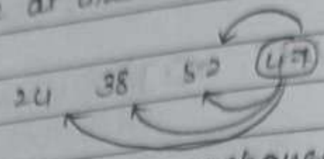
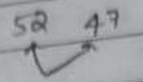
ele at index 3 will be compared with 0,1,2

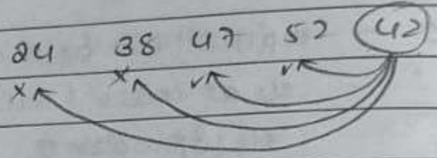




exchange

Pass 3 - 24 38 47 52 42

ele in index 4 will be compared with all the element



*5 elements

5-1 = 4 pass ← pass 4 - 24 38 42 47 52 Sorted//

function.

```
void insertion_sort ( int a[] , int n )
{
    int pass , j , key;
    for ( pass =1 ; pass<n ; pass++)
    {
        key = a[pass];          more condition
        for(j=pass-1 ; j>=0 && key<a[j] ; j--)
        {                          compare
            a[j+1] = a[j] } shifting values.
        }
        a[j+1]=key;
    }
}
```

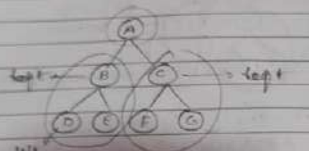
```
    while(i< = mid)
    {
      b[k] = a[i];
      i++;
      k++;
    }
   while(j< = high)
   {
     b[k] - a[j];
     j++;
     k++;
   }
  for (i=0 ; i<n ; i++)
     a[i] = b[i]
}
```

All
9.42 - 1
9.41 - 2
9.44 - 3
9.45 - 4

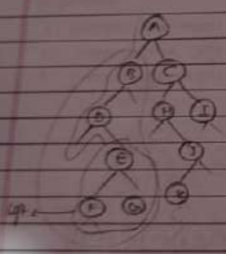
...ate
...preue...
...duca...
90...

02 Inorder

Step 1- process left sub tree in inorder.
2. process the root node
3. process the right subtree in inorder.



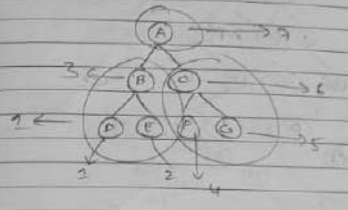
root
DBEAFCG
root root



FEGDBAHKJCI

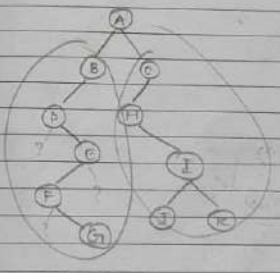
03 Postorder.

step 1 - process left
process right
process the root node



DEBFGCA

ex -

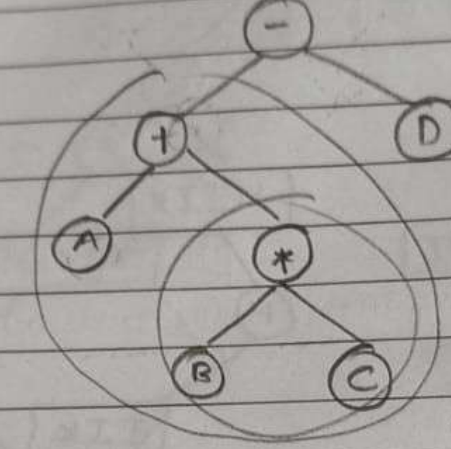


GFEDBJKIHCA

Construct the binary tree for the arithmetic expression

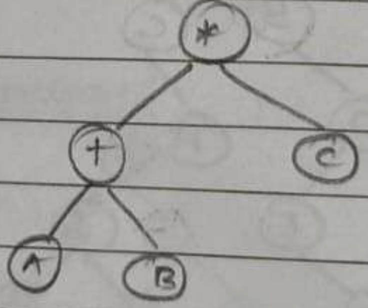
01. A + B * C - D go from bottom to up

highest p



postorder - A B C * + D -

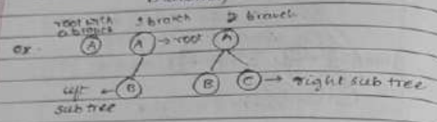
02. (A+B) * C



postfix - AB +C *

All
9.42 - I
9.41 - 2
9.44 - 3
9.45 - 4
9.42 - 6

Binary tree (can have a root with 0, 1 or many 2 branches)

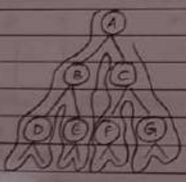


Binary tree traversal
- → pre order
- → in order
- → post order.

