ASSIGNMENT :- 1

1 Design an algorithm to perform deletion operation by searching an item in an array A[100] having elements from lower bound LB to upper bound UB.

Ans [Consider an array A[100] having elements from lb to ub and item is given to be deleted from the array.]

Delete (A[100], lb, ub, item)

step 1 : start

step 2 : let i, flag = 0

step 3 : for (i = lb to ub), incr by 1

step 3.1 : if (A[i] = item), then

step 3.1.1 : flag = 1

step 3.1.2 : break

[end of if]

[end of for]

step 4 : if (flag = 0), then

step 4.1 : Display "Item to be deleted is not found"

step 5 : else

step 5.1 : while (i <= ub)

step 5.1.1 : A[i] = A[i+1]

step 5.1.2 : i = i+1

[end of while]

step 5.2 : ub = ub - 1

[end of if]

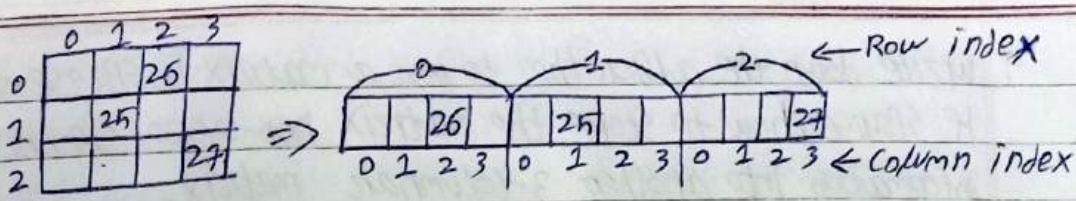
step 6 : exit

2 write an algorithm to sort a list of elements present in an array X[0] in ascending order.

Ans [Consider an array X[0] having elements from lb to ub for sorting in ascending order.]
step 1: start
step 2: let i, j, temp
step 3: for (i=lb to ub-1), incr by 1
  step 3.1: for (j = i+1 to ub), incr by 1
    step 3.1.1: if (a[i] > a[j]), then
      step 3.1.1.1: temp=a[i], a[i]=a[j], a[j]=temp
    [end of if]
  [end of for]
[end of for]
step 4: exit

3 what is row-major order and column-major order? Given a matrix a[7][8] having base address 1000. if the size of each memory is 4 bytes, and we need to find the address of a[4][3] then find the address in row-major order and also in column-major order.

Ans ① Row-major order:- when the memory is occupied row by row serially then it is called the row major order. To find the address the formula is: A[i][j] = Base + w * (n*i + j)
where, Base = the starting or base address
w = the size of each element
n = the total number of columns
i and j = the corresponding row and column address we are finding.



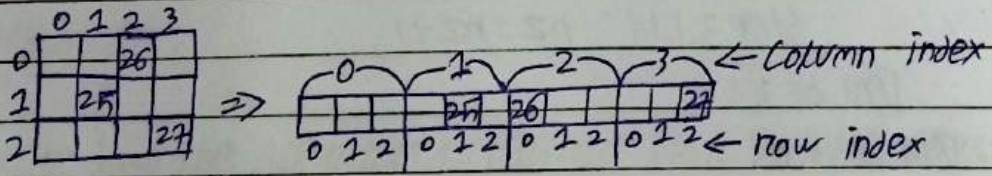
(ii) Column major order :- when the memory of a matrix occupied serially column bit column then it is called column major order. To find the address the formula is : A[i][j] = Base + w * (i + m * j)

where, Base = the starting or base address

w = the size of each element

m = the total number of rows

i and j = the corresponding row and column address we are finding.



Given, Base address = 1000, w = 4 bytes, m = 7, n = 8, i = 4, j = 3

The address in row-major order is :-

$$Q[i][j] = Base + w * (n * i + j)$$

$$Q[4][3] = 1000 + 4 * (8 * 4 + 3)$$

$$= 1000 + 4 * 35$$

$$= 1000 + 140$$

$$= 1140$$

The address in column-major order is :-

$$Q[i][j] = Base + w * (i + m * j)$$

$$Q[4][3] = 1000 + 4 * (4 + 7 * 3)$$

$$= 1000 + 4 * 25$$

$$= 1000 + 100$$

$$= 1100$$

Date
Page

4 Write down the algorithm to test a matrix is sparse or not. If sparse, how to store the matrix non-zero elements information into another 3-columnar matrix.

Ans [Consider a matrix A[m][n] having elements. Declare a 3-columnar matrix SP[nz+1][3] where nz is non-zeros count.]

Sparse (A[m][n], SP[nz+1][3])
Step 1: Start
Step 2: Let nz=0, i, j
Step 3: for (i=0 to m-1), incr by 1
  Step 3.1: for (j=0 to n-1), incr by 1
    Step 3.1.1: If (A[i][j] != 0), then
      Step 3.1.1.1: nz=nz+1
    [End of if]
  [End of for]
[End of for]
Step 4: If (nz >= (m*n)/2), then
  Step 4.1: Display "It is not sparse"
Step 5: else
  Step 5.1: Display "It is sparse"
  Step 5.2: Call Alternate (A[m][n], SP[nz+1][3])
[End of if]
Step 6: Stop

Alternate (A[m][n], SP[nz+1][3])
Step 1: Start
Step 2: Let i, j, k = 1
Step 3: SP[0][0] = m, SP[0][1] = n, SP[0][2] = nz
Step 4: for (i=0 to m-1), incr by 1
  Step 4.1: for (j=0 to n-1), incr by 1
    Step 4.1.1: if (A[i][j] ≠ 0), then
      Step 4.1.1.1: SP[k][0] = i, SP[k][1] = j, SP[k][2] = A[i][j]
      Step 4.1.1.2: k = k+1
    [end of if]
  [end of for]
[end of for]
Step n: exit

5 Write down the algorithm to perform PUSH and POP operations on a given stack using an array.

Ans Push :-
[Consider a stack[size], initially top = -1]
Push (stack[size], top, item)
Step 1: Start
Step 2: If (top = size-1), then
  Step 2.1: Display "Stack is full or overflow"
Step 3: else
  Step 3.1: top = top + 1
  Step 3.2: stack[top] = item
[end of if]
Step 4: Stop

Pop :-

[Consider a stack[size] having elements and top contains index of top most element.]

Pop (Stack[size], top)

Step 1 : Start

Step 2 : if (top = -1), then

Step 2.1 : Display "Stack is empty on underflow"

Step 3 : else

Step 3.1 : let temp = stack[top]

Step 3.2 : Display "popped value is", temp

Step 3.3 : top = top - 1

[end of if]

Step 4 : Stop

6 write down the algorithm to convert a given infix expression into equivalent postfix notation using stack.

Ans [Given an infix expression Q and we need to find equivalent postfix expression P. Declare a stack where top represents top most element. Here i is the index for Q and j is the index for P.]

InfixtoPostfix (P, j, Q, i, Stack, top)

Step 1: Start

Step 2: Push '(' to stack and Put ')' at the end of Q

Step 3: while (Stack is not empty)

Step 3.1 : Scan Symbol for Q[i]

Step 3.2: If Q[i] is operand then put the operand at the end of P[j]

Step 3.3: else if Q[i] is '(' then push '(' into the stack [top]

Step 3.4: else if Q[i] is operator ⊗, then

Step 3.4.1: while (Stack[Top] is operator AND
Priority(Stack[Top]) >= Priority(Q[i]))
Step 3.4.1.1: Pop operator from Stack[Top] and
put at the end of P[j].
[End of while]
Step 3.4.2: Push operator Q[i] into Stack[Top]
Step 3.5: else if Q[i] is ')' then
Step 3.5.1: while (Stack[Top] not equal to '(')
Step 3.5.1.1: Pop operator from Stack[Top] and
put at the end of P[j]
[End of while]
Step 3.5.2: Delete '(' from Stack[Top]
[End of if]
[End of while]
Step 4: Display "postfix Expression is", P
Step 5: Stop

7 write down the algorithm to perform Insertion and Deletion operations on a linear queue implemented using an array.

Ans Insertion Algorithm:

Consider a queue[Size], Front=-1, Rear=-1 initially. Given an Item for insertion

Insertion (Q[Size], Front, Rear, Item)
Step 1: Start
Step 2: if (Rear = Size-1), then
Step 2.1: Display "queue overflow"
Step 3: else if (Front = -1 AND Rear = -1), then
Step 3.1: Front = 0, Rear = 0
Step 3.2: Q[Rear] = Item
Step 4: else
Step 4.1: Q[++Rear] = Item
[End of if]
Step 5: Stop

Deletion Algorithms:-

[Consider a queue Q[size], initially Front = -1, Rear = -1. Temp declared to hold the deleted value.]

Deletion (Q[size], Front, Rear, Temp)

Step 1: Start

Step 2: if (Front = -1 AND Rear = -1), then

Step 2.1: Display "queue underflow"

Step 3: else if (Front = Rear), then

Step 3.1: Temp = Q[Front]

Step 3.2: Front = -1, Rear = -1

Step 3.3: Display "deleted value is", temp

Step 4: else

Step 4.1: Temp = Q[Front]

Step 4.2: Front = Front + 1

Step 4.3: Display "deleted value is", temp

[End of if]

Step 5: Stop

8 given infix expression Q = A-S/D+(E*F^G)-H

Find its equivalent postfix using stack.

Ans: First put ')' at the end of infix expression Q and push '(' to stack[top].

Symbol	Stack	Postfix Expression
	(	
A	(	A
-	(-	A
S	(-	AS
/	(-/	AS
D	(-/	ASD
+	(+	ASD/-

(	(+(	ASD/-
E	(+(	ASD/-E
*	(+(*	ASD/-E
F	(+(*	ASD/-EF
^	(+(*^	ASD/-EF
G	(+(*^	ASD/-EFG
)	(+	ASD/-EFG^*
-	(-	ASD/-EFG^*+
H	(-	ASD/-EFG^*+H
)	empty	ASD/-EFG^*+H-

9 Given two arrays A[5] and B[7] having elements in ascending order. Design an algorithm to apply merging operation on both arrays and generate the resultant list into array C[12] in ascending order.

Ans Consider A[5] and B[7] having elements in ascending order from lb1 to ub1 and lb2 to ub2 and a resultant array C[12] having lb3 and ub3]

Merging (A[5], lb1, ub1, B[7], lb2, ub2, C[12], lb3, ub3)

Step 1: Start

Step 2: Let i, j, k

Step 3: i = lb1, j = lb2, k = 0

Step 4: while (i <= ub1 AND j <= ub2)

Step 4.1: if (A[i] < B[j]), then

Step 4.1.1: C[k] = A[i], i = i+1, k = k+1

Step 4.2: else

Step 4.2.1: C[k] = B[j], j = j+1, k = k+1

[end of if]

[end of while]

Step 5: while (j <= ub1)

Step 5.1: C[k] = A[i], k = k+1, i = i+1

[end of while]

Step 6: while (j <= ub2)
step 6.1: C[k] = B[j], k=k+1, j=j+1
[end of while]
Step 7: lb3 = 0, ub3 = k-1
Step 8: Stop

10 write down the algorithm to evaluate a given ~~postfix~~ expression using stack.

Ans Consider a postfix expression P given for evaluation.
Declare stack[size], top, i, A, B, R.]
PostfixEvaluation (P, stack, i, top, A, B, R)
Step 1: Start
Step 2: put ')' at the end of postfix P.
Step 3: while (P[i] != ')')
Step 3.1: If (P[i] is operand), then
Step 3.1.1: push P[i] into stack[top]
Step 3.2: else if (P[i] is operator) then
Step 3.2.1: A = Pop 1st element from stack
Step 3.2.2: B = Pop 2nd element from stack
Step 3.2.3: R = B P[i] A
Step 3.2.4: push R into stack[top]
[end of if]
[end of while]
Step 4: Display "Result is", stack[top]
Step 5: Stop

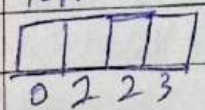
11 Given a stack STK[size] where size=4. Initially the top=-1. Apply the below list of operations on the stack and elaborate the execution process in detail. Push(10), Push(20), POP(), Push(30), Push(40), Push(50), Push(60), POP(), POP(), POP(), Push(70).

Ans

Initially:-

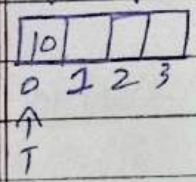
STK[4]

TOP=-1



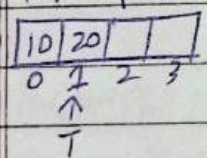
Push(10):-

TOP=-1 → 0



Push(20):-

TOP=0 → 1

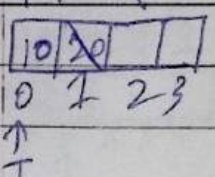


POP():-

TOP=1

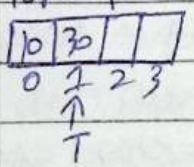
let temp=STK[TOP] =20

TOP=1 → 0



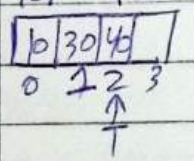
Push(30):-

TOP=0 → 1



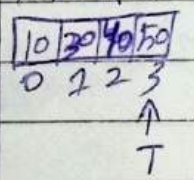
Push(40):-

TOP=1 → 2



Push(50):-

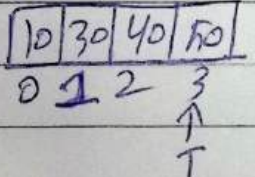
TOP=2 → 3



Push(60):-

TOP=3=size-1

As TOP is equal to size-1. Therefore the stack is full.

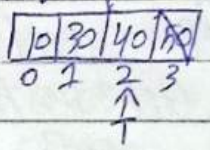


POP():-

TOP=3

let temp=STK[TOP] =50

TOP=3 → 2

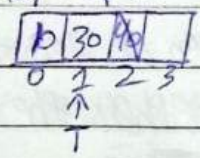


POP():-

TOP=2

let temp=STK[TOP] =40

TOP=2 → 1

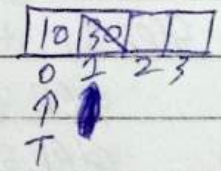


POP():-

TOP=1

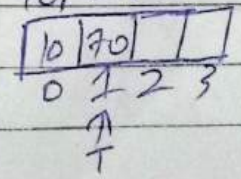
let temp=STK[TOP] =30

TOP=1 → 0



Push(70):-

TOP=0 → 1



12 Given a list of elements from lb to ub in an array A[100]. write two algorithms to perform:
(a) Insertion of an item at a specific location
(b) Search for a given item.

Ans (a) Given an array A[100] having elements from lb to ub and an item to be inserted at a specific location between lb to ub]

Insertion (A[100], lb, ub, item, position)
Step 1: start
Step 2: let i
Step 3: for (i = ub to position), decr by 1
  Step 3.1: A[i+1] = A[i]
[end of for]
Step 4: A[position] = item
Step 5: ub = ub+1
Step 6: stop

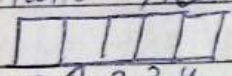
(b) [Given an item to search for its position. Here let us search whether the element is exist or not.]

Search (A[100], lb, ub, item)
Step 1: start
Step 2: let i, flag=0
Step 3: for (i=lb to ub), incr by 1
  Step 3.1: if (A[i] = item), then
    step 3.1.1: flag = 1
    step 3.1.2: break
  [end of if]
[end of for]

Step 4: if (flag = 0), then
    Step 4.1: Display "Item is not found"
Step 5: else
    Step 5.1: Display "Item found"
[End of if]
Step 6: stop

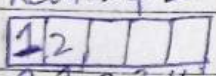
13 Given a linear queue QUEUE[size] where size = 5.
Initially Front = -1 and Rear = -1
Apply the below lits of operations on the queue and elaborate the execution process in details. Insert(1), Insert(2), Insert(3), Delete(), Delete(), Insert(4), Insert(5), Delete(), Insert(6), Insert(7).

