Date 20/08/19

PAGE NO.: 01
DATE: / /

DATA STRUCTURE (DS) & Algorithm Using "C"

Q:- What is Data structure?

→ The logical and mathematical model of data organisation is called data structure.
It is a branch of "Computer Science" in which we study the various ways/methods to organise data into memory and the method to access them. The data into memory should be organised in such a manner that they can be accessed quickly at the time of processing.
In some processing environment, data must be organised in some specific form.

Categories of Data structure
(I) Linear Data structure
(II) Non-Linear Data structure

(I) Linear Data structure :- A kind of data structure which organises the data items in ordered manner is called linear data structure.

Teacher Signature

Examples:
- Array
- Linked-List
- Stack
- Queue

(ii) Non-Linear Data Structure:- A kind of data structure that organises the data items in unordered manner is called non-linear data structure.
Examples:
- Tree
- Graph

* Operations that can be performed on any Data structure

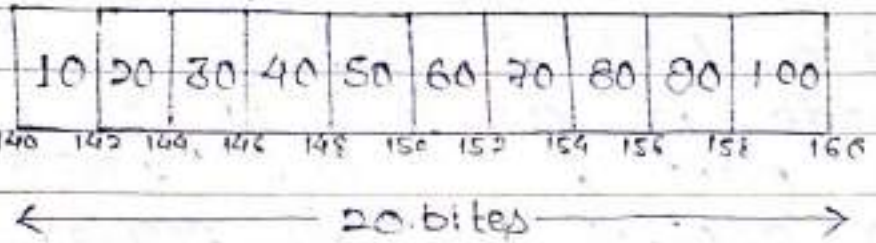
1) Inserting data items into data structure.
2) Deleting data items from data structure
3) Updating data items of data structure.
4) Finding / searching a particular data items in the list.
5) Arranging data items in a particular order.
6) Merging one or more list into one.

PAGE NO.: 03
DATE: / /

ARRAY

→ An array is a linear data structure which organises homogeneous data items at contiguous memory location.

eg: int a[10] = {10, 20, 30, ... 100};



Date 22/06/19

One of the major drawback of the array is that the insertion and deletion of data items are very difficult. Inserting a data item at specific location requires shifting of all data item forward by one position. Similarly, deleting a particular data item requires shifting of all data item backward by one position. This shifting operation takes lots of time and also requires complex algorithm. Hence, array data structure is not suitable when insertion and deletion of data items are more frequent.

Teacher Signature _____

LINKED LIST

→ A linked list is a linear data structure that organises the data items at non-contiguous memory location. The linear order is maintained by pointer.

The data item in linked list is referred as "Node". A Node is divided into two parts. One part is called "INFO" part and another part is called "LINK" part. The INFO part holds information and LINK part holds the address of next NODE (successor node). The link part of last node contains NULL which indicates end of list.

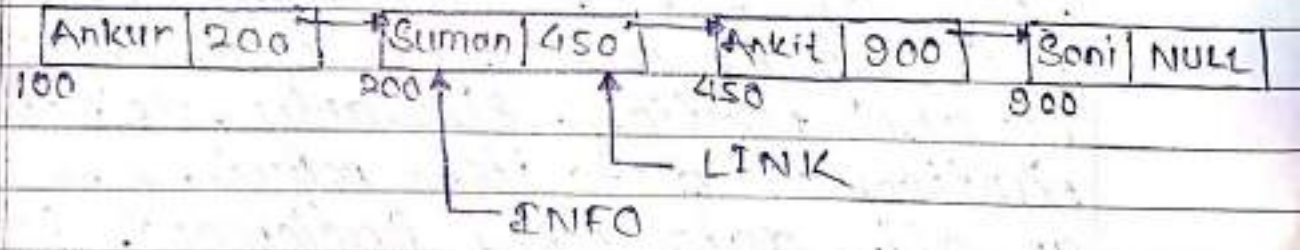


Fig:- Schematic diagram of linked-list.

Since the nodes of linked-list are not adjacent to each other and therefore insersation and deletion of data items (nodes) are very simple.

Teacher Signature

Hence the drawback encountered in array got eliminated in linked-list.

⇒ Drawbacks of Linked-List

(i) Linked list occupies more memory as every node has an extra field to store address.

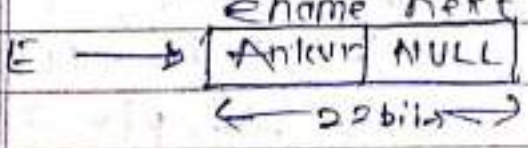
(ii) Any data item can't be accessed quickly / directly because traversal begins from the first node.

(iii) Little programming overhead because of complex pointer application.

⇒ Defining / Creating Node

The self-referencial structure is used to define node: A structure that has one of its members as a pointer to same structure then the structure is called self-referencial structure.

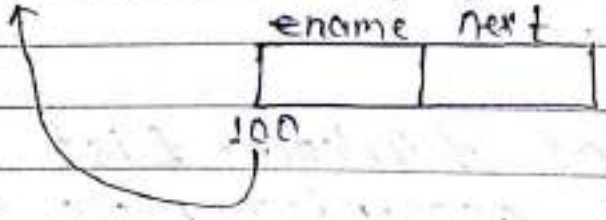
eg:-
```
struct employee
{
char ename[20];
struct employee *next;
};
```




```
Structure Employee E;
E.name
E.next
```

Teacher Signature

struct Employee *p;
p = (struct Employee) malloc (sizeof (struct employee));



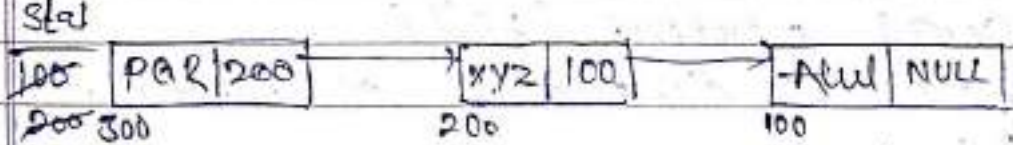
Date 23/Oct/19

Linked List

- Singly linked list
- Doubly linked list
- Circular linked-list

(1) Singly linked list :- It is also referred to as simply "linked-list". It is a kind of linked-list that allows traversal only in forward-direction i.e every node has only one link-field that holds address of successor node only.

Q:- Create a linked-list to maintain name of students. Add node at the beginning of list.



Output
PAR
XYZ
ANU

Traversed

```
Ptr = start;
while (ptr != NULL)
{
printf ("%s", ptr → name);
ptr = ptr → next;
}
```

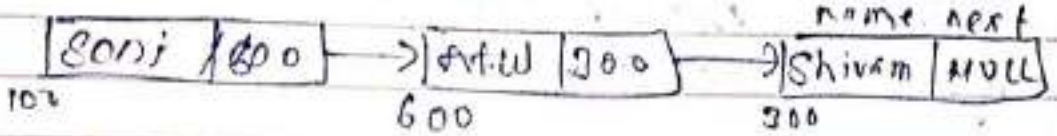
Teacher Signature

PAGE NO.: 07
DATE: / /

```
#include <stdio.h>
#include <conio.h>
#include <alloc.h>
struct student
{
 char name[20];
struct student *next;
};
void main()
{
struct student *start, *pta;
char ch;
clrscr();
start = NULL;
 do
{
pta = (struct student *)malloc(sizeof(struct student));
printf("Enter student name:");
scanf("%s", pta -> name);
fflush(stdin);
pta -> next = start;
start = pta;
printf("Do you wish add more(Y/n):");
 ch = getchar();
}while(ch == 'Y' || ch == 'y');
printf("List of students \n");
 pta = start;
while(pta != NULL)
{
printf("%s\n", pta -> name);
pta = pta -> next;
}
getch();
```

Teacher Signature

start	ptr	cf	Traversal
~~NULL~~	~~300~~	Y	ptr
~~300~~	~~600~~	Y	~~300~~
~~600~~	100	N	600
100			~~300~~
			NULL



Output
Soni
Atul
Shivam

Algorithm (Add node in beg Beggtng)
beginning
1. Start
2. Initialize start = NULL
3. Create node
ptr = (struct student*) malloc (sizeof (struct student))
4. Set value in ptr
5. Set ptr -> next = start
6. Set start = ptr
7. If more to add then goto step 3
8. End

Q:- Adding node at the end of list.

```
#include <stdio.h>
#include <conio.h>
#include <alloc.h>
struct student
{
```

```
    char name[20];
struct student * next;
};
void main()
{
struct student *start, *ptr, *last;
char ch;
clrscr();
start = NULL;
do
{
ptr=(struct student*)malloc(sizeof(struct student));
printf("enter student name :");
scanf("%s", ptr->name);
ptr->next = NULL
fflush(stdin);
if(start==NULL)
{
  start = ptr;
}
else
{
  last->next = ptr;
}
 last = ptr;
printf("Wish to add more(y/n):");
   ch = getchar();
}while(ch=='y'||ch=='y');
printf("List of students \n");
 ptr = start;
while(ptr!=NULL)
```

Teacher Signature

```
{
printf("%s\n", ptr->name);
ptr = ptr->next
}
getch();
}
```

Date 26/08/19

Inserting Node At Specific position in the list

Start 100

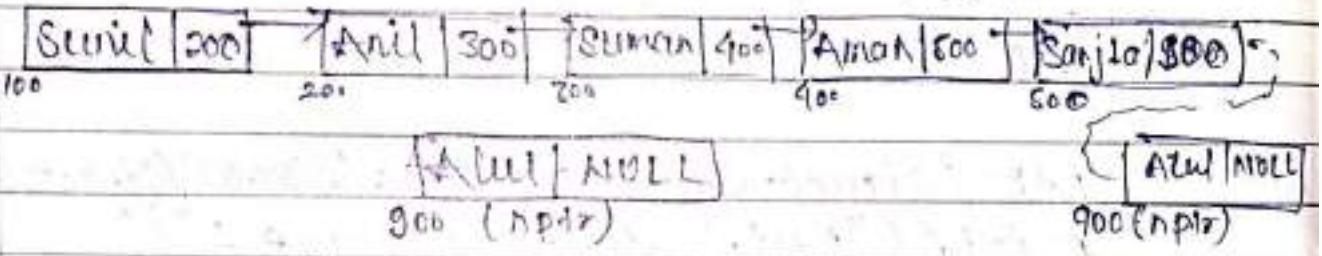


Fig:- Node to insert

Start 100 900

Inserting in the beginning

```
nptr -> next = start
start = nptr;
```

Prev 100 900 300 Start 100

Inserting in the middle (pos=4)

Find the address of node before insert position i.e. 3rd node

```
prev = start
for(i=1; i<pos-1; i++)
{
    prev = prev -> next;
}
```

Now insert

```
nptr -> next = prev -> next;
prev -> next = nptr
```

Teacher Signature

* Insert at the end

```
prev → next = nptr;
```

* Counting nodes in list

ptr	count
~~100~~	0
~~200~~	1
~~300~~	2
~~400~~	3
~~500~~	4
~~600~~	5
NULL	6

```
count = 0;
ptr = start;
while (ptr != NULL)
{
    count++;
    ptr = ptr → next;
}
print count;
```

Q:- Write an algorithm / function to insert a player at specific position in the list represent by linked-list.

```
typedef struct player
{
    char pname [40];
    struct player *next;
} pnode;
```

Sol":-

```
pnode * InsertNode (pnode *s)
{ /* s holds the address of 1st node */
  pnode *ptr, *nptr, *prev;
  int count = 0, pos, i;
    if (s == NULL)
    {
    printf ("List not found. Plz create first");
```

Teacher Signature

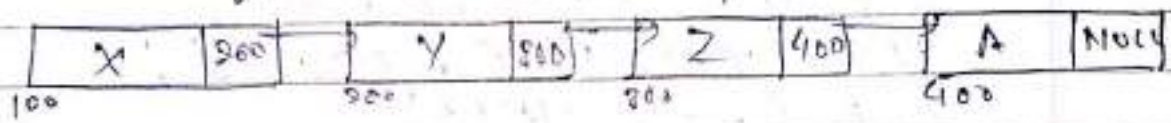
```
    return(0);
}
ptr = 0;
while (ptr != NULL)
{
    ccount++;
    ptr = ptr -> next;
}
printf("Enter position to insert player node:");
scanf("%d", &pos);
if (pos > ccount+1)
{
    printf("Invalid position \n");
    printf("Total player = %d \n", count);
    return(0);
}
nptr = (pnode *)malloc(sizeof(pnode));
nptr -> next = NULL;
printf("Enter player name:");
scanf("%s", nptr -> pname);
if (pos == 1)
{
    nptr -> next = 0;
    0 = nptr;
}
else
{
    prev = 0;
    for(i=1; i<pos-1; i++)
    {
    prev = prev -> next;
```

Teacher Signature

```
    }
if(prev -> next == NULL)
  {
    prev -> next = npt;
  }
else
  {
     npt -> next = prev -> next;
     prev -> next = npt;
  }
}
  return(0);
}
```

Deleting Node from specific position

start
100



Deleting 1st Node

```
pt = start;
start = start -> next
free(ptr);
```

pt
100

start
~~100~~ 200

Deleting middle Node (pos = 3)

Find address of 2nd & 3rd node

```
pt = start;
for(i=1; i<pos; i++)
  {
     prev = ptr;
     pt = pt -> next;
  }
```

ptr prev
~~100~~ ~~100~~
~~200~~ 200
300

Teacher Signature

Node Delete

```
prev->next = ptr->next;
free(ptr);
```

Deleting Last Node

```
prev->next = NULL
free(ptr);
```

Algorithm to delete Node from specific position

```
pnode * DelNode(pnode *p)
{
/* p holds the address of 1st node */
    pnode *ptr, *prev;
    int count = 0, pos, i;
    if(p == NULL)
    {
printf("List not found");
    return(p);
}
    ptr = p;
  while(ptr != NULL)
    {
        count++;
        ptr = ptr->next;
    }
printf("Enter player position to delete:");
  scanf("%d", &pos);
  if(pos) count)
```

Teacher Signature

```
}
printf("Invalid position");
printf("\n Total players = %d", count);
return(0);
}
if (pos == 1)
{
 pto = 0;
 0 = 0 -> next;
 free(pto);
}
else
 {
    pto = 0;
for(i=1; i<pos; i++)
{
   prev = pto;
   pto = pto -> next;
}
if (pto -> next == NULL)
{
    prev -> next = NULL;
     free(pto);
}
else
{
     prev -> next = pto -> next;
      free(pto);
   }
}
   return(0);
}
```

Teacher Signature

Date 27/06/19

PAGE NO.: 16
DATE: / /

2:- Develop a software using linked list to main players list of EPL-2019.