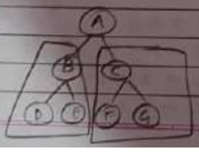
1. Pre order

algorithm
1. process the root node
2. process the left subtree in preorder
3. process the right subtree in preorder.

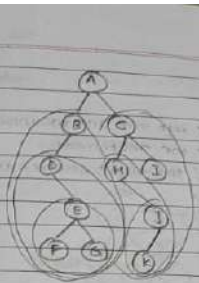
Q1.



ABDECFG
↓
pre order.

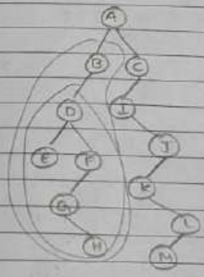


Q2.



= ABDEFGCHIJKI //

Q3.



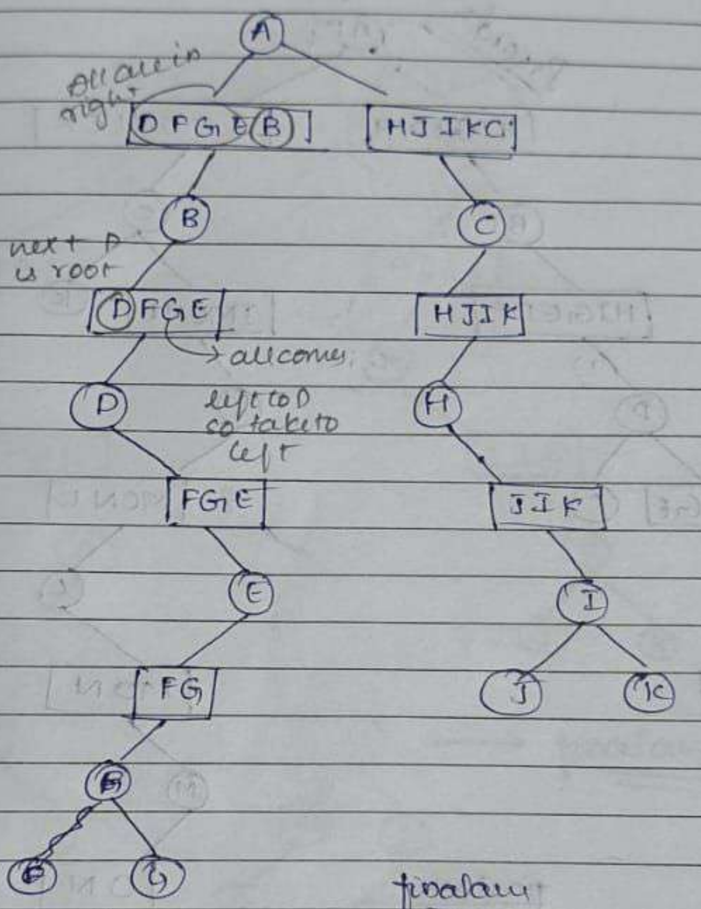
ABDEFG HC IJKLM

root

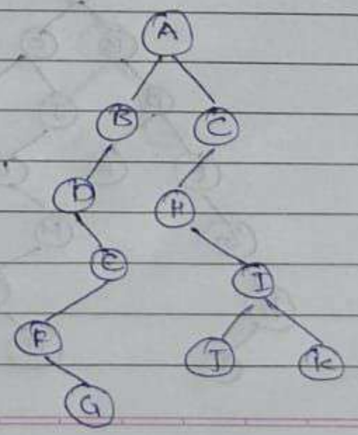
Preorder - A B D E F G C H I J K

Inordo D F G E B A H J I K C

B in righ



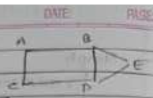
final ans



All
9.42 - 1
9.41 - 2
9.44 - 3
9.45 - 4

9.42 - 6

Adjacency linked list



A → B → C null

B → A → D → E null

C → A → D null

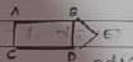
D → B → C → E null

E → B → D null

Graph traversal (visiting a node

- → DFS → stack
- → BFS → Queue

Depth first search

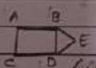


adjacency

traversal → ABDCE

A → B, C
B → D, E
D → ~~C~~, E → remov
C → A, ~~D~~

Breadth first search.