Ans Initially :-
Queue[5]
Front = -1, Rear = -1

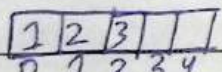


Insert(1) :-
Front = ~~-1~~ 0
Rear = ~~-1~~ 0

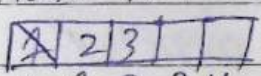
Insert(2):-
Front = 0
Rear = ~~0~~ 1



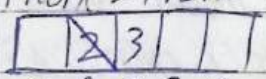
Insert(3):-
Front = 0, Rear = ~~1~~ 2



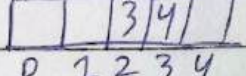
Delete():-
Front = 0
Rear = 2
Let temp = QUEUE[Front]
= 1
Front = Front + 1 = 1



Delete():-
Front = 1
Rear = 2
Let temp = QUEUE[Front]
= 2
Front = Front + 1 = 2



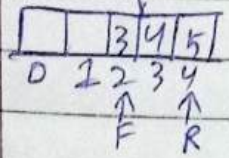
Insert(4):-
Front = 2
Rear = ~~2~~ 3



Insert(5):-

Front = 2

Rear = ~~3~~ 4



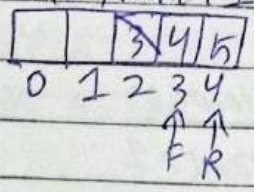
Delete():-

Front = 2

Rear = 4

let temp = QUEUE[Front]
= 3

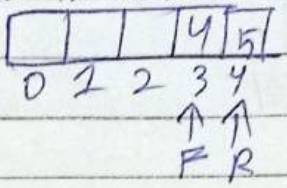
Front = Front + 1 = 3



Insert(6):-

Front = 3, Rear = 4

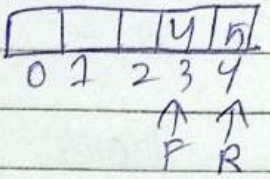
As Rear = size - 1. Therefore stack is full on overflow



Insert(7):-

Front = 3, Rear = 4

As Rear = size - 1. Therefore stack is full on overflow



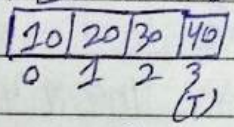
14 List out:-

(a) The stack overflow and underflow conditions and

(b) Linear queue overflow and underflow conditions.

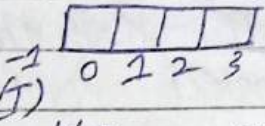
Ans (a) Stack overflow:- When the stack is full then "top = size-1", then it is called stack overflow.

Let stack[4] having elements.



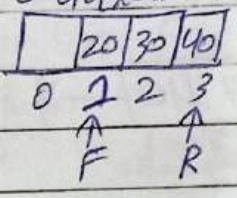
If we try to insert (50), then it will not be possible and it will display "Stack overflow".

Stack underflow :- when the "top = -1 or NULL", then the stack is empty or underflow. Let stack[4] is declared with the top = -1 initially



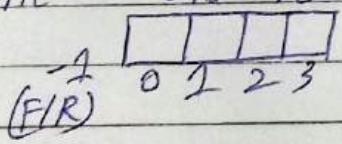
If we try to delete(), then it will not be possible and it will display "stack underflow".

(B) Linear queue overflow :-
when the queue is full then "Rear = size-1", then it is called queue overflow. Let a queue[4] having elements



If we try to insert(50), then it will not be inserted and it will display "queue overflow".

Linear queue ~~over~~ underflow :-
when the "Front = -1 AND Rear = -1", then the queue is empty or underflow. Let a queue[4] is declared with the Front = -1 and Rear = -1 initially



If we try to delete(), then it will not be executed and it will display "queue underflow".

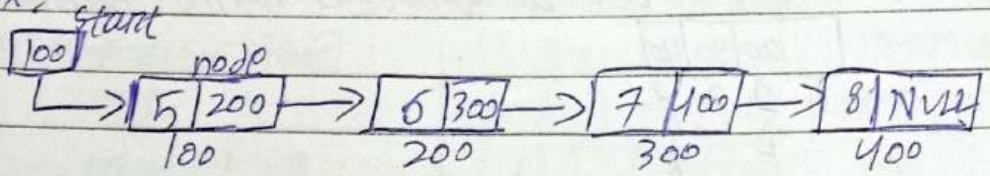
Date
Page

1) Define the terms:
● Abstract Datatype, Linked list, queue, Tree, Stack.

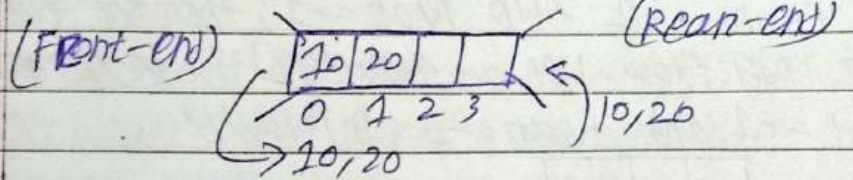
Ans Abstract Datatype:- It is the most suitable datatype for a data model.

Linked List:- It is the collection of non-consecutive nodes connected in serial manner. A node contains 2 parts:-
ⓐ Info part:- It hold the value
ⓑ Linked part:- It hold the address of next node.

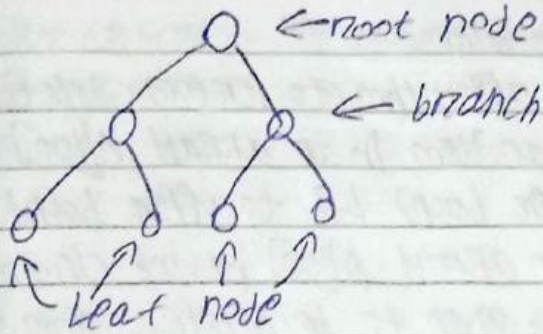
EX:-



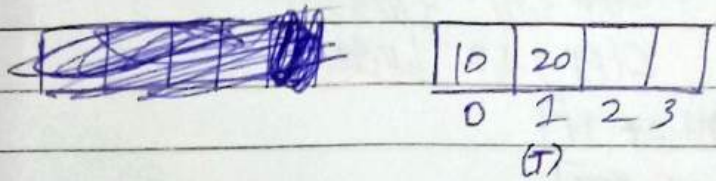
Queue:- A Linear datastructure (i.e. either array or linkedlist) which implements "First in First out" concept (FIFO).
Let us consider a Q[4] having elements.



Tree:- A group of memory locations (node) in which if there is hierarchical relationship exists or top-to-down relationship exists among them, then it is called Tree. Here the top most memory is the root node. From root node it is connected to different nodes called branches. At the bottom level whatever the nodes available called leafs (leaf node). Let us consider a binary tree.



Stack:- A linear datastructure that implements "Last In First out" concept is called stack. The concept of stack can be implemented using either arrays or linked list. Let us consider S[4] having elements



Assignment :- 11

1 Differentiate between Linear search operation and Binary search operation implements on an array of elements.

Ans

Linear search	Binary search
(i) A searching technique that checks each element one by one until the desired element is found.	(i) A searching technique that divides the array into halves and checks the middle element to decide which half to search next.
(ii) works on both sorted and unsorted arrays.	(ii) works only on sorted arrays.
(iii) It is simple and easy to implement.	(iii) If the array is not already sorted then it requires sorting first.
(iv) Inefficient for large datasets.	(iv) more efficient for large datasets.
(v) It is most suitable for small or unsorted datasets.	(v) It is most suitable for large and sorted datasets.
(vi) It works by starting comparing from the first element and check each element sequentially.	(vi) It works by comparing the middle element with the target and eliminates half of the array in each step.
(vii) Time complexity in worst case is O(n).	(viii) Time complexity in worst case is O(log n).

2 Given a list of elements: 70, 40, 50, 30, 35, 25, 45. write down the algorithm for applying insertion sort on the elements to sort them in ascending order.

Ans Consider an array A[size] having elements and it is to be sorted in ascending order

InsertionSort(A[size])

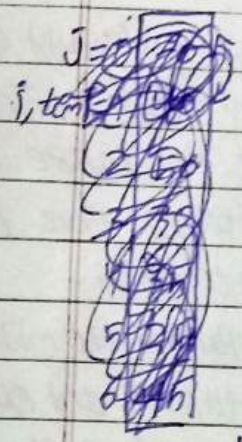
step 1: start

step 2: let i, j, temp

step 3: for(i=1 to size-1), incr by 1

step 3.1: temp = A[i]

Step 3.2: j = i-1
Step 3.3: while (j >= 0 AND A[j] > temp)
Step 3.3.1: A[j+1] = A[j]
Step 3.3.2: j = j-1
[end of while]
Step 3.4: A[j+1] = temp
[end of for]
Step 4: stop



working :-

when i = 1 temp = 40

40 70

70	40	50	30	35	25	45
0	1	2	3	4	5	6
	j					

when i = 2 temp = 50

50 70

40	70	50	30	35	25	45
0	1	2	3	4	5	6
	j					

when i = 3 temp = 30

30 40 50 70

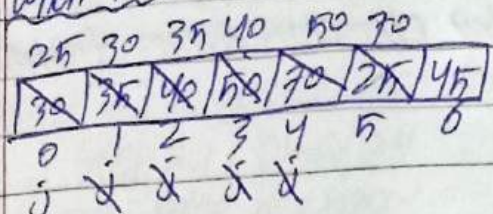
40	50	70	30	35	25	45
0	1	2	3	4	5	6
j	✗	✗				

when i = 4 temp = 35

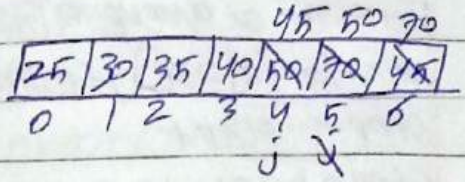
35 40 50 70

30	40	50	70	35	25	45
0	1	2	3	4	5	6
	j	✗	✗			

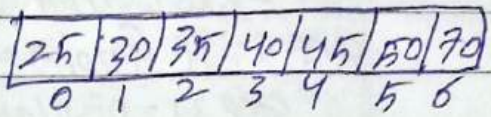
when i=5 temp=25



when i=6 temp=45



so, the final sorted array is



3 write algorithms to implement insertion and deletion operations on a circular queue using array.

Ans Insertion Algorithm :-

Consider q[size], initially front = -1 and rear = -1, item given for insertion

Insertion (q[size], front, rear, item)

Step 1: Start

step 2: if (front = (Rear+1) % size), then

step 2.1: display "Queue is full or overflow"

step 3: else if (front = -1 AND rear = -1)

step 3.1: front = 0, rear = 0, q[rear] = item

step 4: else

step 4.1: rear = (front + 1) % size

step 4.2: q[rear] = item

[end of if]

step 5: stop.

Deletion Algorithm:-

Consider a queue Q[size] having elements from front to rear]

deletion (Q[size], front, rear)

Step 1: Start

Step 2: Let temp

Step 3: if (front = -1 and rear = -1), then

Step 3.1: Display "Queue is empty or underflow"

Step 4: else if (front = rear), then

Step 4.1: temp = Q[front]

Step 4.2: front = -1, rear = -1

Step 4.3: Display "Deleted value is", temp

Step 5: else

Step 5.1: temp = Q[front]

Step 5.2: front = (front + 1) % size

Step 5.3: Display "Deleted value is", temp

[End of if]

Step 6: Stop

4 write down the algorithm to perform Insertion and Deletion operations on it. Execute the below operations on a linear queue Q[5], then find out the sequence of deleted elements and also the elements held by the queue. The operations are:- Insert(S), Insert(W), Insert(E), Insert(R), delete(), delete(), Insert(I), delete(), delete().

Ans Algorithm for insertion in a linear queue:-

[Consider a queue Q[size], front = -1, rear = -1 initially, Given an item for insertion]

Insertion (Q[size], front, rear, item)
Step 1: Start
Step 2: If (rear = size - 1), then
    Step 2.1: Display "Queue overflow"
Step 3: else if (front = -1 AND rear = -1), then
    Step 3.1: front = 0, rear = 0
    Step 3.2: Q[rear] = item
Step 4: else
    Step 4.1: rear = rear + 1
    Step 4.2: Q[rear] = item
[end of if]
Step 5: exit

Algorithm for deletion in a linear queue:-
[consider a queue Q[size], initially front = -1, rear = -1.
-temp declared to hold the deleted value]
Deletion (Q[size], front, rear, temp)
Step 1: Start
Step 2: If (front = -1 AND rear = -1), then
    Step 2.1: Display "queue underflow".
Step 3: else if (front = rear), then
    Step 3.1: temp = Q[front]
    Step 3.2: front = -1, rear = -1
    Step 3.3: Display "Deleted value is", temp.
Step 4: else
    Step 4.1: temp = Q[front]
    Step 4.2: front = front + 1
    Step 4.3: Display "Deleted value is", temp.
[end of if]
Step 5: Stop.

Insert(Q), Insert(W), Insert(E), Insert(R), delete(), delete(), Insert(T), delete(), delete().

Initially:-

Front = -1

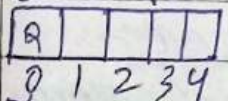
Rear = -1

Q[5]

Insert(Q):-

Front = -1 → 0

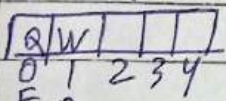
Rear = -1 → 0



Insert(W):-

Front = 0

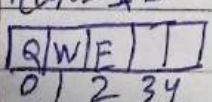
Rear = 0 → 1



Insert(E):-

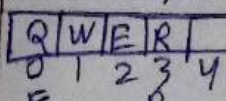
Front = 0

Rear = 1 → 2



Insert(R):-

Front = 0, Rear = 2 → 3

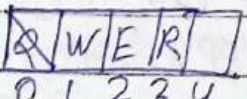


delete():-

Front = 0 → 1

Rear = 3

temp = Q

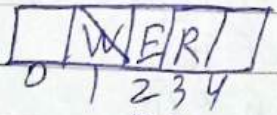


delete():-

Front = 1 → 2

Rear = 3

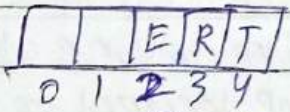
temp = W



Insert(T):-

Front = 2

Rear = 3 → 4

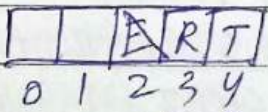


delete():-

Front = 2 → 3

Rear = 4

temp = E

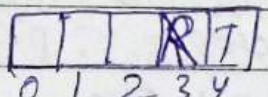


delete():-

Front = 3 → 4

Rear = 4

temp = R



∴ The sequence of Deleted Element:

Q → W → E → R

and elements remaining in queue:

[T]

5 Given a list of elements as follows: 670, 903, 786, 451, 234, 432, 975, 444, 810, 339, 287. Explain the process of Radix sort for sorting the above elements in ascending order.

Ans Algorithm for Radix sort:-

Consides an array a[size] having elements

RadixSort [a(size)]

Step 1: Start

Step 2: Let bucket (0 to 9), digit, max, pass, i

Step 3: max = largest element in array a[size]

Step 4: digit = count of digit in max.

Step 5: for (pass = 1 to digit), incr by 1

Step 5.1: Initialize all the buckets from 0 to 9

Step 5.2: for (i=0 to size-1), incr by 1

Step 5.2.1: digit = find a digit from a[i] as per pass

Step 5.2.2: Store a[i] into corresponding bucket [digit].