EPL-2019
MAIN MENU
1. create players list
2. Insert player
3. Delete player
4. Show list of players
5. Search player by position
6. Exit
choice (1-6):-

```
#include <stdio.h>
#include <conio.h>
#include <alloc.h>
#include <stdlib.h>
typedef struct Players
{
char pname[40];
struct Player *next;
}Pnode;

Pnode * create(Pnode*);
Pnode * insert(Pnode*);
Pnode * delp(Pnode*);
void show(Pnode*);
void search(Pnode*);
```

Teacher Signature

```
void main()
{
Pnode * start = NULL;
int n;
while(1)
{
clrscr();
printf("\t\t IPL-2019 \n");
printf("\t\t MAIN MENU \n");
printf("\t 1. Create Players List \n");
printf("\t 2. Insert Player \n");
printf("\t 3. Delete Player \n");
printf("\t 4. Show List of players \n");
printf("\t 5. Search Player by Position \n");
printf("\t 6. Exit \n");
printf("\t Choice (1-6): ");
scanf("%d", &n);
switch(n)
{
case 1:
start= create(start);
break;
case 2:
start= insert(start);
break;
case 3:
start = delp(start);
break;
case 4:
show(start);
break;
```

Teacher Signature

```
case 5:
search (start);
break;
case 6:
exit(0);
break;
default:
printf("Invalid choice... ");
}
getch();
}
}
/* Function to create players list */
Pnode * create (Pnode * s)
{
Pnode *last, *ptr;
char ch;
if (s != NULL)
{
printf("List is already created. Plz
  insert now");
return (s);
}
do
{
ptr = (Pnode *)malloc (sizeof (Pnode));
ptr -> next = NULL;
printf("Enter Players Name :");
scanf("%s", ptr -> pname);
fflush (stdin);
if (s == NULL)
```

Teacher Signature

```
{
s = ptr;
}
else
{
last = ptr;
printf("Wish to enter more (y/n):");
ch = getchar();
}while(ch == 'Y' || ch == 'y');
return(s);
}

/* Function to show player list */
void show(Pnode *s)
{
Pnode *ptr;
if(s == NULL)
{
printf("List is empty...");
return;
}
ptr = s;
while(ptr != NULL)
{
printf("%s\n", ptr->pname);
ptr = ptr->next;
}
}
```

Teacher Signature

```
/* Function to insert players */
Pnode* insert (Pnode *s)
{
Pnode *ptr, *nptr, *prev;
int ecount = 0, i, pos;
if (s == NULL)
{
printf ("List no found. Plz create
 first...");
return(s);
}
ptr = s;
while (ptr != NULL)
{
ecount++;
ptr = ptr -> next;
}
printf ("Enter position to insert: ");
scanf ("%d", &pos);
if (pos > ecount + 1)
{
printf ("Invalid Position...\n");
printf ("Total Players = %d \n", count);
return(s);
}
nptr = (Pnode*) malloc (sizeof (Pnode));
nptr -> next = NULL;
printf ("Enter player name :");
scanf ("%s", nptr -> pname);
if (pos == 1)
{
```

Teacher Signature

```
nptr -> next = s;
s = nptr;
}
else
{
        prev = s;
    for(i=1 ; i<pos-1 ; i++)
    {
    prev = prev -> next;
    }
    if (prev -> next == NULL)
    {
    prev -> next = nptr;
    }
    else
    {
    nptr -> next = prev -> next;
    prev -> next = nptr;
    }
}
return(s);
}

/* function to delete playes */
Pnode * delp(Pnode *s)
{
Pnode *ptr, *prev;
int count = 0, i, pos;
if (s == NULL)
{
```

Teacher Signature

```
printf("List is empty...");
return(s);
}
ptr = s;
while (ptr != NULL)
{
ecount++;
ptr = ptr -> next;
}
printf("Enter position to delete from:");
scanf("%d", &pos);
if(pos > ecount)
{
printf("Invalid Position:..\n");
printf("Total Players = %d \n", ecount);
return(s);
}
if (pos == 1)
{
ptr = s;
s = s -> next;
free(ptr);
}
else
{
        ptr = s;
      for(i=1; i<pos; i++)
      {
       prev = ptr;
       ptr = ptr -> next;
       }
```

Teacher Signature

```
    if (ptr -> next = NULL)
     {
     prev -> next = NULL;
     free (ptr);
     }
     else
     {
     prev -> next = ptr -> next;
     free (ptr);
     }
}
return (s);
}

/* function to search player */
void search (Pnode *s)
{
Pnode *ptr;
int pos, i, ccount = 0;
if (s == NULL)
{
printf ("List is empty...");
return;
}
ptr = s;
while (ptr != NULL)
{
count++;
ptr = ptr -> next;
}
```

Teacher Signature

```
printf("Enter Player Position:");
scanf("%d", &pos);
if(pos > count)
{
printf("Invalid Position \n");
printf("Total player = %d", count);
return;
}
ptr = s;
for(i=1; i<pos; i++)
{
ptr = ptr -> next;
}
printf("Player at position %d =
        %s", pos, ptr -> pname);
}
```

Date 28/08/19

(11) Doubly Linked List

In doubly linked-list is a kind of linked-list in which every node is divided into three parts. Two "linked" parts and one "info" part. One of the link parts holds the address of successor node and another linked part holds the address of predecessor node. Hence, doubly linked list allow traversal in both directions (in forward direction as well as in backward direction).

Teacher Signature

In contracts to this, singly linked-list allows traversal only in forward direction because singly linked-list contains the address of successor node only.

The drawbacks of doubly linked-list is that it takes more memory than singly linked-list.

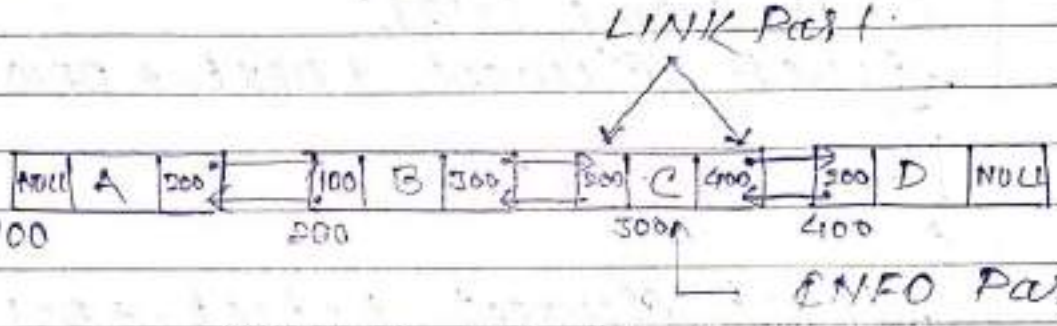
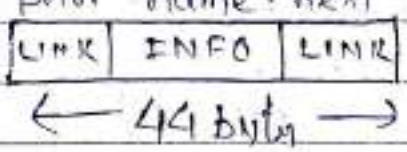


Fig:- Doubly linked-list.

* Structure for doubly linked list Node

```
struct student
{
  char name[40];
  struct student *next, *prior;
};
```



Teacher Signature

Q1 Create a doubly linked - list to store name of students Display names in forward direction as well as in backward direction

```
#include <stdio.h>
#include <conio.h>
#include <alloc.h>
struct student
{
char name [40];
struct student *next, *prior;
};
void main()
{
struct student *start, *ptr, *last;
char ch;
clrscr();
start = NULL;
do
{
ptr = (struct student *) malloc (sizeof (struct student));
ptr -> next = NULL;
printf ("Enter student name :");
scanf ("%s", ptr -> name);
fflush (stdin);
if (start == NULL)
{
    start = ptr;
    ptr -> prior = NULL;
}
```

Teacher Signature_____

```
else
{
 last -> next = ptr;
 ptr -> prior = last;
}
 last = ptr;
printf("Do you want to add more
                    (Y/n):");
 ch = getchar();
}while(ch == 'y' || ch == 'Y');
printf("Student Name: in foreward
          direction \n");
      ptr = start;
  while(ptr != NULL)
  {
printf("%s\n", ptr -> name);
  ptr = ptr -> next;
}
printf("Student names in backward
    direction \n");
      ptr = start;
while(ptr -> next != NULL)
{
ptr = ptr -> next;
}
while(ptr != NULL)
{
printf("%s", ptr -> name);
  ptr = ptr -> prior;
}
getch();
}
```

ptr
100
200
300
NULL
NR
SR
PR

ptr
100
200
300
200
100
NULL
PR
SR
NR

Last node address

Teacher Signature

Date 30/08/18

PAGE NO.: 28
DATE: / /

Q?:- Create a doubly linked-list to maintain name of employee. Add node in the beginning of list. Finally display names in forecoard as coell as in backward direction

Q?:- Inserting Node at specific position in Doubly linked-list

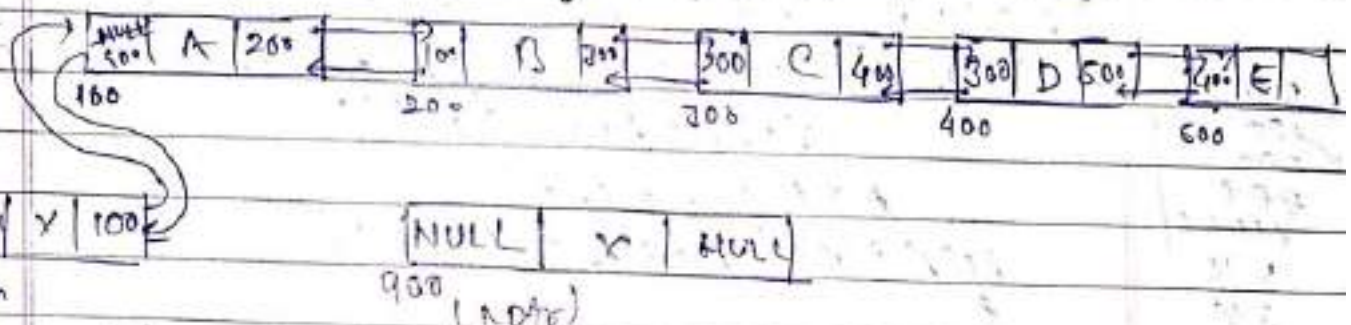


Fig:- Node to insert

⇐ At the Beginning (pos = 1)

```
start → prior = nptr;
nptr → next = start;
start = nptr;
```

⇐ On the middle (pos = 4)

Find address of 3rd Node

```
prev = start
for(i=1; i<pos-1; i++)
{
    prev = prev → next;
}
```

Teacher Signature

Node insert

```
nptr → next = prev → next;
nptr → prior = prev;
prev → next → prior = nptr;
prev → next = nptr;
```

ii) Inserting at the end

```
prev → next = nptr;
nptr → prior = prev;
```

Q1:- Write algorithm to insert node in doubly linked list at specific position of player list.

```
typedef struct Player
{
    struct pname [10];
  struct Player * next, * prior;
} pnode
```

Sol1:-

```
pnode *insert (pnode *s)
{ /* s is a pointer to 1st node*)
pnode *nptr, *ptr, *prev;
  int count = 0, pos, c;
    if (0 == NULL)
{
  printf("List is empty. Plz create first");
  return(s);
  }
ptr = s;
while (ptr != NULL)
```