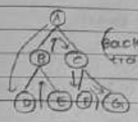


not adjacency ? required

(visit) A → B, C
B → D, E
C → A, D
D → ~~B~~, ~~B~~, E

queue
traversal: ABCDE

01. DFS

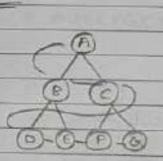


(Back tracking)

A → BC
B → DE
C → FG

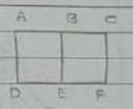
ABDECFG.

BFS



ABCDEFG

02. DFS



ABCFEDF

BFS A → BD

ABDCEF

A → B D
B → ~~A~~ C E
D → ~~A~~ ~~E~~
C → ~~B~~ F

A → BD
B → ~~A~~ C E
C → ~~B~~ F
F → ~~C~~ E
E → ~~B~~ D F
D → ~~A~~ ~~E~~

DFS

just put node left & right it is not binary search tree.

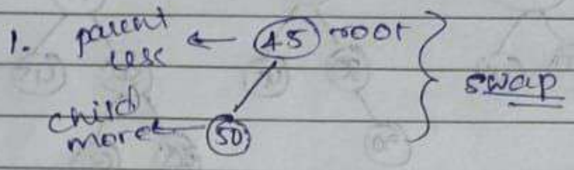
heap → complete binary tree.

- → maximum heap → roots will having max value compared to child
- → minimum heap → roots will having less value compared to child.

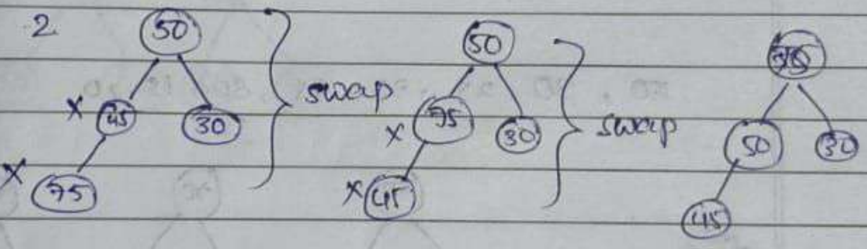
1. maximum heap

ex 45 ,50 ,30 ,75 , 60, 80 , 70

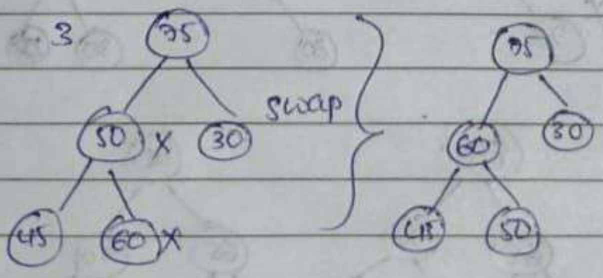
1.