[end of for]

Step 5.3: Restore elements from bucket (0 to 9) into array a[size]

[end of for]

Step 6: Stop

670, 903, 786, 451, 234, 432, 975, 444, 810, 339, 287

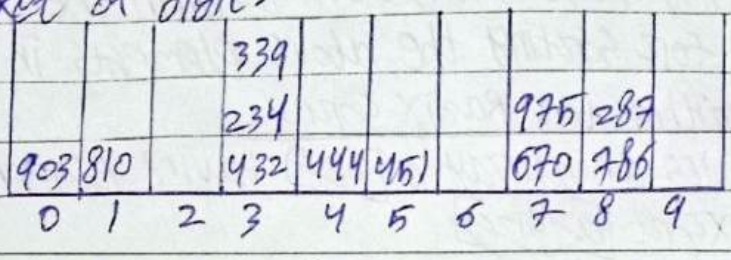
Pass 1:- Identify the unit position digit and store in corresponding bucket digit.

0	1	2	3	4	5	6	7	8	9
670	451	432	903	234	975	786	287		339
810				444					

Restore all the elements from bucket 0 to 9 serially into array:

0	1	2	3	4	5	6	7	8	9	10
670	810	451	432	903	234	444	975	786	287	339

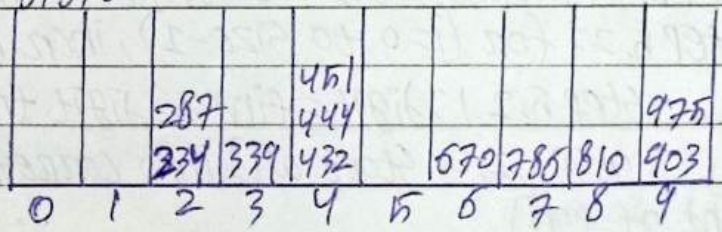
Pass 2:- Identify 10's position digit and store in corresponding bucket of digit.



Restore all the elements from bucket 0 to 9 serially into array.

903	810	432	234	339	444	451	670	975	786	287
0	1	2	3	4	5	6	7	8	9	10

Pass 3:- Identify 100's position digit and store in corresponding bucket of digit.



Restore all the elements from bucket 0 to 9 serially into array.

234	287	339	432	444	451	670	786	810	903	975
0	1	2	3	4	5	6	7	8	9	10

6 Explain the process of quick sort on the given a list of elements: 34, 56, 12, 23, 90, 78, 67 and 45. Write down the algorithm for quick sort to get the elements in ascending order.

Ans Quick Sort Algorithm:-

[Consider an array A[size]. Here l=lower bound, h=upper bound]

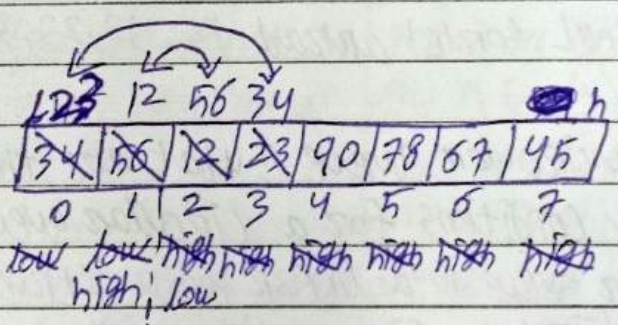
quicksort (A[size], l, h)

step 1: Start

step 2: let low, high, key, temp

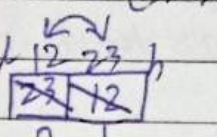
step 3: low = l, high = h, key = A((l+h)/2)

Step 4: while (low <= high)
  step 4.1: while (A[low] < key)
    step 4.1.1: low = low + 1
  [end of while]
  step 4.2: while (A[high] > key)
    step 4.2.1: high = high - 1
  [end of while]
  step 4.3: if (low <= high), then
    step 4.3.1: temp = A[low], A[low] = A[high], A[high] = temp,
      low = low + 1, high = high - 1
  [end of if]
[end of while]
step 5: if (l < high), then
  step 5.1: Call quicksort (A, l, high)
[end of if]
step 6: if (low < h), then
  step 6.1: Call quicksort (A, low, h)
[end of if]
step 7: exit.



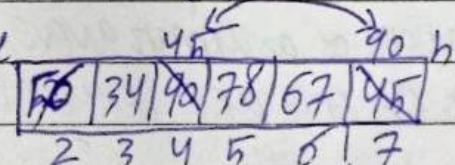
key = a[(l+h)/2]
= a[(0+7)/2]
= a[3]
= 23

quicksort(A, 0, 1)



key = A[(0+1)/2]
= A[0]
= 23

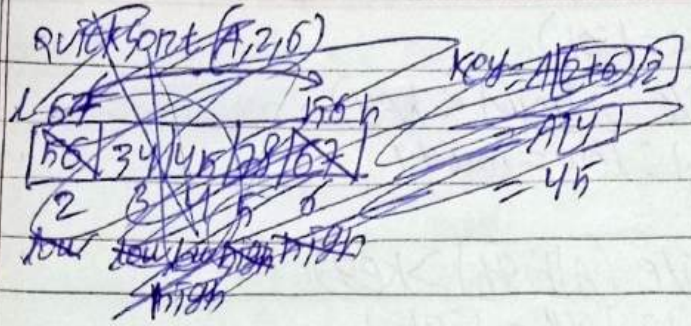
quicksort(A, 2, 7)

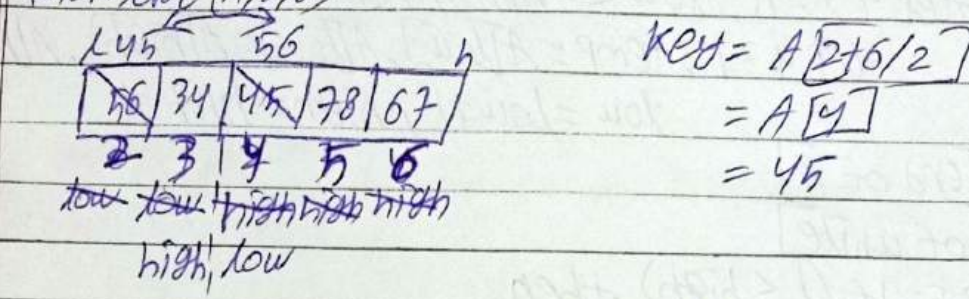


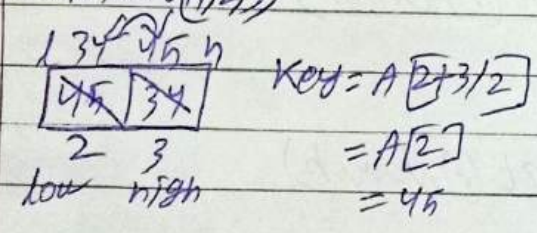
key = A[(2+7)/2]
= A[4]
= 90

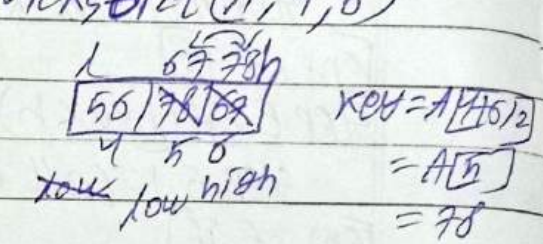
quicksort(A, 2, 6)

quick(A, 7, 7)









So the final sorted array is 12, 23, 34, 45, 56, 67, 78, 90

7 What is a circular queue? what are the overflow and underflow conditions for a circular queue using array.

Ans A circular queue is a linear data structure that follows the FIFO (First-In-First-out) Principle but connects the last position back to the first to form a circle. This overcomes the limitations of a linear queue, which may leave unused space when elements are dequeued.

Queue overflow:-

Situation 1:-

when Front = 0 AND Rear = size-1, then queue is full.

Situation 2:

when front = rear+1, then queue is full

Both situation 1 and 2 can be dealt using a single formula:

Front = (rear+1)%size, then queue is full.

Queue underflow:-

when front = -1 AND Rear = -1, then queue is empty

8 State the difference between queue using Array and Linked queue. What are the drawbacks of a linear queue compared to circular queue.

Queue using Array	Linked queue
(i) The size is fixed because it must be declared in advance.	(i) The size is dynamic and it can grow or shink as needed.
(ii) It is implemented by using a fixed-size array.	(ii) It is implemented by using dynamically allocated linked nodes.
(iii) It may waste memory if elements are deavered but space isn't reused.	(iii) Efficient memory usage, as nodes are allocated only when needed.
(iv) Insertion and deletion are done using array indices.	(iv) Insertion and deletion are done using linked list pointers.
(v) Overflow condition can occur if the array is full.	(v) No overflow condition can occur unless memory is exhausted.
(vi) Flexibility is limited by pre-defined size	(vi) more flexible, as memory is allocated dynamically.

Drawbacks of a linear queue compared to a circular queue:

1. wasted Space:-
In a linear queue, when elements are dequeued, the vacant space at the front is not reused, leading to inefficient memory utilization. But in a circular queue, the rear wraps around and reuses space.

2. queue overflow issue:-
Even if there are empty spaces in a linear queue (due to dequeuing), new elements cannot be inserted unless all elements are shifted. But in a circular queue, the rear pointer wraps around, preventing overflow unless truly full.

3 Not Suitable for continuous Data Streams:-
In applications like network buffering or real-time scheduling, a linear queue is inefficient because it does not recycle memory like a circular queue.

9 write down the algorithm to perform Insertion and Deletion operations on a linear queue implemented using an array.

Ans Insertion Algorithm:-
[Consider a queue Q[size], Front = -1, Rear = -1 initially. given an item for insertion]
Insertion (Q[size], front, Rear, Item)
Step 1: start
Step 2: If (Rear = size-1), then
  Step 2.1: Display "queue overflow".
Step 3: else If (Front = -1 AND Rear = -1), then
  Step 3.1: Front = 0, Rear = 0
  Step 3.2: Q[Rear] = Item.

step 4: else
step 4.1: Rear = Rear + 1
step 4.2: Q [Rear] = Item
[End of if]
step 5: exit

Deletion Algorithms:-
[Consider a queue Q[Size], initially Front = -1, Rear = -1.
Temp declared to hold the deleted value.]
Deletion (Q[Size], Front, Rear, Temp)
step 1: start
step 2: if (Front = -1 AND Rear = -1), then
step 2.1: Display "queue underflow"
step 3: else if (Front = Rear), then
step 3.1: Temp = Q[Front]
step 3.2: Front = -1, Rear = -1
step 3.3: Display "deleted value is", Temp
step 4: else
step 4.1: TEMP = Q[Front]
step 4.2: Front = Front + 1
step 4.3: Display "Deleted value is", Temp.
[End of if]
step 5: exit.

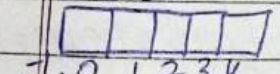
Date
Page

10 Given a linear queue queue[size] where size = 5. Initially Front = -1 and Rear = -1. Apply the below list of operations on the queue and elaborate the execution process in details. Insert(1), Insert(2), Insert(3), delete(), delete(), Insert(4), Insert(5), Delete(), Insert(6), Insert(7).

Ans

Initially

queue[5]



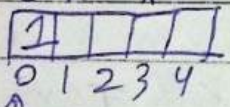
Front = -1

Rear = -1

Insert(1):-

Front = ~~-1~~ 0

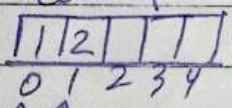
Rear = ~~-1~~ 0



Insert(2):-

Front = 0

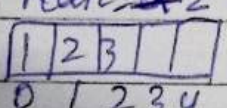
Rear = ~~0~~ 1



Insert(3):-

Front = 0

Rear = ~~1~~ 2



delete():-

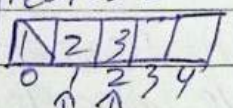
~~Front~~

Temp = queue[Front]

= 1

Front = ~~0~~ 1

Rear = 2



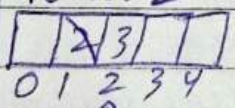
Delete():-

Temp = queue[Front]

= 2

Front = ~~1~~ 2

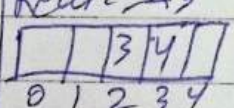
Rear = 2



Insert(4):-

Front = 2

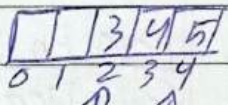
Rear = ~~2~~ 3



Insert(5):-

Front = 2

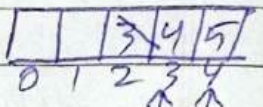
Rear = ~~3~~ 4



Delete():- Temp = queue[Front] = 3

Front = ~~2~~ 3

Rear = 4

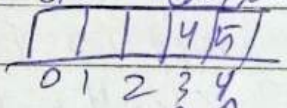


Insert(6):-

Front = 3

Rear = 4

As Rear = size - 1, so queue is full or overflow.

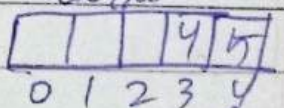


Insert(7):-

Front = 3

Rear = 4

As Rear = size - 1, so queue is full or overflow.



Assignment :-3

1 Design algorithm for performing push and pop operations in a linked STACK.

Ans Push operation :-

[Consider *Top points to top most node. Initially TOP = NULL]

push (TOP, Item)

step 1: start

step 2: let fresh

step 3: fresh = malloc()

step 4: if (fresh = NULL), then

    step 4.1: Display "stack is full or memory is full".

    step 4.2: exit

step 5: else

    step 5.1: Info [fresh] = item

    step 5.2: Link [fresh] = Top

    step 5.3 : Top = fresh

[end of if]

step 6: stop

Pop operation :-

[Consider *Top points to Top most node in a stack.]

pop (TOP)

step 1: start

step 2: let ptr

step 3: if (Top = NULL), then

    step 3.1: Display "stack is empty or underflow"

    step 3.2: exit

step 4: else

    step 4.1: ptr = top

    step 4.2: TOP = Link [Top]

    step 4.3 = free (ptr)

[end of if]

step n: exit

2 Design algorithm for performing insertion and deletion operations in a Linked queue.

Ans Insertion operation:-

[Initially Front=NULL, Rear=NULL]

Insertion (Front, Rear, Item)

step 1: start

step 2: let fresh

step 3: fresh=malloc()

step 4: if (fresh = NULL), then

step 4.1: display "queue overflow"

step 4.2: exit

step 5: else

step 5.1: info [fresh] = item

step 5.2: Link [fresh] = NULL

step 5.3: if (Front = NULL AND Rear=NULL), then

step 5.3.1: Front = fresh, Rear = fresh

step 5.4: else

step 5.4.1: Link [Rear] = fresh

step 5.4.2: Rear = fresh

[end of if]

[end of if]

step 6: stop

Deletion operation:-

Deletion (Front, Rear)

step 1: start

step 2: let Ptr

step 3: if (Front = NULL AND Rear = NULL), then

step 3.1: Display "queue underflow"

step 3.2: exit.

Step 4: else if (Front = Rear), then
Step 4.1 : Ptr = Front
Step 4.2: Front = NULL, Rear = NULL
Step 4.3 : free(Ptr)
Step 5: else
Step 5.1 : Ptr = Front
Step 5.2 : Front = Link [Front]
Step 5.3 : free(Ptr)
[end of if]
Step 6 : Stop

3 write down the algorithm for insertion of an item at the END of a single Linked List.

Ans [Consider *start for holding address of 1st node.
An item is given for insertion.]
InsertAtEnd (start, Item)
Step 1: start
Step 2 : Def fresh, Ptr
Step 3: fresh = malloc ( )
Step 4: if (fresh = NULL), then
Step 4.1: Display "memory is full"
Step 4.2 : exit
Step 5: else
Step 5.1 : Info [fresh] = item
Step 5.2 : Link [fresh] = NULL
Step 5.3 : If (start = NULL), then
Step 5.3.1 : start = fresh
Step 5.4 : else
Step 5.4.1 : Ptr = start.