Teacher Signature

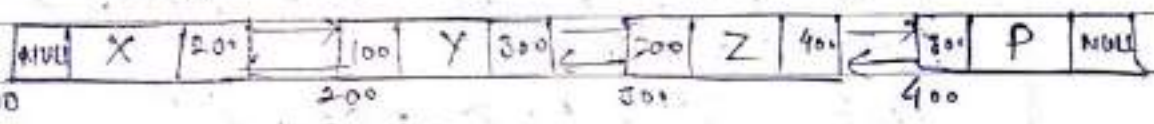
```
{
coount ++;
 ptr = ptr -> next;
}
printf("Enter position to insert :");
scanf("%d", &pos);
if(pos > coount+1)
{
printf("Invalid Position \n");
printf("Total players = %d", count);
 return(0);
}
nptr = (pnode *)malloc(sizeof(pnode));
npto -> next = nptr -> prior = NULL;
printf("Enter players name :");
scanf("%s", npto -> pname);
if(pos == 1)
{
    nptr -> next = 0;
     0 -> prior = nptr;
      0 = nptr;
  }
else
{
        prev = 0;
  for(i=1; i<pos-1; i++)
   {
     prev = prev -> next;
   }
if(prev -> next == NULL)
 {
```

Teacher Signature

```
        nptr → prior = prev;
        prev → next = nptr;
    }
    else
    {
        nptr → next = prev → next;
        nptr → prior = prev;
        prev → next → prior = nptr;
        prev → next = nptr;
    }
}
return(s);
}
```

Deleting Node from Doubly Linked-List

start 100 200 | ptr 100



Deleting 1st Node (pos = 1)

```
ptr = start;
start = start → next;
start → prior = NULL
free(ptr);
```

Deleting Middle Node (pos = 3)

ptr 300 pos 3

Find address of 3rd Node

```
ptr = start;
for(i=1; i<pos; i++)
    ptr = ptr → next;
```

Teacher Signature

```
}
```

Now Delete

```
ptr -> next -> prior = ptr -> prior;
ptr -> prior -> next = ptr -> next;
free(ptr);
```

* Deleting Last Node (pos = 4)

```
ptr -> prior -> next = NULL
free(ptr);
```

Q:- Write algorithm to delete node from doubly linked-list.

Sol:-
```
Pnode delnode(Pnode q)
{ /* q points to 1st node */
  Pnode ptr;
  int count = 0, pos, i;
  if(q == NULL)
  {
printf("List is empty...");
    return(0);
  }
  ptr = q;
while(ptr != NULL)
  {
    count++;
    ptr = ptr -> next;
  }
printf("Enter position of node to delete:");
scanf("%d", &pos);
```

Teacher Signature

```
if (pos>ccount)
{
printf (" Invalid position ");
printf ("\n Total Nodes = %d", ccount);
return(0);
}
if(pos==1)
{
   ptr = 0;
   0 = 0 -> next;
   0 -> prior = NULL
      free(ptr);
}
else
{
         ptr = 0;
for(i=1 ; i<pos ; i++)
  {
      ptr = ptr -> next;
  }
if(ptr -> next == NULL)
  {
    ptr -> prior -> next = NULL
      free (ptr);
}
else
{
      ptr -> next -> prior = ptr -> prior;
      ptr -> prior -> next = ptr -> next;
         free(ptr);
   }
```

Teacher Signature

```
}
return 0;
}
```

Q7:-

CPL-2019
Players Maintenance
MAIN MENU

1. Create Players list
2. Insert Player
3. Delete Player
4. Display Players list (Foreward)
5. Display Players list (Backward)
6. Search player
7. Exit

choice (1-7):-

Teacher Signature

Date 31/08/19

Circular Linked-List

A linked-list that allows traversal in circular manner is called circular linked-list. The link-field of last node contains address of 1st node instead of NULL. Hence, we can directly jumps from last node to 1st node to have traversal in circular manner.

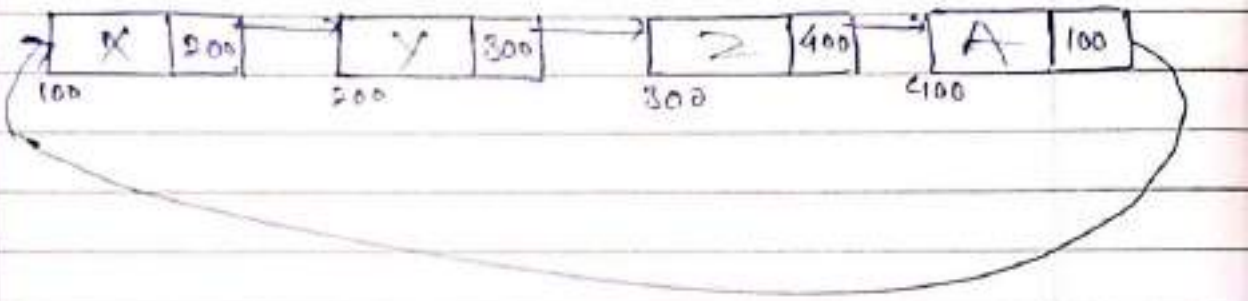


Fig:- Circular Linked-List

While traversing circular linked-list, NULL value is no longer checked.

Traversal Algorithm

```
ptr = start;
do
{
printf("%d\n", ptr->data);
ptr = ptr->next;
} while (ptr != start);
```

~~ptr~~
~~100~~
~~200~~
~~300~~
~~400~~
100

Output
X
Y
Z
A

Teacher Signature

Q:- Create a circular linked-list to maintain employees name.

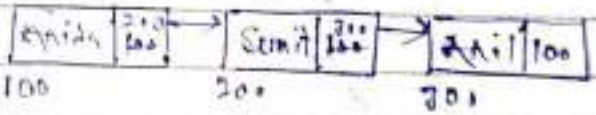
```
#include <stdio.h>
#include <conio.h>
typedef struct employee
 {
    char ename[40];
struct employee *next;
}EMP;
void main()
{
EMP *start, *ptr, *last;
   char ch;
   clrscr();
   start = NULL;
   do
 {
  ptr = (EMP*)malloc(sizeof(EMP));
printf(" Enter employee name :");
scanf("%s", ptr -> ename);
fflush (stdin);
if (start == NULL)
{
      start = ptr;
}
else
{
    last -> next = ptr;
}
      ptr -> next = start;
```

Teacher Signature

```
        last = ptr;
printf("Do you wish to enter more (y/n):");
   ch = getchar();
}while (ch == 'y' || ch == 'Y');
printf("List of Names \n");
        ptr = start;
   do
{
printf("%s\n", ptr->name);
   ptr = ptr -> next;
}while (ptr != start);
   getch();
}
```

Dry Run

start	ptr	last
NULL	100	100
100	~~200~~ 300	~~200~~ 300



Teacher Signature

(iii) STACK :- Stack is a linear data structure in which data items are inserted and deleted at only one end of list called "top" of the list.

The data items are removed in the reverse order of ~~insertion and therefore~~ insertion i.e the last item inserted will be the first item to be removed. Hence, stack is also called LIFO (Last-In-First-Out) list

⇒ Insertion

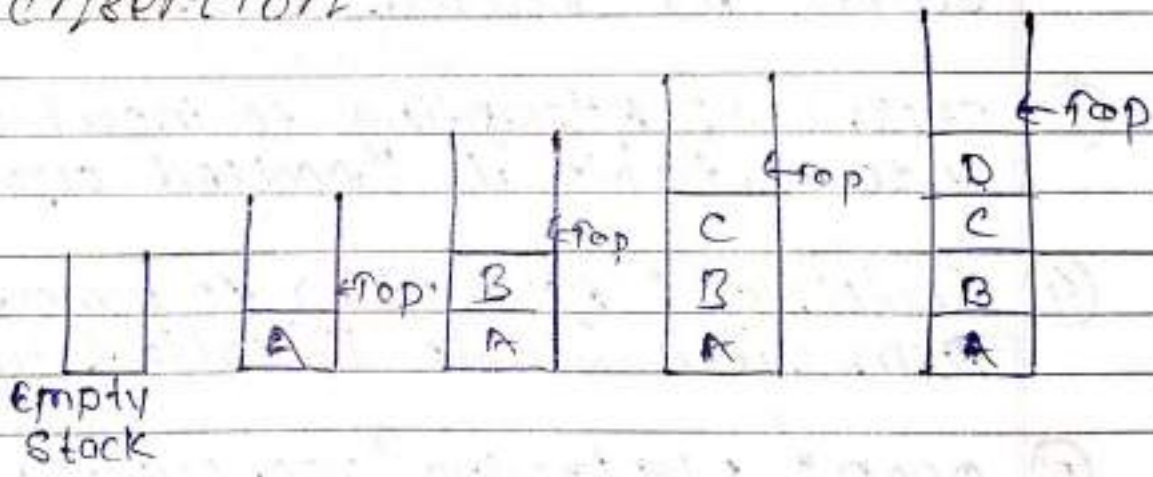


Fig:- Insertion into stack

⇒ Deletion

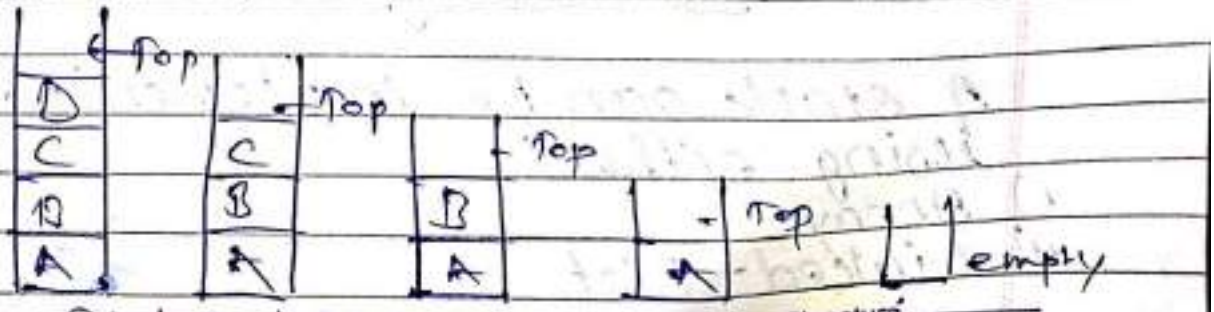


Fig:- Deletion from stack

Teacher Signature

Application of STACK

1. Number base conversion
2. Evaluating arithmetic exp^r
   - Infix to postfix
   - Post evaluation
3. Recursion

Date 03/09/19

Terms used in stack-operation

1. push: Inserting data item into stack is called push.
2. pop: Deleting data item from stack is called pop.
3. overflow: Attempting to insert data-item into stack is called overflow.
4. Underflow: Attempting to remove data-item from stack is called underflow.
5. peep: Displaying/accessing a particular data item from stack is called peep

Implementing STACK

A stack can be implemented using either:

i) Array
ii) Linked-List

Teacher Signature

Array Implementation

Push Algorithm

```
set pop = -1
void push()
{
    if(top = = size-1)
  {
printf("stack is overflow");
 return;
 }
printf("Enput item to insert:");
scanf("%d", &s[++top]);
}
```

Pop Algorithm

```
  set top = -1
void pop()
{
 if(top = = -1)
 {
printf("stack underflow..");
 return;
}
printf("Popped item = %d", s[top]);
 top--;
}
```

Teacher Signature

8. Create a stack using array to maintain mark of 5 students.

```
STACK Operation
Main Menu
1. Push
2. Pop
3. Display
4. Exit
Choice (1-4):
```

```
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
#define size 5

void push();
void pop();
void display();

int top = -1;
int stack[size];

void main()
{
int n;
while(1)
{
clrscr();
printf("\t\t STACK Operation \n");
```

Teacher Signature

```
printf("\t\t MAIN MENU \n");
printf("\t 1. PUSH \n");
printf("\t 2. POP \n");
printf("\t 3. Display \n");
printf("\t 4. Exit \n");
printf("\t CHOICE (1-4):");
scanf("%d", &n);
switch(n)
{
case 1:
push();
break;
case 2:
pop();
break;
case 3:
display();
break;
case 4:
exit(0);
break;
default:
printf("Invalid Choice");
}
getch();
}
}

void push()
{
if(top == size-1)
```

Teacher Signature

```
}
printf("stack is overflow");
return;
}
printf("Enter marks to push:");
scanf("%d", &S[++top]);
}

void pop()
{
if(top==-1)
{
printf("stack is Underflow");
return;
}
printf("popped marks: %d.", S[top]);
top--;
}
void display()
{
int i;
if(top==-1)
{
printf("stack is Empty...");
return;
}
for(i=top; i>=0; i--)
{
printf("%d, ", s[i]);
}
}
```

Teacher Signature

Date 05/09/19

PAGE NO.: 53
DATE: / /

Stack Implementation using Linked-List

```
Stack Operation
   Main Menu
1. Push
2. Pop
3. Display
4. Exit
Choice (1-4):
```

```
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
#include <alloc.h>
typedef struct Exam
{
int marks;
struct Exam *next;
}Enode;

Enode *push(Enode*);
Enode *pop(Enode*);
void display(Enode*);
void main()
{
int n;
Enode *start=NULL;
while(1)
{
```