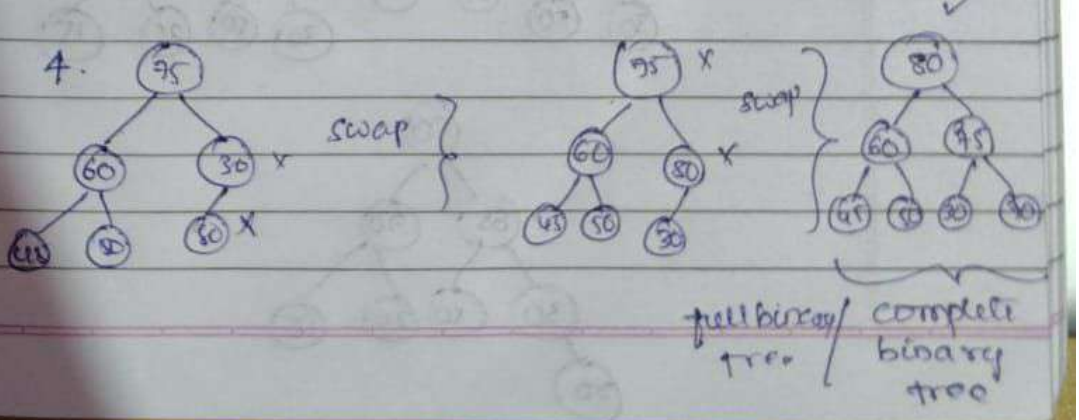
2

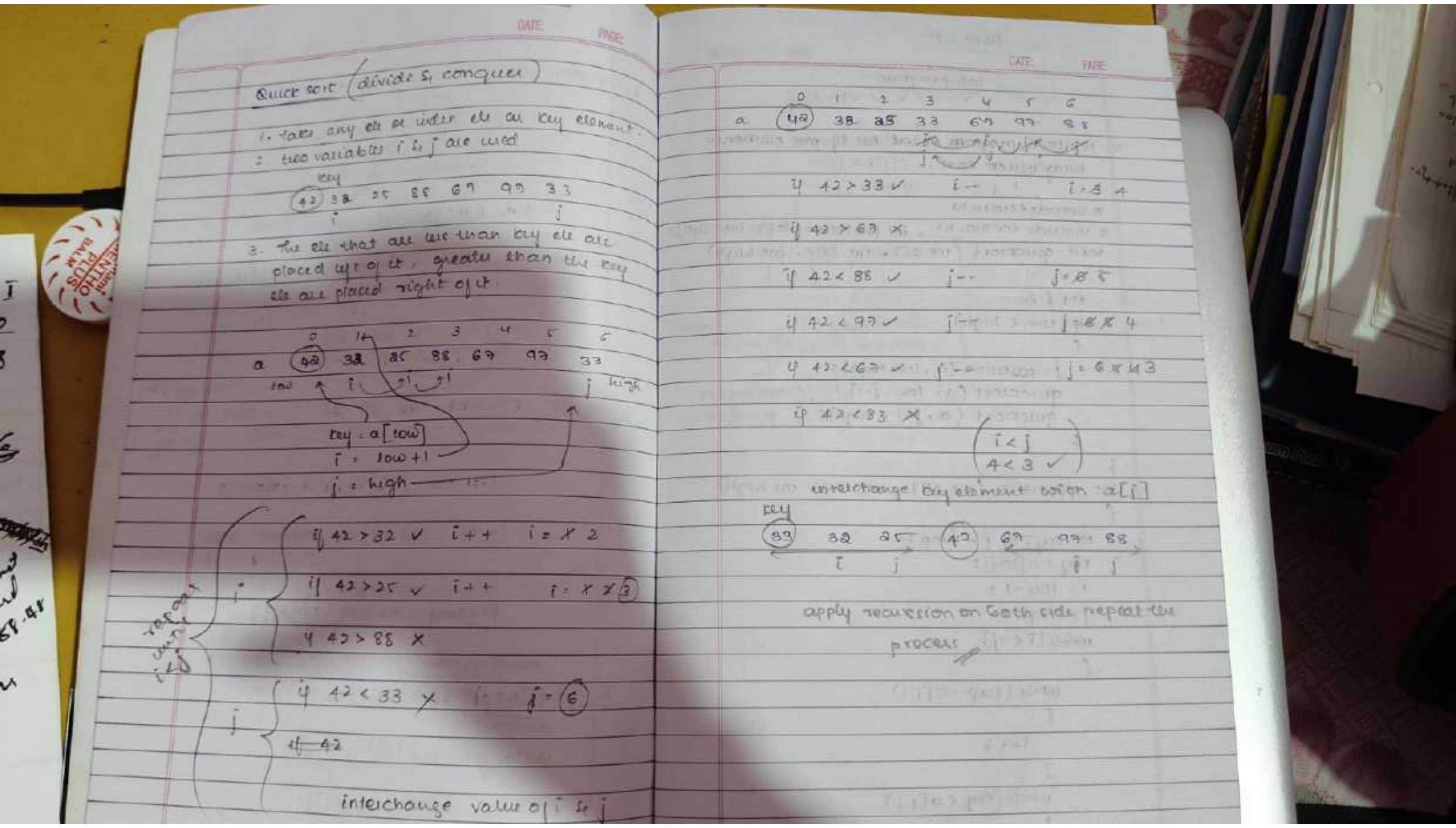


3



4.





Quick sort (divide & conquer)

1. take any ele or index ele as key element.
2. two variables i & j are used

key
$\boxed{42}$ 32 25 88 67 97 33
i j

3. The ele that are less than key ele are placed left of it, greater than the key ele are placed right of it.

	0	1	2	3	4	5	6
a	42	32	25	88	67	97	33
	low	i	i	i			j high

key : a[low]
i : low+1
j : high

repeat until i<j

i
- if 42 > 32 ✓ i++ i = ~~1~~ 2
- if 42 > 25 ✓ i++ i = ~~1~~ ~~2~~ $\boxed{3}$
- if 42 > 88 ✗