Step 5.4.2 : while (link[Ptr] ≠ NULL)
Step 5.4.2.1 : Ptr = link[Ptr]
[End of while]
Step 5.4.3 : link[Ptr] = fresh
[End of if]
[End of if]
Step 6 : Stop

4 Design the algorithm to display all the node items of a double linked list and find the largest info present in it.

Ans: [Consider *start points to double linked list and max contains the largest info present in it]
Display (start)
Step 1 : start
Step 2 : let Ptr, max
Step 3 : if (start = NULL), then
Step 3.1 : Display "Double linked list is empty"
Step 3.2 : exit
Step 4 : else
Step 4.1 : Ptr = start
Step 4.2 : while (Ptr ≠ NULL)
Step 4.2.1 : Display Info[Ptr]
Step 4.2.2 : if (Info[Ptr] > max), then
Step 4.2.2.1 : max = Info[Ptr]
[End of if]
Step 4.2.3 : Ptr = next[Ptr]
[End of while]
Step 4.3 : Display "The largest value is", max
[End of if]
Step 5 : Stop.

5 Design an algorithm to search an item and then delete that node in a single linked list.

Ans [Consider *start points linked list and an item given to search and delete]

Deletion (start, Item)

step 1: start

step 2: let Ptr, Prev

step 3: Ptr = start

step 4: while (ptr ≠ NULL AND item ≠ info[Ptr])

step 4.1: Prev = Ptr

step 4.2: Ptr = Link [Ptr]

[End of while]

step 5: if (Ptr = NULL), then

step 5.1: Display "Item not found"

step 6: else if (Ptr = start), then

step 6.1: start = Link [start]

step 6.2: free (Ptr)

step 7: else

step 7.1: Link [Prev] = Link [Ptr]

step 7.2: free (Ptr)

[End of if]

step 8: stop.

6. write down the algorithm for Reversing a single Linked List.

Ans [Consider *start Points to a Linked List]
Reverse(start)
step 1: Start
step 2: let Ptr = start, Prev = NULL, Ptr 1
step 3: while (Ptr ≠ NULL)
    step 3.1: Ptr 1 = Link [Ptr]
    step 3.2: Link [Ptr] = Prev
    step 3.3: Prev = Ptr, Ptr = Ptr 1
[end of while]
step 4: start = Prev
step 5: stop

7. write algorithms for counting the nodes and Finding the sum of all Info parts of a Double Linked List.

Ans [Consider *start Points to a Linked List]
Counting and DisplaySum (start)
step 1: Start
step 2: let Ptr, count = 0, sum = 0
step 3: Ptr = start
step 4: while (Ptr ≠ NULL)
    step 4.1: sum = sum + info[Ptr], count = count + 1
    step 4.2: Ptr = next [Ptr]
[end of while]
step 5: Display "The total no of nodes are", count
step 6: Display "The sum of all nodes are", sum
step 7: stop.

8 write an algorithm to delete a given item from a Double Linked List.

Ans [Consider *start points to a linked list and item is given for deletion]

DeletionItem (start, searchItem)

step 1: start

step 2: let Ptr, Ptr1, Ptr2

step 3: Ptr = start

step 4: while (Ptr ≠ NULL AND Info [Ptr] ≠ searchItem)

step 4.1: Ptr = next [Ptr]

[End of while]

step 6: if (Ptr = NULL), then

step 6.1: Display "search Item not exist"

step 6: else if (Ptr = start), then

step 6.1: start = next [start]

step 6.2: Prev [start] = NULL

step 6.3: free (Ptr)

step 7: else if (next [Ptr] = NULL), then

step 7.1: Ptr 1 = prev [Ptr]

step 7.2: next [Ptr1] = NULL

step 7.3: free (Ptr)

step 8: else

step 8.1: Ptr 1 = Prev [Ptr]

step 8.2: Ptr 2 = next [Ptr]

step 8.3: next [Ptr1] = Ptr2

step 8.4: Prev [Ptr2] = Ptr 1

step 8.5: free (Ptr)

[End of if]

step 9: stop.

9 write an algorithm to Insertion at the beginning and Insertion at the end in a Double Linked List.

Ans Insertion of a node at the beginning:-

[Consider *start points to a linked list and an item is given to be inserted at the beginning.]
InsertFirst (start, item)
Step 1: start
Step 2: let fresh
Step 3: fresh = malloc()
Step 4: Info [fresh] = item
Step 5: next [fresh] = NULL
Step 6: prev [fresh] = NULL
Step 7: if (start = NULL), then
    step 7.1: start = fresh
Step 8: else
    Step 8.1: next [fresh] = start
    Step 8.2: prev [start] = fresh
    Step 8.3: start = fresh
[end of if]
Step 9: Stop

Insertion of a node at the end:-
[Consider *start points to a linked list and an item is given to be inserted at the end]
InsertLast (start, item)
Step 1: start
Step 2: let fresh, ptr
Step 3: fresh = malloc()
Step 4: Info [fresh] = item

Step 5: next [fresh] = NULL
Step 6: Prev [fresh] = NULL
Step 7: if (start = NULL), then
    step 7.1: start = fresh
Step 8: else
    step 8.1: Ptr = start
    step 8.2: while (next [Ptr] ≠ NULL)
        step 8.2.1: Ptr = next [Ptr]
    [end of while]
    step 8.3: next [Ptr] = fresh
    step 8.4: Prev [fresh] = Ptr
[end of if]
Step 9: Stop

10 Design an algorithm to display all the Info part of Double Linked List from last node to 1st node.

Ans [Consider *start points to a linked list]
DisplayInReverse(start)
Step 1: Start
Step 2: Let Ptr = start
Step 3: while (next [Ptr] ≠ NULL)
    Step 3.1: Ptr = next [Ptr]
[end of while]
Step 4: while (Ptr ≠ NULL)
    Step 4.1: Display Info [Ptr]
    Step 4.2: Ptr = Prev [Ptr]
[end of while]
Step 5: Stop.

Sample Questions For Practice

Date
Page

1 Design algorithm for performing PUSH and POP operations in a Linked STACK.

Ans Push operation:-

[Consider * TOP points to top most node. Initially TOP = NULL]

Push (TOP, Item)

Step 1: Start

Step 2: let fresh

Step 3: fresh = malloc()

Step 4: if (fresh = NULL), then

Step 4.1: Display "Stack is full or memory is full".

Step 4.2: exit

Step 5: else

Step 5.1: Info [fresh] = item

Step 5.2: Link [fresh] = TOP

Step 5.3: TOP = fresh

[end of if]

Step 6: Stop

Pop operation:-

[Consider * TOP points to TOP most node in a stack]

Pop (TOP)

Step 1: Start

Step 2: let Ptr

Step 3: if (TOP = NULL), then

Step 3.1: Display "Stack is empty or underflow"

Step 3.2: exit

Step 4: else

Step 4.1: Ptr = TOP

Step 4.2: TOP = Link [TOP]

Step 4.3: free (Ptr)

[end of if]

Step 5: exit.

2. Design algorithm for performing insertion and deletion operations in a Linked QUEUE.

Ans Insertion operation :-

[Initially Front = NULL, Rear = NULL]

Insertion(Front, Rear, Item)

Step 1: Start

Step 2: let fresh

Step 3: fresh = malloc()

Step 4: if (fresh = NULL), then

Step 4.1: Display "queue overflow"

Step 4.2: exit

Step 5: else

Step 5.1: info [fresh] = item

Step 5.2: Link [fresh] = NULL

Step 5.3: if (Front = NULL AND Rear = NULL), then

Step 5.3.1: Front = fresh, Rear = fresh

Step 5.4: else

Step 5.4.1: Link [Rear] = fresh

Step 5.4.2: Rear = Fresh

[end of if]

[end of if]

Step 6: Stop

Deletion operation :-

Deletion (Front, Rear)
Step 1 : Start
Step 2 : Let Ptr
Step 3 : if (Front = NULL AND Rear = NULL), then
    Step 3.1 : Display "queue underflow"
    Step 3.2 : exit
Step 4 : else if (Front = Rear), then
    Step 4.1 : Ptr = Front
    Step 4.2 : Front = [redacted] NULL, Rear = NULL
    Step 4.3 : free (Ptr)
Step 5 : else
    Step 5.1 : Ptr = Front
    Step 5.2 : Front = Link [Front]
    Step 5.3 : free (Ptr)
End of if
Step 6 : Stop

(3) write down the algorithm for insertion of an item at the END of a single Linked List.

Ans [Consider *start for holding address of 1st node. An item is given for insertion]
InsertEnd (Start, Item)
Step 1 : Start
Step 2 : Let fresh, Ptr
Step 3 : fresh = malloc ()
Step 4 : Info [fresh] = item
Step 5 : Link [fresh] = NULL
Step 6 : If (Start = NULL), then
    Step 6.1 : Start = Fresh.

Step 7: else
Step 7.1: Ptr = start
Step 7.2: while (Link [Ptr] ≠ NULL)
Step 7.2.1: ptr = link [Ptr]
[end of while]
Step 7.3: Link [Ptr] = fresh
[End of if]
Step 8: stop

4 Design an algorithm to display all the Info part of Double linked list from Last node to 1st node.

Ans [Consider *start for holding address of 1st node]
~~Step 1~~ DisplayReverse (start)
Step 1: start
Step 2: let fresh
Step 3: If (start = NULL), then
Step 3.1: display "Double linked list is empty"
Step 3.2: exit
Step 4: else
Step 4.1: fresh = start
Step 4.2: while ([redacted] next [fresh] ≠ NULL), then
Step 4.2.1: fresh = [redacted] next [fresh]
[end of while]
Step 4.3: while (fresh ≠ NULL), then
Step 4.3.1: display Info [fresh]
Step 4.3.2: fresh = Prev [fresh]
[end of while]
[End of if]
Step n: stop.

5 write algorithms for counting the nodes and finding the sum of all info parts of a Double Linked List.

Ans [consider *start for holding address of 1st node]
counter and sum (start)
step 1: start
step 2: let ptr, count=0, sum=0
step 3: if (start = NULL), then
    step 3.1: Display "Double Linked List is empty"
    step 3.2: exit
step 4: else
    step 4.1: ptr = start
    step 4.2: while (Ptr ≠ NULL)
        step 4.2.1: count = count + 1
        step 4.2.2: sum = sum + info[ptr]
        step 4.2.3: ptr = next[ptr]
    [end of while]
    step 4.3: Display "Total number of nodes are", count
    step 4.4: Display "sum of all info parts are", sum
[end of if]
step 5: stop

6 what is a Binary Search Tree? write the steps of algorithm for inserting an element into a BST.

Ans A binary tree is termed as Binary Search Tree (BST) when each left child of Parent is less than its Parent node and right child of Parent is greater than its parent node.

Insertion operation:-

[Assume root represents root node and item given for insertion]

InsertBST(Root, item)

Step 1: Start

step 2: let Ptr, fresh, Prev

Step 3: fresh = malloc()

Step 4: info[fresh] = item

Step 5: left[fresh] = NULL

Step 6: right[fresh] = NULL

Step 7: if (root = NULL), then

    Step 7.1: root = fresh

Step 8: else

    Step 8.1 : Ptr = root

    Step 8.2 : while (ptr ≠ NULL)

        Step 8.2.1 : Prev = Ptr

        Step 8.2.2 : if (info[Ptr] > info[fresh]), then

            Step 8.2.2.1 : Ptr = left[Ptr]

        Step 8.2.3 : else

            Step 8.2.3.1 : Ptr = right[Ptr]

        [end of if]

    [end of while]

[end of if]

Step 9: if (Ptr = NULL), then

    Step 9.1 : if (info[Prev] > info[fresh]), then

        Step 9.1.1 : left[Prev] = fresh

    Step 9.2 : else

        Step 9.2.1 : right[Prev] = fresh

    [end of if]

[end of if]

step 10 : exit.

7 Briefly elaborate the memory representation of a binary tree and their types with suitable example.

Ans A binary tree can be stored in memory in 2 ways:-

(i) Array representation OR sequential representation

(ii) Linked representation

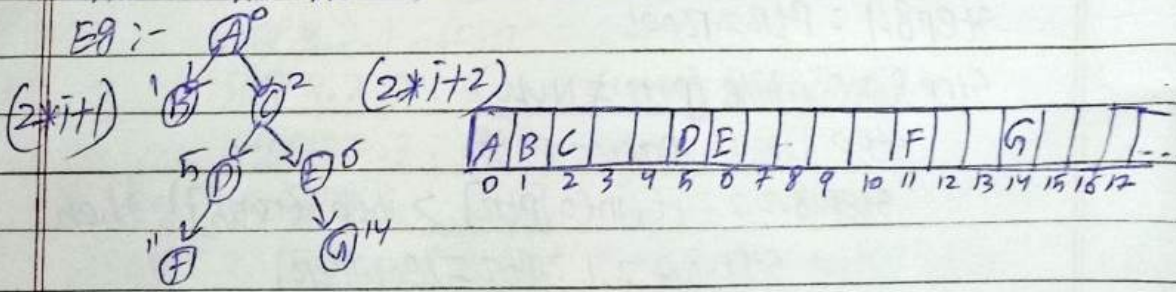
(i) Array representation :-

→ We can store a binary tree nodes in the following manner:

1. Store the next node at i^{th} location.
2. Store the left child at $2*i+1$ location.
3. Store the right child at $2*i+2$ location.

continue this process for every parent and child nodes.

Eg :-



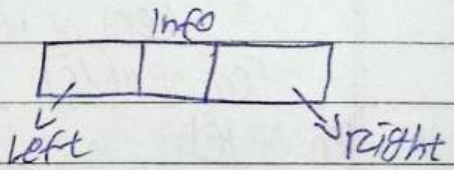
(ii) Linked Representation :-

→ A binary tree can be stored using a collection of nodes.

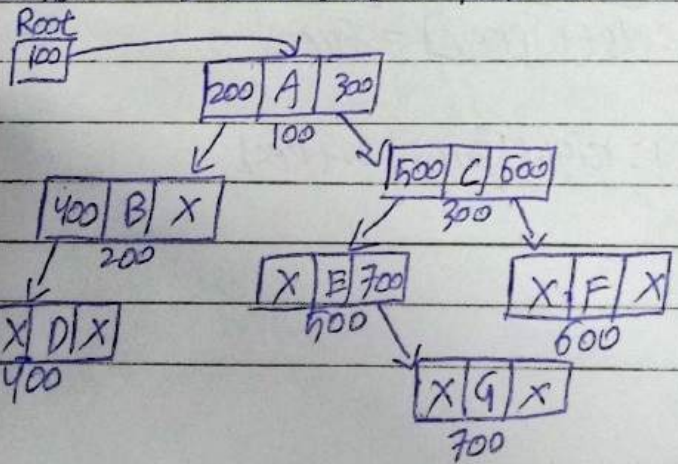
A node contains:- (1) Info

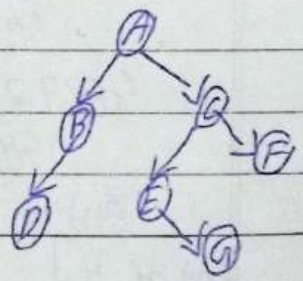
(2) Left Pointer

(3) Right Pointer



consider a binary tree to represent in linked form.





8 write down the algorithm for Inorder traversal for a binary tree. When an inorder and preorder sequence of nodes given then how to construct a binary tree.

