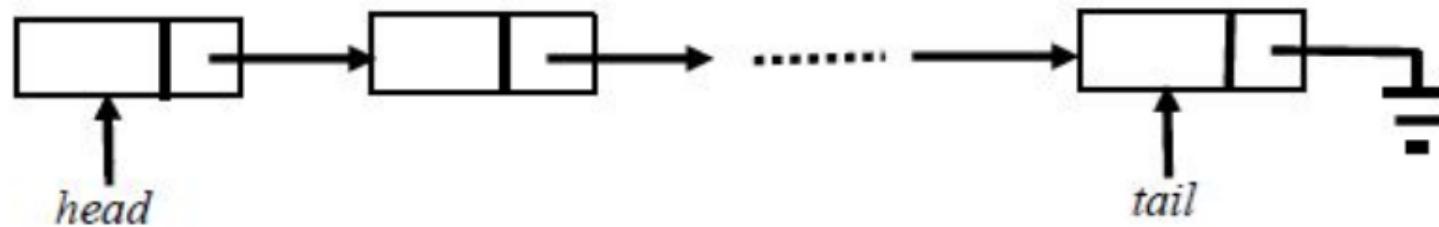
}



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 elements with the same priority are processed according to the order in which they were added to the queue.

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)** [Asked in Accenture 2020]

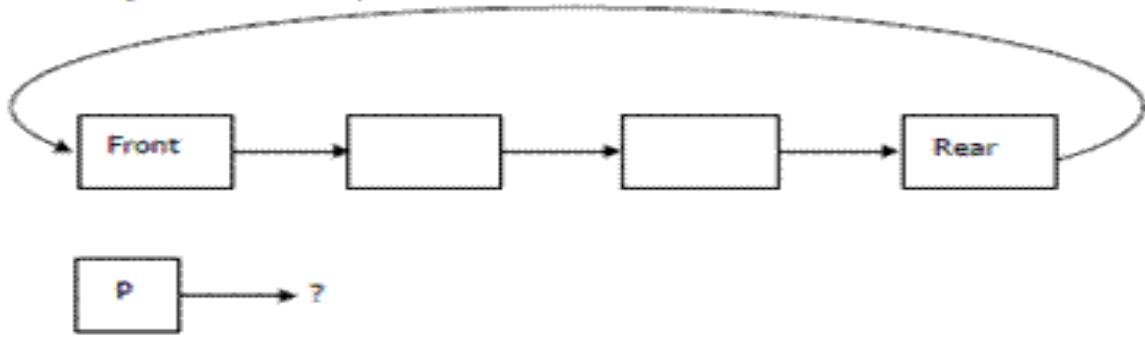
- a) $\Theta(1), \Theta(1)$
- b) $\Theta(1), \Theta(n)$
- c) $\Theta(n), \Theta(1)$
- d) $\Theta(n), \Theta(n)$

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

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

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 $\Omega(n)$
- c) The worst case time complexity for both operations will be $\Omega(n)$
- d) Worst case time complexity for both operations will be $\Omega(\log n)$

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) [Asked in Goldman Sachs 2018]**

- 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.

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

Q Consider the following operation along with Enqueue and Dequeue operations on queues, where k is a global parameter.

MultiDequeue(Q)

{

 m = k

 while (Q is not empty and m > 0)

 {

 Dequeue(Q)

 m = m - 1

 }

}

What is the worst case time complexity of a sequence of n MultiDequeue() operations on an initially empty queue? (GATE - 2013) (2 Marks)

a) $O(n)$

b) $O(n + k)$

c) $O(n k)$

d) $O(n^2)$

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);  
    }
```

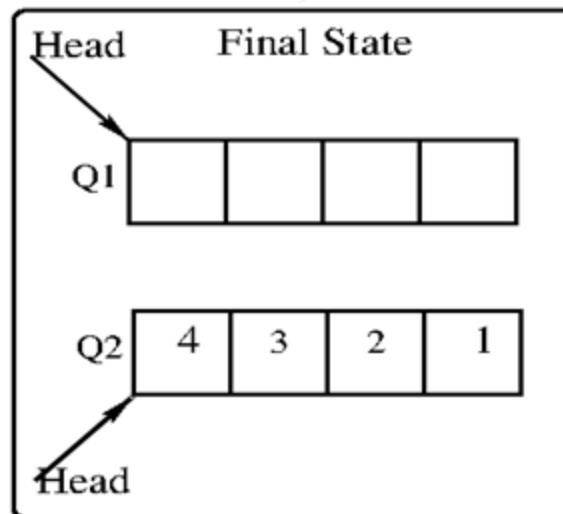
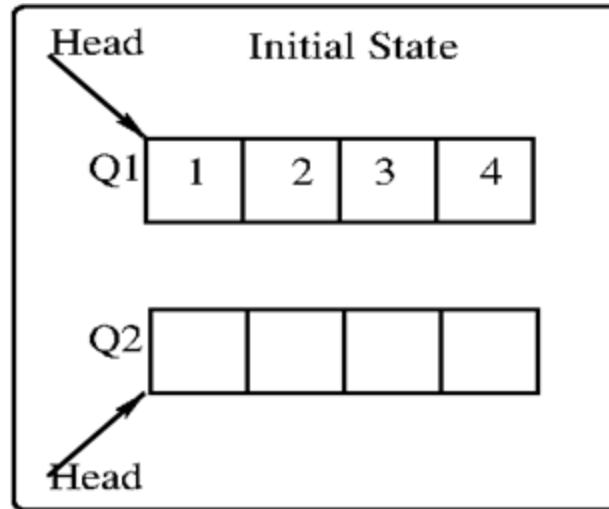
```
}
```

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

[Asked in AMCAT 2017]

- (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

Q Consider the queues Q_1 containing four elements and Q_2 containing none (shown as the Initial State in the figure). The only operations allowed on these two queues are Enqueue(Q , element) and Dequeue(Q). The minimum number of Enqueue operations on Q_1 required to place the elements of Q_1 in Q_2 in reverse order (shown as the Final State in the figure) without using any additional storage is _____. (GATE 2022) (2 MARKS)



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

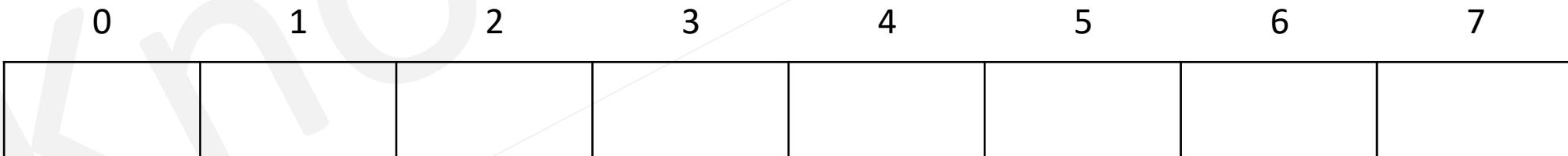
(B) Non-decreasing order

(C) strictly increasing order

(D) strictly decreasing order

Problem with Array

- **Fixed size and reallocation:** Arrays have a fixed size, which can lead to memory waste if the allocated size is larger than the actual data. Resizing an array often requires creating a new one and copying elements, which can be inefficient.
- **Inefficient insertion and deletion:** Adding or removing elements in the middle of an array requires shifting the remaining elements, resulting in a higher time complexity ($O(n)$) compared to linked lists.
- **Less flexible:** Arrays can only store elements of the same data type, and their structure cannot be easily adapted to different types (e.g., singly, doubly, circular) like linked lists.

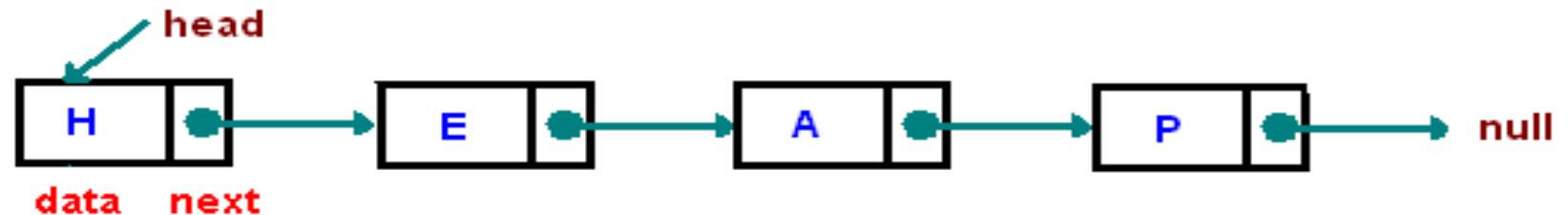


Linked list



- A linked list is a dynamic data structure consisting of elements called **nodes** that are connected in a linear sequence. Each node contains two main components:
- **Data**: This part stores the value or information of the node, which can be any data type like integers, characters, or objects.
- **Next Pointer**: Known as the link field or next pointer, this part holds the address of the next node in the list.
- The **last node** in a linked list has its next pointer set to NULL, indicating the end of the list. This means there is no subsequent node to reference.
- A special pointer variable, commonly referred to as head (also known as start or first), points to the first node in the list. If the list is **empty**—meaning it contains no nodes—the head pointer is set to NULL.

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



- **Advantages of Linked Lists:**
 - **Dynamic Sizing:** They can grow or shrink at runtime, efficiently using memory without needing a predefined size.
 - **Efficient Insertions and Deletions:** Adding or removing elements is fast ($O(1)$) when the position is known, without shifting other elements as in arrays.
 - **Versatility:** Linked lists support various forms—singly, doubly, circular—and can store different data types, offering flexibility for diverse applications.
- **Disadvantages of Linked Lists:**
 - **Slower Access Times:** Accessing elements requires sequential traversal from the head, resulting in ($O(n)$) time complexity.
 - **Memory Overhead:** Each node needs extra memory for pointers, increasing overall usage compared to arrays.
 - **Complex Pointer Management:** Handling pointers adds code complexity and risks errors like memory leaks or segmentation faults.

Q Write a C-style pseudocode for Traversing a link list iteratively, where pointer head have the address of the first node of the list?

```
void traverse(Node* head)
```

```
{
```

```
    Node* ptr = head;
```

```
    while (ptr != NULL)
```

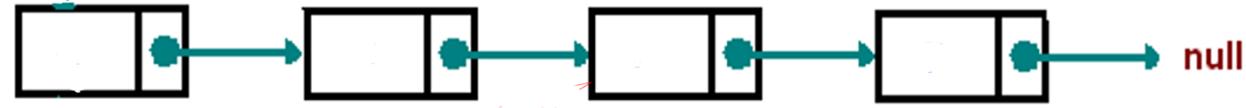
```
{
```

```
        printf("%d ", ptr->data);
```

```
        ptr = ptr->next;
```

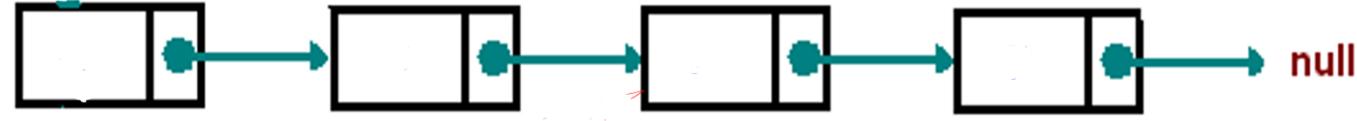
```
}
```

```
}
```



Q Write a C-style pseudocode for Traversing a link list recursively, where pointer head have the address of the first node of the list?

```
void traverse(Node* ptr)
{
    if (ptr == NULL)
        return;
    printf("%d ", ptr->data);
    traverse(ptr->next);
}
```



Q Write a C-style pseudocode for searching a key in a link list iteratively, where pointer head have the address of the first node of the list?

```
Node* search(Node* head, int key)
```

```
{
```

```
    Node* ptr = head;
```

```
    while (ptr != NULL)
```

```
{
```

```
        if (ptr->data == key)
```

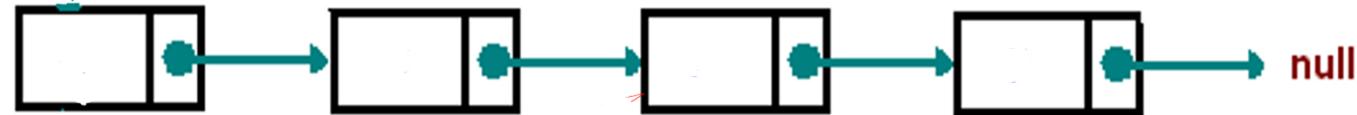
```
            return ptr;
```

```
        else
```

```
            ptr = ptr->next;
```

```
}
```

```
return NULL;
```



Q Write a C-style pseudocode for inserting a node with a key in the starting of link-list?

Typedef struct node

{

```
    int data;  
    struct node* next;
```

}

Node* CreateNewNode(int key)

{

```
    Node* new = (Node*) malloc(sizeof(Node));  
    new → data = key;  
    new → next = NULL;  
    return new;
```

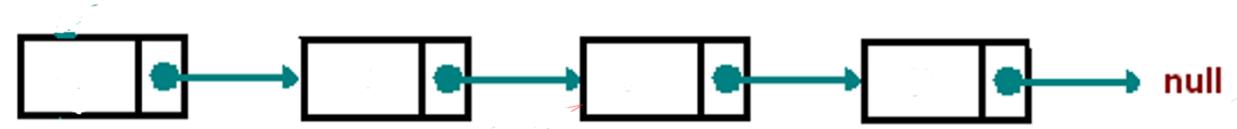
}

Void Insert(Node** head, int key)

{

```
    node* new = CreateNewNode(key);  
    new → next = *head;  
    *head = new;
```

}



Q Write a C-style pseudocode for inserting a node with a key after a location in a link-list?

```
Void InsertAfter(Node* Loc, int key)
```

```
{
```

```
    if(Loc == null)
```

```
{
```

```
        return;
```

```
}
```

```
    node* new = CreateNewNode(key);
```

```
    new → next = loc → next ;
```

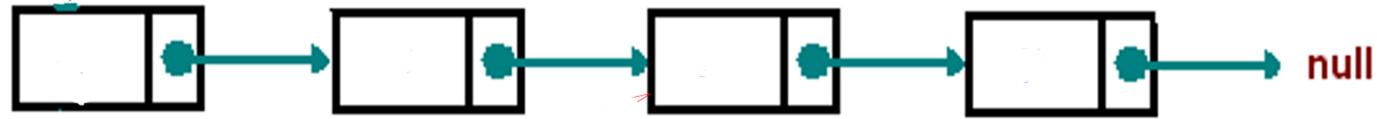
```
    loc → next = new;
```

```
}
```



Q Write a C-style pseudocode for deleting a node from the starting of link-list?

```
void deleteAtHead(Node** head)
{
    if (*head == NULL)
        return;
    Node* ptr = *head;
    *head = (*head)->next;
    free(ptr);
}
```



Q Write a C-style pseudocode for deleting a node after a given location from the starting of link-list?

```
void deleteAfter(Node* loc)
```

```
{  
    if (loc == NULL)  
        return;  
    if (loc->next == NULL)  
        return;  
    Node* ptr = loc->next;  
    loc->next = ptr->next;  
    free(ptr);  
}
```



Q Write a C-style pseudocode for reversing a link-list in a iteratively?

Void reverse(node** head)

{

if(*head!=null)

{

Struct node* p = *head;

Struct node* q = null;

Struct node* r;

While(p !=null)

{

r=q;

q=p;

p=p->next;

q->next = r;

}

*head = q;

}



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) [Asked in Accenture 2020]

```
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
(C) 1,3,2,5,4,7,6

- (B) 2,1,4,3,6,5,7
(D) 2,3,4,5,6,7,1

Q The following C function takes a singly-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) $q = \text{NULL}; p->\text{next} = \text{head}; \text{head} = p;$
- (B) $q->\text{next} = \text{NULL}; \text{head} = p; p->\text{next} = \text{head};$
- (C) $\text{head} = p; p->\text{next} = q; q->\text{next} = \text{NULL};$
- (D) $q->\text{next} = \text{NULL}; p->\text{next} = \text{head}; \text{head} = p;$

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.

Q Consider the function f defined below. (GATE-2003) (2 Marks) [Asked in Cognizant 2020]

```
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.

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

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;
```

```
}
```

```
}
```

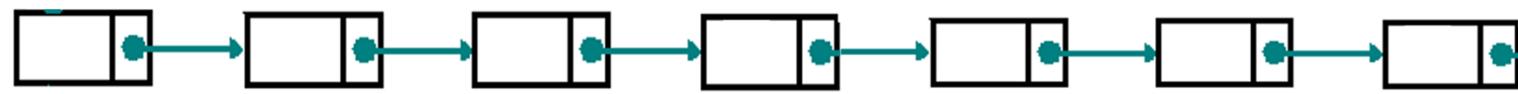
Which of the following is **FALSE** about above function?