j
- if 42 < 33 ✗ j = 6
- ~~if 42~~

interchange value of i & j

	0	1	2	3	4	5	6
a	42	32	25	33	67	97	88
				j	i		

if 42 > 33 ✓ i++ i = ~~3~~ 4

if 42 > 67 ✗

if 42 < 88 ✓ j-- j = ~~6~~ 5

if 42 < 97 ✓ j-- j = ~~6~~ ~~5~~ 4

if 42 < 67 ✓ j-- j = ~~6~~ ~~5~~ ~~4~~ 3

if 42 < 33 ✗

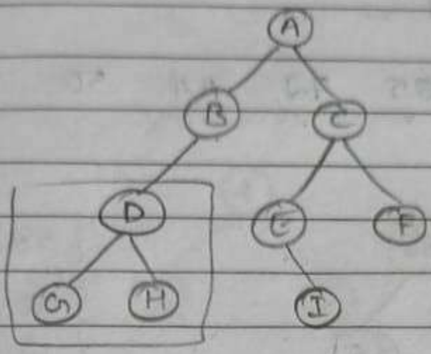
(i < j
4 < 3 ✓)

interchange key element with a[j]

key
33 32 25 42 67 97 88
i j i j

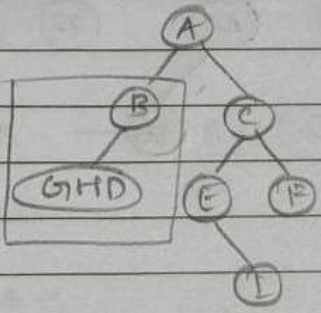
apply recursion on both side repeat the process

Postorder

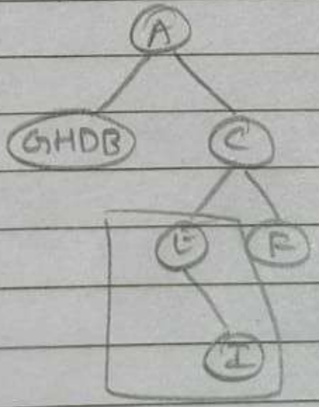


Postorder - lt rg root

GHD

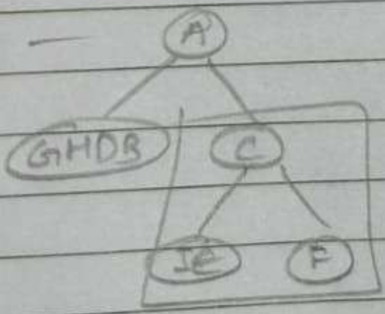


GHDB

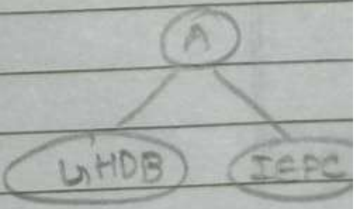


IC

IC —



IeFC



postor - GHDBIEFCA

*Tree - acyclic graph

non linear data structu

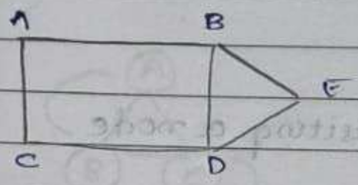
↳ tree
↳ graph

Graph
↳ adjacency matrix } representing graph
↳ adjacency linked list } in memory.

G = (V, E)

a graph with 0 edges are called null graph.
where E = {Φ}

. . } no edges → nullgraph
. .

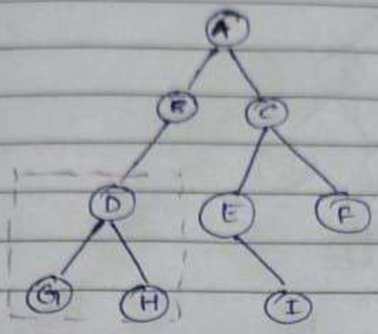


nxn vertices

	A	B	C	D	E	degree
A	0	1	1	0	0	A = 2
B	1	0	0	1	1	B = 3
C	1	0	0	1	0	C = 2
D	0	1	1	0	1	D = 3
E	0	1	0	1	0	E = 2 //

no. of edge dependid on vertex.