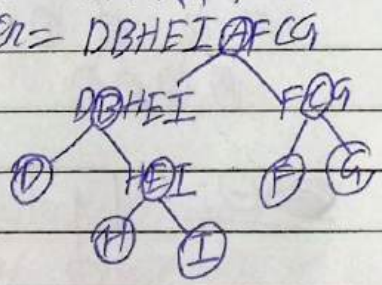
Ans Inorder traversal algorithm :-

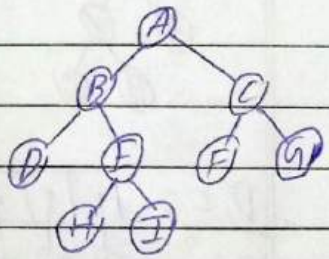
Inorder(node)
step 1: start
step 2: If (node ≠ NULL), then
    step 2.1: Inorder (left[node])
    step 2.2: Process node
    step 2.3: Inorder (right[node])
[End of if]
step 3: stop.

~~construction of a binary tree~~
using Inorder and Preorder traversal method :-

① 1st node of the PREORDER traversal is the root node of the tree.
② Find the position of the root node in the INORDER traversal.
   @ Nodes preceding the root in the INORDER construct the left subtree.
   ⓑ Nodes succeeding the root in the INORDER construct the right subtree
③ Find out the next roots from each sub trees using PREORDER.
④ Continue the above process until leaf nodes are found.

EX:- Preorder = ~~A B D E H I C F G~~
Inorder = D B H E I A F C G





Date
Page

9 Given an infix expression (A+B/C-D^E*F-G). Find its equivalent postfix notation and then using stack construction an expression tree.

Ans Given infix expression

A+B/C-D^E*F-G

= A+B/C-[DE^]*F-G

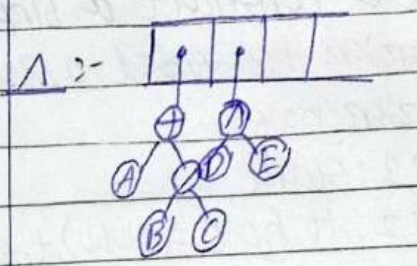
= A+B/C-[DE^F*]-G

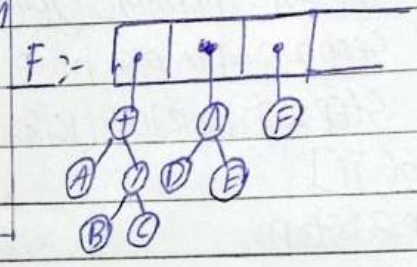
= A+[BC/]-[DE^F*]-G

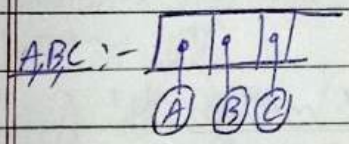
= [ABC/+]-[DE^F*]-G

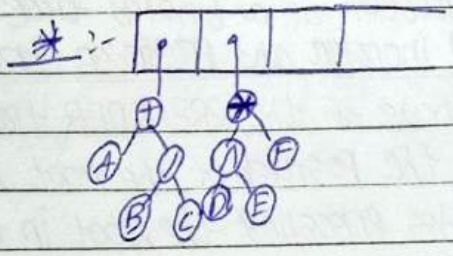
= [ABC/+DE^F*-]-G

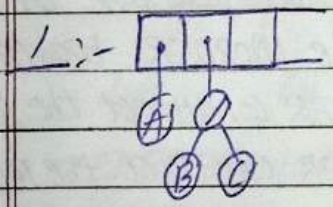
= ABC/+DE^F*-G-

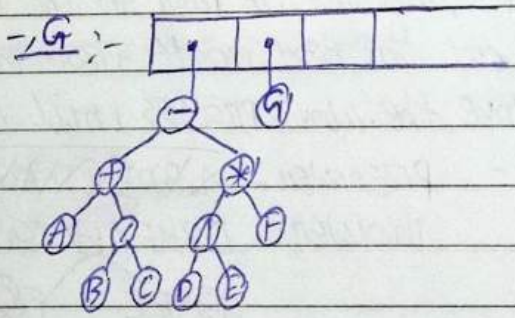


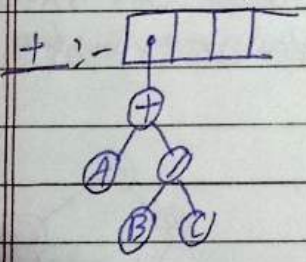


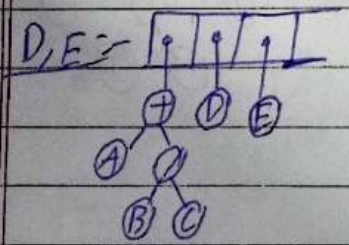


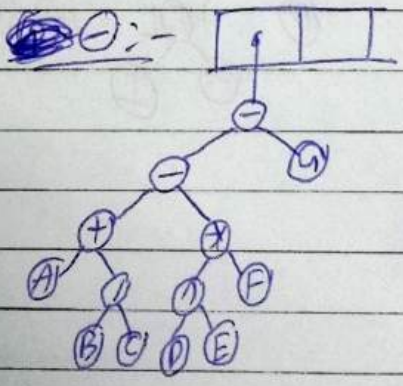












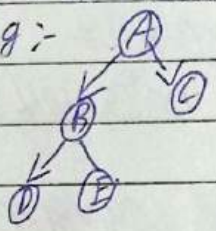
Q. Write down all the 3 different recursive traversal methods for traversing nodes of a binary tree.

Ans The 3 different recursive traversal methods for traversing nodes of a binary tree are :-

① Inorder Traversal method
② Preorder Traversal method
③ Postorder Traversal method.

① Inorder Traversal method :-

Here, the order of traversing the nodes will be left, root, right order. Eg :-



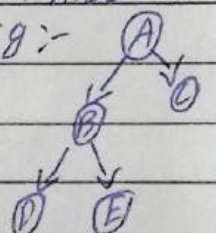
The inorder sequence will be: DBEAC.

Algorithm :-

Inorder(node)
step 1 : Start
step 2 : if (node ≠ NULL), then
    step 2.1 : Inorder (left[node])
    step 2.2 : Process node.
    step 2.3 : Inorder (right[node])
[end of if]
step 3 : Stop

② Preorder Traversal method :-

Here Traversal of node will be in the sequence of root, left, right order. Eg :-



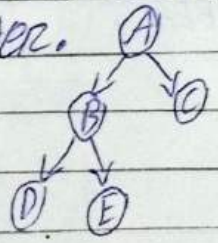
The Preorder sequence will be :- ABDEC

Algorithm :-

Preorder(node)
Step 1 : Start
Step 2 : if (node ≠ NULL), then
    Step 2.1 : Process node
    Step 2.2 : Preorder (left [node])
    Step 2.3 : Preorder (right [node])
[end of if]
Step 3 : Stop

③ Postorder Traversal Method :-

Here, traversal sequence of nodes will be as : left, right, root order.



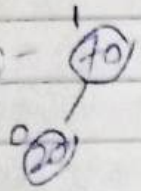
The sequence of nodes = DEBCA.

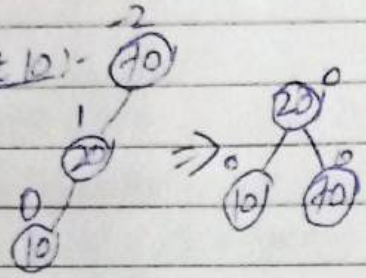
Algorithm :-

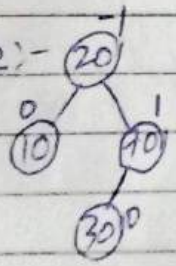
postorder (node) :-
Step 1 : Start
Step 2 : if (node ≠ NULL), then
    Step 2.1 : Postorder (left [node])
    Step 2.2 : Postorder (right [node])
    Step 2.3 : Process node
[end of if]
Step 3 : Stop.

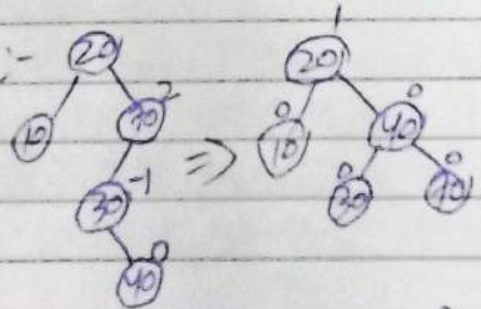
11 Given a sequence of elements: 70, 20, 10, 30, 40, 50, 60, 25, 35, 45, 75, 95, 10, 52, 92. construct an AVL tree by inserting each element.

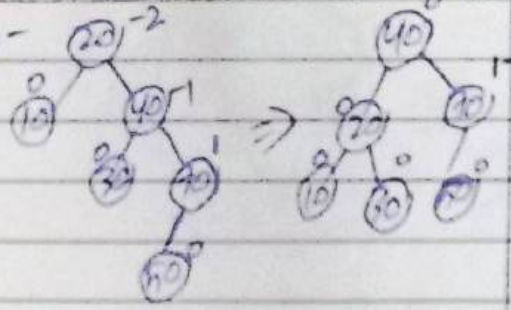
Ans Insert 70:-  ← pivot

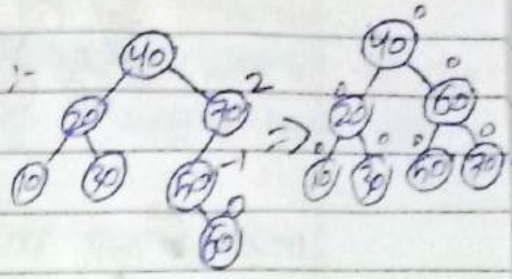
Insert 20:- 

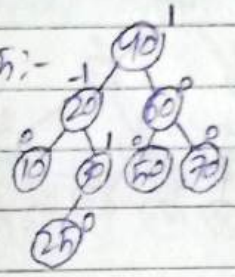
Insert 10:- 

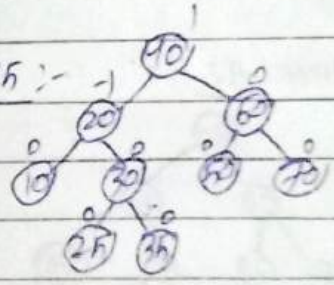
Insert 30:- 

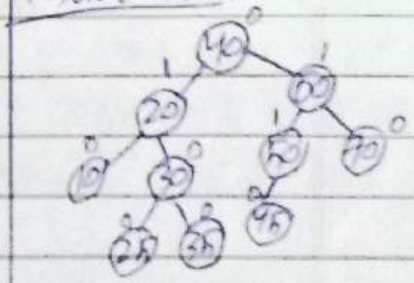
Insert 40:- 

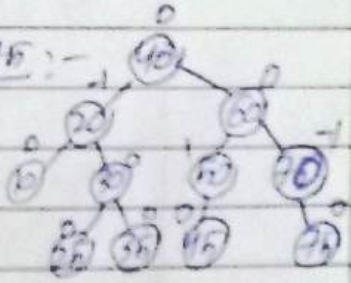
Insert 50:- 

Insert 60:- 

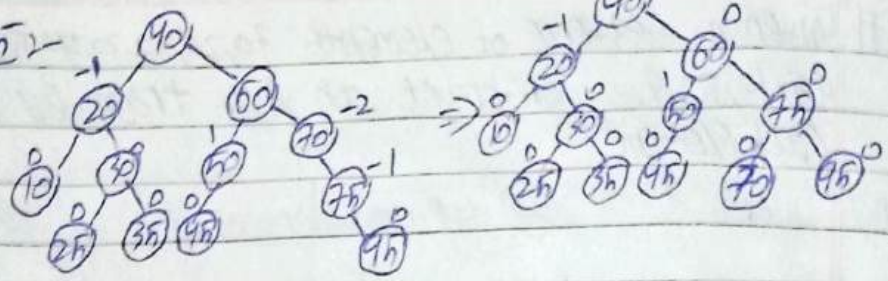
Insert 25:- 

Insert 35:- 

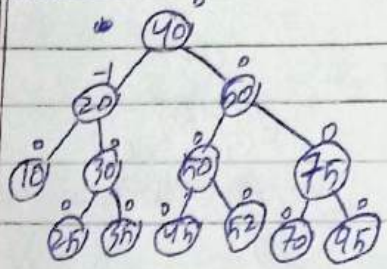
Insert 45:- 

Insert 75:- 

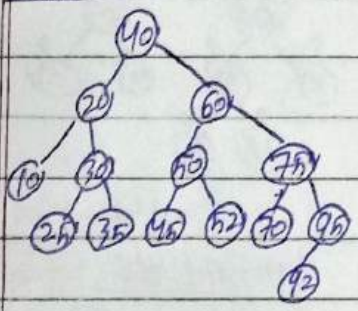
Insert 95 :-



Insert 52 :-



Insert 92 :-



(12) What is Heap Tree and its types? Explain the procedure for construction of max heap tree and min heap tree both with suitable example.

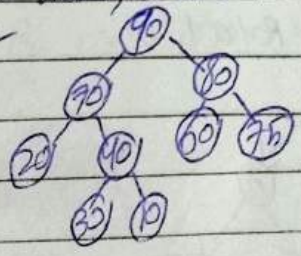
Ans. A binary tree in which from root node to leaf node if we can get an ordered list of elements then it is called a heap tree. There are 2 types of heap tree:-

(i) max heap tree (or) Descending Heap Tree
(ii) min heap tree (or) Ascending Heap Tree

max Heap Tree:-

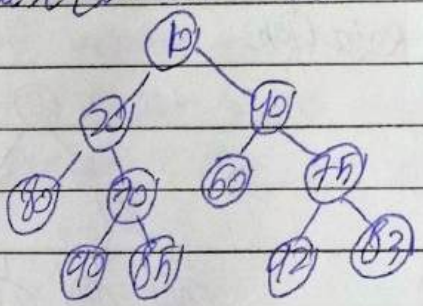
A binary tree in which from root node to leaf node, if we get descending order of elements then it is called max heap tree. Here, every parent node is greater than child node.

Eg:-



Min Heap Tree:-

A binary tree in which from root node to leaf node, if we get ascending order of elements, then it is called min heap tree. Here, every parent is smaller than both child node and root node contains smallest element. Eg:-



ASSIGNMENT :- IV

1 What is Height Balanced BST ? How to find the balance factor of a node ? what are the rotations exist to balance a node ?

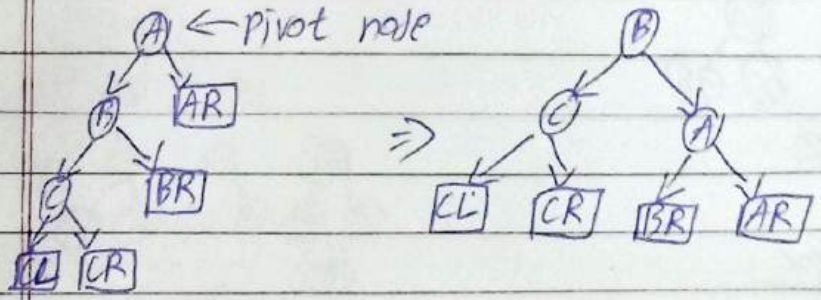
Ans A Height Balanced Binary search Tree is a binary search tree in which the height of the left and right subtrees of any node can differ by -1, 0, 1. This ensures that the tree remains balanced and provides efficient operations such as insertion, deletion and lookup. To find the balance factor of a node.