(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

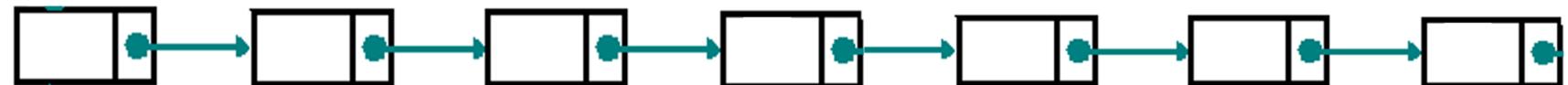
(C) The function is implemented incorrectly because it changes head

(D) none



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

```
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)
```

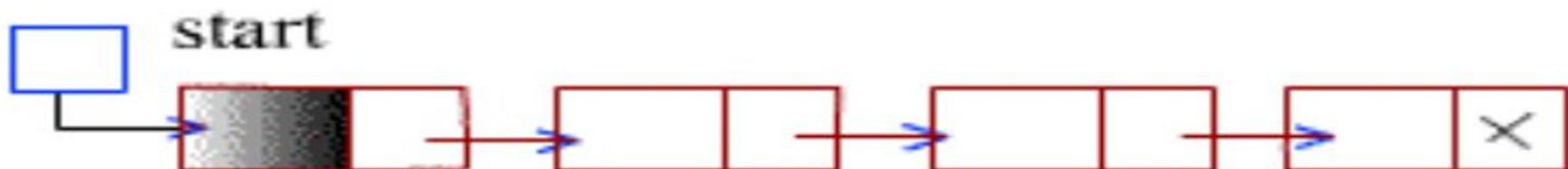
```
{
    struct node* prev = NULL;
    struct node* current = *head_ref;
    struct node* next;
    while (current != NULL)
    {
        next = current->next;
        current->next = prev;
        prev = current;
        current = next;
    }
    /*ADD A STATEMENT HERE*/
}
```

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;

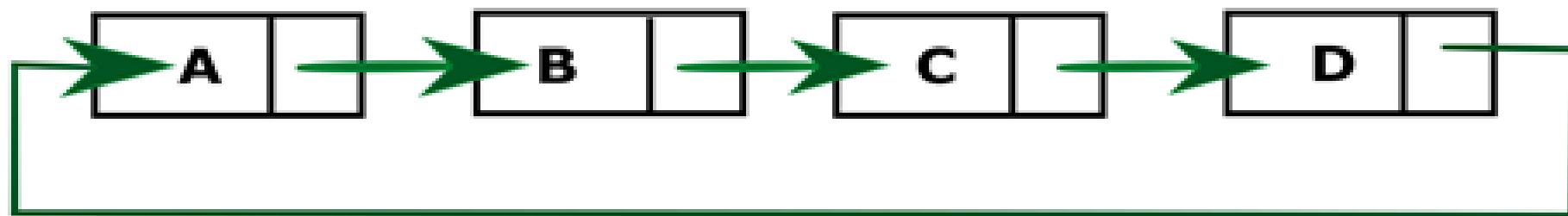
Header link list

- A header linked list is a variation of a standard linked list that begins with a special node called the header node. This node doesn't store actual data but serves as a fixed reference point, simplifying operations like insertion or deletion at the beginning of the list.
 - **Consistent Starting Point:** The header node is always present, even when the list is empty, providing a consistent starting point for traversal and modification.
 - **Pointer to First Data Node:** Its next pointer points to the first actual data node or NULL if the list is empty.
 - **Simplified Operations:** It simplifies insertion and deletion at the beginning, middle, or end of the list due to the consistent structure.
 - **Metadata Storage:** Importantly, the header node can store metadata about the list, such as its length or other properties, enhancing efficiency in operations that require this information.



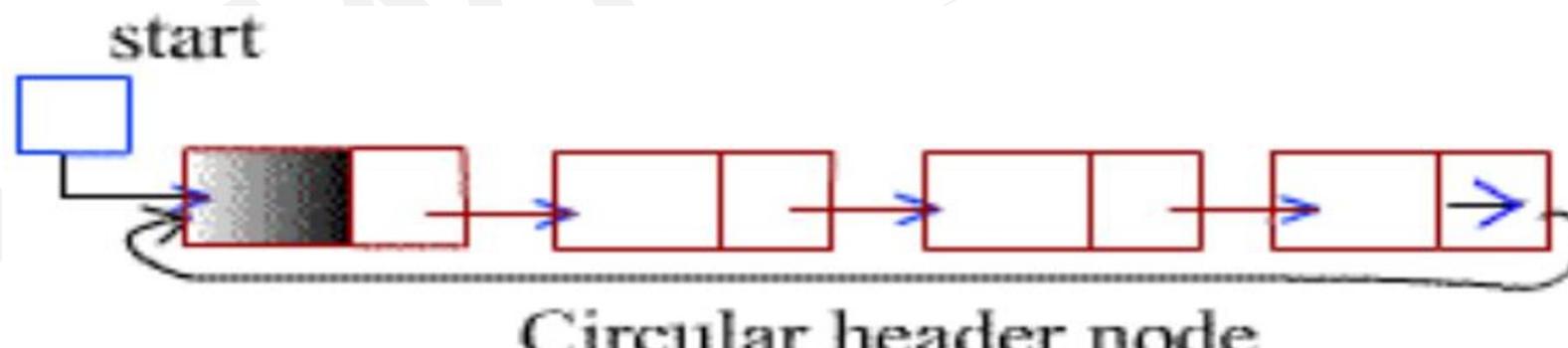
Circular Linked List

- A singly circular linked list is a variation of a singly linked list where the last node points back to the first node, forming a continuous loop. Unlike a standard singly linked list that ends with a NULL pointer, the last node in a circular linked list references the first node.
 - **Circular Structure:** The next pointer of the last node points to the first node, eliminating the NULL reference and creating a circular flow.
 - **Efficient Operations:** Useful for implementing data structures like queues and circular buffers, allowing constant-time addition at the end and removal from the front.
 - **Traversal Considerations:** Requires a stopping condition during traversal—such as returning to the starting node or using a counter—to prevent infinite loops.



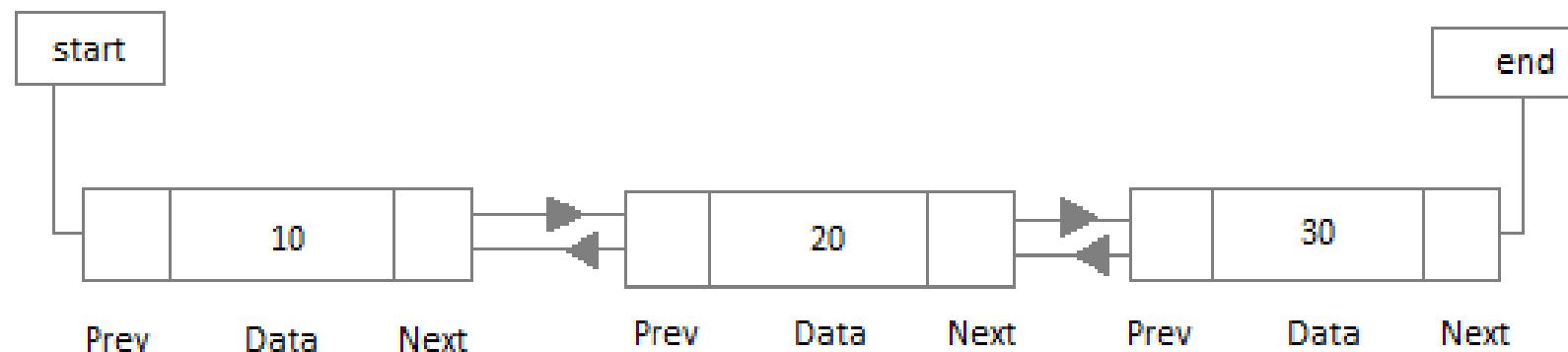
Header circular link list

- A header circular linked list is a singly circular linked list that begins with a special node called the **header node**, which doesn't contain actual data but serves as a fixed reference point. This simplifies operations like insertion and deletion at the beginning or end of the list by eliminating special cases.
 - **Consistent Starting Point:** The header node provides a uniform starting point, and the last node's next pointer points back to it, maintaining the circular structure.
 - **Simplified Operations:** With the header node always present, there's no need to handle special cases for empty lists or edge nodes, reducing code complexity.
 - **Metadata Storage:** The header node can store metadata such as the list's length, optimizing performance for certain operations.



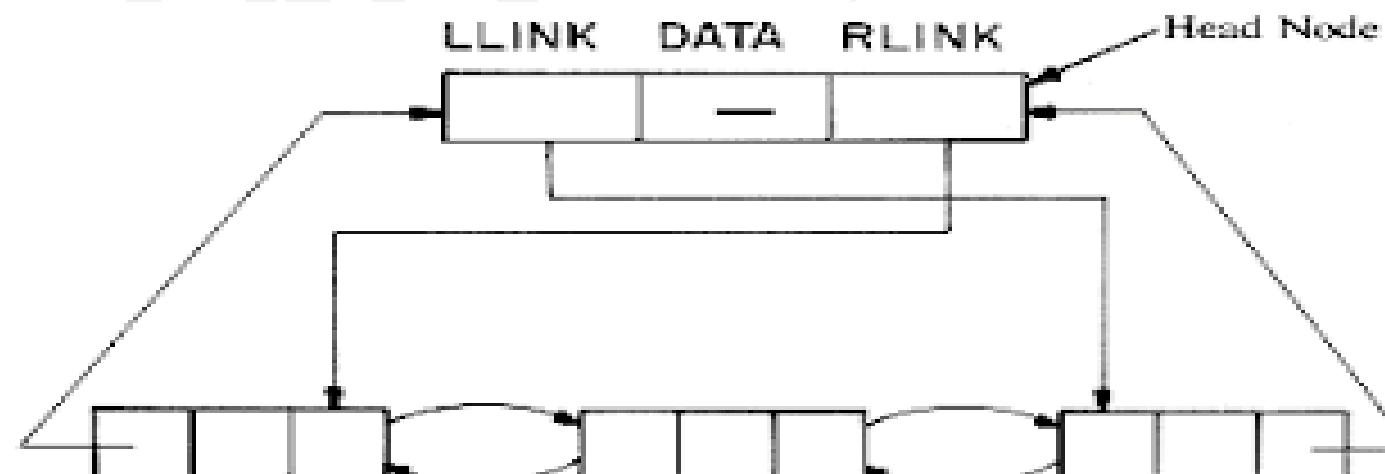
Doubly link list

- A doubly linked list is a data structure where each node contains a data element and two pointers: one pointing to the previous node and one to the next node. This bidirectional linking allows for traversal and manipulation of the list in both forward and backward directions, simplifying operations like insertion or deletion at any position.
 - **Bidirectional Pointers:** Each node has a next pointer to the subsequent node and a previous pointer to the preceding node.
 - **Terminal Nodes:** The previous pointer of the first node and the next pointer of the last node are set to NULL, indicating the start and end of the list.
 - **Ease of Traversal:** Enables efficient traversal and manipulation in both directions compared to singly linked lists.
 - **Memory Overhead:** Consumes more memory than singly linked lists due to the additional previous pointer.



Header Circular Doubly Link List

- A header circular doubly linked list is a doubly linked list that includes a special header node and forms a circular structure. The header node does not contain data but serves as a fixed reference point, simplifying operations by eliminating special cases when inserting or deleting at the beginning or end.
 - **Header Node:** Provides a consistent starting point for traversal and manipulation, and can store metadata like the list's length.
 - **Circular Structure:** The last node's next pointer points back to the header node, and the header's previous pointer points to the last node, allowing continuous traversal without encountering NULL pointers.
 - **Bidirectional Links:** Each node has next and previous pointers, enabling efficient traversal in both forward and backward directions.