degree of vertex A = 2
" B = 3
" C = 2
" D = 3
" E = 2



Pre - r L t

~~AGDGHBIEFC~~

Inorder - L r t

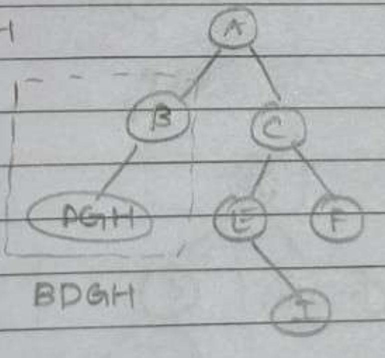
~~GDHBAEIFCFA~~

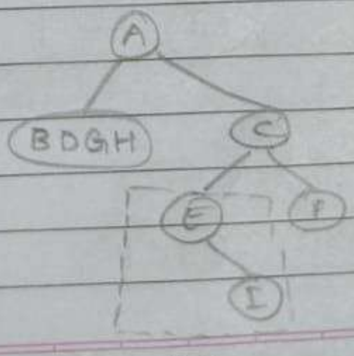
Postorder - L t r

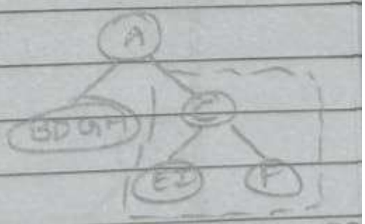
→ R L Rg

Preorder → ABDGHCEIF

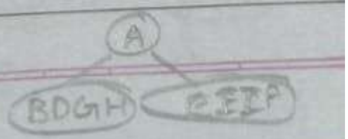
DGH



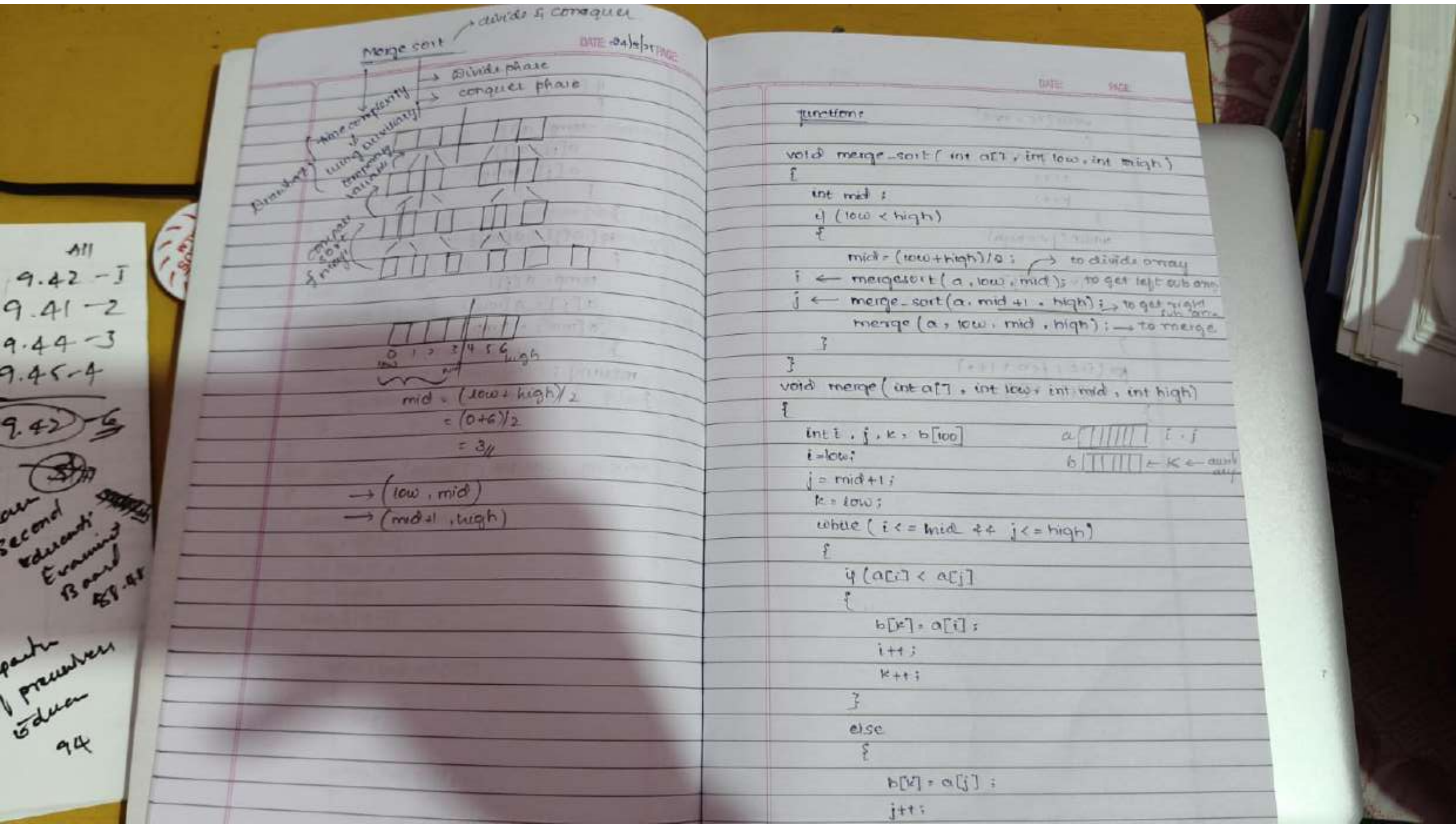




BICAP CEIF



EI



All
9.42 - 1
9.41 - 2
9.44 - 3
9.45 - 4
9.42 - 6

Second Education Examination Board 58.48

Edu 94

Merge sort → divide & conquer

DATE: 04/8/21

→ Divide phase
→ conquer phase

Drawback { time complexity & using auxiliary temporary array

combine sort & merge

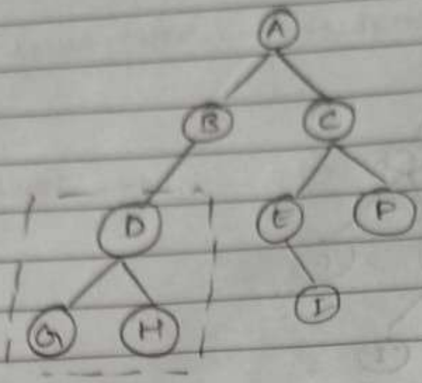
0 1 2 3 | 4 5 6
low mid high

mid = (low + high)/2
= (0+6)/2
= 3

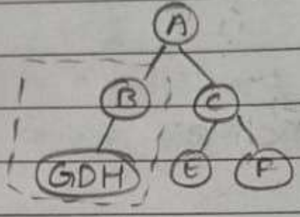
→ (low , mid)
→ (mid+1 , high)

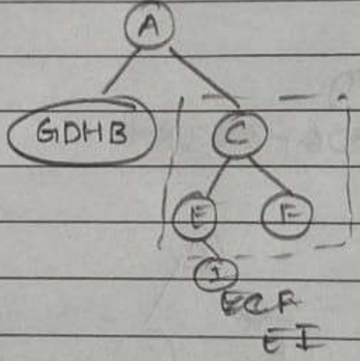
function:

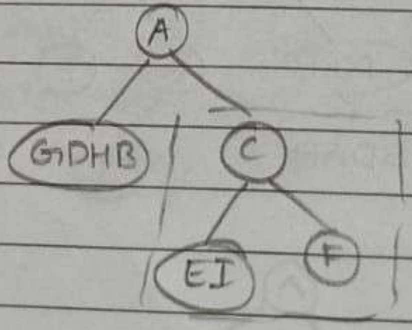
```
void merge_sort( int a[] , int low , int high )
{
   int mid ;
   if (low < high)
   {
      mid = (low+high)/2 ;   → to divide array