Balance factor (Bf) = Height of left branch - Height of right branch

The rotations exist to balance a node are :-

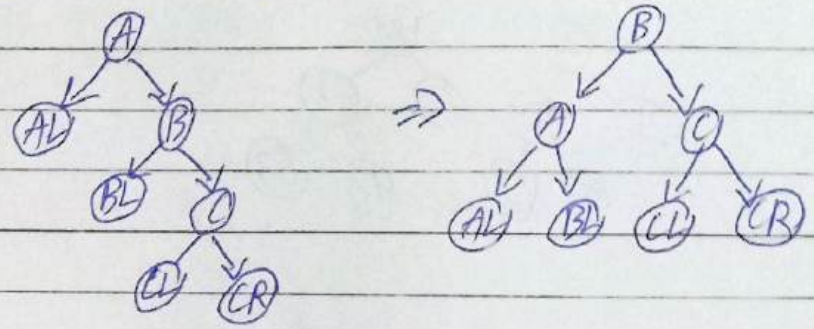
① LL Rotation :-

When the tree gets imbalanced due to insertion of a node at left side of the left sub tree of the pivot node, then we need to follow the LL Rotation.

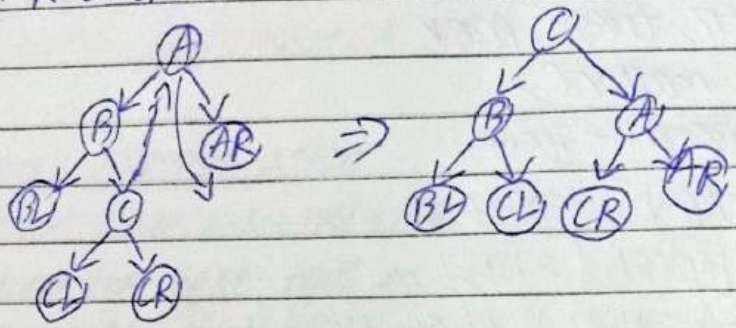


② RR Rotation :-

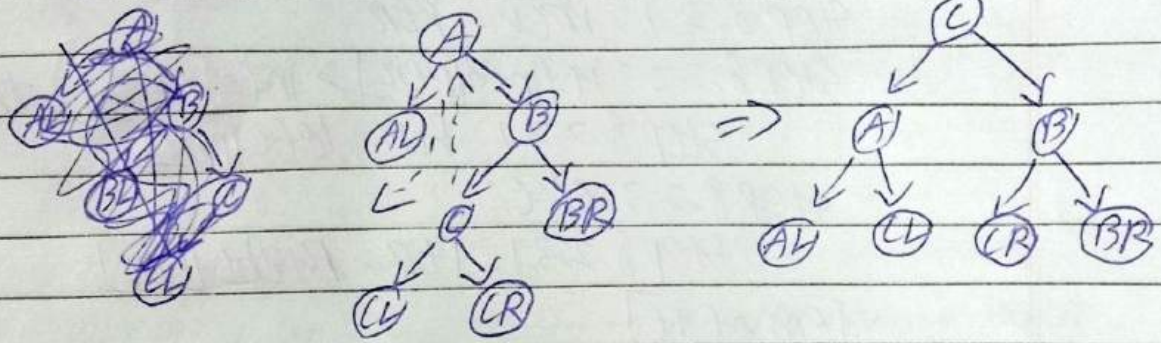
When the tree gets imbalanced due to insertion of a node at right side of the right subtree of the pivot node, we need to follow the RR Rotation.



③ LR Rotation :-
When tree gets imbalanced due to insertion of a node at right side of the left subtree of the pivot node, we need to follow the LR Rotation.



④ RL Rotation :-
When tree gets imbalanced due to insertion of a node at left side of the right subtree of the pivot node, we need to follow the RL rotation.



2 What is a binary search tree? write the steps of algorithms for inserting an element into a BST.

Ans A binary tree is termed as Binary Search Tree (BST) when each left child of Parent is less than its Parent node and right child of Parent is greater than its Parent node.

insertion operation:-

[Assume root represents root node and an item given for insertion]

Insert BST (root, item)
Step 1: Start
Step 2: Let Ptr, fresh, Prev
Step 3: fresh = malloc()
Step 4: info[fresh] = item
Step 5: left[fresh] = NULL
Step 6: right[fresh] = NULL
Step 7: if (root = NULL), then
    Step 7.1: root = fresh
Step 8: else
    Step 8.1: Ptr = root
    Step 8.2: while (Ptr ≠ NULL)
        Step 8.2.1: Prev = Ptr
        Step 8.2.2: if (info[Ptr] > info[fresh]), then
            Step 8.2.2.1: Ptr = left[Ptr]
        Step 8.2.3: else
            Step 8.2.3.1: Ptr = right[Ptr]
        [end of if]
    [end of while]
[End of if]
Step 9: If (Ptr = NULL), then
    step 9.1: if (info[Prev] > info[fresh]), then
        Step 9.1.1: left[Prev] = fresh
    step 9.2: else
        Step 9.1.2: right[Prev] = fresh
    [end of if]
[end of if]
Step 10: exit

Briefly elaborate the memory representation of a binary tree and their types with suitable example.

A binary tree can be stored in memory in 2 ways:

1. Array representation (OR) sequential representation
2. Linked representation.

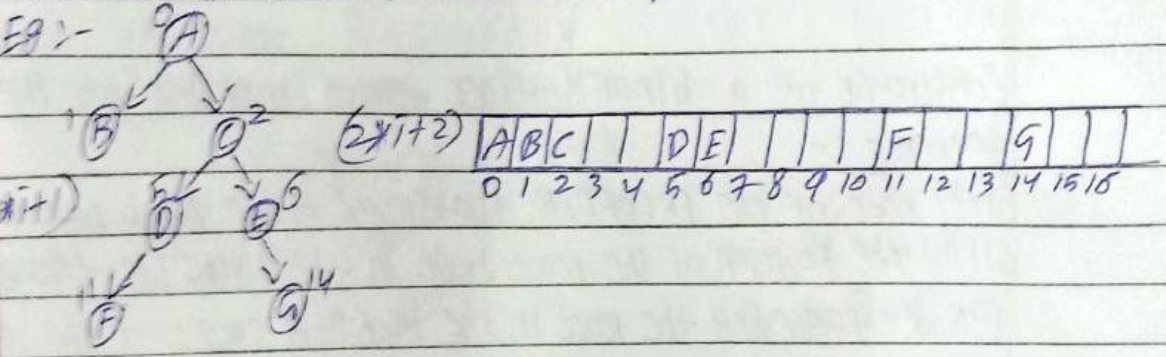
Array representation :-

We can store a binary tree nodes in the following names:-

1. Store the next node at i^{th} location.
2. Store the left child at $2*i+1$ location.
3. Store the right child at $2*i+2$ location.

Continue this process for every parent and child nodes.

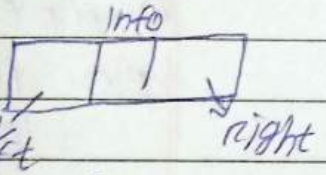
Eg :-



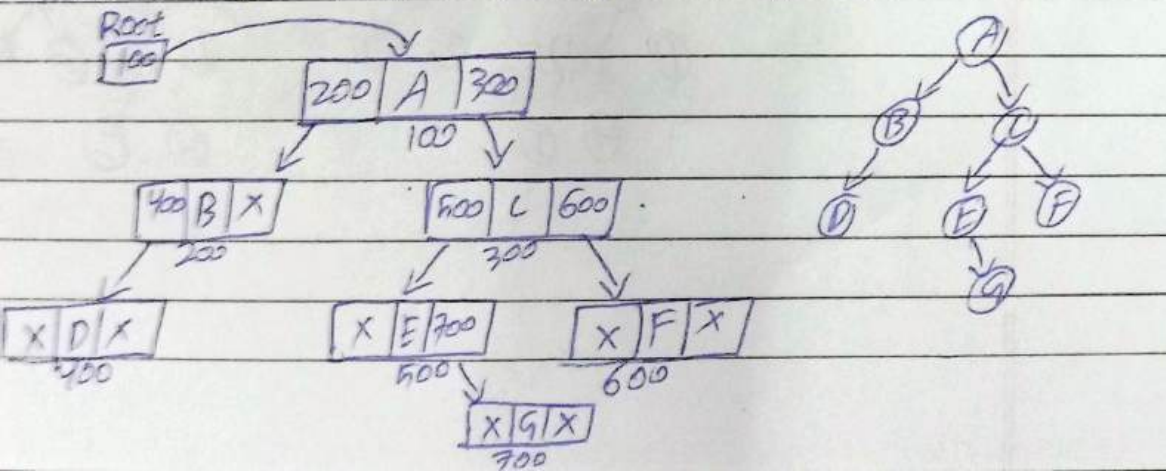
Linked representation :-

⇒ A binary tree can be stored using a collection of nodes.

A node contains :- ① Info
② Left pointer
③ Right pointer



Consider a binary tree to represent in linked form.



4 write down the algorithm for Inorder traversal for a binary tree. when an inorder and preorder sequence of nodes given then how to construct a binary tree.

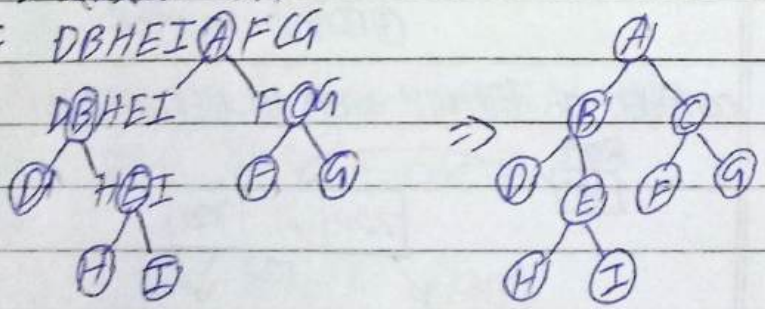
Ans Inorder traversal for a binary algorithm :-

Inorder(node)
step 1: start
step 2: if (node ≠ NULL), then
    step 2.1 : Inorder (left [node])
    step 2.2 : process node
    step 2.3 : Inorder (right [node])
[end of if]
step 3 : stop.

Construct of a binary tree from inorder and Preorder sequence :-

① 1st node of the PREORDER traversal is the root node of the tree.
② Find the position of the root node in the INORDER traversal.
③ Nodes preceding the root in the INORDER construct the left subtree
④ Nodes succeeding the root in the INORDER construct the right subtree
⑤ Find out the next roots for each sub trees using PREORDER.
⑥ Continue the above process until leaf nodes are found.

Ex:- Preorder : ~~ABDEHICFG~~
Inorder : DBHEIAFCG



5 How to construct a binary tree when Inorder and post order sequence of node given? Also write down the algorithm for preorder traversal of a binary tree.

Ans Constructing binary tree using postorder and inorder traversal:-

① Here, last node in the POSTORDER traversal is considered as root node of the tree.

② Find the position of the root node in the INORDER traversal.

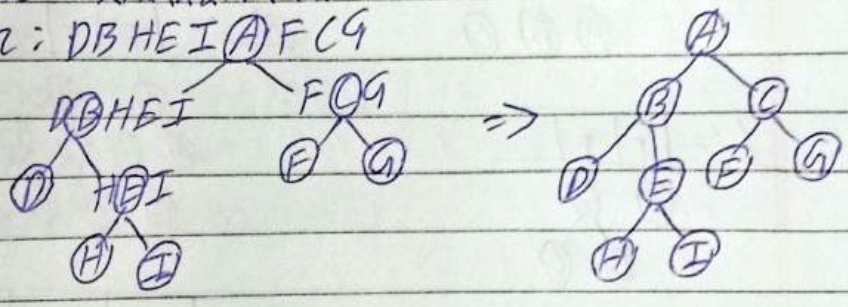
ⓐ Nodes Preceding the root in the INORDER construct the left subtree.

ⓑ Nodes succeeding the root in the INORDER construct the right subtree.

③ Find out the next roots for each sub tree using POSTORDER.

Ex:- Postorder : ~~RAAAEBFGCX~~

Inorder : DB HE I (A) F C G



Preorder traversal for a binary tree:-

Preorder (node)
Step 1: Start
Step 2: if (node ≠ NULL), then
  Step 2.1: Process node
  Step 2.2: Preorder (left [node])
  Step 2.3: preorder (right [node])
[End of if]
Step 3: Stop.

(6) Given an infix expression (A+B/C-D^E*F-G) Find its equivalent postfix notation and then using stack construction an expression tree.

Ans Given infix expression:

A+B/C-D^E*F-G

= A+B/C-[DE^]*F-G

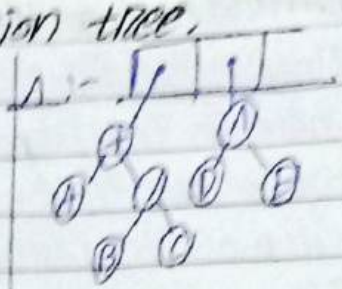
= A+B/C-[DE^F*]-G

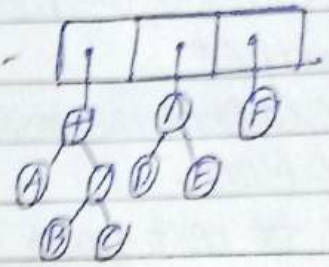
= A+[BC/]-[DE^F*]-G

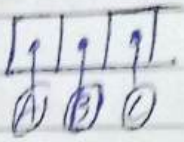
= [ABC/+]-[DE^F*]-G

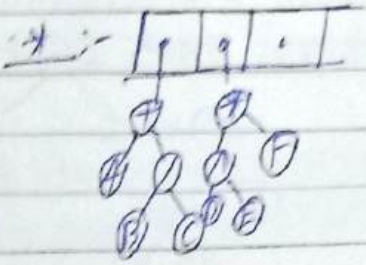
= [ABC/+DE^F*-]-G

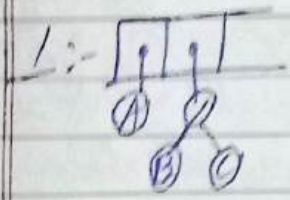
= ABC/+DE^F*G-

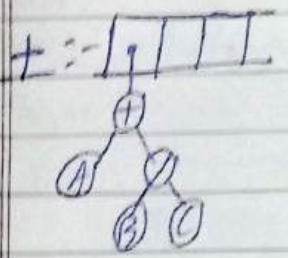


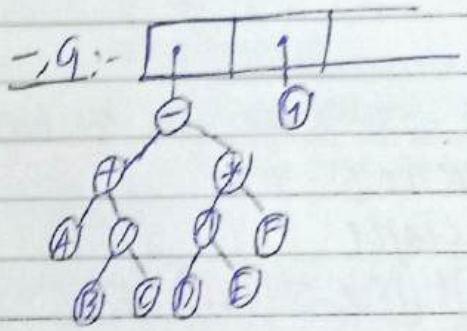


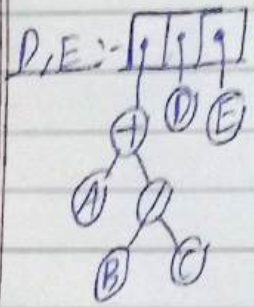


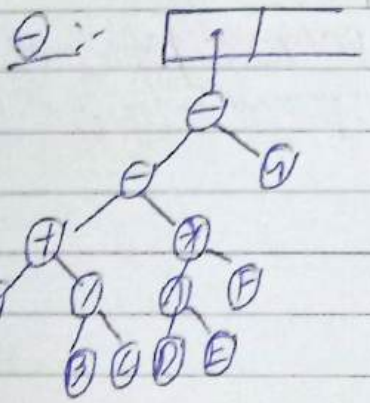










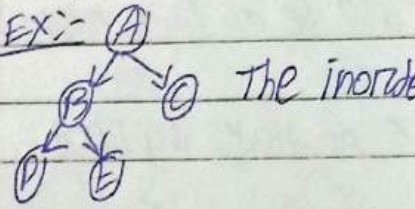


Q) write down all the 3 different recursive traversal methods for traversing nodes of a binary tree.