Teacher Signature

```
clrscr();
printf("\t\t STACK Operation \n");
printf("\t\t MAIN MENU \n");
printf("\t 1. PUSH \n");
printf("\t 2. POP \n");
printf("\t 3. Display \n");
printf("\t 4. Exit \n");
printf("\t Choice (1-4) :");
scanf("%d", &n);
switch(n)
{
case 1:
start = push(start);
break;
case 2:
start = pop(start);
break;
case 3:
display(start);
break;
case 4:
exit(0);
break;
default:
printf("Invalid choice");
}
getch();
}
}
```

Teacher Signature

```
Enode * push(Enode *s)
{
Enode *ptr;
ptr = (Enode *)malloc(sizeof(Enode));
if (ptr == NULL)
{
printf("Memory is full...");
return(s);
}
printf("Enter marks to push:");
scanf("%d", &ptr->marks);
ptr->next=s;
s=ptr;
return(s);
}

Enode * pop(Enode *s)
{
Enode *ptr;
if(s==NULL)
{
printf("STACK UNDERFLOW...");
return(s);
}
printf("Popped marks:%d", s->marks);
ptr=s;
s=s->next;
free(ptr);
return(s);
}
```

Teacher Signature

```
void display(Enode *s)
{
Enode *ptr;
if(s==NULL)
{
printf("stack is empty...");
return;
}
for(ptr=s; ptr!=NULL; ptr=ptr->next)
{
printf("%d,", ptr->marks);
}
}
```

* Arithmetic Expression
It consists of operators & operands which evaluates to a single value.

eg. 9+3*4
21

* Operators
  - ↑, ^, ** - power/exponential
  - %
  - /
  - *
  - +
  - -

Teacher Signature

Operator	Precedence	
↑, ** , ^	1	Left to Right
%, /, *	2	
+, −	3	

Notation of Arithmetic Expression

(i) Infix Notation :- Operator is placed between operands.
ex:- A+B, C*D, F/E etc

(ii) Polish/Prefix Notation :- Operator is placed before operands.
ex:- +AB, *CD, /FE etc

(iii) Reverse Polish Notation/Postfix Notation
Operator is placed after operands.
eg:- AB+, CD*, FE/ etc

Evaluation of Arithmetic Expression

An arithmetic exp^n in computer is evaluated in two steps :-

i) Infix exp^n is transformed to postfix exp^n.
ii) Evaluates postfix exp^n.
Computer uses stack data structure in both the above steps.

Teacher Signature

Date 04/09/19

PAGE NO.: 58
DATE: / /

Infix to Postfix Conversion (Using trick)

Q:- Convert following infix into postfix.

$9*(5+7)-2\uparrow 3/4+5$

Soln:- $9*\underset{T1}{(5+7)}-2\uparrow 3/4+5$

$\Rightarrow 9*T1-\underset{T2}{2\uparrow 3}/4+5$

$\Rightarrow \underset{T3}{9*T1}-T2/4+5$

$\Rightarrow T3-\underset{T4}{T2/4}+5$

$\Rightarrow \underset{T5}{T3-T4}+5$

$\Rightarrow T5+5$

$\Rightarrow T5,5+$

$\Rightarrow T3,T4,-,5,+$

$\Rightarrow 9,T1,*,T2,4,/,-,5,+$

$\Rightarrow 9,5,7,+,*,2,3,\uparrow,4,/,-,5,+$

Show the evaluation of following postfix

9, 5, 7, +, *, 2, 3, ↑, 4, /, -5, +

add ')' at the end of postfix to indicate expⁿ ending :

9, 5, 7, +, *, 2, 3, ↑, 4, /, -5, +,)

Teacher Signature

BODMAS

Scan postfix Left-to-Right	STACK
9	9
5	9, 5
7	9, 5, 7
+	9, 12
*	108
2	108, 2
3	108, 2, 3
↑	108, 8
4	108, 8, 4
/	108, 2
−	106
5	106, 5
+	111
)	↳ pop & set into result

Q": convert following infix into postfix
A * B - C ↑ D + E / F

Sol":-
A * B - C ↑ D + E / F
⇒ A * B − T1 + E / F (T2 = A * B)
⇒ T2 − T1 + E / F (T3 = E / F)
⇒ T2 − T1 + T3
⇒ ~~A, B, *, −, C, D, ↑, +, E, F, /~~ (T4 = T1 ↑ T)
⇒ T4 + T3, +
⇒ T2, T1, −, E, F, /, +
⇒ A, B, *, C, D, ↑, −, E, F, /, +

Teacher Signature

Date 06/09/19

PAGE NO.: 60
DATE: / /

Q:- Convert the following into postfix
A+B*C/E↑F*G+H

Q:- Convert the following into postfix and also evaluates the postfix.
3*(4+2) - 5/2 * 9 + 4

(1) A+B*C/E↑F*G+H
T1

=) A+B*C/T1*G+H
T2

=) A+T2/T1*G+H
T3

=) A+T3*G+H
T4

=) A+T4+H
T5

=) T5+H

=) T5, H, +

=) A, T4, +, H, +

=) A, T3, G, *, +, H, +

=) A, T2, T1, /, G, *, +, H, +

=) A, B, C, *, E, F, ↑, /, G, *, +, H, +

Teacher Signature

(2) $3 * \underbrace{(4+2)}_{T1} - 5/2 * 9 + 4$

$\Rightarrow \underbrace{3 * T1}_{T2} - 5/2 * 9 + 4$

$\Rightarrow T2 - \underbrace{5/2}_{T3} * 9 + 4$

$\Rightarrow T2 - \underbrace{T3 * 9}_{T4} + 4$

$\Rightarrow T2 - \underbrace{T4 + 4}_{T5}$

$\Rightarrow T2 - T5$

$\Rightarrow T2, T5, -$

$\Rightarrow 3, T1, *, T4, 4, *, +, -$

$\Rightarrow 3, 4, 2, +, *, T3, 9, *, 4, +, -$

$\Rightarrow 3, 4, 2, +, *, 5, 2, /, 9, *, 4, +, -)$

Scan Postfix Left to Right	STACK
3	3
4	3, 4
2	3, 4, 2
+	3, 6
*	18
5	18, 5
2	18, 5, 2
/	18, 2
9	18, 2, 9
*	18, 18
4	18, 18, 4
+	18, 22
−	4
)	

Teacher Signature: Pops set into result

Q:- Convert the following infix into postfix and evaluate

$5*(6+2)-12/4$

Sol:

$5*\underbrace{(6+2)}_{T1}-12/4$

$\Rightarrow \underbrace{5*T1}_{T2}-\underbrace{12/4}_{T2}$

~~$5*T1-T2$~~

$\Rightarrow T2-\underbrace{12/4}_{T3}$

$\Rightarrow \underline{T2-T3}$

$\Rightarrow T2,T3,-$

$\Rightarrow$ ~~$T2,T3,-$~~

$\Rightarrow 5,T1,*,12,4,/,-$

$\Rightarrow 5,6,2,+,*,12,4,/,-$

Scan Postfix	STACK
5	5
6	5, 6,
2	5, 6, 2
+	5, 8
*	40
12	40, 12
4	40, 12, 4
/	40, 3
-	37
)	

↳ pop & set into Result

Teacher Signature

Q?: convert into postfix & evaluate
15*(2+3)/2-1*4%3*2

Sol?:- 15*(2+3)/2-1*4%3*2
T1

=> 15*T1/2-1*4%3*2
T2

=> T2/2-1*4%3*2
T3

=> T3-1*4%3*2
T4

=> T3-T4%3*2
T5

=> T3-T5*2
T6

=> T3-T6
=> T3,T6,-
=> T2,2,/,T5,2,*,-
=> 15,T1,*,2,/,T4,3,%,2,*,-
=> 15,2,3,+,*,2,/,1,4,*,3,%,2,*,-
=> 15,2,3,+,*,2,/,1,4,*,3,%,2,*,-

Scan Postfix	STACK
15	15
2	15,2
3	15,2,3
+	15,5
*	75
2	75,2
/	37
1	37,1

Teacher Signature

4	37, 1, 4
*	37, 4
3	37, 4, 3
-	37, 1
2	37, 1, 2
*	37, 2
-	35
)	35 — pop & set into result

H.W

(1) Convert the following into postfix

A + (B * C - (D / E ↑ F) * G) * H

(2) Convert the following into postfix & also evaluate:-

19 * 2 * 4 / 3 - 6 % 7 + 4

Soln:- (1) A + (B * C - (D / E ↑ F) * G) * H
T1

=> A + (B * C - (D / T1) * G) * H
T2

=> A + (B * C - T2 * G) * H
T3

=> A + (T3 - T2 * G) * H
T4

=> A + (T3 - T4) * H
T5

=> A + T5 * H
T6

=> A + T6

=> A, T6, +

=> A, T5, H, *, +

Teacher Signature

⇒ A + T3, T4, -, d1, *, +
⇒ A, B, C, *, T2, G, *, -, d1, *, +
⇒ A, B, C, *, D, T1, /, G, *, -, d1, *, +
⇒ A, B, C, *, D, E, F, ↑, /, G, *, -, d1, *, +

(2) $\underbrace{19 * 2}_{T1} * 4 / 3 - 6 \% 7 + 4$

⇒ $T1 * \underbrace{4 / 3}_{T2} - 6 \% 7 + 4$

⇒ $T2 - \underbrace{6 \% 7}_{T3} + 4$

⇒ $T3 - \underbrace{T2 / 3}_{T4} + 4$

⇒ $\underbrace{T3 - T4}_{T5} + 4$

⇒ T5 + 4
⇒ T5, 4, +
⇒ T3, T4, -, 4, +
⇒ T2, 3, /, 6, 7, %, -, 4, +
⇒ T1, 4, *, 3, /, 6, 7, %, -, 4, +
⇒ 19, 2, *, 4, *, 3, /, 6, 7, %, -, 4, +

Scan Postfix	STACK
19	19
2	19,2
*	38
4	38,4
*	152
3	152,3
/	50
6	50,6
7	50,6,7
%	50,6
-	44
4	44,4
+	48

← pop & set into result

Teacher Signature

Algorithm to evaluate Postfix

1. Start
2. Input Postfix into P.
3. Add ")" at the end of P
4. Scan P from left to Right & repeate set step 5 and 6 until ")" is found.
5. If operand is found, push it into stack.
6. If operator (X) is found then
   a. Pop 2-operands from STACK: A & B
   b. Evaluate A (X) B and push the result into stack.
   [End. - if]
   [End of loop - step 4]
7. Pop & set into result
8. End

Date 07/09/19

Convert infix into postfix using computer algorithm
A+(B*C.-(D/E↑F)*G)*H

Soln:- Add ")" into infix & push "(" into stack.

A+(B*C-(D/E↑F)*G)*H)

Teacher Signature

Scan infix Left-to-Right	STACK	Postfix
A	(	A
+	(+	A
(	(+(	A
B	(+(	A,B
*	(+(*	A,B
C	(+(*	A,B,C
-	(+(-	A,B,C,*
(	(+(-(	A,B,C,*
D	(+(-(	A,B,C,*,D
/	(+(-(/	A,B,C,*,D
E	(+(-(/	A,B,C,*,D,E
↑	(+(-(/↑	A,B,C,*,D,E
F	(+(-(/↑	A,B,C,*,D,E,F
)	(+(-	AB,C,*,D,E,F,↑,/
*	(+(-*	A,B,C,*,D,E,F,↑,/
G	(+(-*	A,B,C,*,D,E,F,↑,/,G
)	(+	A,B,C,*,D,E,F,↑,/,G,*,-
*	(+*	A,B,C,*,D,E,F,↑,/,G,*,-
H	(+*	A,B,C,*,D,E,F,↑,/,G,*,-,H
)	Empty	A,B,C,*,D,E,F,↑,/,G,*,-,H,*,+

Algorithm (Infix to Postfix):
Let I be the infix exp^n & P be the equivalent postfix.

1. Start
2. Add ")" at the end of "I" and push "(" into STACK

Teacher Signature

(3) Scan Infix (Q) from left to right & repeat steps (4) to (7) until stack is empty.
(4) If operand is found, add into P
(5) If "(" is found, push into STACK
(6) If operator (X) is found then
(a) Pop each operator having equal or higher precedence & add into P
(b) Push (X) into stack.
[End. - If]
(7) If ")" is found, pop each operator & add to P until "(" is found. Remove "(" But don't add to P.
(8) End.