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*. **(GATE-1996) (2 Marks)**

```
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

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:

$\Theta(N)$ delete, $O(\log N)$ insert, $O(\log N)$ find, and $\Theta(N)$ decrease-key

What is the time complexity of all these operations put together (**Gate-2016**) (1 Marks)

(A) $O(\log^2 N)$

(B) $O(N)$

(C) $O(N^2)$

(D) $\Theta(N^2 \log N)$

Q The concatenation of two lists is to be performed in O(1) time. Which of the following implementations of a list should be used? **(Gate-1997) (1 Marks)**

- (A) singly linked list
- (B) doubly linked list
- (C) circular doubly linked list
- (D) array implementation of lists

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

Q In the worst case, the number of comparisons needed to search a single linked list of length n for a given element is **(Gate-2002) (1 Marks)**

a) $\log n$

b) $n/2$

c) $\log_2 n - 1$

d) n

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) [Asked in AMCAT 2017]**

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

Q Consider the following statements: (GATE - 1996) (2 Marks) [Asked in TCS NQT 2019]

- 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

Q Consider the following ANSI C program: Which one of the following statements below is correct about the program? (GATE 2021) (2 MARKS)

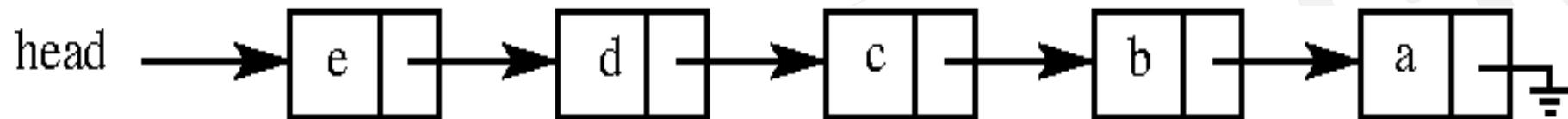
```
#include <stdio.h>
#include <stdlib.h>
struct Node
{
    int value;
    struct Node *next;
};
int main( )
{
    struct Node *boxE, *head, *boxN; int index=0;
    boxE=head= (struct Node *) malloc(sizeof(struct Node));
    head → value = index;
    for (index =1; index<=3; index++)
    {
        boxN = (struct Node *) malloc (sizeof(struct Node));
        boxE → next = boxN;
        boxN → value = index;
        boxE = boxN;
    }
    for (index=0; index<=3; index++)
    {
        printf("Value at index %d is %dn", index, head → value);
        head = head → next;
        printf("Value at index %d is %dn", index+1, head → value);
    }
}
```

- (a)** Upon execution, the program creates a linked-list of five nodes
- (b)** Upon execution, the program goes into an infinite loop
- (c)** It has a missing return which will be reported as an error by the compiler
- (d)** It dereferences an uninitialized pointer that may result in a run-time error

Q Consider the problem of reversing a singly linked list. To take an example, given the linked list below,



the reversed linked list should look like



Which one of the following statements is TRUE about the time complexity of algorithms that solve the above problem in O(1) space? **(GATE 2022) (1 MARKS)**

- (A)** The best algorithm for the problem takes $O(n)$ time in the worst case.
- (B)** The best algorithm for the problem takes $O(n \log n)$ time in the worst case.
- (C)** The best algorithm for the problem takes $O(n^2)$ time in the worst case.
- (D)** It is not possible to reverse a singly linked list in $O(1)$ space.

Q Let SLLdel be a function that deletes a node in a singly-linked list given a pointer to the node and a pointer to the head of the list. Similarly, let DLLdel be another function that deletes a node in a doubly-linked list given a pointer to the node and a pointer to the head of the list. Let n denote the number of nodes in each of the linked lists. Which one of the following choices is TRUE about the worst-case time complexity of SLLdel and DLLdel? **(Gate-2023) (1 Marks)**

- (a)** SLLdel is $O(1)$ and DLLdel is $O(n)$
- (b)** Both SLLdel and DLLdel are $O(\log(n))$
- (c)** Both SLLdel and DLLdel are $O(1)$
- (d)** SLLdel is $O(n)$ and DLLdel is $O(1)$

Q find n^{th} node from the end of a link list?

Q Check whether the link list is either null terminated or ends in a cycle?

Q find the intersection of the two-link list?

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.

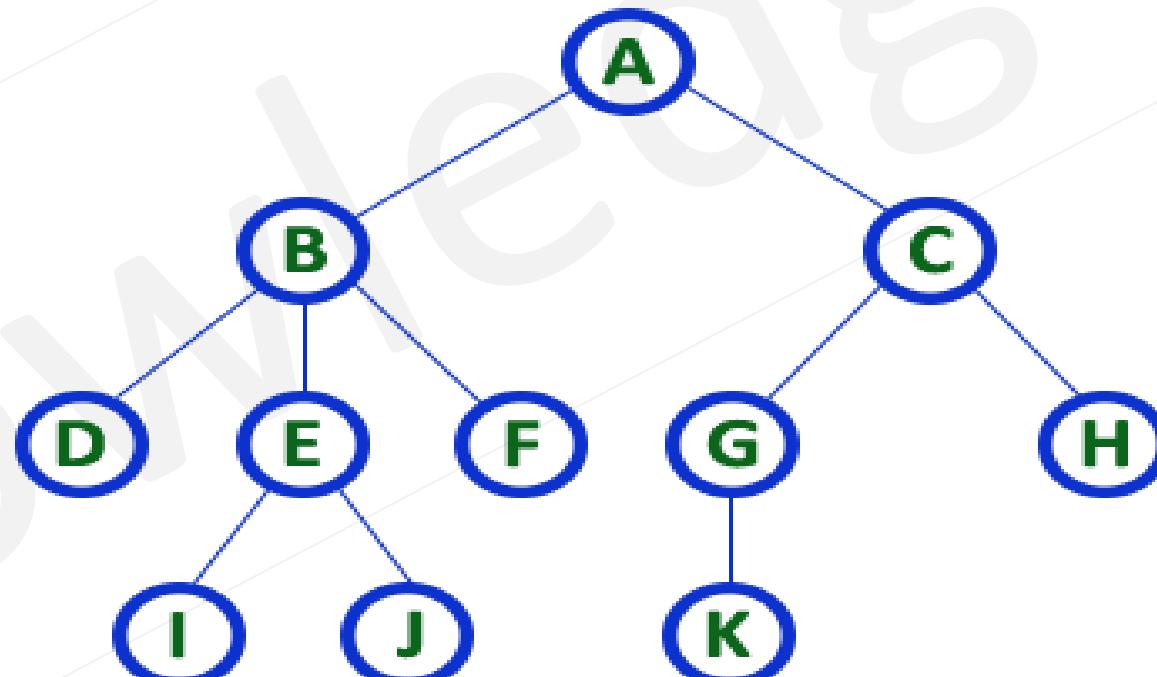
(B) Possible if size of linked list is even.

(C) Possible if size of linked list is odd

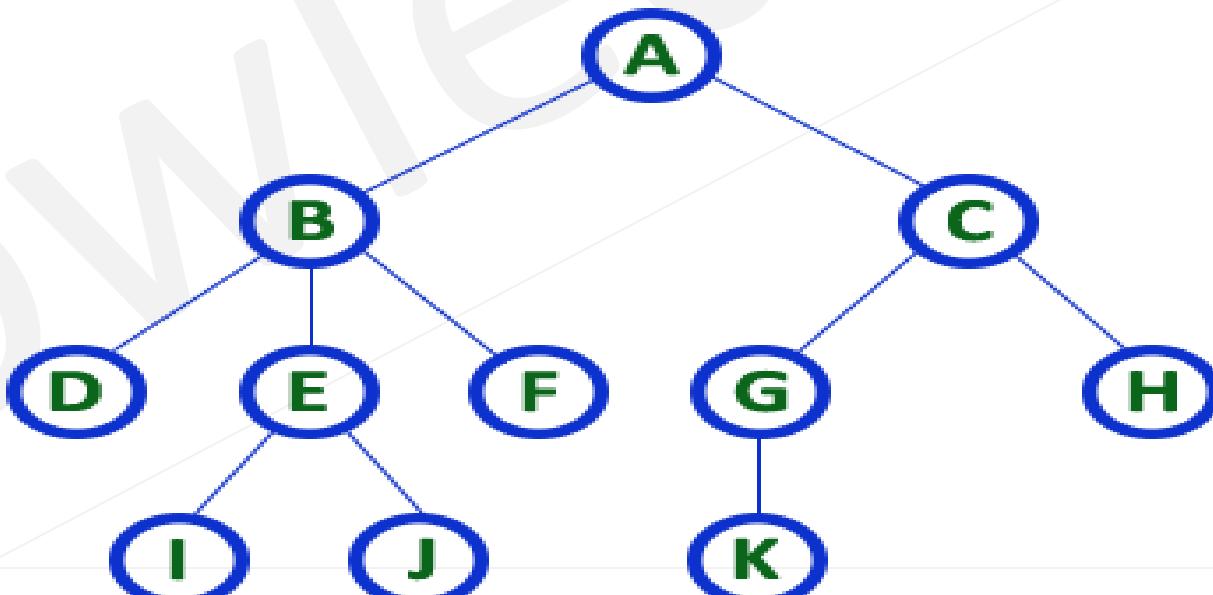
(D) Possible if X is not first node. Use following two steps (a) Copy the data of next of X to X. (b) Delete next of X.

Tree

- A **tree** is a nonlinear data structure that represents hierarchical relationships between data items. It is widely used due to its flexibility and versatility. The **root node** is the starting point, and each node (except the root) is connected to exactly one parent node, forming a **parent-child** relationship.
- **Root Node:** The top-most node in the tree.
- **Subtrees:** A tree can be divided into smaller subsets called **subtrees**, each acting as an independent tree.
- **Applications:** Trees are used in file systems, database indexing, decision-making, and many more.



- **Root Node:** The top-most node, serving as the origin of the tree.
- **Edge:** The connection between two nodes. A tree with 'N' nodes has exactly 'N-1' edges.
- **Parent Node:** A node that has one or more children.
- **Child Node:** A node that descends from a parent node.
- **Leaf (External) Node:** A node with no children.
- **Internal Node:** A node with at least one child.
- **Degree of Node:** The number of children a node has.
- **Level/Depth:** Indicates the step count from the root (Level 0) to a particular node.
- **Path:** A sequence of nodes connected by edges.
- **Subtree:** A tree formed from a child node and its descendants.



Binary tree

- A **binary tree** is a type of tree data structure where each node can have at most **two children**, commonly referred to as **left** and **right** child nodes. It is structured as a set of nodes with the following properties:
- **Empty Tree:** A binary tree can be empty (no nodes).
- **Root Node:** If not empty, the tree has a unique root node (R).
- **Subtrees:** The remaining nodes are partitioned into two disjoint subtrees, **T1** (left) and **T2** (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 Let T be a binary search tree with 15 nodes. The minimum and maximum possible heights of T are: _____ (GATE-2017) (1 Marks)

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

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}

Q 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 (GATE-2005) (2 Marks)

- a) 2^{h-1}
- b) $2^{h-1} + 1$
- c) $2^h - 1$
- d) 2^h

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)

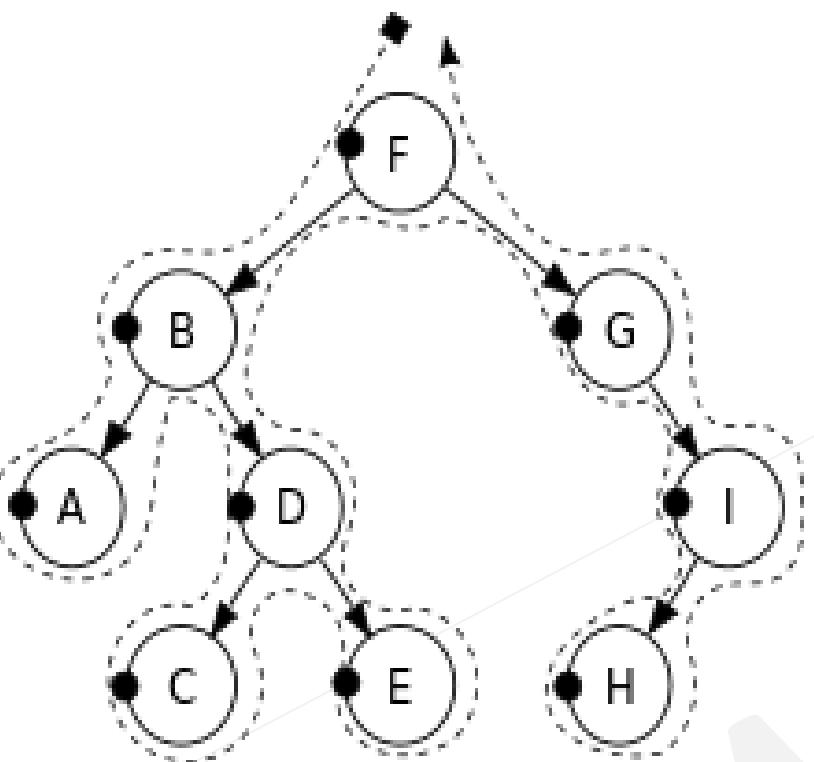
- (A) 0
- (B) 1
- (C) $(n-1)/2$
- (D) $n-1$

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)**

Traversal of binary tree

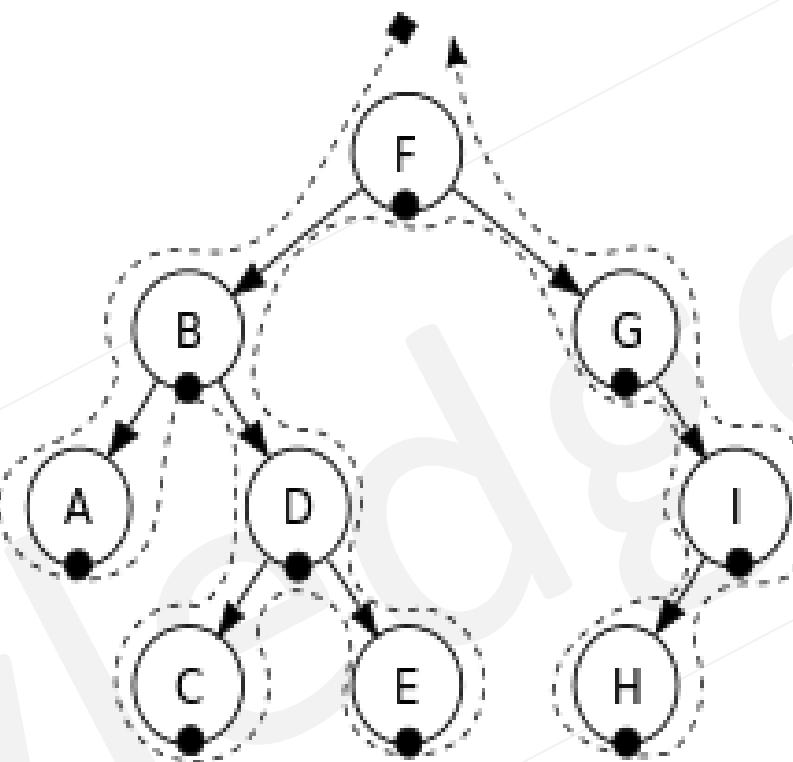
- The process of visiting (checking and/or updating) each node in a tree data structure, exactly once in called tree traversal. Such traversals are classified by the order in which the nodes are visited. Unlike linked lists, one-dimensional arrays and other linear data structures, which are traversed 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.
- **Pre-order (Root-Left-Right):** Visit root, then left subtree, followed by right subtree.
- **In-order (Left-Root-Right):** Visit left subtree, then root, and finally right subtree.
- **Post-order (Left-Right-Root):** Visit left subtree, then right subtree, and root last.

Pre-order (Root L R)



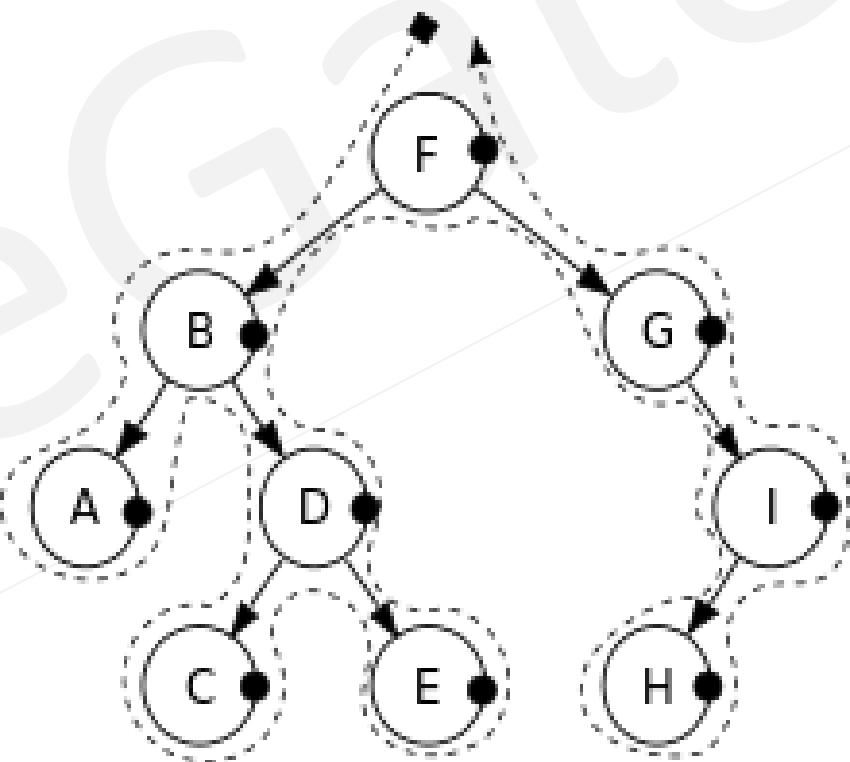
F, B, A, D, C, E, G, I, H

In-order (L root R)



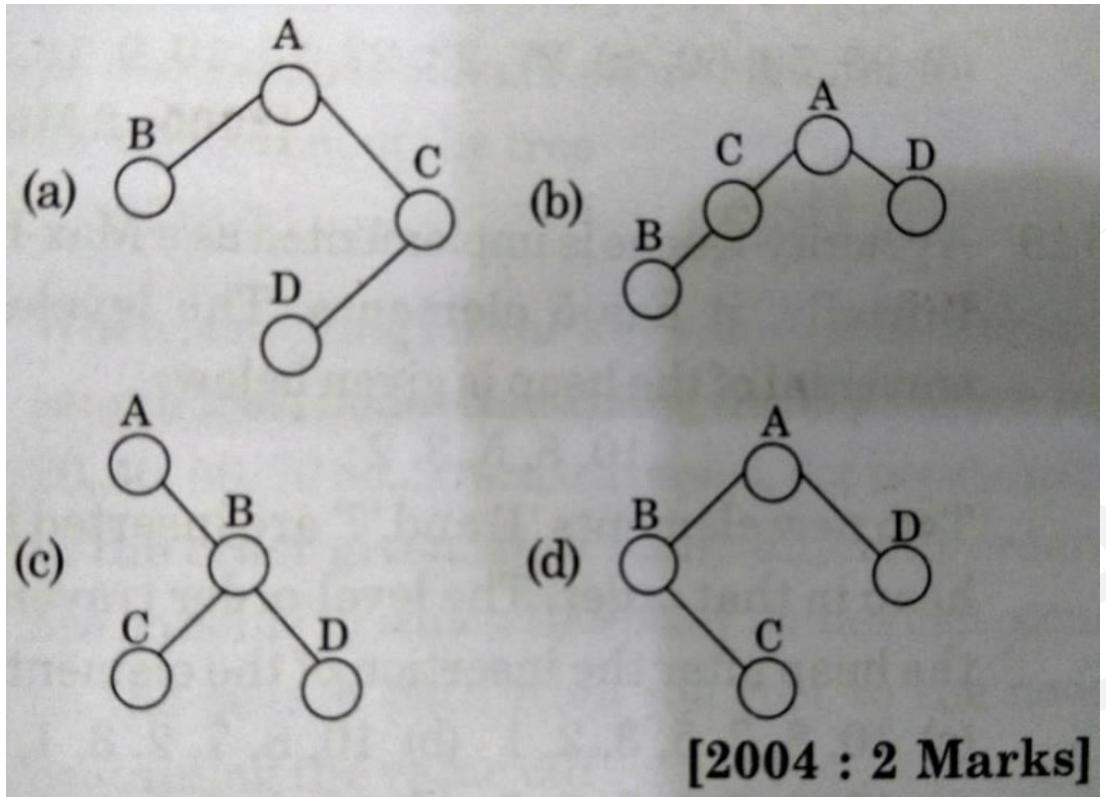
A, B, C, D, E, F, G, H, I

Post-order (L R Root)



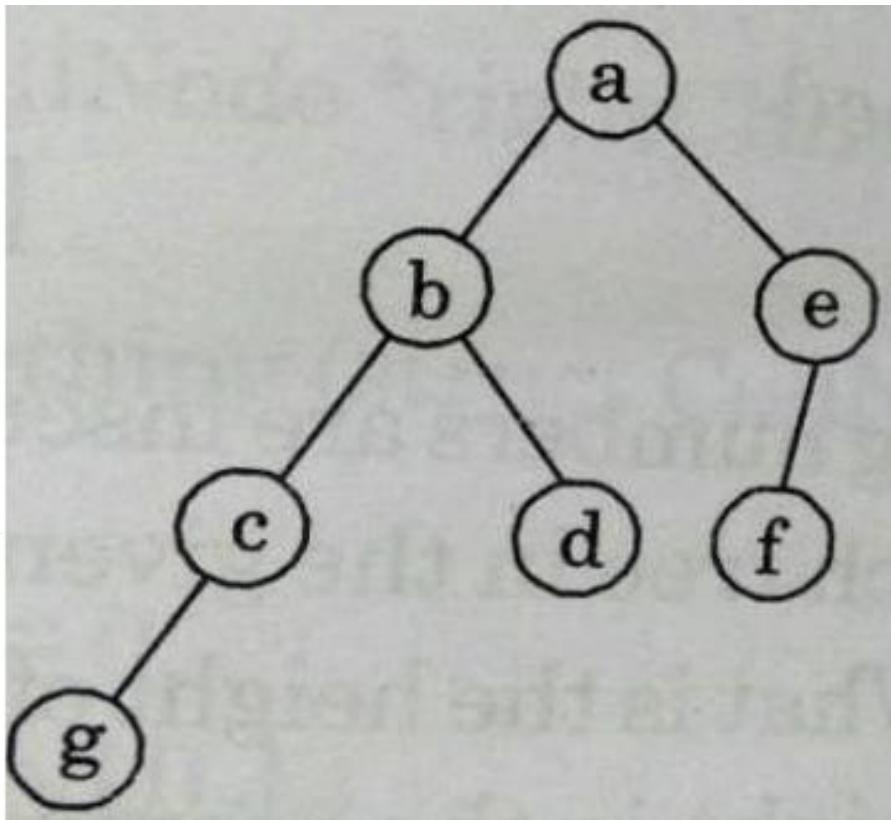
A, C, E, D, B, H, I, G, F

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



Q which of the following is post order traversal of the above tree (**GATE-1991**) (1 Marks)

- a) fegcbdba
- b) gcbdafe
- c) gcdbfeca
- d) fedgcba



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

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

Q The following three are known to be the preorder, inorder and post order 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. **(GATE-2008) (2 Marks)**

- a) I and II are preorder and inorder sequences, respectively**
- b) I and III are preorder and post order 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**

Q Consider the following statements.

- S_1 : The sequence of procedure calls corresponds to a preorder traversal of the activation tree.
- S_2 : The sequence of procedure returns corresponds to a postorder traversal of the activation tree.

Which one of the following options is correct? (GATE 2021)

- (a) S_1 is true and S_2 is false
- (b) S_1 is false and S_2 is true
- (c) S_1 is true and S_2 is true
- (d) S_1 is false and S_2 is false

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)**

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

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

Q Which of the following pairs of traversals is not sufficient to build a binary tree from the given traversals?

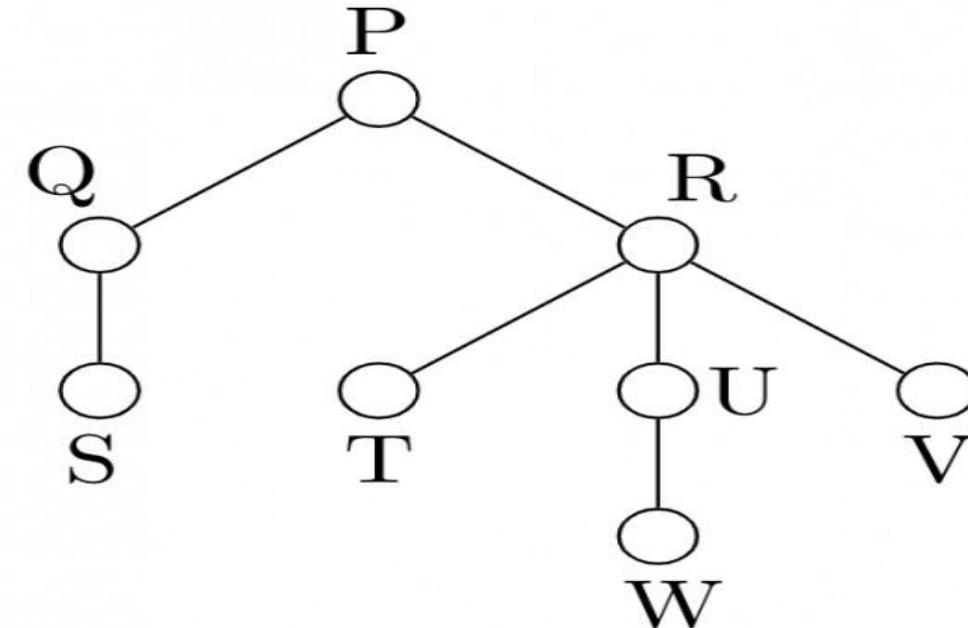
- (A) Preorder and Inorder
- (B) Preorder and Post order
- (C) Inorder and Post order
- (D) level order and post order

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) $1 \ 7 \ 6 \ * \ + \ 2 \ 5 \ 4 \ 3 \ * \ - \ ^ \ -$

Q 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 (GATE-2014) (1 Marks)

- a) SQPTRWUV
- b) SQPTUWRV
- c) SQPTWUVR
- D) SQPTRUWV



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 B_1 and B_2 are

- (A) $B_1 : (1 + \text{height}(n \rightarrow \text{right}))$, $B_2 : (1 + \max(h_1, h_2))$
- (B) $B_1 : (\text{height}(n \rightarrow \text{right}))$, $B_2 : (1 + \max(h_1, h_2))$
- (C) $B_1 : \text{height}(n \rightarrow \text{right})$, $B_2 : \max(h_1, h_2)$
- (D) $B_1 : (1 + \text{height}(n \rightarrow \text{right}))$, $B_2 : \max(h_1, h_2)$

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

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) Returns diameter where diameter is number of edges on the longest path between any two nodes.

Q Following function is supposed to calculate the maximum depth or height of a Binary tree —(root is at level 1)

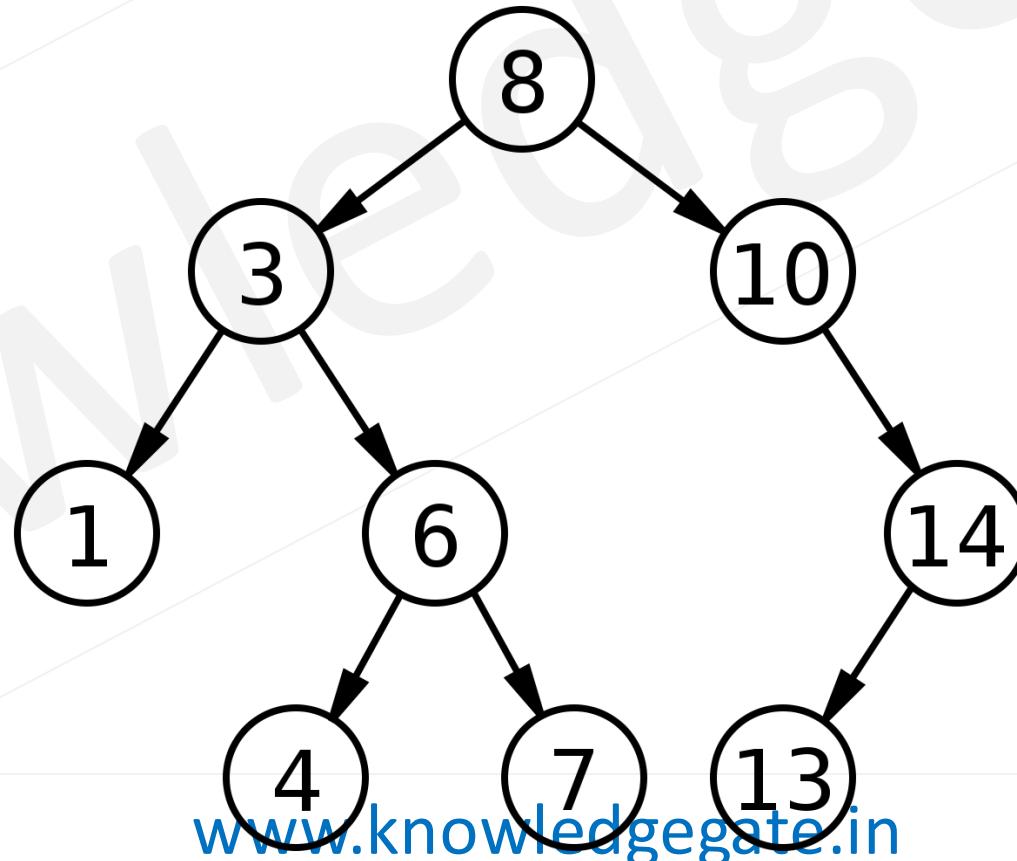
```
int maxDepth (struct node* node)
{
    if (node==NULL)
        return 0;
    else
    {
        /* compute the depth of each subtree */
        int lDepth = maxDepth(node->left);
        int rDepth = maxDepth(node->right);
        /* use the larger one */
        if (lDepth > rDepth)
           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