 i ← mergesort( a , low , mid );   to get left sub array
 j ← merge_sort(a , mid +1 , high) ;  → to get right sub arr.
      merge (a , low , mid , high) ; → to merge
   }
}
void merge( int a[] , int low , int mid , int high)
{
   int i , j , k , b[100]          a [ | | | | | | ]  i , j
   i=low;                           b [ | | | | | | ] ← K ← auxiliary
   j = mid+1 ;
   k = low ;
   while ( i <= mid && j <= high)
   {
      if (a[i] < a[j]
      {
         b[k] = a[i] ;
         i++ ;
         k++ ;
      }
      else
       {
          b[k] = a[j] ;
          j++ ;
```

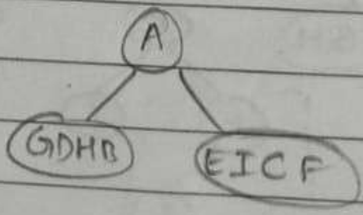


inorder = ~~R~~ Lt r rig









In - GDHBAEICF

tion sort

```
    printf(" sorted array elements are : \n");
    for (i=0; i<n; i++)
    {
        printf("%4d", a[i]);
    }
   getch();
  }
```

19/03/25 Selection sort

Insertion sort ★ (Number of pass = n - 1)

pass 0 - kept 1st ele as it is

Pass 1 - ele at index 1 will be compared with ele of index 0.

ex - 38 52 24 47 42

0 1

compared, so no exchange.