(IV) Queue

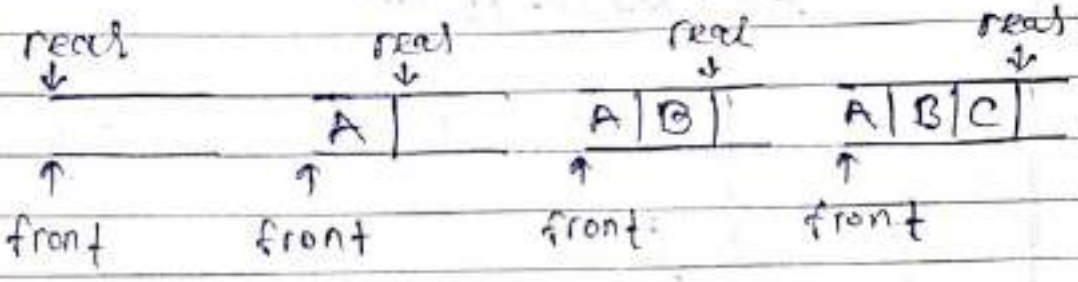
→ Queue is a linear data structure in which data items are inserted at one end of list and deleted from another end of list. The end of list where data-items are inserted is called "Rear" and the end of list from where data items are deleted is called "Front"

The data-items are removed in the order in which

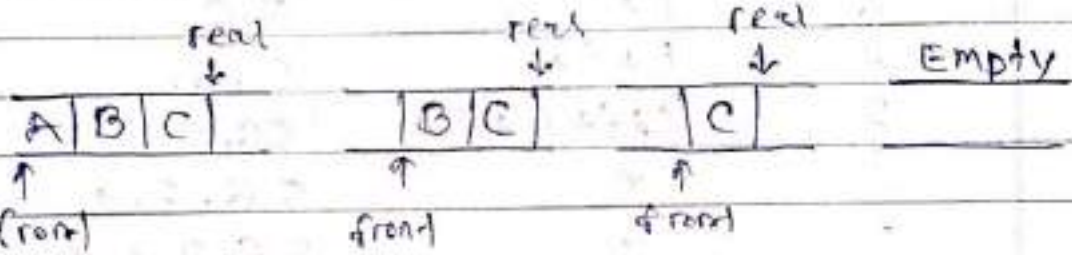
Teacher Signature_____

they are inserted into the queue. In other words, the first item inserted will be the first item to be removed. i.e Hence, queue is also called FIFO (First-In-First-Out) list.

= Insertion



= Deletion



= Application

- Used in job scheduling where jobs are processed in FCFS manner.
- Used in disk scheduling.
- Print spooling
- Synchronous data transfer

Teacher Signature

Date 09/10/19

PAGE NO.: 70
DATE: / /

Implementation of Queue

A queue can be implemented using either

A) Array
B) Linked - list

Q:- 5*(6+2)-12/4

Add ")" at the end of infix

5*(6+2)-12/4)

Scan L-to-R	STACK	Postfix
5	(	5
*	(*	5
(	(*(	5
6	(*(	5,6,
+	(*(+	5,6,
2	(*(+	5,6,2
)	(*	5,6,2,+
-	(-	5,6,2,+,*
12	(-	5,6,2,+,*,12
/	(-/	5,6,2,+,*,12
4	(-/	5,6,2,+,*,12,4
)	Empty	5,6,2,+,*,12,4,/,-

Queue using Array

Q:- Create a queue to maintain marks of 10 - students.

Teacher Signature

Queue:
Main Menu
1. Insert
2. Delete
3. Display
4. Exit
Choice (1-4):

Insert Algorithm
Let Q[10], front = -1, rear = -1;
void Qinsert()
{
if (rear == size - 1)
{
printf("Input marks:");
scanf("%d", &Q[++rear]);
if (front == -1)
front = 0
}

Delete Algorithm
void Qdelete()
{
if (front == -1)
{
printf("Queue is empty ..");
return;
}
printf("Deleted item = %d", Q[front]);
front++;
if (front == rear + 1)
front = rear = -1;
}

Teacher Signature

80 M°:-

```
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
#define size 10

int a[size];
int front = -1;
int rear = -1;
void insert();
void delete();
void display();

void main()
{
int n;
while(1)
{
clrscr();
printf("\t\t Queue \n");
printf("\t\t MAINMENU \n");
printf("\t 1. Insert \n");
printf("\t 2. Delete \n");
printf("\t 3. Display \n");
printf("\t 4. Exit \n");
printf("\t Choice(1-4):");
scanf("%d", &n);
switch(n)
{
case 1:
insert();
break;
```

Teacher Signature

Soln:-

```
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
#define size 10

int a[size];
int front = -1;
int rear = -1;
void insert();
void delete();
void display();

void main()
{
int n;
while(1)
{
clrscr();
printf("\t\t Queue \n");
printf("\t\t MAINMENU \n");
printf("\t 1. Insert \n");
printf("\t 2. Delete \n");
printf("\t 3. Display \n");
printf("\t 4. Exit \n");
printf("\t Choice(1-4):");
scanf("%d", &n);
switch(n)
{
case 1:
insert();
break;
```

Teacher Signature

```
case 2:
@delete();
Break;
case 3:
@display();
break;
case 4:
Exit(0);
Break;
default;
printf("Invalid Choice...");
}
getch();
}
}
void @insert()
{
if(rear==size-1)
{
printf("Queue is full...");
return;
}
printf("Enter marks:");
scanf("%d",&q[++rear]);
if(front==-1)
front=0;
}
void @delete()
{
if(front==-1)
{
```

Teacher Signature

```
printf("Queue is empty..");
return;
}
printf("Deleted item = %d", Q[front]);
front++;
if(front == rear+1)
front = rear = -1;
}
void Qdisplay()
{
int i;
if(front == -1)
{
printf("Queue is empty..");
return;
}
for(i=front; i<=rear; i++)
{
printf("%d,", Q[i]);
}
}
```

Date 13/09/19

4. Create a queue to maintain marks of 10 students through linked list.

```
Queue
Main Menu
1. Insert
2. Delete
3. Display
4. Exit
Choice (1-4):
```

Teacher Signature

Sol^n:-

```
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
typedef struct Exam
{
int mark;
struct Exam * next;
}Sexam;
Sexam * Qinsert(Sexam*);
Sexam * Qdelete(Sexam*);
void Qdisplay(Sexam*);

void main()
{
int n;
Sexam * start = NULL;
while (1)
{
clrscr();
printf("\t Queue using Linked List \n");
printf("\t\t MAIN MENU \n");
printf("\t 1. Insert \n");
printf("\t 2. Delete \n");
printf("\t 3. Display \n");
printf("\t 4. Exit \n");
printf("\t Choice (1-4): ");
scanf("%d", &n);
switch(n)
{
case 1:
start = Qinsert(start);
```

Teacher Signature

```
break;
case 2:
start = &delete(start);
break;
case 3:
&display(start);
break;
case 4:
exit(0);
break;
default:
printf("Invalid choice");
}
getch();
}

/* function to insert into Queue */
Sexam *&insert(Sexam *s)
{
Sexam *ptr, *last;
ptr = (Sexam *)malloc(sizeof(Sexam));
ptr -> next = NULL;
printf("Enter mark to insert:");
scanf("%d", &ptr -> mark);
if (s == NULL)
{
s = ptr;
}
else
{
```

Teacher Signature

```
last = s;
while (last -> next != NULL)
{
last = last -> next;
}
last -> next = ptr;
}
return (s);
}

/* Function to delete from Queue */
Sexam * Qdelete (Sexam *s)
{
Sexam * ptr;
if (s == NULL)
{
printf("Queue is empty ...");
return(s);
}
printf(" Deleted Item = %d ", s -> mark);
ptr = s;
s = s -> next;
free (ptr);
return (s);
}

/* function to display queue */
void Qdisplay (Sexam *s)
{
Sexam * ptr;
if (s == NULL)
```

Teacher Signature

```
{
printf("Queue is empty...");
return;
}
ptr=S;
while(ptr!=NULL)
{
printf("%d -> ", ptr->mark);
ptr = ptr->next;
}
}
```

Date 14/09/19

Circular Queue

When a queue is implemented through array then there is problem that once the queue is full and someone item has been deleted he cannot insert item even through some rooms are vacant insertion can be done only when all items are deleted. This problem is result through circular queue.

Circular queue is a kind of queue which allows insertion and deletion of a data items in circular manner. In circular queue, the last element appear to be just before the first element as at shown below:

Teacher Signature

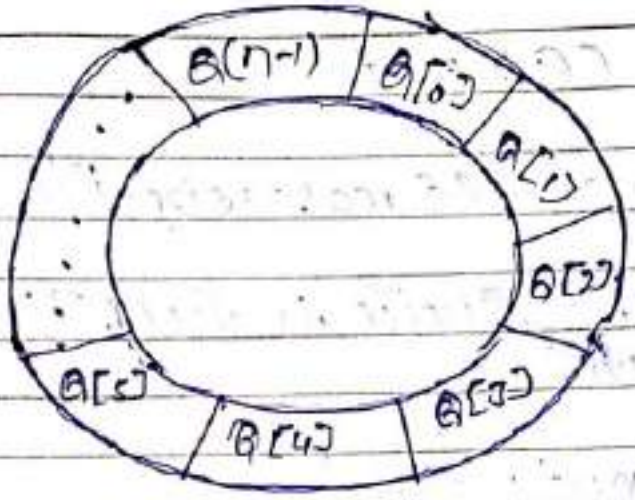
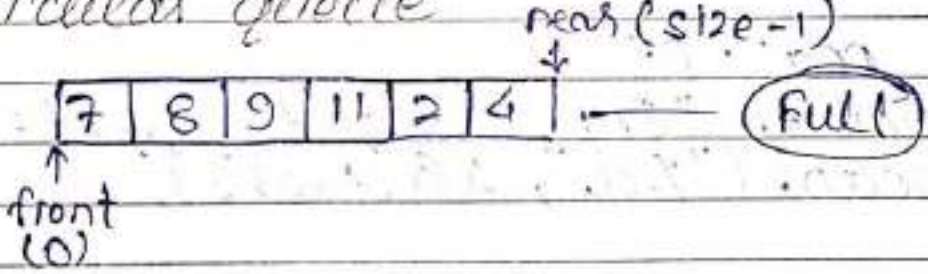


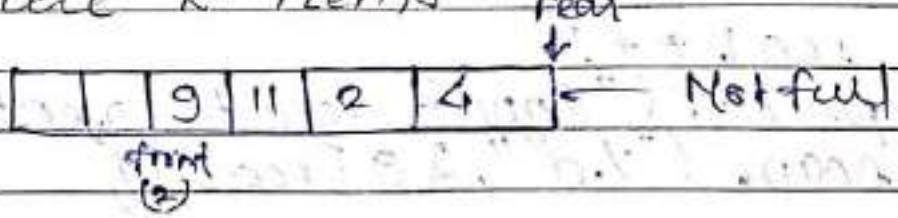
Fig:- Circular Queue

In case of circular queue, once the queue is full and few items are deleted then we can insert as many items as deleted.

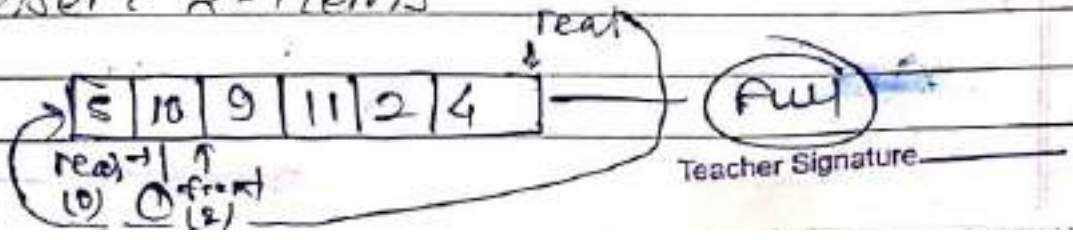
* Circular queue



* Delete 2 items



Insert 2 items



Teacher Signature

Insert Algorithm (Circular Queue)

```
void cQinsert
{
if((front==0&&rear==size-1)||front==rear+1)
{
  printf("Queue is full");
  return;
}
 if(front==-1)
 {
    front=0;
    rear=0;
printf("Input data to insert:");
scanf("%d",&q[rear]);
}
else if((rear==size-1)&&front>0)
{
   rear=0;
 printf("Input data to insert:");
 scanf("%d",&q[rear]);
}
else
{
    rear++;
   printf("Input data to insert:");
   scanf("%d",&q[rear]);
}
}
```

Teacher Signature

Delete Algorithm (Circular Queue)

```
void CQdelete()
{
  if (front == -1)
{
  printf ("Queue is empty ... ");
return;
}
printf ("Deleted data = %d", Q[front]);
  front++;
if (front == rear)
  {
     front = -1;
     rear = -1;
  }
else if (front == size - 1)
  {
  front = 0;
  }
}
```

Teacher Signature

Display Algorithm (Circular Queue)

```
void CQdisplay()
{
  int i;
if(front==-1)
{
 printf("Queue is empty..");
return;
}
if(rear>front)
{
 for(i=front;i<=rear;i++)
{
printf("%d,",Q[i]);
 }
}
else
{
 for(i=front;i<=size-1;i++)
{
   printf("%d,",Q[i]);
}
for(i=0;i<=rear;i++)
{
printf("%d,",Q[i]);
 }
}
}
```

Teacher Signature_____

Q: Create circular queue to maintain marks of 10 students.

```
Circular Queue
Main Menu
1. Insert
2. Delete
3. Display
4. Exit
Choice (1-4):
```

Sol^n:-

```
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
# define size 10

int cq[size];
int front = -1;
int rear = -1;
void cqinsert();
void cqdelete();
void cqdisplay();

void main()
{
int n;
while(1)
{
clrscr();
printf("\t\t Circular Queue\n");
printf("\t Main Menu \n");
```