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



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

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)? **(GATE-2004) (1 Marks)**

- a) 2
- b) 3
- c) 4
- d) 6

Q A binary search tree is generated by inserting in order 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 **(GATE-1996) (2 Marks)**

- a) (4,7)
- b) (7,4)
- c) (8,3)
- d) (3,8)

- **Insertion**

- The insertion operation in a binary tree is similar to a search.
- Start from the root and traverse down the tree:
 - **Left Subtree:** Insert into the left if the new node's key is less than the current node's key.
 - **Right Subtree:** Insert into the right if it's greater than or equal to the current node's key.
- Add the new node at an external position once the correct spot is found.

- **Deletion**

- Three main cases need to be handled when deleting a node:
- **Node with No Children**: Directly remove the node.
- **Node with One Child**: Remove the node and connect its child to its parent.
- **Node with Two Children**:
 - Do not delete the node.
 - Choose the **in-order predecessor** (maximum node of the left subtree) or the **in-order successor** (minimum node of the right subtree).
 - Replace the node's value with the selected node.
 - Remove the selected node from its original position (which will be either a leaf or a node with one child).

- 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

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

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 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

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

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

Q A binary search tree contains the numbers 1,2,3,4,5,6,7,8. When the tree is traversed in pre-order 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 post-order, the sequence obtained would be **(GATE - 2005) (1 Marks)**

- 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

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

Q A binary search tree contains the value 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? **(Gate-1997) (2 Marks)**

a) 5 3 1 2 4 7 8 6

b) 5 3 1 2 6 4 8 7

c) 5 3 2 4 1 6 7 8

d) 5 3 1 2 4 7 6 8

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]

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]

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)

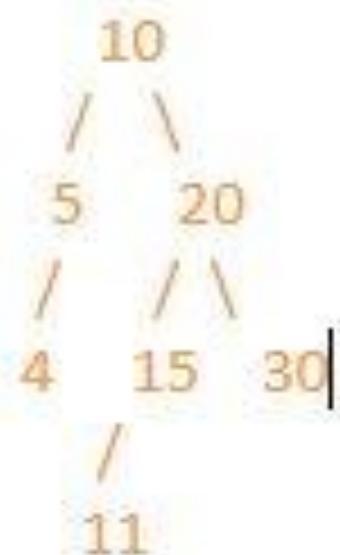
Q When searching for the key value 60 in a binary search tree, nodes containing the key values 10, 20, 40, 50, 70, 80, 90 are traversed, not necessarily in the order given. How many different orders are possible in which these key values can occur on the search path from the root to the node containing the value 60? (**GATE-2007**) (2 Marks)

- a) 35
- b) 64
- c) 128
- d) 5040

Q 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 **(GATE-2005) (1 Marks)**

- a) P
- b) $p+1$
- c) $n-p$
- d) $n-p+1$

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

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) $O(\log n)$ for all
- (C) $O(\log n)$ for search and insert, and $O(n)$ for delete
- (D) $O(\log n)$ for search, and $O(n)$ for insert and delete

Q What are the worst-case complexities of insertion and deletion of a key in a binary search tree? **(GATE-2015) (1 Marks)**

- a) $\Theta(\log n)$ for both insertion and deletion
- b) $\Theta(n)$ for both insertion and deletion
- c) $\Theta(n)$ for insertion and $\Theta(\log n)$ for deletion
- d) $\Theta(\log n)$ for insertion and $\Theta(n)$ for deletion

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)**

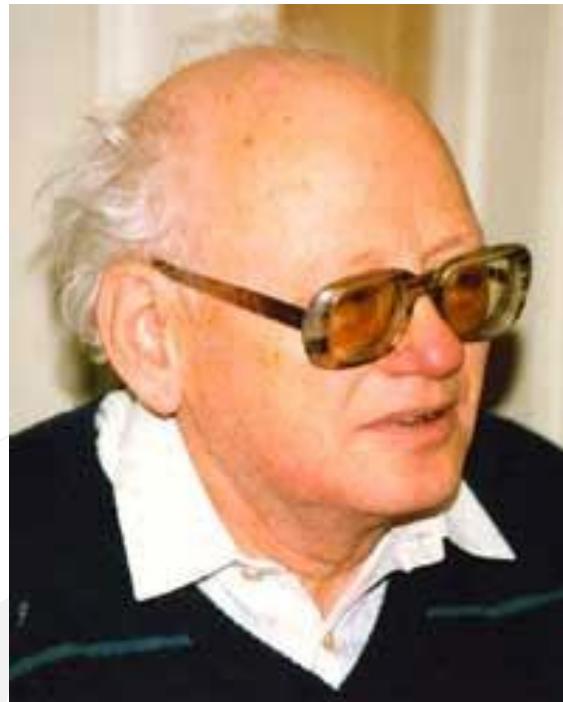
- a) $O(1)$
- b) $O(\log n)$
- c) $O(n)$
- d) $O(n \log n)$

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)** $\Theta(\log n)$
- b)** $\Theta(n)$
- c)** $\Theta(n \log n)$
- d)** None of the above, as the tree cannot be uniquely determined

AVL tree

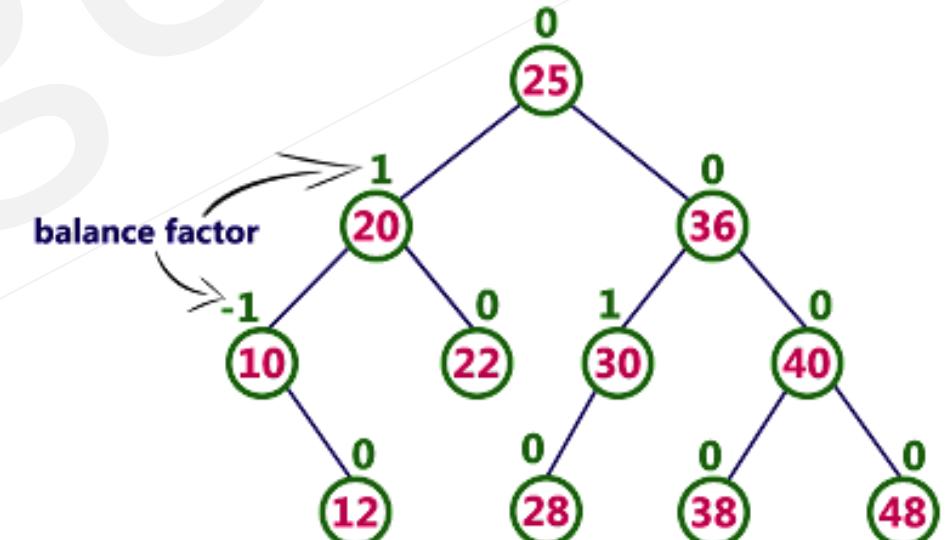
- In computer science, an **AVL tree** (named after inventors **Adelson-Velsky** and **Landis**) is a self-balancing binary search tree. It was the first such data structure to be invented. In an AVL tree, the heights of the two child subtrees of any node differ by at most one; if at any time they differ by more than one, rebalancing is done to restore this property.



Adelson-Velsky



Landis



Balance factor

- In a binary tree the *balance factor* of a node N is defined to be the height difference $\text{Balance Factor}(N) := \text{Height}(\text{LeftSubtree}(N)) - \text{Height}(\text{RightSubtree}(N))$ of its two child subtrees.
- A binary tree is defined to be an *AVL tree* if the $\text{invariant_Balance Factor}(N) \in \{-1, 0, +1\}$ holds for every node N in the tree.
- A node N with $\text{Balance Factor}(N) > 0$ is called "left-heavy"
- One with $\text{Balance Factor}(N) < 0$ is called "right-heavy"
- One with $\text{Balance Factor}(N) = 0$ 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.
- After every insertion at most two rotations are sufficient to balance the AVL tree

Problem	Solution
LL	R
RR	L
LR	LR
RL	RL

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

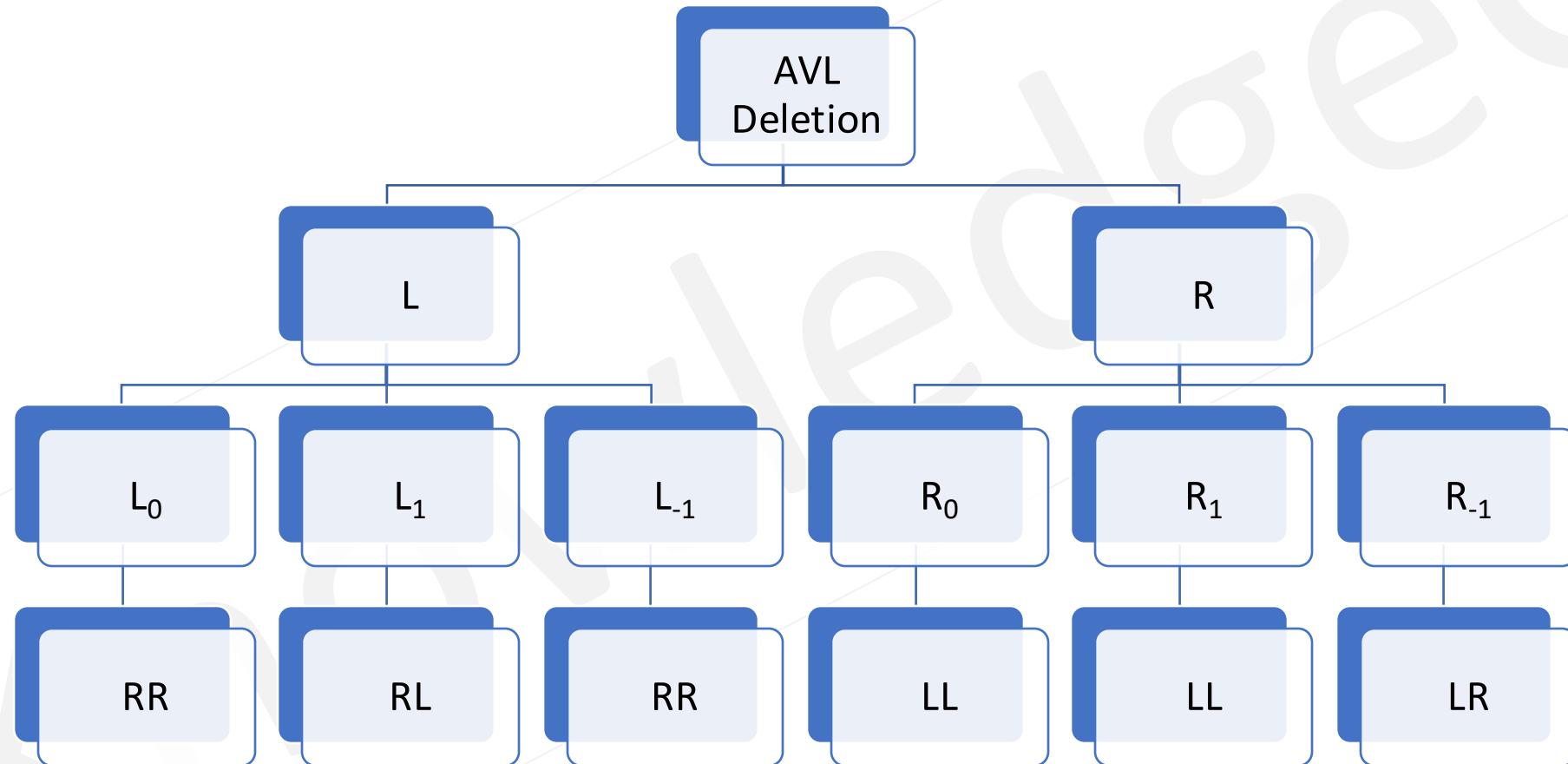
Q What is the worst-case possible height of AVL tree?