Ans The 3 different recursive traversal methods for traversing nodes of a binary tree are:-

(i) Inorder traversal method:-

Here, the order of traversing the nodes will be left, root, right order.

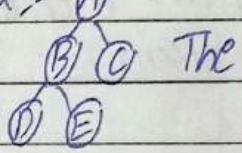
Ex:- 

The inorder sequence will be : DBEAC

Algorithm:-

Inorder(node)
Step 1: Start
Step 2: if (node ≠ NULL), then
  Step 2.1: Inorder (left node)
  Step 2.2: Process node
  Step 2.3: Inorder (right node)
[End of if]
Step 3: Stop.

(ii) Preorder Traversal method:-

=> Here, traversal of node will be in the sequence of root, left, right order. Ex:- 

The preorder sequence of node = ABDEC

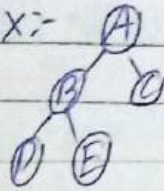
Algorithm:-

Preorder(node)
Step 1: Start
Step 2: if (node ≠ NULL), then
  Step 2.1: Process node.

Step 2.2 : Preorder (left[node])
Step 2.3 : Preorder (right[node])
[end of if]
step 3 : Stop.

(iii) Postorder Traversal method :-

Here, traversal sequence of nodes will be as left, right, root order. EX :-



The sequence of nodes : DEBCA.

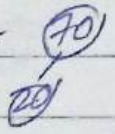
Algorithm :-

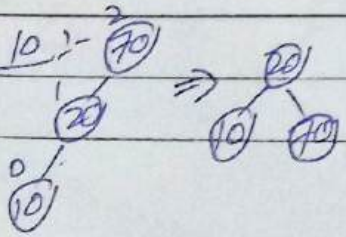
postorder (node)
Step 1 : Start
step 2 : if (node ≠ NULL), then
  step 2.1 : Postorder (left[node])
  step 2.2 : Postorder (right[node])
  step 2.3 : Process node.
[End of if]
step 3 : Stop.

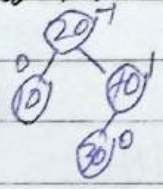
8 Given a sequence of elements : 70, 20, 10, 30, 40, 50, 60, 25, 35, 45, 75, 95, 52, 92. Construct an AVL tree by inserting each element.

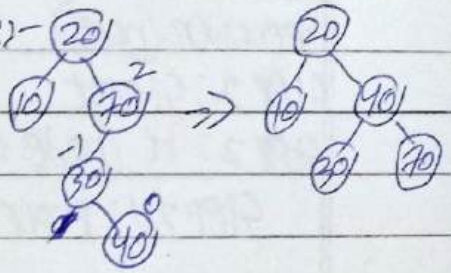
Ans Insert 70 :- 

← Pivot

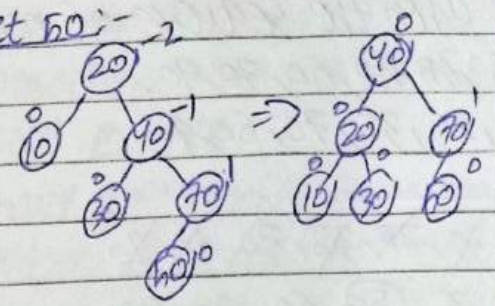
Insert 20 :- 

Insert 10 :- 

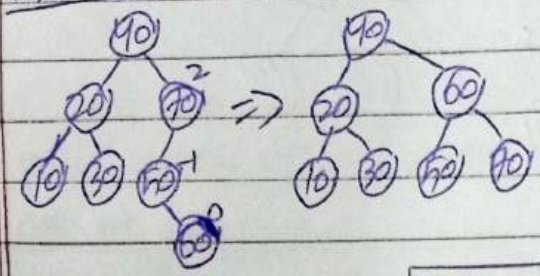
Insert 30 :- 

Insert 40 :- 

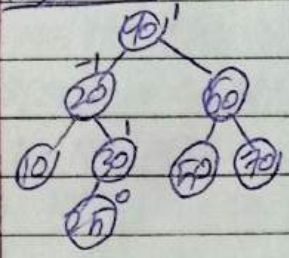
Insert 50 :-



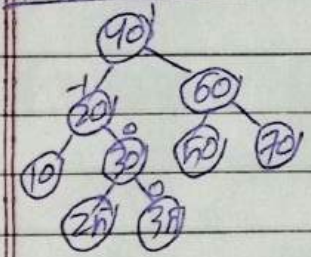
Insert 60 :-



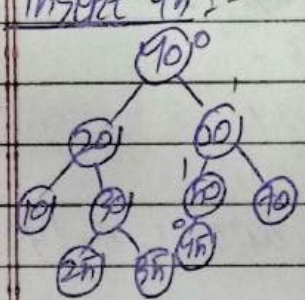
Insert 25 :-



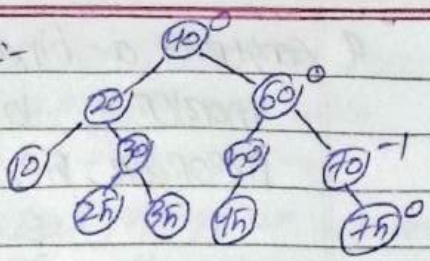
Insert 35 :-



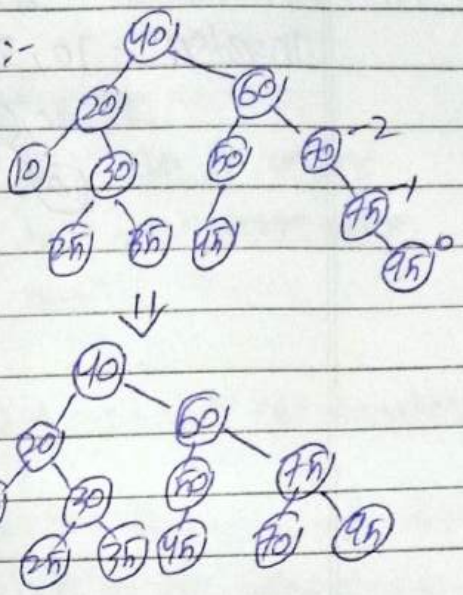
Insert 45 :-



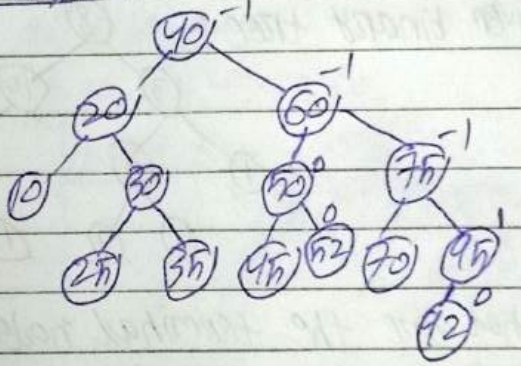
Insert 75 :-



Insert 95 :-



Insert 52, 92 :-



Date
Page

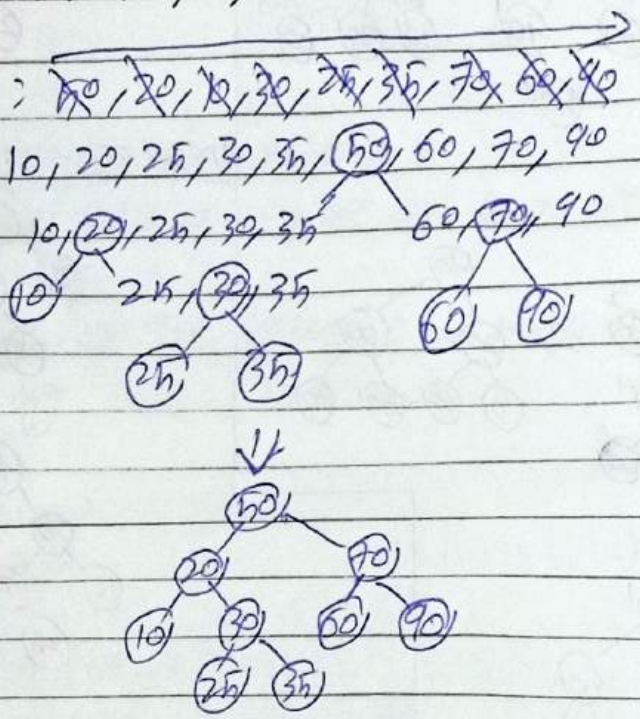
9 Construct a binary tree when the sequence of nodes given as.

Inorder : 10, 20, 25, 30, 35, 50, 60, 70, 90.

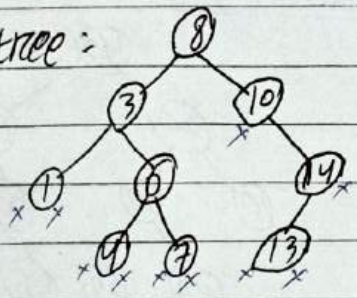
Preorder : 50, 20, 10, 30, 25, 35, 70, 60, 90.

Ans preorder : 50, 20, 10, 30, 25, 35, 70, 60, 90

Inorder :- 10, 20, 25, 30, 35, 50, 60, 70, 90



10 Given binary tree :



what are the terminal nodes?

what are the internal nodes?

How many NULL pointers exist?

What are the nodes available at level 2?

Ans The terminal nodes are 1, 4, 7, 13.

The internal nodes are 1, 3, 6, 10, 14.

There are 10 NULL pointers exists.

The nodes available at level 2 are 1, 6, 14.

Assignment - V

1 What is Heap Tree and its types? Explain the Procedure for construction of max heap Tree and min Heap Tree both with suitable Example.

Ans A binary tree in which from root node to leaf node if we can get an ordered list of elements then it is called a heap tree.

There are 2 types of heap tree :-

① max Heap Tree (or) descending Heap Tree

A binary tree in which from root node to leaf node, if we get descending order of elements then it is called max heap tree. Here, every Parent node is greater than child node.

construction :-

→ Insert elements one by one into the tree by maintain the complete binary tree property.

→ If the inserted element is greater than its Parent, swap them.

→ Continue swapping until the root is reached or the heap property is restored.

→ Repeat this process for all elements.

Ex :- Construct a max heap tree :- 10, 20, 40, 30, 80.

Insert 10 :- (10)

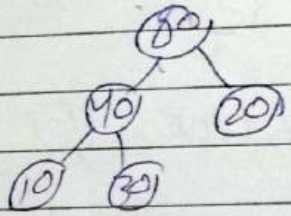
Insert 20 :- 20, 10

Insert 40 :- 40, 20, 10, 20

Insert 30 :- 40, 30, 20, 10

Insert 80 :-

∴ The final max heap tree is



(11) Min heap tree (or) Ascending Heap tree:-

A binary tree in which from root node to leaf node, if we get ascending order of elements, then it is called min heap tree. Here, every parent is smaller than both child node and root node contains smallest element.

Construction:-

→ Insert elements one by one into the tree by maintaining the complete binary tree property.

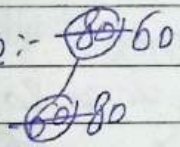
→ If the inserted element is smaller than its parent, swap them.

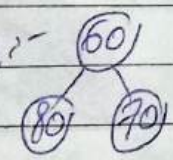
→ Continue swapping until the root is reached or the heap property is restored.

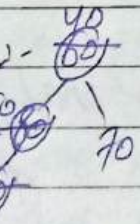
→ Repeat this process for all elements.

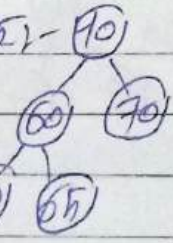
Ex:- Construction of a min heap tree :- 80, 60, 70, 40, 65, 35

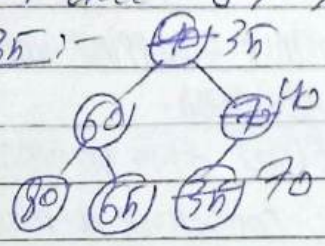
Insert 80:- 

Insert 60:- 

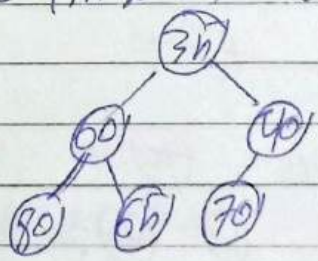
Insert 70:- 

Insert 40:- 

Insert 65:- 

Insert 35:- 

∴ The final min heap tree is



2 write down the process of mid square method and folding method in Hash functions.

Ans Mid square method:-

Here, we can apply hash function to find the middle value of k^2 where k is the key element

① Find k^2

② Identify middle value of k^2 as hash address.

Consider an example:-

Say there are 20 key elements for storage, some of them are: 27, 37, 43, 78, 92

key	mid square method	Address
27	$(27)^2 = 0729$	72
37	$(37)^2 = 1369$	36
43	$(43)^2 = 1843$	84
78	$(78)^2 = 6084$	08
92	$(92)^2 = 8464$	46

hash table:-

Address	key
08	78
36	37
46	92
72	27
84	43

Folding method:-

→ In this method we need to divide a key into number of partitions and then add them to identify address. $H(k) = k_1 + k_2 + k_3 \cdots k_n$

Example:- Consider 30 numbers for storage, some of them are:- 1234, 2345, 4567, 6789. Here we consider partition of 2 digits each.

key	Folding method	Address
1234	12+34 = 46	46
2345	23+45 = 68	68
4567	45+67 = 112	12
6789	67+89 = 156	56

hash table:-

Address	key
12	4567
46	1234
56	6789
68	2345

③ what is hash function? use the division method to generate hash addresses for the given key values : 10, 19, 35, 43, 62, 59, 31, 49, 77, 33

Ans It is a searching technique whose time complexity is O(1). Here to search for any value the time taken is constant. Hashing is the most effective technique and it is not dependent on N.

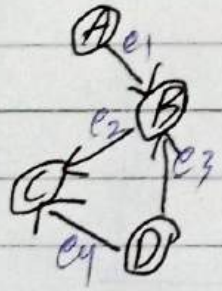
10, 19, 35, 43, 62, 59, 31, 49, 77, 33 N=10 m=11

key (k)	Division method H(k)=k%.m	generated Address
10	10%.11 = 10	10
19	19%.11 = 8	8
35	35%.11 = 2	2
43	43%.11 = 10	10
62	62%.11 = 7	7
59	59%.11 = 4	4
31	31%.11 = 9	9
49	49%.11 = 5	5
77	77%.11 = 0	0
33	33%.11 = 0	0

Hashtable :-

Address	key
0	77, 33
1	
2	35
3	
4	59
5	49
6	
7	62
8	19
9	31
10	10, 43

④ what is Path matrix? write down adjacency matrix and incidence matrix of the following graph. Also represent the linked concept for memory representation of below graph.



Ans A matrix which contain information about there exist a path or not between every pair of vertices is called path matrix.