Pass 1 38 52 24 47 42

ele at index 2 will be compared with index 0 & 1.

a[0] a[1] a[2]

ex - 38 52 24 47 42

54 24 24 is smaller so exchange

38 24 24 is smaller so exchange

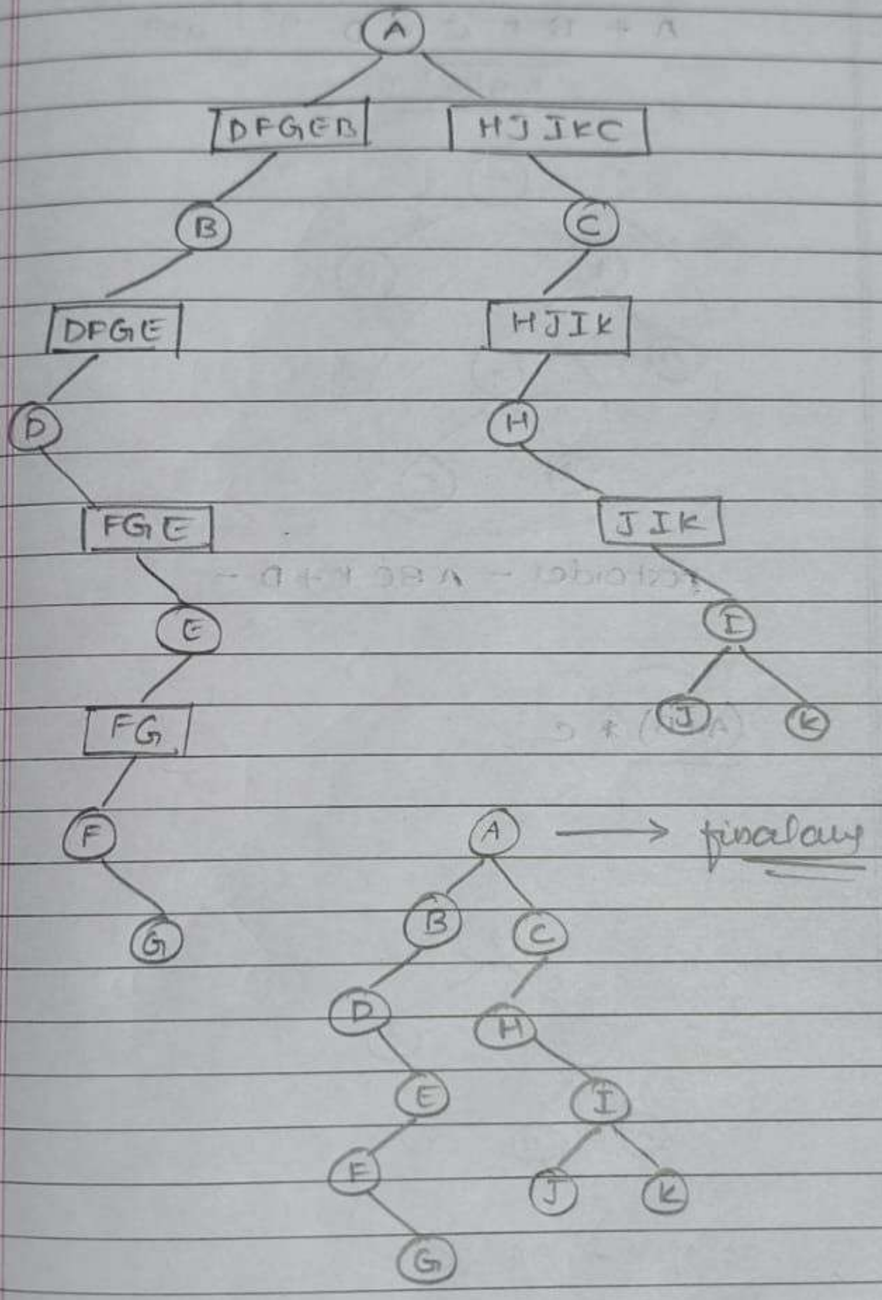
38 52 24 47 42

pass 2 - 24 38 52 47 42

come from the PAGE

Postorder - G F E D B J K I H C A → rootnode

Inorder - D F G E B A H J I K C


A
DFGEB
HJIKC
B
C
DFGE
HJIK
D
H
FGE
JIK
E
I
FG
J
K
F
G
A
B
C
D
H
E
I
F
J
K
G
finalans.