Ans-(1) **Bobble sort**

```
n=len(arr)
for (i=0; i<n-1; i++) {
   flag= 0;
  for(j=0; j<n-1; j++) {
     if(arr[j] > arr[j+1])
     { swap(arr[j], arr[j+1]);
         flag=1; }
  if (flag=0)
       break;
```

=> arr[] = {25, 16, 23, 64, 31, 86, 28, 88}
n = len(arr) = 8

=> i=0 =>

j=0 => | 16 | 25 | 23 | 64 | 31 | 86 | 28 | 88 |

j=1=> | 16 | 23 | 25 | 64 | 31 | 86 | 28 | 88 |

j=2 => | 16 | 23 | 25 | 64 | 31 | 86 | 28 | 88 |

j=3=> | 16 | 23 | 25 | 31 | 64 | 86 | 28 | 88 |

j=4 => | 16 | 23 | 25 | 31 | 64 | 86 | 28 | 88 |

j=5 => | 16 | 23 | 25 | 31 | 64 | 28 | 86 | 88 |

j=6 => | 16 | 23 | 25 | 31 | 64 | 28 | 86 | 88 |

19DCS098

i=1;

Here, for j=0,1,2,3 array will not swap elements as the elements till the iteration are sorted.

j=4

16	23	25	31	28	64	86	88

j=5 }
j=6 } ->

16	23	25	31	28	64	86	88

for, j=0,1,2 array will be unchanged

j=3 ->

16	23	25	28	31	64	86	88

For i=3; flag=0; so, loop will be terminated

* Insertion Sort:-

```
   n= len (arr)
 for( int i= 1; i<len; i++ ) {
     key = arr[i];
       j = i-1;
 while (j>=0 && arr[j] > key )
 { arr[j+1]= arr[j];
         j--;
   }
    arr[j+1]= key;
  }
```

arr[]= { 25, 16, 23, 64, 31, 86, 28, 88 }

i=1; key = 16;

j=0 ⇒ |16|25|23|64|31|86|28|88|

i=2; key = 23

j=1; ⇒ |16|23|25|64|31|86|28|88|

j=0 ⇒ loop terminates;

i=3; key=64
⇒ loop terminated

i=4 ; key= 31
j⇒3 |16|23|25|64|64|86|28|88|

j= 2; loop terminated.

arr = |16|23|25|31|64|86|28|88|

i=5=> key=86

arr[4]< key; loop terminated

i=6; key= 28; j=5

j=5=> |16|23|25|31|64|86|86|88|

j=4=> |16|23|25|31|64|64|86|88|

j=3 => |16|23|25|31|31|64|86|88|

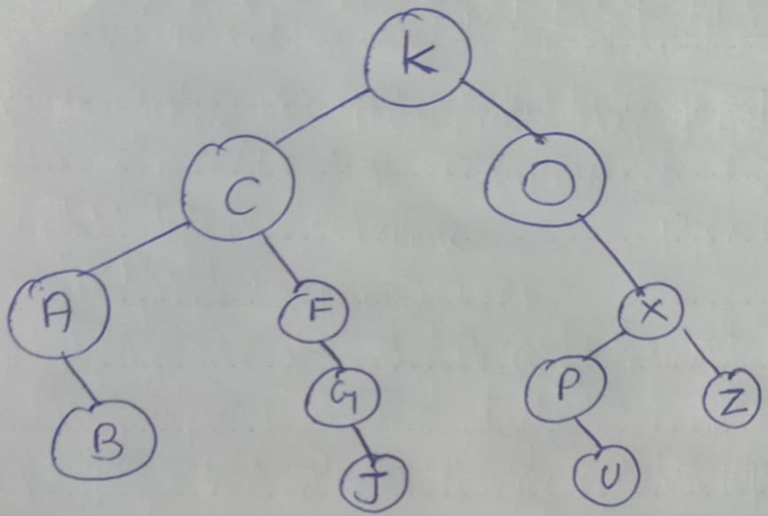
j=2 > loop terminated

arr: |16|23|25|28|31|64|86|88|

i=7=> loop terminated as array is sorted

Ans-(2)

19DCS098



PREORDER : K, C, A, B, F, G, J, O, X, P, U, Z

INORDER : A, B, C, F, G, J, K, O, P, U, X, Z

POSTORDER : B, A, J, G, F, C, U, P, Z, X, O, K

19DCS098

Page
Date

Ans-(3) A+(((B-C)*(D-E)+F)/G)^(H-J)

* Prefix:-

→ In prefix, we start from last element.