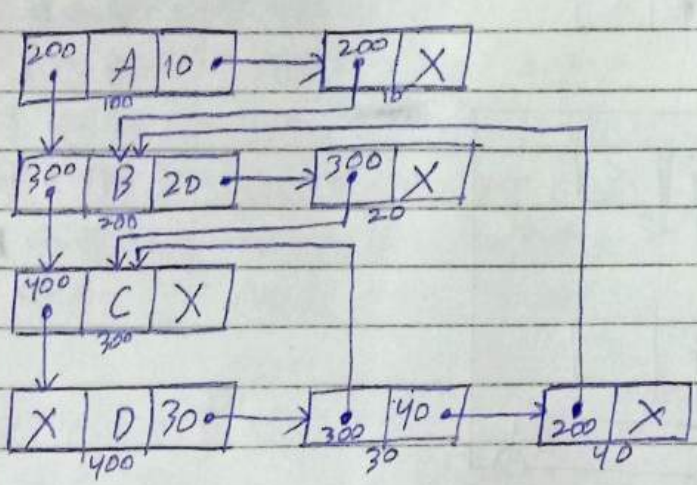
Adjacency matrix :-

	A	B	C	D
A	0	1	0	0
B	0	0	1	0
C	0	0	0	0
D	0	1	1	0

Incidence matrix :-

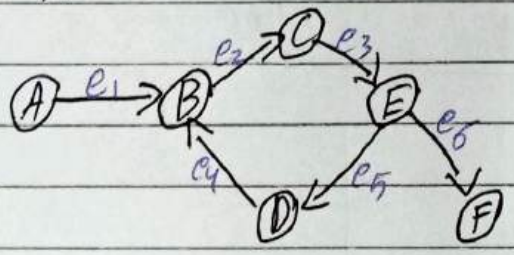
	e_1	e_2	e_3	e_4
A	1	0	0	0
B	-1	1	-1	0
C	0	-1	0	-1
D	0	0	1	1

Linked representation :-



Adjacency Table
A → B
B → C
C →
D → C, B

Q Define the Path matrix ; state the difference by writing down adjacency matrix and incidence matrix of the follow graph. Also represent the linked concept for memory representation of below graph.



Ans A matrix which contain information about there exist a path or not between every pair of vertices is called path matrix.

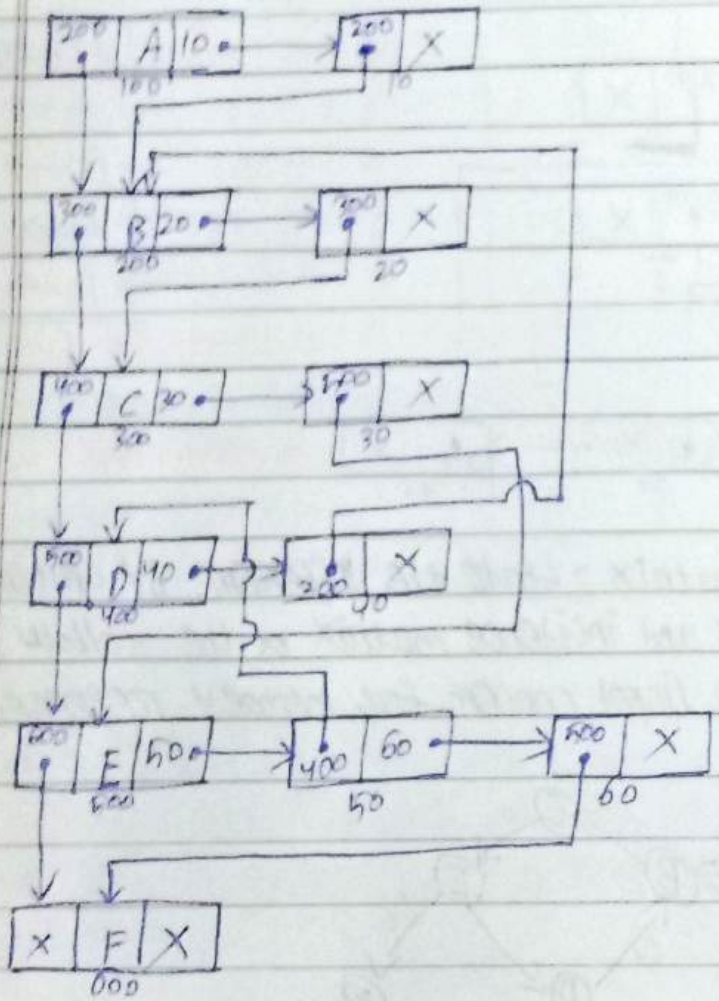
Adjacency matrix :-

	A	B	C	D	E	F
A	0	1	0	0	0	0
B	0	0	1	0	0	0
C	0	0	0	0	1	0
D	0	1	0	0	0	0
E	0	0	0	1	0	1
F	0	0	0	0	0	0

Incidence matrix :-

	e_1	e_2	e_3	e_4	e_5	e_6
A	1	0	0	0	0	0
B	-1	1	0	-1	0	0
C	0	-1	1	0	0	0
D	0	0	0	1	-1	0
E	0	0	-1	0	1	1
F	0	0	0	0	0	-1

Linked representation :-



Adjacency Table

A → B

B → C

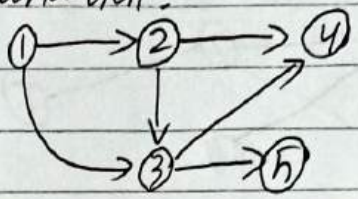
C → E

D → B

E → D, F

F →

⑥ write the algorithm for BFS on a directed graph and write down the explanation.



Ans In this approach we can traverse all the nodes of a given graph using a queue. Here, a graph need to be connected one.

Algorithm:=

step 1: start

step 2: Initialize all the nodes to ready state

step 3: consider a node x as 1st node and insert into queue.

step 4: change the state of node x into waiting state.

step 5: while (queue is not empty)

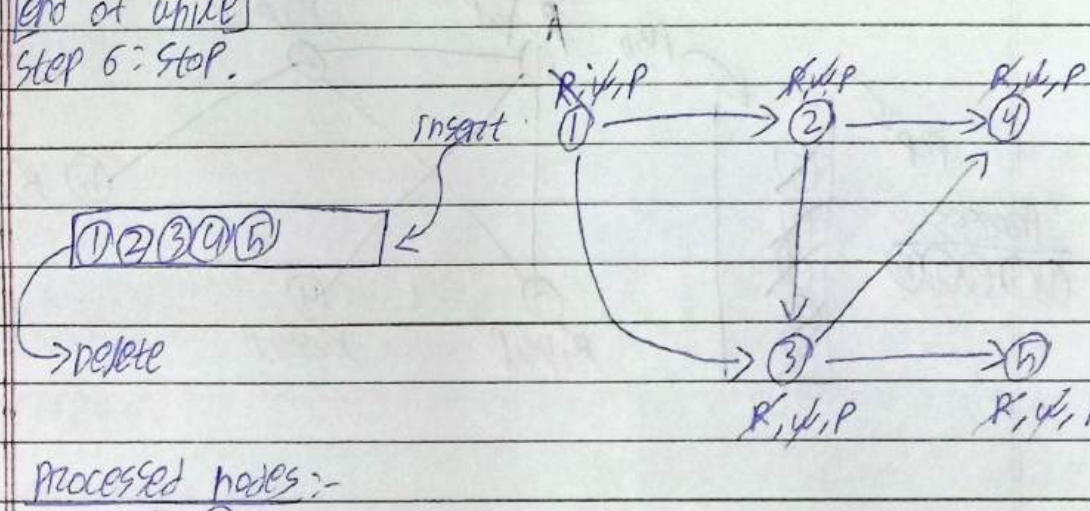
step 5.1: delete a node x from queue and process it

step 5.2: change the state of node x into a processed state

step 5.3: for each adjacent node y of node x which is in ready state; insert into queue and change its state to waiting state.

[end of while]

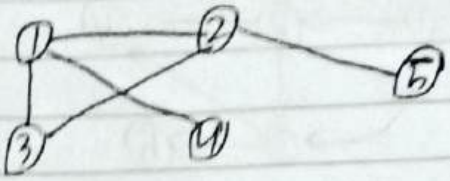
step 6: stop.



Processed nodes:-

1 2 3 4 5

④ write the algorithm for depth first search of a graph. write down the DFS order of the graph given below.



Ans Algorithm:-

Step 1: Start

Step 2: Initialize all the nodes to ready state

Step 3: Consider a node as 1st node and push into stack

Step 4: Change state of 1st node into waiting state

Step 5: while (Stack is not empty)

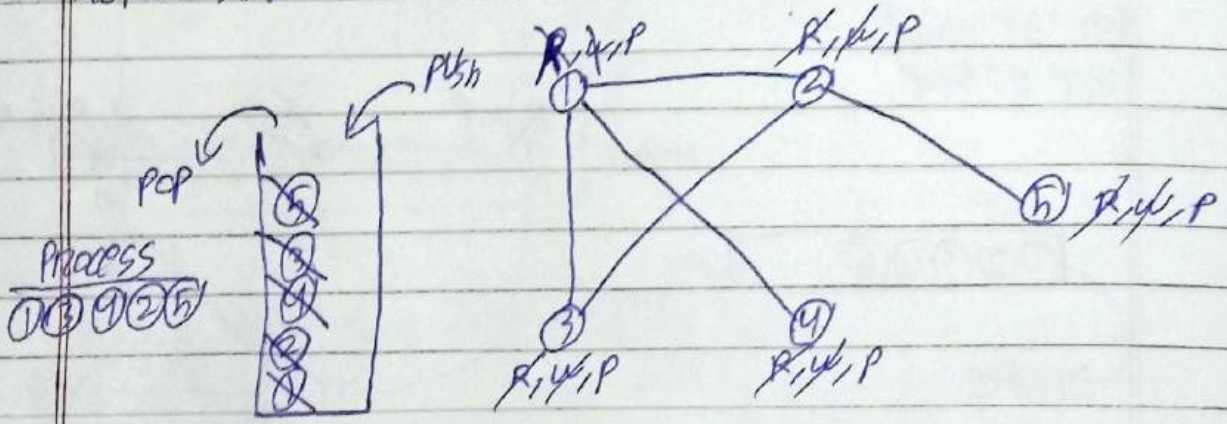
Step 5.1: pop a node (x) and process it

Step 5.2: change state of node (x) into processed state.

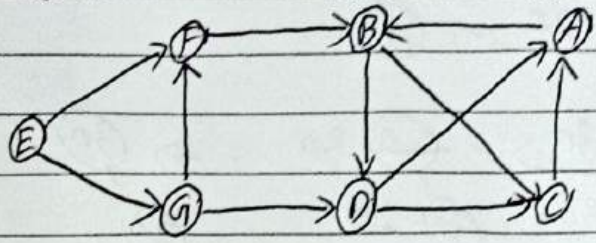
Step 5.3: For each adjacent node (x) which are in ready state, push into stack and change their state to waiting state.

End of while

Step 6: Stop.

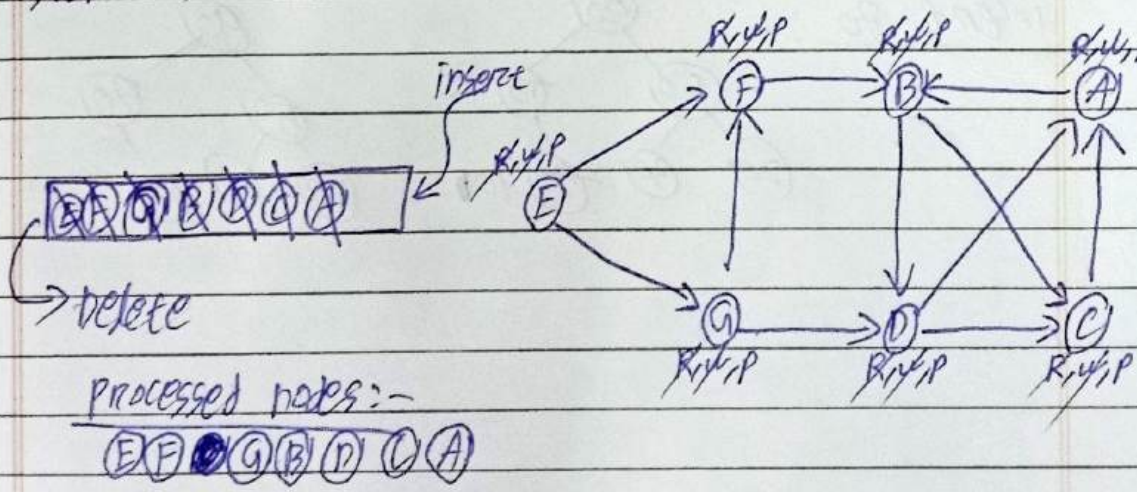


(8) write the algorithm for implementing Breadth First search in a graph. write down BFS order for the following graph.



Ans Algorithm :-

Step 1: start
step 2: initialize all the nodes to ready state
step 3: consider a node x as 1st node and insert into queue
step 4: change the state of node x into waiting state.
step 5: while (queue is not empty)
    step 5.1: delete a node x from queue and process it
    step 5.2: change the state of node x into a processed state.
    step 5.3: for each adjacent node y of node x which is in ready state; insert into queue and change its state to waiting state
[end of while]
step 6: stop.

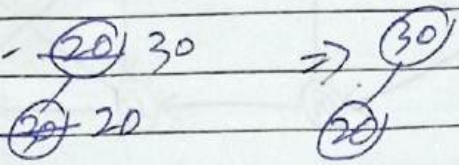


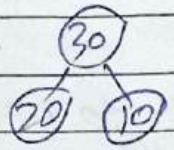
Processed nodes :-

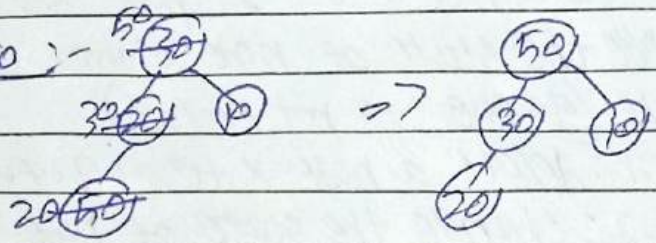
E F G B D C A

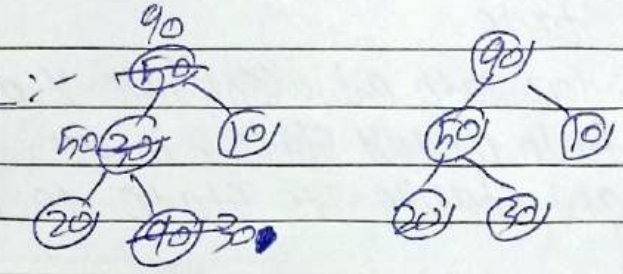
Q construct a Max-heap tree for a given list of elements: 20, 30, 10, 50, 90, 70.

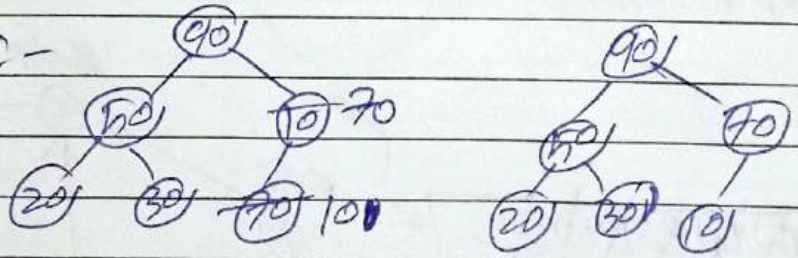
Ans Insert 20 :- 20

Insert 30 :- 

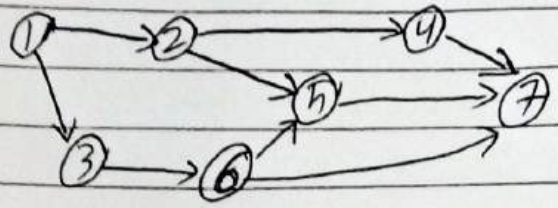
Insert 10 :- 

Insert 50 :- 

Insert 90 :- 

Insert 70 :- 

⑩ Find the in-degree and out-degree of each vertex of the graph given below:



Ans

Node 1 :-
In-order = 0
out-order = 2

Node 2 :-
In-order = 1
out-order = 2

Node 3 :-
In-order = 1
out-order = 1

Node 4 :-
In-order = 1
out-order = 1

Node 5 :-
In-order = 2
out-order = 1

Node 6 :-
In-order = 1
out-order = 2

Node 7 :-
In-order = 3
out-order = 0