- (A) $2 \log_2 n$
- (B) $1.44 \log_2 n$
- (C) Depends upon implementation
- (D) $\Theta(n)$

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)

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

Deletion in an AVL tree



Deletion in an AVL tree

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

Analysis of AVL tree

- 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	$O(\log n)$	$O(n)$
Insert	$O(\log n)$	$O(n)$
Delete	$O(\log n)$	$O(n)$

Binary search tree

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

AVL tree

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

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)

Q The worst-case running time to search for an element in a balanced binary search tree with n^{2^n} elements is (GATE-2013) (1 Marks)

- (A) $\Theta(n \log n)$
- (B) $\Theta(n^{2^n})$
- (C) $\Theta(n)$
- (D) $\Theta(\log n)$

Q Which of the following is TRUE? (GATE-2008) (1 Marks)

- 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 $O(n)$

Q A binary search tree T contains n distinct elements. What is the time complexity of picking an element in T that is smaller than the maximum element in T? **(GATE 2021)**

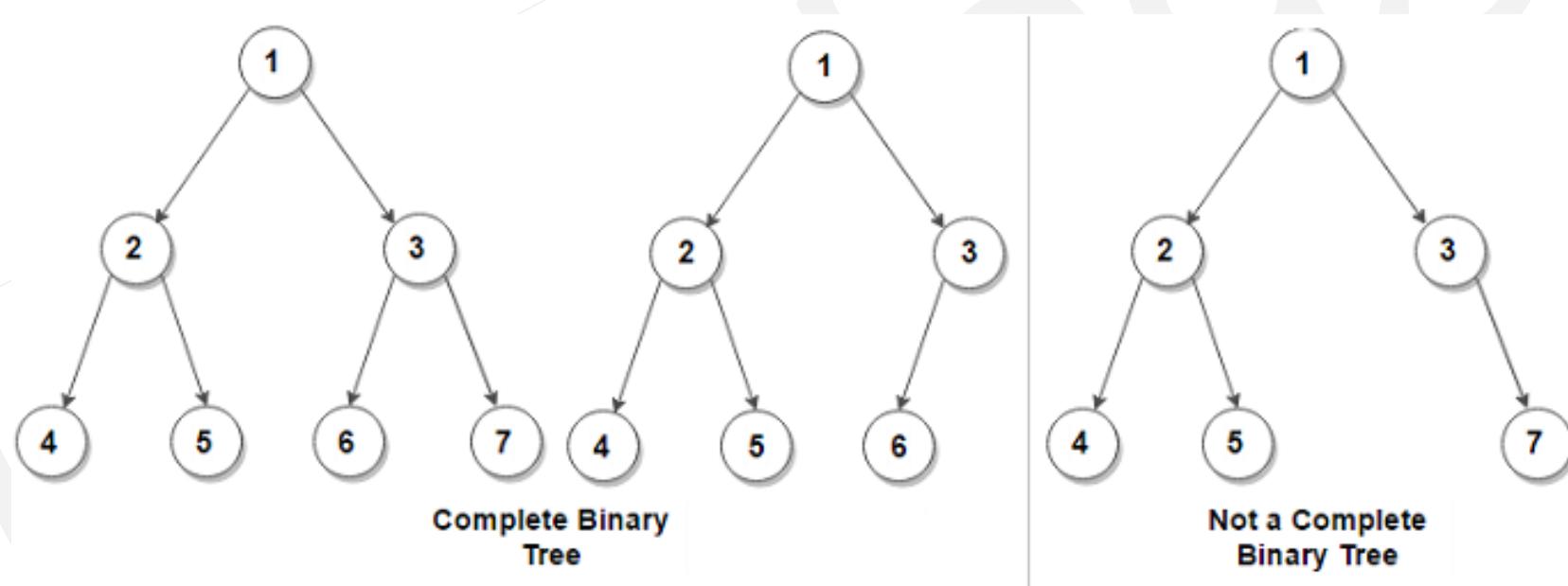
- (a) $\Theta(n \log n)$
- (b) $\Theta(n)$
- (c) $\Theta(\log n)$
- (d) $\Theta(1)$

Q Which one of the following sequences when stored in an array at locations $A[1], \dots, A[10]$ forms a max-heap? **(Gate-2023) (1 Marks)**

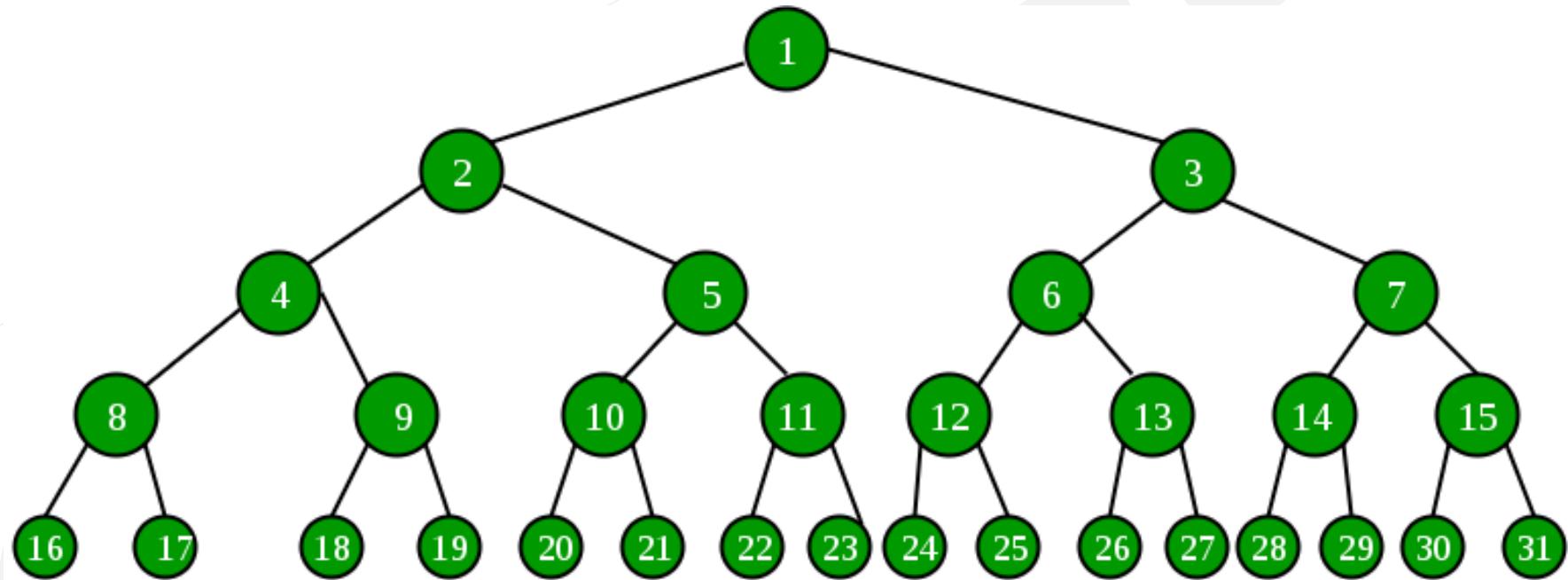
- (a)** 23, 17, 10, 6, 13, 14, 1, 5, 7, 12
- (b)** 23, 17, 14, 7, 13, 10, 1, 5, 6, 12
- (c)** 23, 17, 14, 6, 13, 10, 1, 5, 7, 15
- (d)** 23, 14, 17, 1, 10, 13, 16, 12, 7, 5

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 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)**

[Asked in Hexaware 2020]

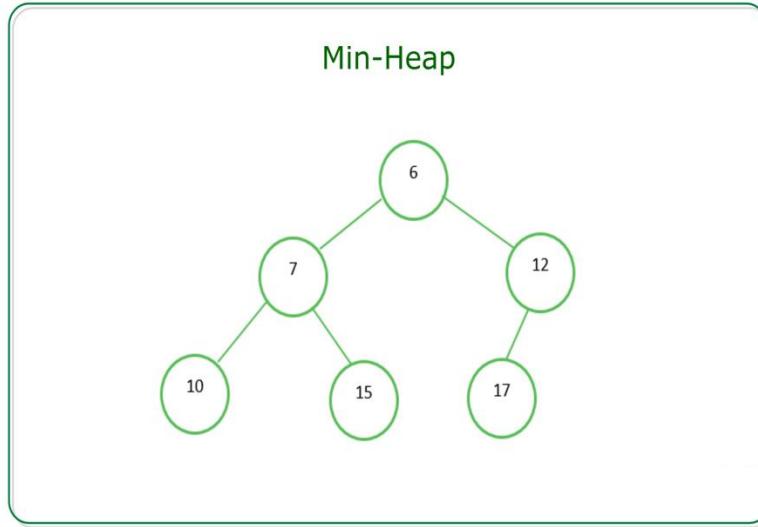
- (A) $\log_2 n$
- (B) n
- (C) $2n + 1$
- (D) $2^n - 1$

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)**

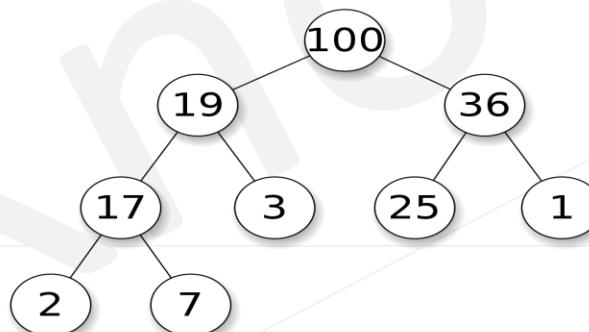
- (A) LASTIN = LASTPOST
- (B) LASTIN = LASTPRE
- (C) LASTPRE = LASTPOST
- (D) None of the above

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.

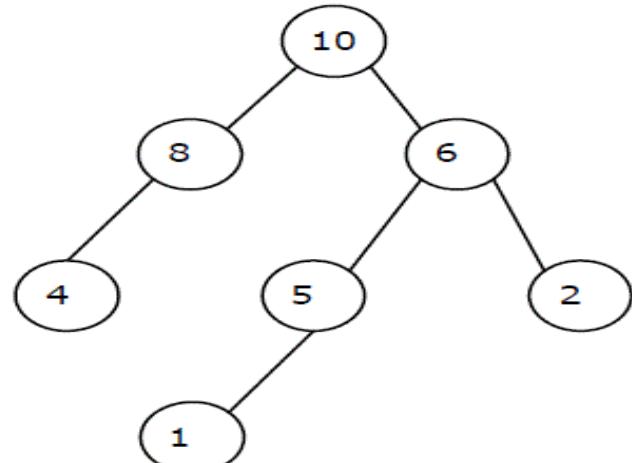


Tree representation

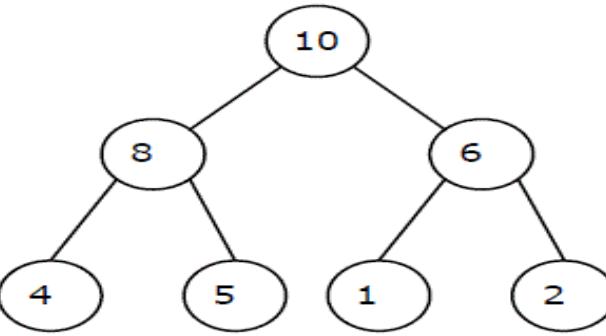


Q A max-heap is a heap where the value of each parent is greater than or equal to the values of its children. Which of the following is a max-heap? (GATE - 2011) (2 Marks)

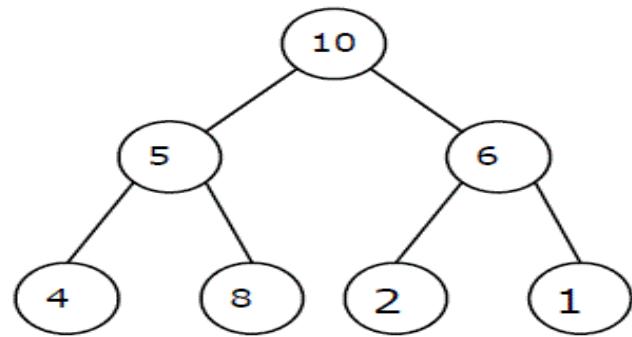
(A)



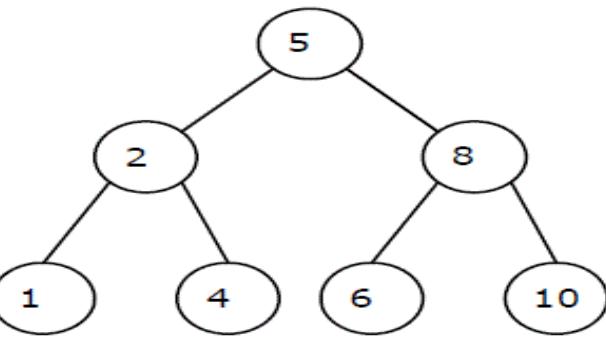
(B)



(C)



(D)



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

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 ($i \leq n$), the index of the parent is

(GATE - 2001) (1 Marks) [Asked in Cognizant 2020]

- (A) $i - 1$
- (B) $\text{floor}(i/2)$
- (C) $\text{ceiling}(i/2)$
- (D) $(i+1)/2$

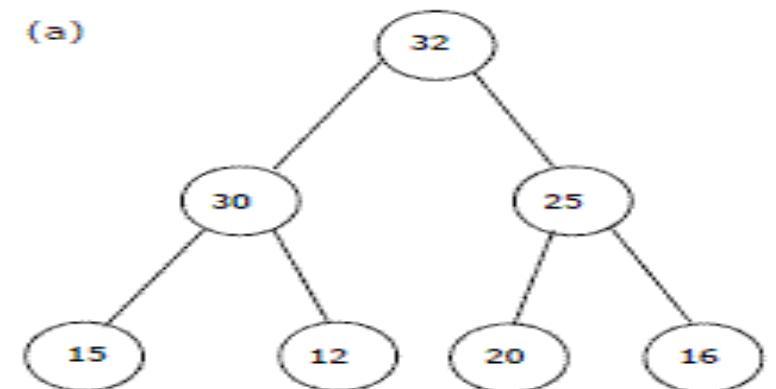
Q The number of possible min-heaps containing each value from {1, 2, 3, 4, 5, 6, 7} exactly once is _____. **(Gate-2018) (1 Marks)**

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

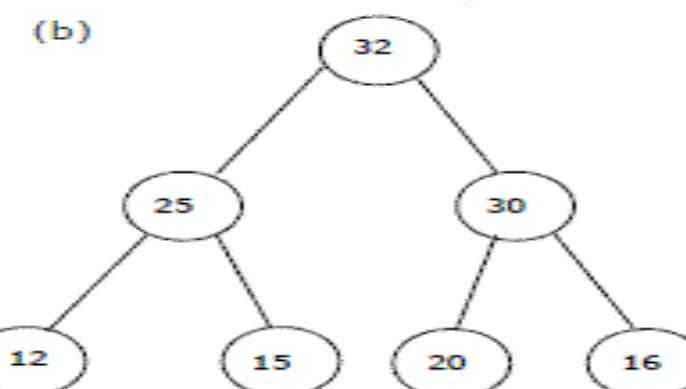
 (Gate-2016) (1 Marks)

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)

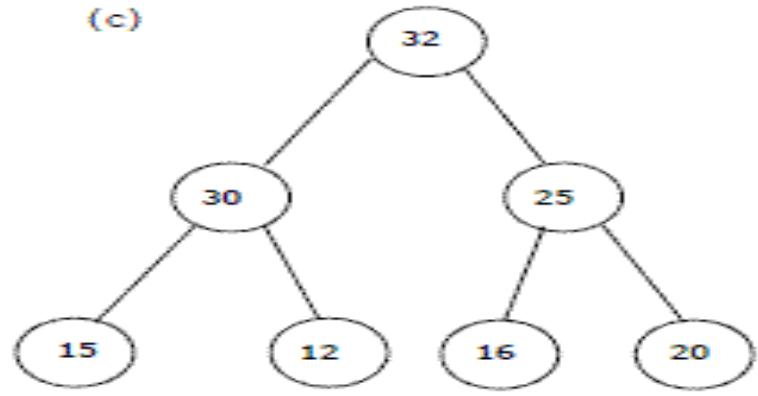
(a)



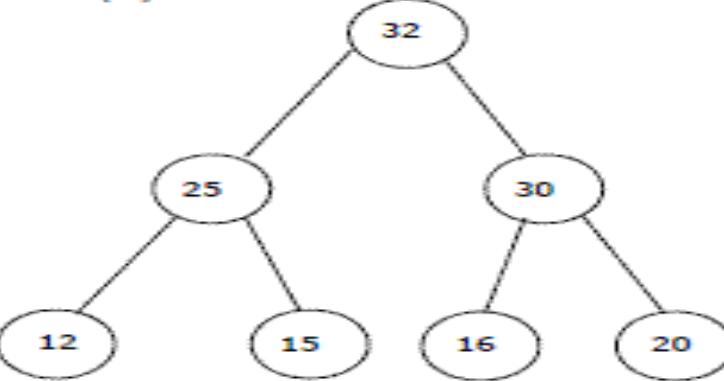
(b)



(c)



(d)



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)**

Q Consider the following array of elements. $\langle 89, 19, 50, 17, 12, 15, 2, 5, 7, 11, 6, 9, 100 \rangle$. 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

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)

Array index	1	2	3	4	5	6	7	8	9
Value	40	30	20	10	15	16	17	8	4

- 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

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)** [Asked in Accenture]

- (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

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**

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

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 $O(n^a \log^b)$. Then the value of $a + 10b$ is _____ (GATE-2015) (1 Marks)

Q 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 (ie., the number of edges in the unique path between a and b) is (rounded off to 2 decimal places) _____. **(GATE-2019) (2 Marks)**

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)

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

- (A) $O(n)$
- (B) $O(\log n)$
- (C) $O(\log \log n)$
- (D) $O(1)$

Q 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.

- I) Deletion of the smallest element
- II) 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? **(GATE - 2003)**
(2 Marks)

- 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 search tree nor heap can be used

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)** [Asked in Capgemini 2016]

- (A) 0 (B) 1 (C) $n!$ (D) $(1/(n+1)).2nC_n$

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

Q How many distinct binary search trees can be created out of 4 distinct keys?

(A) 4

(B) 14

(C) 24

(D) 42

Q how many distinct BST can be constructed with 3 distinct keys?