Teacher Signature

```
printf("\t 1. Insert \n");
printf("\t 2. Delete \n");
printf("\t 3. Display \n");
printf("\t 4. Exit \n");
printf("\t Choice (1-4): ");
scanf("%d", &n);
switch(n)
{
case 1:
cirinsert();
break;
case 2:
cirdelete();
break;
case 3:
cirdisplay();
break;
case 4:
Exit(0);
break;
default:
printf("Invalid Choice...");
}
getch();
}
}

void cirinsert()
{
if((front==0 && rear==size-1) || front==rear+1)
```

Teacher Signature

```
{
printf("Queue is full");
  return;
}
  if(front==-1)
  {
     front=0;
     rear=0;
printf("Input data to insert:");
scanf("%d",&CQ[rear]);
 }
else if((rear==size-1)&&front>0)
{
 rear=0;
printf("Input data to insert:");
scanf("%d",&CQ[rear]);
}
else
{
     rear++;
printf("Input data to insert:");
scanf("%d",&CQ[rear]);
}
}

void CQdelete()
{
if(front==-1)
{
```

```
printf("Queue is empty...");
return;
}
printf("Deleted data = %d", CQ[front]);
front++;
if(front == rear)
{
    front = -1;
    rear = -1;
}
else if (front == size-1)
{
 front = 0;
  }
}

void CQdisplay()
{
 int i;
if(front == -1)
{
printf("Queue is empty...");
 return;
}
 if (rear > front)
{
for(i=front; i<=rear; i++)
{
printf("%d,", CQ[i]);
```

Teacher Signature

```
}
}
else
{
for(i=front; i<=size-1; i++)
{
  printf("%d ", CQ[i]);
}
for(i=0; i<=rear; i++)
{
printf("%d ", CQ[i]);
}
}
}
```

— x —

Teacher Signature

SORTING :- The process of arranging the data items of list in a particular order: Ascending or descending is called sorting.

There are two types of sorting

a) Internal sorting
b) External sorting

a) Internal sorting :- Sorting the list items present in main memory is called internal sorting.

b) External sorting :- Sorting the list items present on secondary memory like disc is called external sorting.

Since the main memory is much faster than secondary memory and therefore internal sorting is faster than external sorting.

Sorting Techniques

i) Selection sort
ii) Bubble sort
iii) Insertion sort
iv) Quick sort
v) Merge sort
vi) Heap sort

Teacher Signature_____

Date 16/09/19

PAGE NO.: 89
DATE: / /

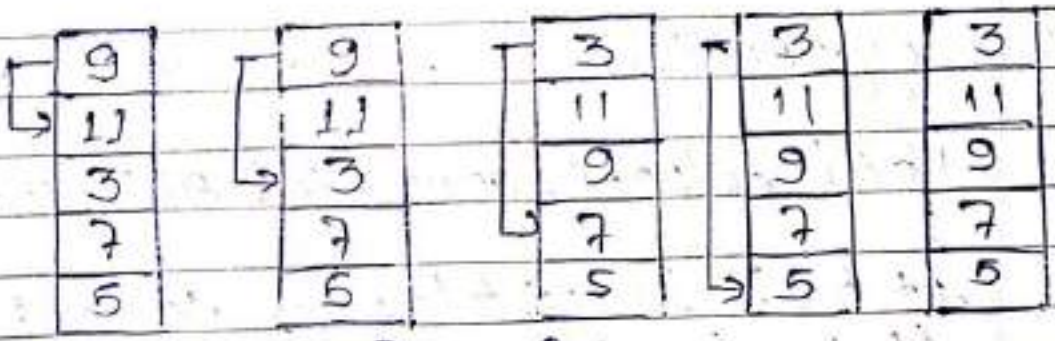
i) Selection sort :- This sorting method sorts the list by comparing each element with the rest of elements and exchange the value if comparing element is larger. After 1st pass, the smallest value get positioned in the 1st element.

If there are n-elements then there can be (n-1)-passes.

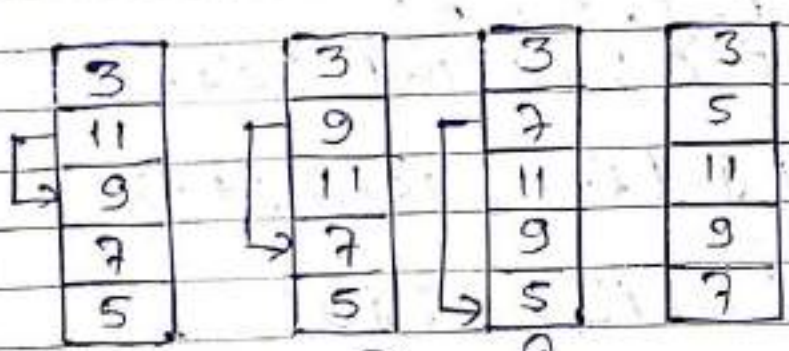
Q:- Sort the following numbers using selection sort.

9, 11, 3, 7, 5

Sol:-



Pass-I



Pass-II

Teacher Signature

Part-III

Part IV

Algorithm (Selection Sort)

```
Void Selectionsort (int A[], int n)
{
/* A[] is list/array to sort */
/* n is the no. of elements */
  int i, j, temp;
 for (i=0; i<(n-1); i++)
  {
 for ( j=(i+1); j<n; j++)
   {
   if( A[i] > A[j])
   {
    temp = A[i];
    A[i] = A[j];
    A[j] = temp;
   }
  }
}
```

Teacher Signature

8?: Write a program in C which accept 5 nos from user & sorts the no. using selection sort.

Sol":-

```
#include <stdio.h>
#include <conio.h>
void selectionsort(int[],int);
void main()
{
int A[5];
int i;
clrscr();
printf("Enter any 5 nos \n");
for(i=0;i<=4;i++)
{
printf("value %d:",i+1);
scanf("%d",&A[i]);
}
printf("List before sorting \n");
for(i=0;i<=4;i++)
{
printf("%4d",A[i]);
}
selectionsort(A,5);
printf("\n List after sorting \n");
for(i=0;i<=4;i++)
{
printf("%4d",A[i]);
}
getch();
}
```

Teacher Signature

```
void selectionsort(int A[], int n)
{
int i, j, temp;
for(i=0; i<n-1; i++)
{
for(j= i+1 ; j<n; j++)
{
    if (A[i]>A[j])
    {
   temp = A[i];
    A[i] = A[j];
    A[j] = temp;
     }
    }
  }
}
```

(1) Sort the following nos. using selection sort.
7, 25, 11, 14, 5, 2, 0, 4, 51, 6

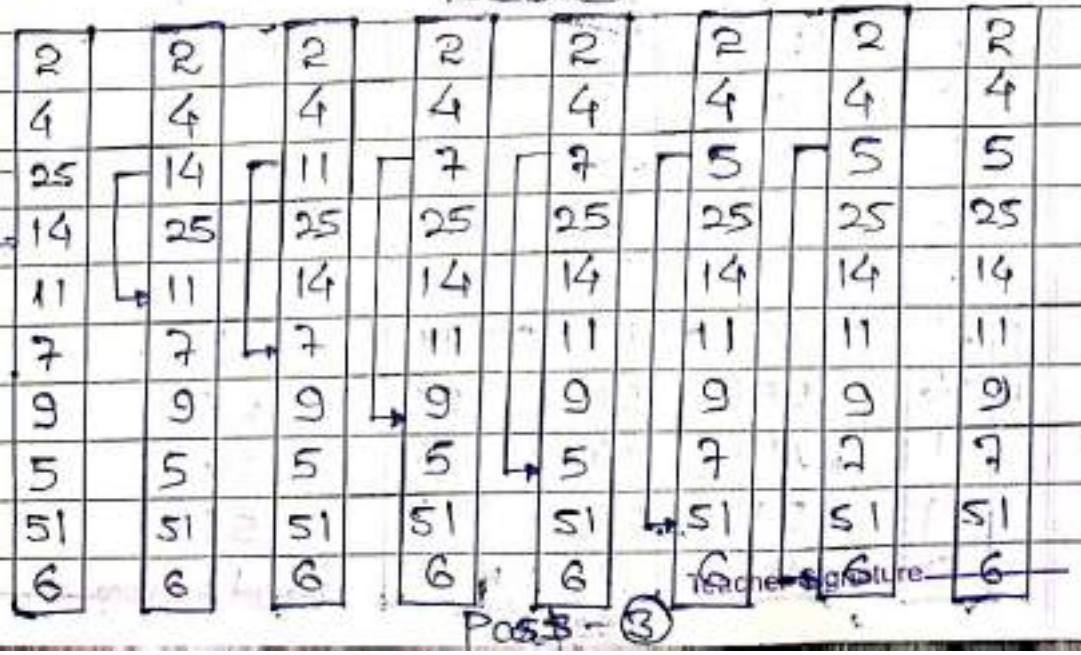
(2) Write C program which accept 10 nos. from user & sorts in descending order.

Teacher Signature

Q) 80]?

Pass-①

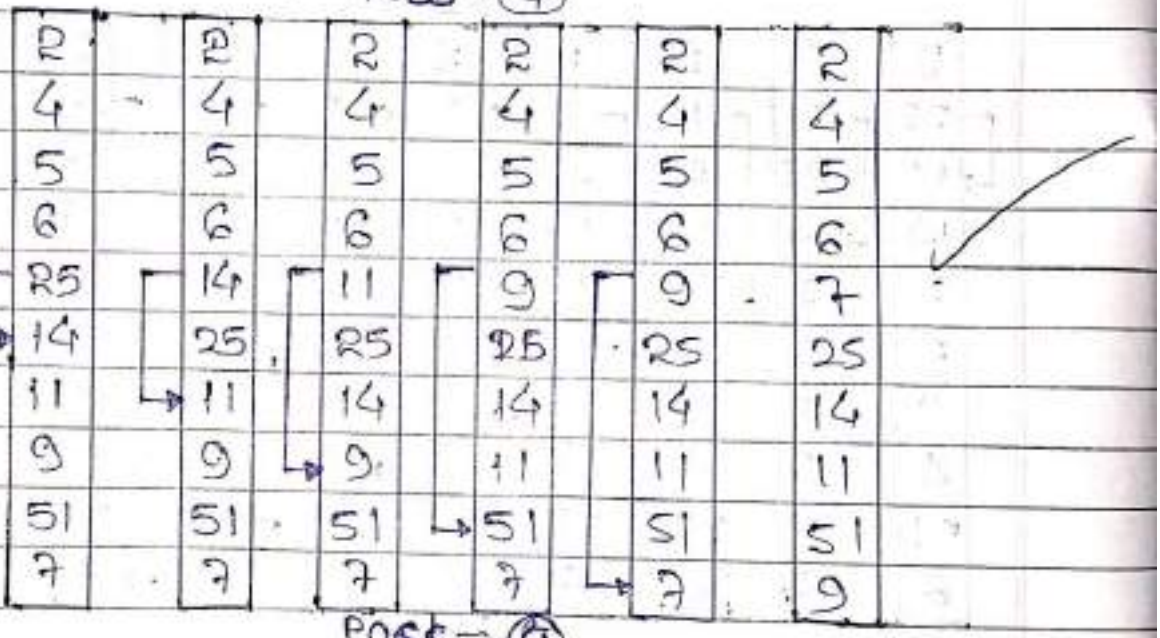
Pass-②



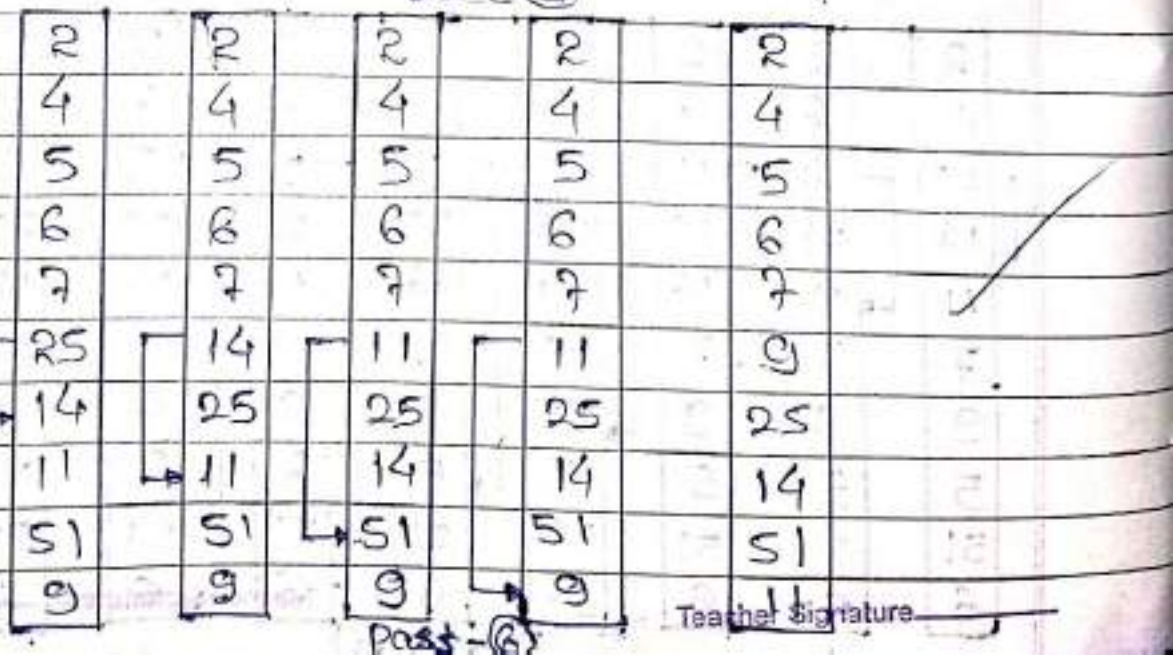
Pass-③

Teacher Signature

Pass - (4)