Infix	Stack	Prefix
)	)	
J	)	J
-	)-	J
H	)-	JH
(		JH-
^	^	JH-
)	^)	JH-
G	^)	JH-G
/	^)/	JH-G
)	^)/)	JH-G
F	^)/)	JH-GF
+	^)/)+	JH-GF
)	^)/)+)	JH-GF
E	^)/)+)	JH-GFE
-	^)/)+)-	JH-GFE
D	^)/)+)-	JH-GFED
(	^)/)+	JH-GFED-
*	^)/)+*	JH-GFED-
)	^)/)+*)	JH-GFED-
C	^)/)+*)	JH-GFED-C
-	^)/)+*)-	JH-GFED-C
B	^)/)+*)-	JH-GFED-CB
(	^)/)+*	JH-GFED-CB-
(	^)/)	JH-GFED-CB-*+
(+	^	JH-GFED-CB-*+/
A +	^ +	JH-GFED-CB-*+/

Final Answer:-

$\boxed{+A^\wedge/+*-BC-DEFG-HJ}$

* **Postfix:-** (Extensively used)

Infix	Stack	Postfix
		A
A		
+	+	A
(	+(	A
(	+((	A
(	+(((	A
B	+(((	AB
-	+(((-	AB
C	+(((-	ABC
)	+((	ABC-
*	+((*	ABC-
(	+((*(	ABC-
D	+((*(	ABC-D
-	+((*(-	ABC-D
E	+((*(-	ABC-DE
)	+((*	ABC-DE-
+	+((+	ABC-DE-*
F	+((+	ABC-DE-*F
)	+(	ABC-DE-*F+
/	+(/	ABC-DE-*F+
G	+(/	ABC-DE-*F+G
)	+	ABC-DE-*F+G/
^	+^	ABC-DE-*F+G/
(	+^(	ABC-DE-*F+G/
H	+^(	ABC-DE-*F+G/H
-	+^(-	ABC-DE-*F+G/H
J	+^(-	ABC-DE-*F+G/HJ
)	+^	ABC-DE-*F+G/HJ-

→ $\boxed{ABC-DE-*F+G/HJ-^\wedge+}$

Page
Date

Ans-(4)

* Stack *

→ Stack is a linear data structure that follows Last In Firstout (LIFO) principle.

→ Stack is only one hand for inserting and deleting data.

→ Mayor operations are:-

(1) push() - to insert new data

(2) pop() - to delete an element

(3) peek() - Returns element at given position

(4) count() - Gives total number of elements

(5) display() - prints/display all elements of stack

(6) isEmpty() - Determines if stack is empty or not

(7) isFull() - Determine if stack is full or not

* Applications:-

(1) Parenthesis counting

(2) Infix to prefix

(3) Infix to postfix

(4) String Reversal

(5) UNDO/REDO

(6) Recursion

Page
Date

* Circular Queue*

→ Circular Queue is similar to linear Queue and based on First In FirstOut (FIFO) principle except the last position is connected to the first position

* Operations:-

(1) Front() - used to get front element
(2) Rear() - used to get Rear element
(3) en Queue( ) - used to insert value (from rear end)
(4) de Queue() - used to delete value (from front end)

* Applications:-

(1) memory management
(2) CPU scheduling
(3) Traffic System

* singly linked list:-

→ Singly linked list is defined as the collection of ordered set of elements

→ It is linear Data Structure.

→ It has two parts: (1) Data part (2) Address part

→ Data part stores actual information

→ Address part contains address of next adjacent node.

→ In Singly linked list, we can traverse in only one direction

* Operations:-

(1) Traverse() → To traverse in the list
(2) Insert() → To insert new element
(3) Detete() → To delete A element
(4) Search() → To search particular element
(5) Update() → To update a node

* Applications:-

(1) Implementation of Stack & Queues
(2) Implementation of Graph
(3) Dynamic memory allocation

Ans-(5) Given 3-D array; A[-2:0, 1:4, 6:9]

=> Assuming $U_a = 0$; $U_b = 4$; $U_c = 9$
$L_a = -2$; $L_b = 1$; $L_c = 6$

=> Total number of elements =>
Total = $(U_a - L_a + 1)(U_b - L_b + 1)(U_c - L_c + 1)$
$= (0-(-2)+1)(4-1+1)(9-6+1)$
$= (2+1)(4)(3+1)$
$= (3)(4)(4)$

=> $\boxed{\text{Total elements} = 48}$

=> To find address of A[-1][3][8];
we need base address, size of each element.

=> Assuming size of each element = 4 bytes
Given base address = 1000

=> The 3-D array is arranged in column major order.

=> Formula for A[i, j, K] in column major is
A[i, j, K] = B.A + W * [(D-D₀)*R*C + (I-R₀) + (J-C₀)*R]
B.A = Base address W = Weight R = total Rows
C = total Columns D = width D_0 = lower bound of width R_0 = lower Bound of Row C_0 = Lower bound of column

=> A[-1, 3, 8] = 1000 + 4 * [(-1-(-2))(4×4) + (1-1) + (3-6)]
$= 1000 + 4*[(8-6)(4) + (2)(3) + 1]$
$= 1000 + 4 * (31) = 1000 + 124 = \boxed{1124}$