- a) 4
- b) 5
- c) 6
- d) 9

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)**

Q The number of leaf nodes in a rooted tree of n nodes, with each node having 0 or 3 children is: **(GATE - 2002) (2 Marks)** [Asked in E-litmus 2018]

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

Q 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 **(GATE - 1998) (2 Marks)**

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

Q In a complete k-ary tree, every internal node(n) has exactly k children or no child. The number of leaves in such a tree with an internal node is:

- (A) $n.k$
- (B) $(n - 1) k + 1$
- (C) $n (k - 1) + 1$
- (D) $n (k - 1)$

Q A binary tree T has 20 leaves. The number of nodes in T having two children is _____. (GATE - 2015) (1 Marks)

Q 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
(GATE - 2006) (1 Marks)

- a) 10
- b) 11
- c) 12
- d) 15

Q A binary tree T has n leaf nodes. The number of nodes of degree 2 in T is **(GATE-1995) (1 Marks)**

- a) $\log_2 n$
- b) $n-1$
- c) n
- d) 2^n

Q 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? **(GATE - 2000) (1 Marks)**

- a) (1 2 (4 5 6 7))
- b) (1 (2 3 4) 5 6) 7)
- c) (1 (2 3 4) (5 6 7))
- d) (1 (2 3 NULL) (4 5))

Q Which of the following statements is false? (GATE - 1998) (1 Marks)

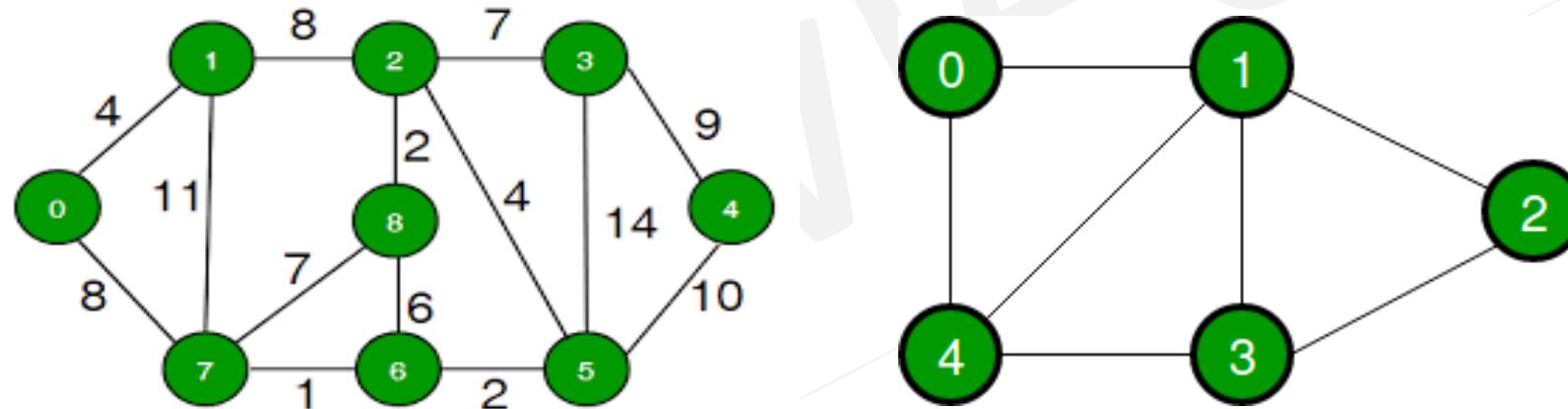
- a) A tree with n nodes has $(n-1)$ edges
- b) A labeled rooted binary tree can be uniquely constructed given its post order and preorder traversal results.
- c) A complete binary tree with n internal nodes has $(n+1)$ leaves.
- d) The maximum number of nodes in a binary tree of height h is $2^{h+1}-1$

Q Let H be a binary min-heap consisting of n elements implemented as an array. What is the worst case time complexity of an optimal algorithm to find the maximum element in H? **(GATE 2021) (1 MARKS)**

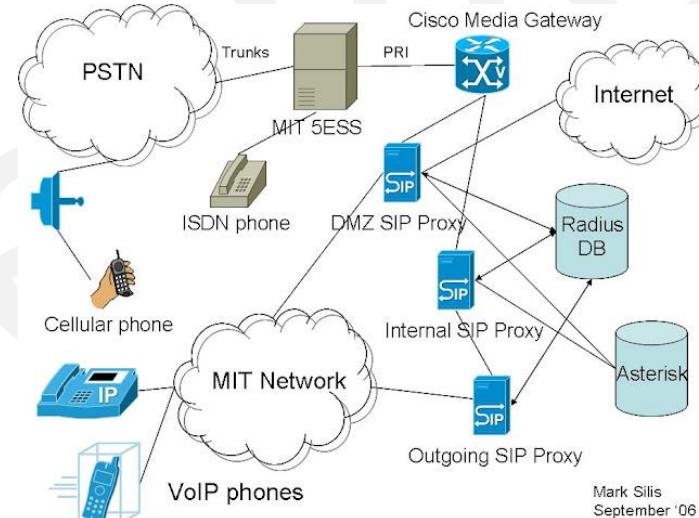
- (A) $\Theta(1)$
- (B) $\Theta(\log n)$
- (C) $\Theta(n)$
- (D) $\Theta(n \log n)$

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.



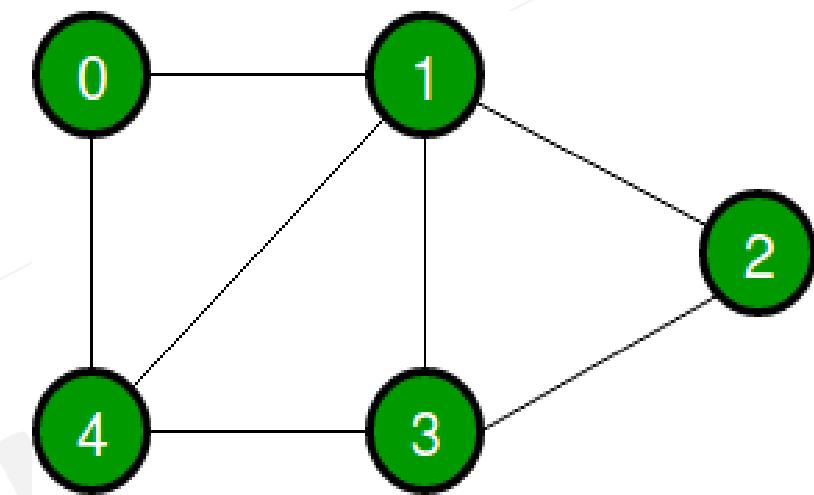
- Graphs are used to represent many real-life applications: Graphs are used to represent networks. The networks may include paths in a city or telephone network or circuit network.
- Graphs are also used in social networks like LinkedIn, Facebook. For example, in Facebook, each person is represented 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 $V \times V$ where V is the number of vertices in a graph. Let the 2D array be $\text{adj}[][]$, a slot $\text{adj}[i][j] = 1$ 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 $\text{adj}[i][j] = w$, then there is an edge from vertex i to vertex j with weight w .



0	1	2	3	4
0				
1				
2				
3				
4				

in

- **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.

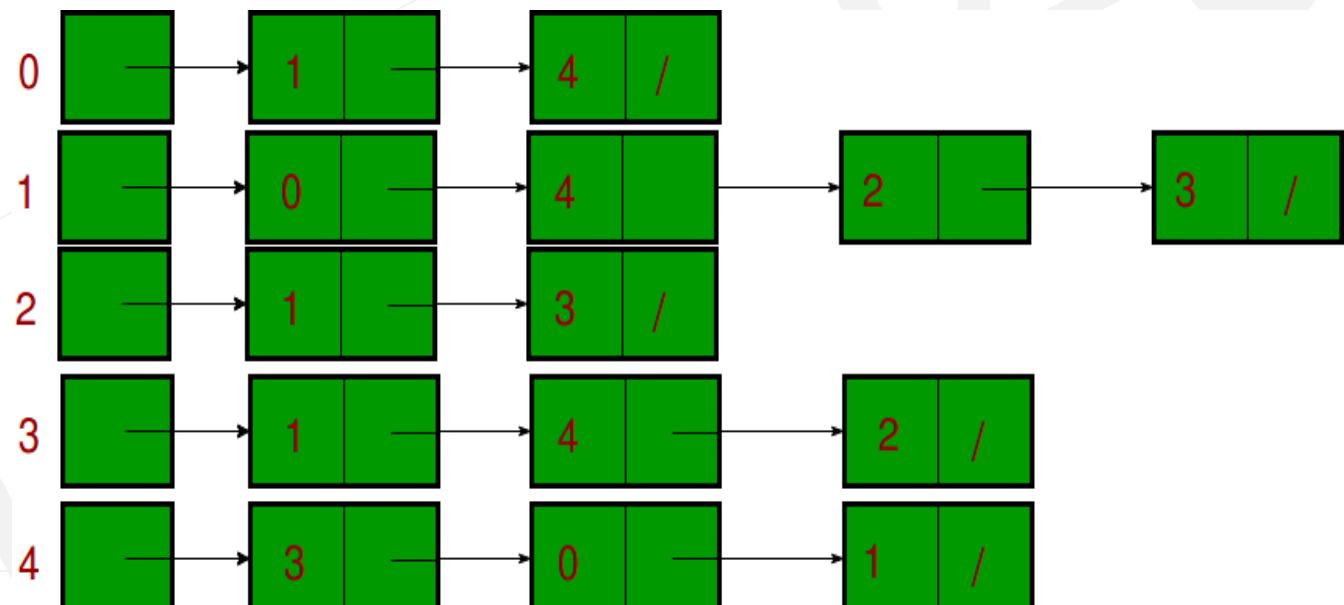
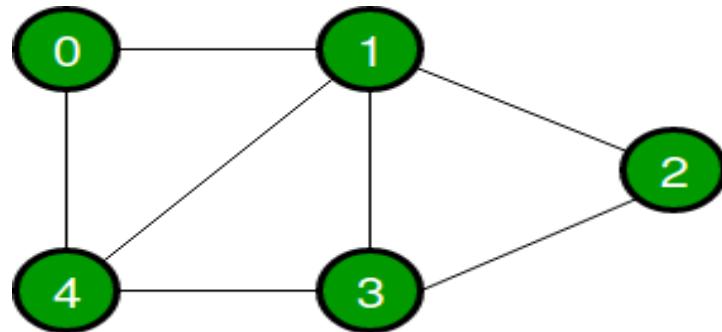
Q Which of the properties hold for the adjacency matrix A of a simple undirected unweighted graph having n vertices? **(GATE 2022) (2 MARKS)**

- (A)** The diagonal entries of A^2 are the degrees of the vertices of the graph.
- (B)** If the graph is connected, then none of the entries of $A^{n-1} + I_n$ can be zero.
- (C)** If the sum of all the elements of A is at most $2(n-1)$, then the graph must be acyclic.
- (D)** If there is at least a 1 in each of A's rows and columns, then the graph must be connected.

Q Consider a simple undirected unweighted graph with at least three vertices. If A is the adjacency matrix of the graph, then the number of 3-cycles in the graph is given by the trace of **(GATE 2022) (2 MARKS)**

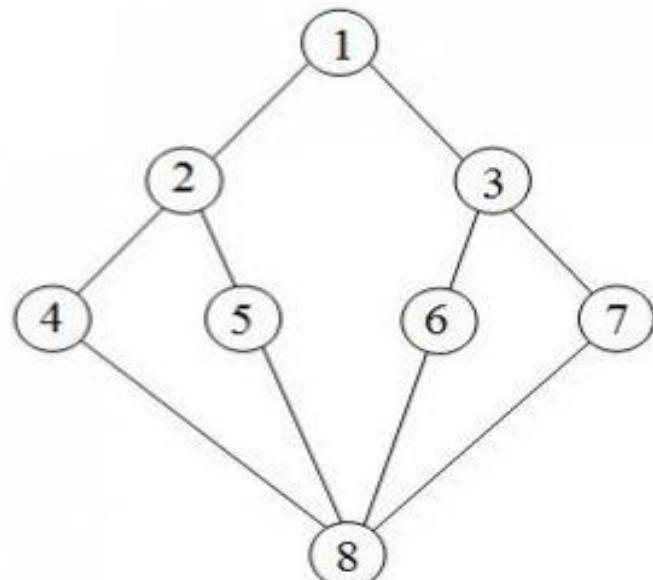
- (a) A^3
- (b) A^3 divided by 2
- (c) A^3 divided by 3
- (d) A^3 divided by 6

- **Adjacency List:** An array of lists is used. Size of the array is equal to the number of vertices. Let the array be $\text{array}[]$. An entry $\text{array}[i]$ 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.



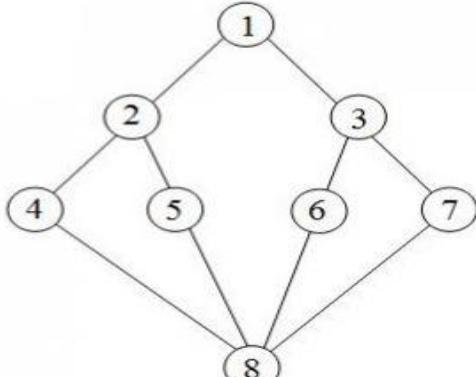
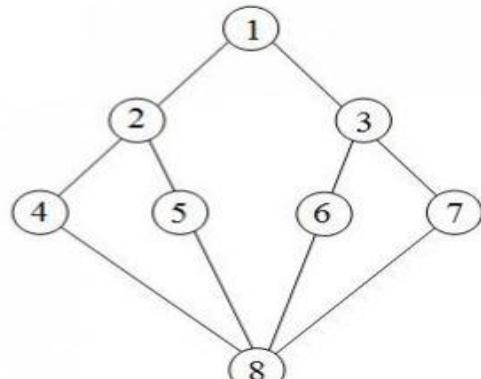
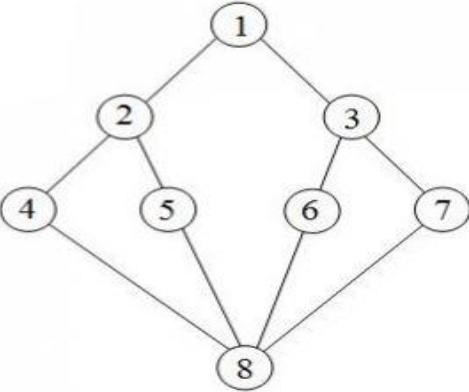
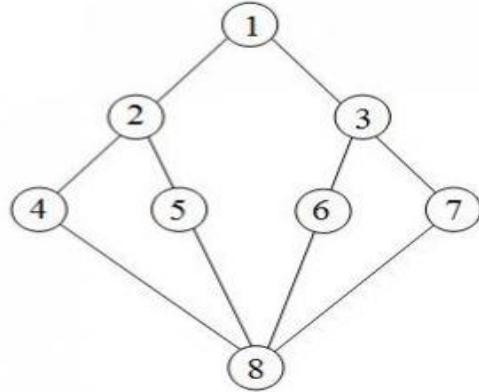
Q Which of the following are valid and invalid DFS traversal sequence

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



- 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(v)

{

 visited(v) = 1

 For all x adjacent to v

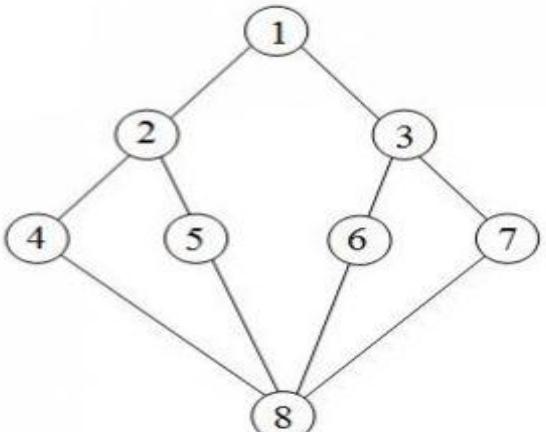
{

 if (x is not visited)

 DFS(x)

}

}



theoretical computer science, DFS is typically used to traverse an entire graph, and takes time $O(|V|+|E|)$, where $|V|$ is the number of vertices and $|E|$ the number of edges.

DFS(G)

```

1  for each vertex  $u \in G.V$ 
2       $u.color = \text{WHITE}$ 
3       $u.\pi = \text{NIL}$ 
4       $time = 0$ 
5  for each vertex  $u \in G.V$ 
6      if  $u.color == \text{WHITE}$ 
7          DFS-VISIT( $G, u$ )

```

DFS-VISIT(G, u)

```

1   $time = time + 1$                                 // white vertex  $u$  has just been discovered
2   $u.d = time$ 
3   $u.color = \text{GRAY}$ 
4  for each  $v \in G.Adj[u]$                       // explore edge  $(u, v)$ 
5      if  $v.color == \text{WHITE}$ 
6           $v.\pi = u$ 
7          DFS-VISIT( $G, v$ )
8   $u.color = \text{BLACK}$                             // blacken  $u$ ; it is finished
9   $time = time + 1$ 
10  $u.f = time$ 

```

What is the running time of DFS? The loops on lines 1–3 and lines 5–7 of DFS take time $\Theta(V)$, exclusive of the time to execute the calls to DFS-VISIT. As we did for breadth-first search, we use aggregate analysis. The procedure DFS-VISIT is called exactly once for each vertex $v \in V$, since the vertex u on which DFS-VISIT is invoked must be white and the first thing DFS-VISIT does is paint vertex u gray. During an execution of DFS-VISIT(G, v), the loop on lines 4–7 executes $|Adj[v]|$ times. Since

$$\sum_{v \in V} |Adj[v]| = \Theta(E) ,$$

the total cost of executing lines 4–7 of DFS-VISIT is $\Theta(E)$. The running time of DFS is therefore $\Theta(V + E)$.