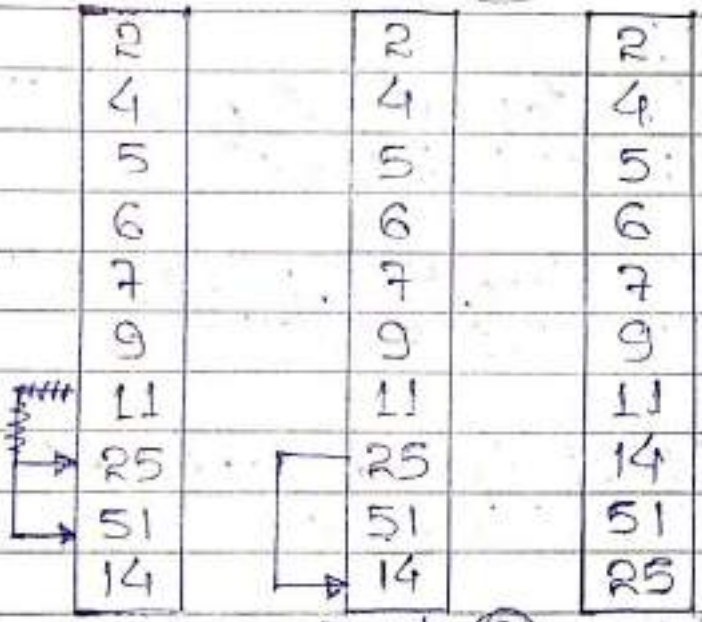
Pass - (5)



Pass - (6)

Teacher Signature

Pass - (7)



Pass - (8)

Pass (9)

→ Sorted

Teacher Signature

(2) Write C- program which accept 5 numbers from user & sorts in descending order.

Sol^n:-

```
#include <stdio.h>
#include <conio.h>
void selectionsort(int[],int);
void main()
{
int A[5];
int i;
clrscr();
printf("Enter any 5 nos\n");
for(i=0;i<=4;i++)
{
printf("value %d.:",i+1);
scanf("%d",&A[i]);
}
printf("List before sorting \n");
for(i=0;i<=4;i++)
{
printf("%4d",A[i]);
}
selectionsort(A,5);
printf("\n List after sorting \n");
for(i=0;i<=4;i++)
{
printf("%4d",A[i]);
}
getch();
}
```

Teacher Signature

```
void selectionsort (int A[], int n)
{
int i, j, temp;
for( i= 0; i<n-1 ; i++)
{
for( j= i+1; j<n; j++)
{
    if( A[i] < A[j] )
    {
  temp = A[i];
    A[i] = A[j];
    A[j] = temp;
     }
   }
 }
```

Date 18/09/19 No. of comparision = $\frac{n \times (n-1)}{2}$
Only applicable for bubble and selection sort.

(ii) Bubble Sort :- This sorting technique sorts the list items by comparing and exchanging adjacent element. On the 1st pass the largest value get positioned in the last element and all smalles value get shifted up (bubbles up) and hence the name is bubble sort. Similarly in the 2nd pass, the second largest value get positioned in the second last element.

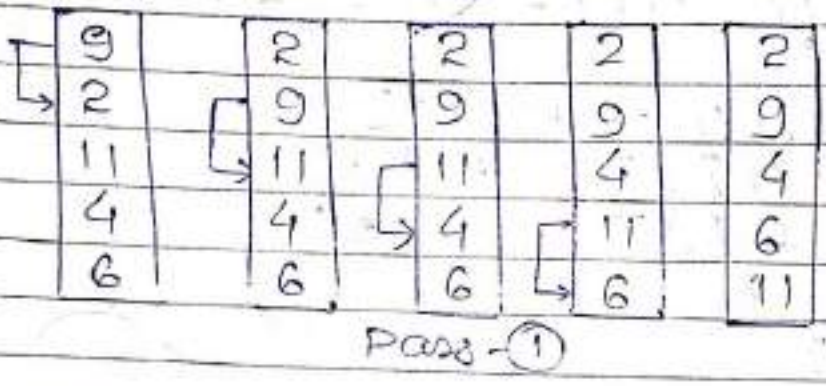
Teacher Signature

This ~~prog~~ process is repeated till (n-1) passes where n is the no. of elements.

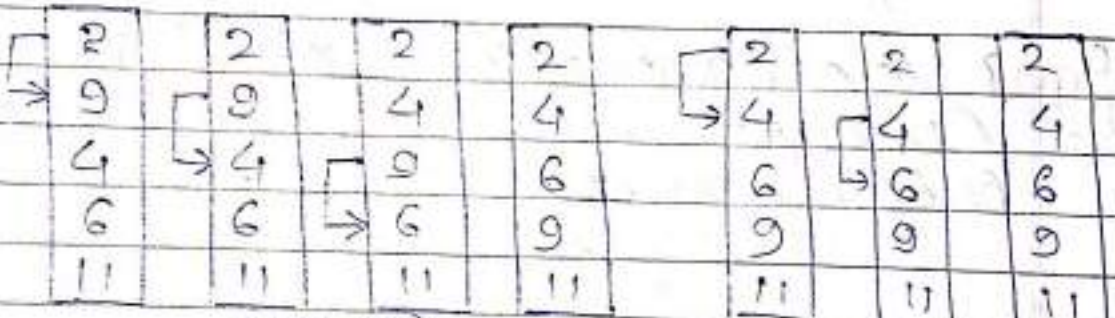
Q:- Sort the following nos. using bubble sort.

9, 2, 11, 4, 6

Sol":-

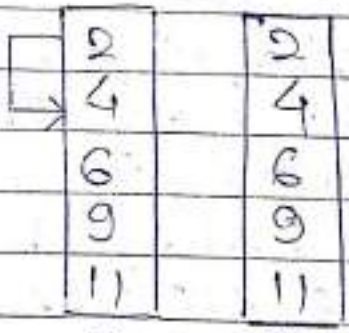


Pass-(1)



Pass-(2)

Pass (3)



Pass-(4)

Teacher Signature

Algorithm (Bubble sort)

```
void Bubblesort(int A[], int n)
{
/* A[] is the array to sort */
/* n- is the no. of element */
    int i, j, temp;
for (i=0; i<(n-1); i++)
{
for (j=0; j<(n-1)-i, j++)
{
    if (A[j] > A[j+1])
    {
        temp = A[j];
        A[j] = A[j+1];
        A[j+1] = temp;
    }
}
}
}
```

Q1: Write C- program that accept 10- nos from user and sorts the number using bubble sort.

Sol^n:-

```
#include <stdio.h>
#include <conio.h>
void bubblesort(int[], int);
void main()
{
int A[10];
int i;
clrscr();
```

Teacher Signature

```
printf("Enter any 10 nos \n");
for(i=0; i<=9; i++)
{
printf("value %d :", i+1);
scanf("%d", &A[i]);
}
printf("List before sorting \n");
for(i=0; i<=9; i++)
{
printf("%4d", A[i]);
}
bubblesort(A,10);
printf("\n List after sorting \n");
for(i=0; i<=9; i++)
{
printf("%4d", A[i]);
}
getch();
}
void bubblesort(int A[], int n)
{
  int i, j, temp;
for(i=0; i<(n-1); i++)
{
for(j=0; j<(n-1)-i, j++)
{
      if(A[j]>A[j+1])
      {
          temp=A[j];
          A[j]=A[j+1];
          A[j+1]=temp;
      }
}
}
```

Teacher Signature

* Time Complexity / Running Time
The time required by an algorithm to complete the task is called time complexity or running time. The time complexity is calculated by calculating the nos of times comparisions are done amount the data items the time complexity increases when the input size is increases

* Big-'O' Notation :- It is also called O-notation. It is a notation to represent time complexity of algorithm.
If 'n' be the algorithm and n-be the no. of data inputs then the time complexity f(x) can be written as

$$f(x) = |O(g(n))|$$

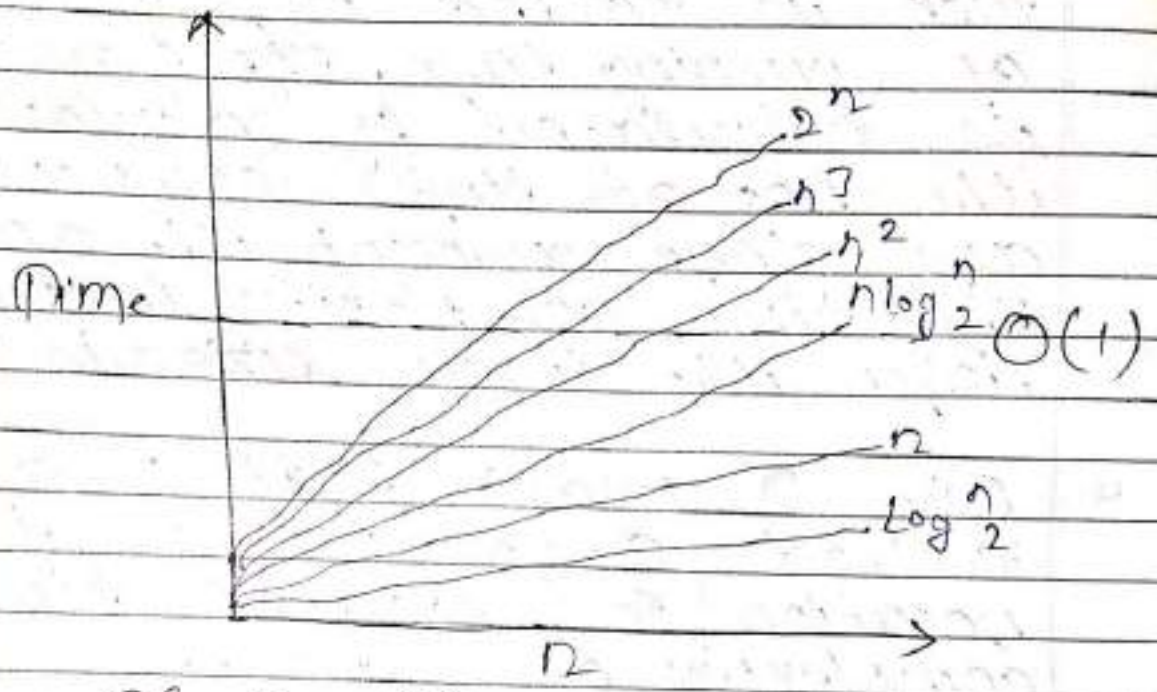
where g(n) is a function which can be one of the following standard specification of time complexity :-

- $O(1)$ - Constant
- $O(n)$ - Linear
- $O(n^2)$ - Quadratic
- $O(n^3)$ = Cubic
- $O(\log_2 n)$ - Logarithmic
- $O(n \log_2 n)$ - Logarithmic & Linear
- $O(2^n)$ - Exponential

Teacher Signature

Rate of Growth

$\log_2 n < n < n\log_2 n < n^2 < n^3 < 2^n$



If $n = 10$

$n = 10$
$\log_2 n = 4$
$n\log_2 n = 40$
$n^2 = 100$
$n^3 = 1000$
$2^n = 2^{10} = 1024$

$x = \log_2 n$
$x = \log_2 10$
$\Rightarrow 2^n = 10$
$\Rightarrow 2^x = 2^4$
$\Rightarrow x = 4$

Date 30/09/19

Running Time (Time complexity) of Selection sort / Bubble Sort.

Total no. of comparism $= \sum_{i=1}^{n} (n-i)$

$\Rightarrow (n-1) + (n+2) + (n-3) + \ldots + 1$

$\Rightarrow \frac{n \times (n-1)}{2}$

$\Rightarrow \frac{n^2 - n}{2}$

Teacher Signature

∴ Running Time $f(x) = O(n^2)$

(iii) Insertion Sort :- This sorting algorithm sorts the list item by inserting the data item at appropriate sorted sub-list. This sorting technique is similar to the way in which playing cards are arranged.

Insertion sorts performs sorting in the given following steps:-

Step-① A[0] is sorted itself.

Step-② A[1] is inserted either before A[0] or after A[1] so that A[0] and A[1] are sorted.