DFS-iterative (G, s)

{

 let S be stack

 Push(s)

 while (S is not empty)

 {

 v = pop(S)

 if v is not marked as visited

 {

 mark v as visited

 for all neighbors w of v in Graph G:

 {

 if w is not marked as visited:

 push(w)

 }

 }

 }

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

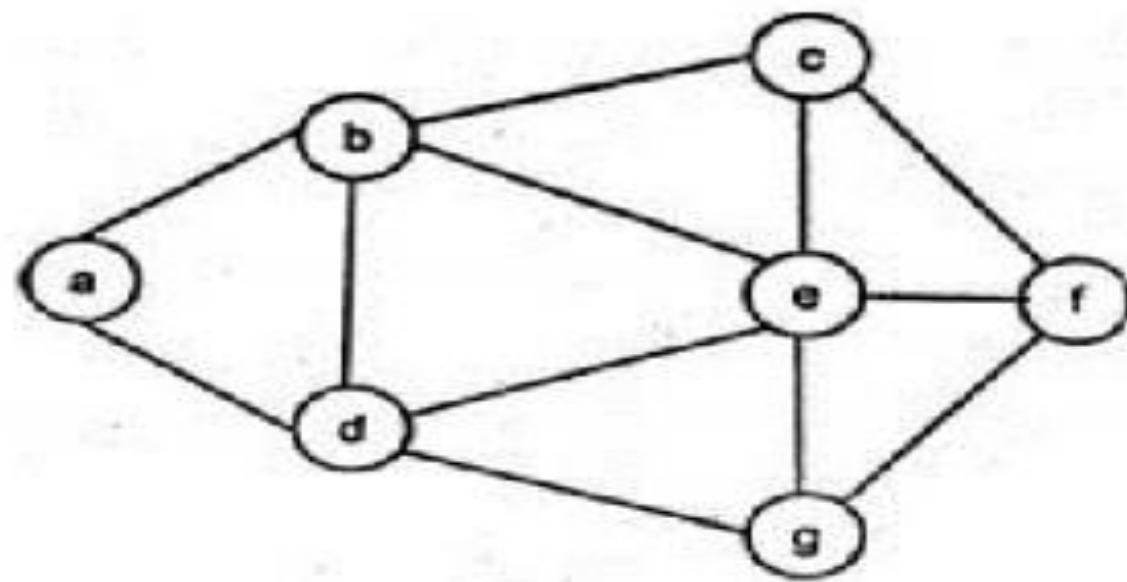
1) a b e f d g c

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

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

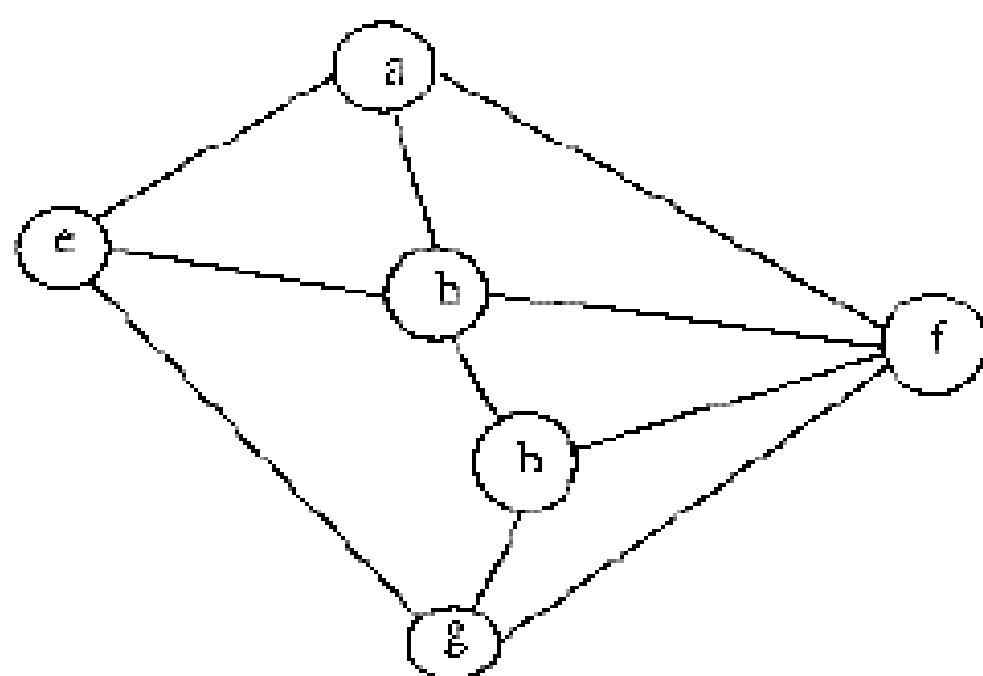
Which are depth first traversals of the above graph? (GATE-2003) (1 Marks)

(A) I, II and IV only

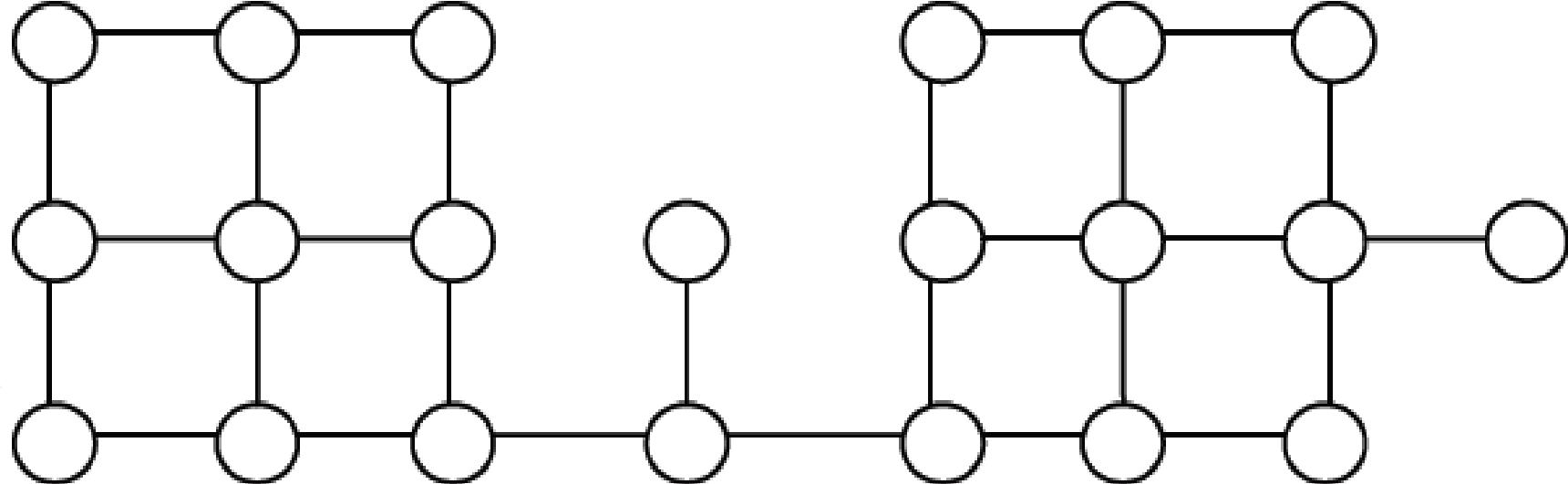
(B) I and IV only

(C) II, III and IV only

(D) I, III and IV only



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)**



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) $O(mn)$

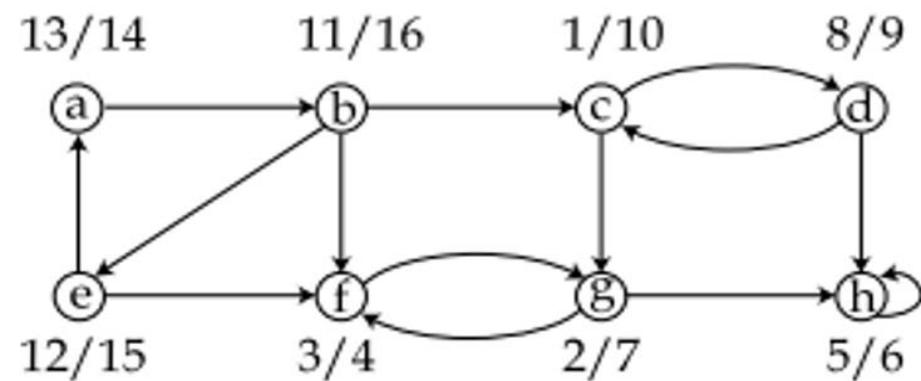
Q Let T be a depth first search tree in an undirected graph G. Vertices u and v are leaves of this tree T. The degrees of both u and v in G are at least 2. which one of the following statements is true? **(Gate-2006) (2 Marks)**

- (A) There must exist a vertex w adjacent to both u and v in G
- (B) There must exist a vertex w whose removal disconnects u and v in G
- (C) There must exist a cycle in G containing u and v
- (D) There must exist a cycle in G containing u and all its neighbors in G.

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

In the following graph, discovery time stamps and finishing time stamps of Depth First Search (DFS) are shown as x/y , where x is discovery time stamp and y is finishing time stamp. (UGC - June – 2017)



It shows which of the following depth first forest ?

- (1) {a, b, e} {c, d, f, g, h}
- (2) {a, b, e} {c, d, h} {f, g}
- (3) {a, b, e} {f, g} {c, d} {h}
- (4) {a, b, c, d} {e, f, g} {h}

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

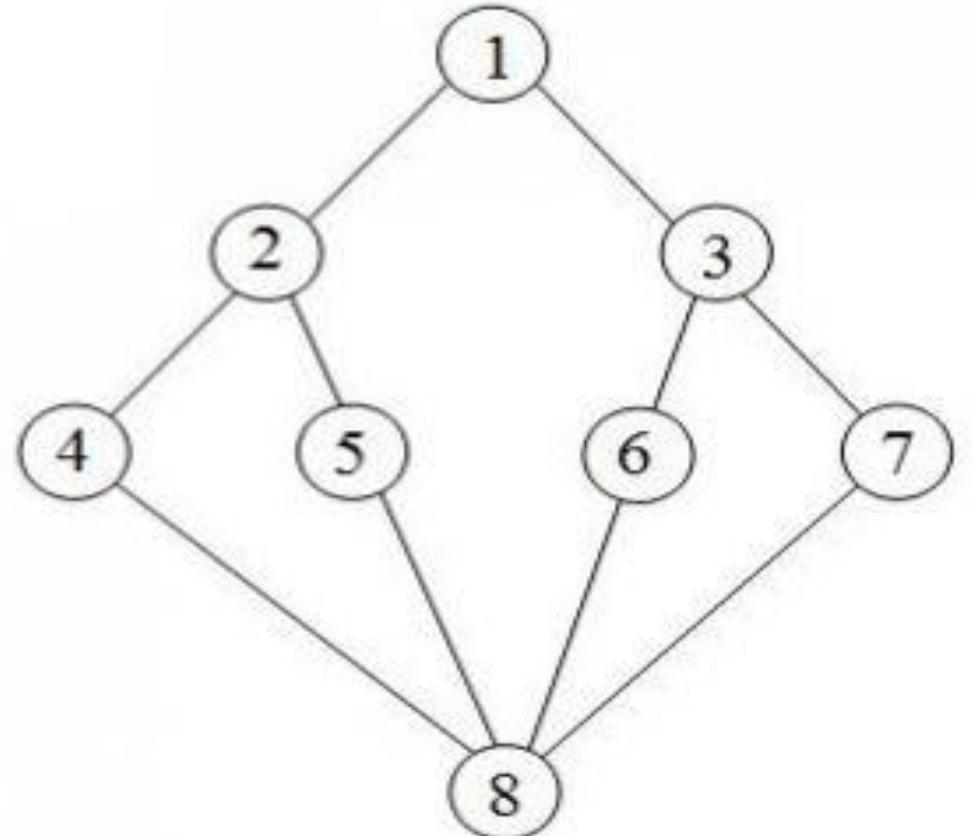
Q Which of the following are valid and invalid BFS traversal sequence

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

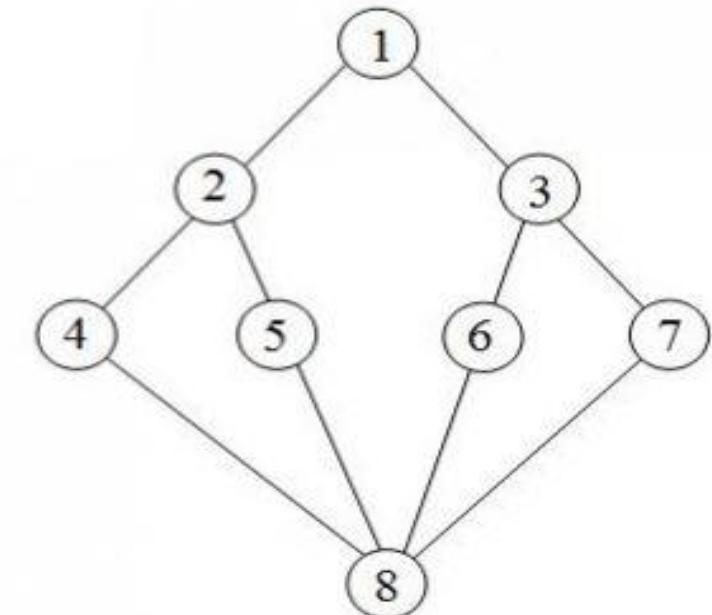
b) 1, 3, 2, 7, 6, 4, 5, 8

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

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



```
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)
            }
        }
    }
}
```



The time complexity can be expressed as $O(|V|+|E|)$, since every vertex and every edge will be explored in the worst case. $|V|$ is the number of vertices and $|E|$ is the number of edges in the graph. Note that $O(|E|)$ may vary between $O(1)$ and $O(|V|^2)$.

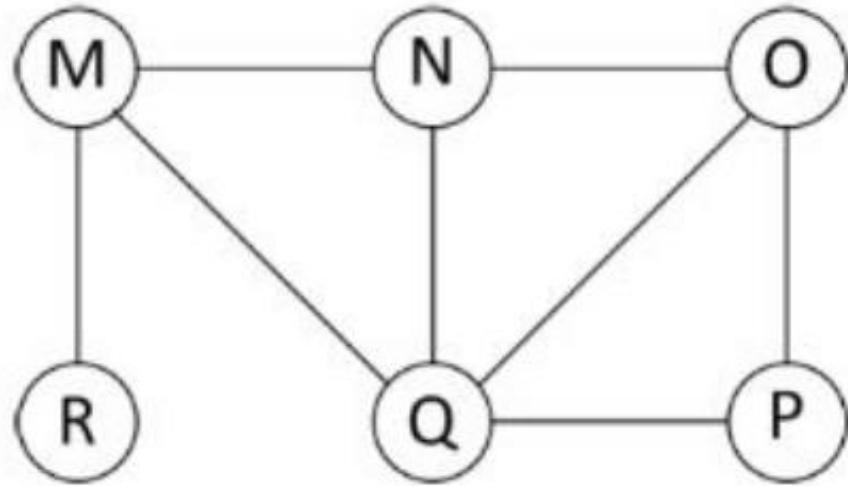
BFS(G, s)

```
1  for each vertex  $u \in G.V - \{s\}$ 
2       $u.color = \text{WHITE}$ 
3       $u.d = \infty$ 
4       $u.\pi = \text{NIL}$ 
5   $s.color = \text{GRAY}$ 
6   $s.d = 0$ 
7   $s.\pi = \text{NIL}$ 
8   $Q = \emptyset$ 
9  ENQUEUE( $Q, s$ )
10 while  $Q \neq \emptyset$ 
11      $u = \text{DEQUEUE}(Q)$ 
12     for each  $v \in G.Adj[u]$ 
13         if  $v.color == \text{WHITE}$ 
14              $v.color = \text{GRAY}$ 
15              $v.d = u.d + 1$ 
16              $v.\pi = u$ 
17             ENQUEUE( $Q, v$ )
18      $u.color = \text{BLACK}$ 
```

Before proving the various properties of breadth-first search, we take on the somewhat easier job of analyzing its running time on an input graph $G = (V, E)$. We use aggregate analysis, as we saw in Section 17.1. After initialization, breadth-first search never whitens a vertex, and thus the test in line 13 ensures that each vertex is enqueueued at most once, and hence dequeued at most once. The operations of enqueueuing and dequeuing take $O(1)$ time, and so the total time devoted to queue operations is $O(V)$. Because the procedure scans the adjacency list of each vertex only when the vertex is dequeued, it scans each adjacency list at most once. Since the sum of the lengths of all the adjacency lists is $\Theta(E)$, the total time spent in scanning adjacency lists is $O(E)$. The overhead for initialization is $O(V)$, and thus the total running time of the BFS procedure is $O(V + E)$. Thus, breadth-first search runs in time linear in the size of the adjacency-list representation of G .

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



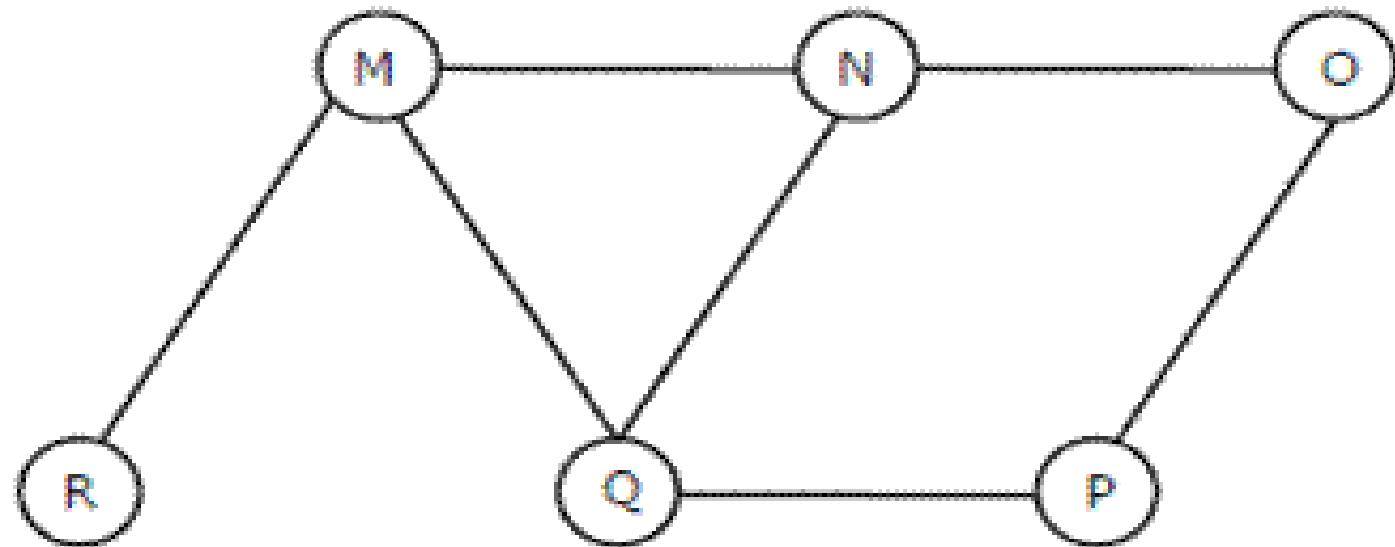
Q The Breadth First Search algorithm has been implemented using the queue data structure. One possible order of visiting the nodes of the following graph is **(Gate-2008) (1 Marks)**

(A) MNOPQR

(B) NQMPOR

(C) QMNPRO

(D) QMNPOR



Q An articulation point in a connected graph is a vertex such that removing the vertex and its incident edges disconnects the graph into two or more connected components. Let T be a DFS tree obtained by doing DFS in a connected undirected graph G. Which of the following options is/are correct? **(GATE 2021)**

(2 MARKS)

- (a)** Root of T can never be an articulation point in G.
- (b)** Root of T is an articulation point in G if and only if it has 2 or more children.
- (c)** A leaf of T can be an articulation point in G.
- (d)** If u is an articulation point in G such that x is an ancestor of u in T and y is a descendent of u in T, then all paths from x to y in G must pass through u.

Q Consider a complete binary tree with 7 nodes. Let A denote the set of first 3 elements obtained by performing Breadth-First Search (BFS) starting from the root. Let B denote the set of first 3 elements obtained by performing Depth-First Search (DFS) starting from the root. The value of $|A - B|$ is _____ . (Gate-2021)
(2 Marks) [Asked in Accenture 2022]

- a) 1
- b) 2
- c) 3
- d) 4

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)

Q Let $G = (V, E)$ be a simple undirected graph, and s be a particular vertex in it called the source. For $x \in V$, 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)**

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

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
- (C)** depth first search
- (B)** inorder traversal
- (D)** breadth first search

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. If 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) $d(r, u) \leq d(r, v)$ (D) None of the above

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) O(n)
- b) O(m)**
- c) O(m+n)
- d) O(m.n)

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) $\Theta(n^4)$

Introduction to hashing

- The main objective of a data structure is to efficiently store data, but the **most frequent operation** on any data structure is **search**. Even for insertion and deletion, searching is required as a preliminary step.
- **Search time** in a data structure primarily depends on the **number of elements** and the **structure type**:
 - **Unsorted Array:** $O(n)$
 - **Sorted Array:** $O(\log n)$
 - **Linked List:** $O(n)$
 - **Binary Tree (BT):** $O(n)$
 - **Binary Search Tree (BST):** $O(n)$ in worst case
 - **AVL Tree:** $O(\log n)$
- Hashing optimizes search operations by providing **constant time complexity** for lookups in the **average case** — making it a powerful tool for managing and accessing data.

- Hashing is a technique where the **search time is independent of the number of items**. The main idea is to use a **key value** (like phone numbers, Aadhar card, roll numbers, etc.) to directly find the address in memory, making the search process fast and efficient.
- **Hash Function:** Converts a large key into a smaller number that serves as an index in the hash table. This transformation is represented as: $H: K \rightarrow L$ where K is the set of keys and L is the set of memory locations.
- **Hash Table:**
 - A special data structure where the keys are used to calculate the index, storing values at these indexes. Unused entries are set to nil or empty pointers.

Q Given the following input (4322, 1334, 1471, 9679, 1989, 6171, 6173, 4199) and the hash function $x \bmod 10$, which of the following statements are true? (Gate-2004) (1 Marks)

- i. 9679, 1989, 4199 hash to the same value
- ii. 1471, 6171 has to the same value
- iii. All elements hash to the same value
- iv. Each element hashes to a different value

(A) i only (B) ii only (C) i and ii only (D) iii or iv

- **Collision**: - It is possible that two different set of keys K_1 and K_2 will yield the same hash address. This situation is called collision. The technique to resolve collision is called collision resolution.
- **Note**: Irrespective of how good a hash function is, collisions are bound to occur. Therefore, to maintain the performance of a hash table, it is important to manage collisions through various collision resolution techniques.
- **Characteristics of good hash function**
 - Easy to compute and understand
 - Efficiently computable- It must take less time to compute
 - Should uniformly distribute the keys (Each table position equally likely for each key) and should not result in clustering.
 - Must have low collision rate

Most popular hash function

- **Division-remainder method**: The size of the number of items in the table is estimated. That number is then used as a divisor into each original value or key to extract a quotient and a remainder.
- The remainder is the hashed value. (Since this method is liable to produce a number of collisions, any search mechanism would have to be able to recognize a collision and offer an alternate search mechanism.)
 - $H(K) = K \bmod m$
 - $H(K) = K \bmod m + 1$

Q Which of the following statement(s) is/are TRUE? (Gate-2006) (1 Marks)

Collision resolution technique

- **Open Addressing/closed hashing** - In Open Addressing, all elements are stored in the hash table itself. i.e. collision is resolved by **probing** or searching through alternate locations in the Hash table itself in a particular *sequence*.
- When searching for an element, we one by one examine table slots until the desired element is found or it is clear that the element is not in the table. So, at any point, size of table must be greater than or equal to total number of keys.
- It is of three types linear probing, quadratic probing, double hashing

Linear probing

- **Linear Probing** is a collision resolution technique in which the hash table is searched **sequentially** starting from the position given by the hash function until a matching key or an empty slot is found. Using linear probing, insert, remove and search operations can be implemented in $O(1)$, as long as the load factor of the hash table is a constant strictly less than one.
 - Insert(k): Keep probing until an empty slot is found. Once an empty slot is found, insert k.
 - Search(k): Keep probing until slot's key doesn't become equal to k or an empty slot is reached.
 - Delete(k): *Delete operation is interesting.* If we simply delete a key, then search may fail. So slots of deleted keys are marked specially as "deleted". Insert can insert an item in a deleted slot, but search doesn't stop at a deleted slot.
- In **Linear Probing**, the position for a key k is calculated using the following hash function:
- $h(k,i) = (h'(k)+i) \bmod m$ Where:
- $h'(k)$ is the **primary hash function** applied to the key k.
- i is the **probe number**, ranging from 0 to $m-1$.
- m is the **size of the hash table**.

Q A hash table contains 10 buckets and uses linear probing to resolve collision. The key values are integers and the hash function used is $\text{key}\%10$, if the values 43 165 62 123 142 are inserted in the table, in what location would the key value 142 be inserted? **(Gate-2005) (1 Marks)**

- A) 2
- b) 3
- c) 4
- d) 6

Q The keys 12, 18, 13, 2, 3, 23, 5 and 15 are inserted into an initially empty hash table of length 10 using open addressing with hash function $h(k) = k \bmod 10$ and linear probing. What is the resultant hash table? **(Gate-2009) (2 Marks)**

0	
1	
2	2
3	23
4	
5	15
6	
7	
8	18
9	

(A)

0	
1	
2	12
3	13
4	
5	5
6	
7	
8	18
9	

(B)

0	
1	
2	12
3	13
4	2
5	3
6	23
7	5
8	18
9	15

(C)

0	
1	
2	12, 2
3	13, 3, 23
4	
5	5, 15
6	
7	
8	18
9	

(D)

Q consider a hash table of size 11 that uses open addressing with linear probing. Let $h(k) = k \bmod 11$ be the hash function used. A sequence of records with keys 43 36 92 87 11 4 71 13 14 is inserted into an initially empty hash table, indexing is from 0, what is the index of the bin into which the last record is inserted? **(Gate-2008) (2 Marks)**

- a) 3
- b) 4
- c) 6
- d) 7

Q Consider a hash table of size seven, with starting index zero, and a hash function $(3x + 4) \bmod 7$. Assuming the hash table is initially empty, which of the following is the contents of the table when the sequence 1, 3, 8, 10 is inserted into the table using closed hashing? Note that ‘_’ denotes an empty location in the table. **(Gate-2007) (2 Marks)**

- (A) 8, _, _, _, _, _, 10
(C) 1, _, _, _, _, _, 3

- (B) 1, 8, 10, _, _, _, 3
(D) 1, 10, 8, _, _, _, 3

Q A hash table of length 10 uses open addressing with hash function $h(k)=k \bmod 10$, and linear probing. After inserting 6 values into an empty hash table, the table is as shown below. Which one of the following choices gives a possible order in which the key values could have been inserted in the table?

(Gate-2010) (2 Marks)

- (A)** 46, 42, 34, 52, 23, 33
- (B)** 34, 42, 23, 52, 33, 46
- (C)** 46, 34, 42, 23, 52, 33
- (D)** 42, 46, 33, 23, 34, 52

0	
1	
2	42
3	23
4	34
5	52
6	46
7	33
8	
9	

Q A hash table with ten buckets with one slot per bucket is shown in the following figure. The symbols S_1 to S_7 , initially entered using a hashing function with linear probing. The maximum number of comparisons needed in searching an item that is not present is
(Gate-1989) (2 Marks)

0	S7
1	S1
2	
3	S4
4	S2
5	
6	S5
7	
8	S6
9	S3

- **Advantages:**
 - **Fast and Simple:** Easy to implement and has good locality of reference.
 - **Efficient on Standard Hardware:** Performs well due to sequential memory access.
- **Disadvantages:**
 - **Primary Clustering Issue:** Collisions can lead to long probe sequences.
 - **Sensitive to Hash Function:** Requires a high-quality hash function to avoid performance degradation.

Q Which among the following statement(s) is(are) true?

- a)** A hash function takes a message of arbitrary length and generates a fixed length code
- b)** A hash function takes a message of fixed length and generates a code of variable length
- c)** A hash function may give same hash value for distinct messages

Choose the correct answer from the options given below: **(Gate-2005) (1 Marks) (NET 2020 OCT)**

- (A)** (a) only
- (B)** (b) and (c) only
- (C)** (a) and (c) only
- (D)** (b) only

- **Primary Clustering:**
 - Occurs in **linear probing** when collisions cause keys to form a **contiguous cluster**.
 - Each new key hashed into the cluster makes it grow, leading to longer probe sequences.
- **Secondary Clustering:**
 - Happens when multiple keys hash to the same location and follow the same **probe sequence**.
 - Affects both linear and quadratic probing, caused by **poor hash functions**, leading to **slow access times**.

Quadratic probing

- **Definition:** Quadratic probing resolves collisions by **adding successive values** of a quadratic polynomial to the original hash index until an empty slot is found.
- **Hash Function:**
 - $h(k,i) = (h'(k) + f(i^2)) \text{mod } m$
- **$h'(k)$:** Primary hash function.
- **i:** Probe number (0, 1, 2, ...).
- **m:** Size of the hash table.

- **Advantage**
 - Quadratic probing can be a more efficient algorithm in a closed hashing table, since it better avoids the clustering problem that can occur with linear probing, although it is not immune.
 - It also provides good memory caching because it preserves some locality of reference; however, linear probing has greater locality and, thus, better cache performance.
- **Disadvantage**
 - Quadratic probing lies between the two in terms of cache performance and clustering.

- **Performance of Open Addressing**: Like Chaining, performance of hashing can be evaluated under the assumption that each key is equally likely to be hashed to any slot of table (simple uniform hashing)
 - m = Number of slots in hash table
 - n = Number of keys to be inserted in hash table
 - Load factor $\alpha = n/m (< 1)$
 - Expected time to search/insert/delete $< 1/(1 - \alpha)$
 - So Search, Insert and Delete take $(1/(1 - \alpha))$ time

Q Given a hash table T with 25 slots that stores 2000 elements, the load factor α for T is _____ (Gate-2015) (1 Marks)

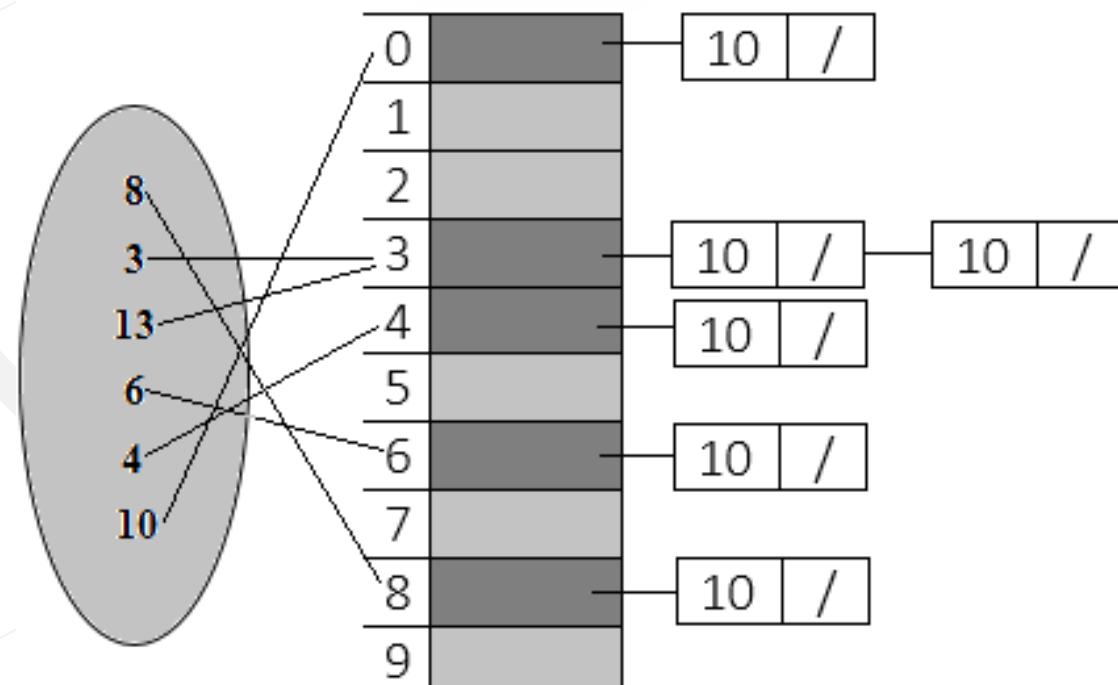
- (A) 80
- (B) 0.0125
- (C) 8000
- (D) 1.25

Q Consider a hash function that distributes keys uniformly. The hash table size is 20. After hashing of how many keys will the probability that any new key hashed collides with an existing one exceed 0.5. **(Gate-2007) (2 Marks)**

- (A) 5
- (B) 6
- (C) 7
- (D) 10

Chaining

- The idea is to make each cell of hash table point to a linked list of records that have same hash function value. In *chaining*, we place all the elements that hash to the same slot into the same linked list.
- **Advantage:** - Chaining is simple
- **Disadvantage:** -but requires additional memory outside the table.



S.N O.	Separate Chaining	Open Addressing
1.	Chaining is Simpler to implement.	Open Addressing requires more computation.
2.	In chaining, Hash table never fills up, we can always add more elements to chain.	In open addressing, table may become full.
3.	Chaining is Less sensitive to the hash function or load factors.	Open addressing requires extra care for to avoid clustering and load factor.
4.	Chaining is mostly used when it is unknown how many and how frequently keys may be inserted or deleted.	Open addressing is used when the frequency and number of keys is known.
5.	Cache performance of chaining is not good as keys are stored using linked list.	Open addressing provides better cache performance as everything is stored in the same table.
6.	Wastage of Space (Some Parts of hash table in chaining are never used).	In Open addressing, a slot can be used even if an input doesn't map to it.
7.	Chaining uses extra space for links.	No links in Open addressing

Q An advantage of chained hash table (external hashing) over the open addressing scheme is **(Gate-1996) (1 Marks)**

- (A)** Worst case complexity of search operations is less
- (B)** Space used is less
- (C)** Deletion is easier
- (D)** None of the above

Q Consider a hash table with 9 slots. The hash function is $h(k) = k \bmod 9$. The collisions are resolved by chaining. The following 9 keys are inserted in the order: 5, 28, 19, 15, 20, 33, 12, 17, 10. The maximum, minimum, and average chain lengths in the hash table, respectively, are **(Gate-2014) (2-Marks)** [Asked in Accenture]

(A) 3, 0, and 1
(C) 4, 0, and 1

(B) 3, 3, and 3
(D) 3, 0, and 2

Q Consider a hash table with 100 slots. Collisions are resolved using chaining. Assuming simple uniform hashing, what is the probability that the first 3 slots are unfilled after the first 3 insertions? **(Gate-2014) (2 Marks)**

- (A) $(97 \times 97 \times 97)/100^3$
- (B) $(99 \times 98 \times 97)/100^3$
- (C) $(97 \times 96 \times 95)/100^3$
- (D) $(97 \times 96 \times 95)/(3! \times 100^3)$

Double Hashing

- **Definition:** Double hashing is a collision resolution technique that uses a **second hash function** to determine the interval for probing, ensuring that consecutive elements use different probe sequences.
- **Hash Function:**
 - $h(i,k) = (h_1(k) + i \cdot h_2(k)) \bmod |T|$
 - $h_1(k)$ and $h_2(k)$ are **two independent hash functions**.
 - i is the **probe number**.
 - $|T|$ is the **size of the hash table**.
- **Key Advantages:**
 - Reduces **clustering** more effectively than linear or quadratic probing.
 - Provides near-random distribution, minimizing collisions and improving search efficiency.

Q Consider a double hashing scheme in which the primary hash function is $h_1(k) = k \bmod 23$, and the secondary hash function is $h_2(k) = 1 + (k \bmod 19)$. Assume that the table size is 23. Then the address returned by probe 1 in the probe sequence (assume that the probe sequence begins at probe 0) for key value $k = 90$ is _____.? **(Gate-2020) (2-Marks)**

Q Suppose we are given n keys, m hash table slots, and two simple uniform hash functions h_1 and h_2 . Further suppose our hashing scheme uses h_1 for the odd keys and h_2 for the even keys. What is the expected number of keys in a slot? **(GATE 2022) (1 MARKS)**

- (A) m/n
- (B) n/m
- (C) $2n/m$
- (D) $n/2m$

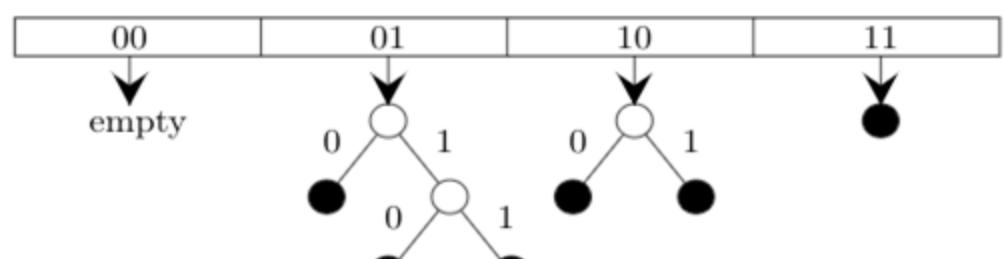
Q Consider a dynamic hashing approach for 4-bit integer keys:

- (A)** There is a main hash table of size 4.
- (B)** The 2 least significant bits of a key is used to index into the main hash
- (C)** Initially, the main hash table entries are empty.
- (D)** Thereafter, when more keys are hashed into it, to resolve collisions, the set of all keys corresponding to a main hash table entry is organized as a binary tree that grows on demand.
- (E)** First, the 3rd least significant bit is used to divide the keys into left and right subtrees.
- (F)** To resolve more collisions, each node of the binary tree is further sub-divided into left and right subtrees based on the 4th least significant bit.

(G) A split is done only if it is needed, i.e., only when there is a collision.

Consider the following state of the hash table. Which of the following sequences of key insertions can cause the above state of the hash table (assume the keys are in decimal notation)? **(GATE 2021) (2 MARKS)**

- (A)** 5,9,4,13,10,7
- (B)** 9,5,10,6,7,1
- (C)** 10,9,6,7,5,13
- (D)** 9,5,13,6,10,14



Q Which one of the following hash functions on integers will distribute keys most uniformly over 10 buckets numbered 0 to 9 for i ranging from 0 to 2020? **(Gate-2015) (2-Marks)**

- (A) $h(i) = i^2 \bmod 10$
- (C) $h(i) = (11 * i^2) \bmod 10$

- (B) $h(i) = i^3 \bmod 10$
- (D) $h(i) = (12 * i) \bmod 10$