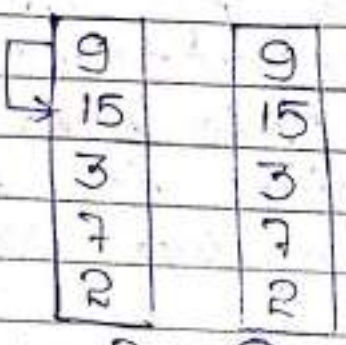
Step-③ A[2] is inserted either before A[0] or after A[1] or in between A[0] and A[1] so that A[0], A[1], A[2] becomes sorted.

.
.
.
.
.

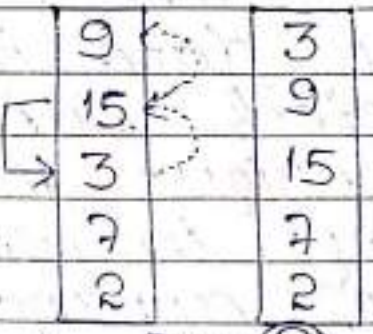
Step-(n) A[n-1] is inserted at appropriate position so that A[0], A[1], A[2] .... A[n-1] becomes sorted.

Teacher Signature

Q:- Sort the following nos. using insertion sort
9, 15, 3, 7, 2



Pass-(1)



Pass-(2)

Pass-(3)

Pass-(4) → Sorted

Algorithm (Insertion sort)

```
void Insertionsort(int A[], int n)
{
/* A[] is the array to sort */
/* n - is the number of element */
    int i, j, key;
for( i = 1; i<n ; i++)
  {
        key = A[i];
for(j=(i-1); j>=0 && key<A[j]; j--)
  {
        A[j+1] = A[j];
```

Teacher Signature

```
    }
    A[j+1] = key;
  }
}
```

Time Complexity of Insertion sort

① Worst Case

$$\text{Total comparism} = \sum_{i=1}^{n} (i-1)$$

$$\Rightarrow 1+2+3+\ldots+(n-1)$$

$$\Rightarrow \frac{n \times (n-1)}{2}$$

$$\Rightarrow \frac{n^2-n}{2}$$

Time complexity $f(x) = O(n^2)$

② Best Case :- If list is already sorted then the total comparism will equal to no. of items.

$\therefore f(x) = O(n)$

Q:- Write C program that accept 10 nos. from user and sorts the numbers in ascending order using insertion sort.

Soln:-

```
#include <stdio.h>
#include <conio.h>
```

Teacher Signature

```
void insertionsort (int [], int)
void main()
{
int A[10];
int i;
clrscr();
printf("Enter any 10 nos\n");
for(i=0; i<=9; i++)
{
printf("Value %d:", i+1);
scanf("%d", &A[i]);
}
printf("List before sorting \n");
for(i=0; i<=9; i++)
{
printf("%4d", A[i]);
}
insertionsort(A, 10);
printf("\n List after sorting \n");
for(i=0; i<=9; i++)
{
printf("%4d", A[i]);
}
getch();
}
void insertionsort(int A[], int n)
{
        int i, j, key;
  for(i=1; i<n; i++)
    {
```

Teacher Signature

```
        key = A[i];
for(j=(i-1); j>=0 && key<A[j]; j--)
  {
      A[j+1] = A[j];
  }
      A[j+1] = key;
    }
}
```

Q?: Write C program that accept 10 numbers from user and displays in descending order. Use insertion sort.

Sol?:

```
#include <stdio.h>
#include <conio.h>
void insertionsort(int[], int)
void main()
{
int A[10];
int i;
clrscr();
printf("Enter any 10 nos\n");
for(i=0; i<=9; i++)
{
printf("Value %d: ", i+1);
scanf("%d", &A[i]);
}
printf("List before sorting\n");
for(i=0; i<=9; i++)
{
printf("%4d", A[i]);
```

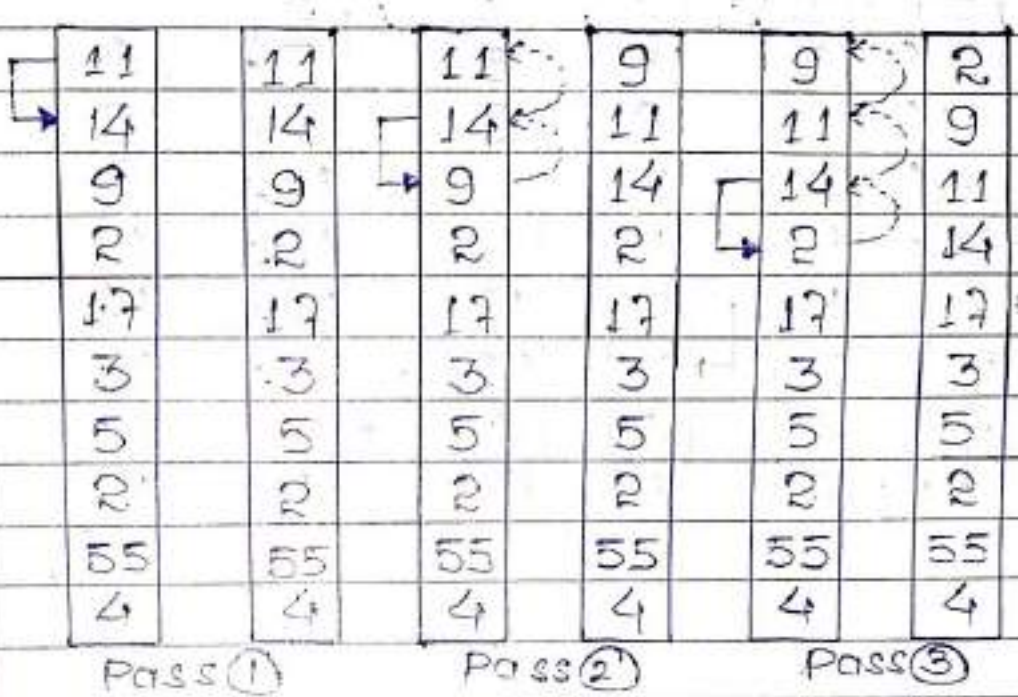
Teacher Signature

```
}
insertionsort (A,10);
printf("\n List after sorting \n");
for(i=0; i<=9; i++)
{
printf("%4d", A[i]);
}
getch();
}
void insertionsort(int A[], int n)
{
 int i, j, key;
 for(i=1; i<=9; i++)
 {
        key = A[j];
for(j=(i-1); j>=0 && key>A[j]; j--)
 {
        A[j+1] = A[j];
 }
        A[j+1] = key;
        }
}
```

Teacher Signature

Q?: Sort the following nos. using insertion sort.

11, 14, 9, 2, 17, 3, 5, 2, 55, 4



Teacher Signature

Date 21/09/19

(IV) Quick Sort :- Quick sort is a very fast sorting method introduce by Hoare in 1962. This sorting method is also called "partition exchange sort" as sorting is done by partitioning an array ~~partitioning an array~~ into two sub-arrays.

Let A[lower-upper] be the array to be sorted. So this algorithm, a = A[lower] is used as a control element (pivot value) and then A[lower:upper] is partitioned into two sub-arrays such that:-

Teacher Signature

i) A[lower, j-1] - Contains all elements smaller than a

ii) A[j+1 : upper] - Contains all elements larger than a where j is the position of a.

iv) Steps for partitioning & positioning control value

(1) Two index pointer "left" and "right" are initialized in such a way that left points to A[lower] and right points to A[upper].

(2) The pointer "left" & "right" are moved to each other in following manner:-

a) Repeatedly increase "left" index by one position until A[left] > a.

b) Repeatedly decrease "right" index by one position until A[right] < a

c) If right > left, swap A[left] & A[right] The above process is repeated until condition (c) fails i.e right > left and in that case A[right] is interchanged with A[lower] and j is set to right.

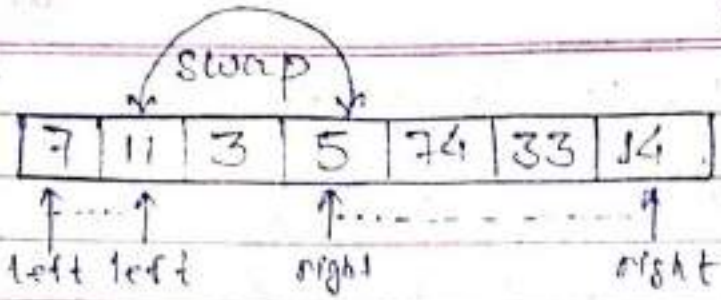
Q:- Sort the following no. using quick sort.

7, 11, 3, 5, 24, 33, 16

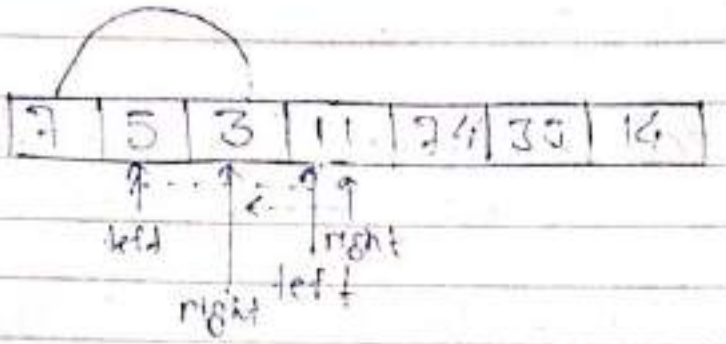
Teacher Signature

Sol^n:-

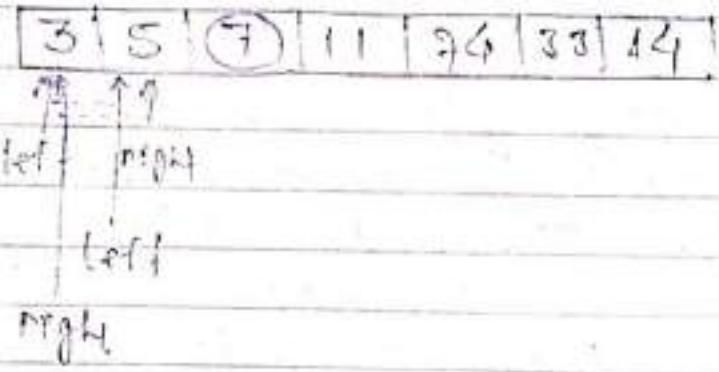
Step-①



a/7

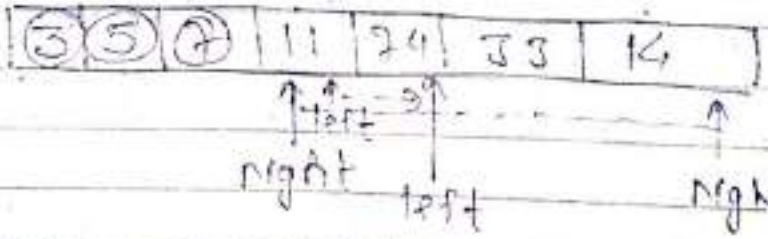


Step-②



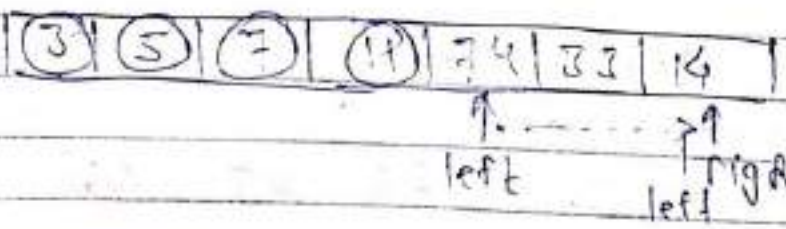
a/3

Step-③



a/11

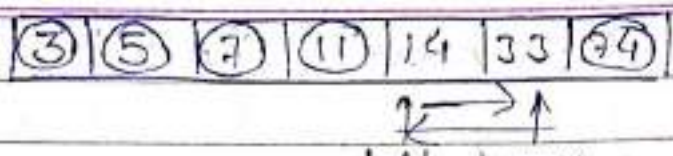
Step-④



a/74

Teacher Signature

Step 5
9/14



Algorithm (Quick sort)

```
void Quicksort (int A[], int lower, int upper)
{
    int j;
if ( lower < upper)
{
    j = partition (A, lower, upper);
Quicksort (A, lower, j-1);
Quicksort (A, j+1, upper);
    }
}
int partition (int A[], int lower, int upper)
{
    int left, right, temp;
     int a;
left = lower;
right = upper;
a = A[lower];
while (right > left)
{
    while (A[left] < a)
{
    left++;
}
```

Teacher Signature

```
while (A[right] > a)
  {
    right--;
  }
if (left < right)
  {
    temp = A[left];
    A[left] = A[right];
    A[right] = temp;
  }
}
  A[left] = A[right];
  A[right] = a;
  return (right);
}
```

Time Complexity

Best Case / Average case

Total comparism $= T(N/2) + T(N/2)$

$\Rightarrow 2T(N/2) + O(n)$

$\Rightarrow N/2^k + O(n)$

$\Rightarrow \log_2 n + O(n)$

$\therefore f(x) = O(n \log_2 n)$

Worst Case (if list is already sorted)

Total comparism $= T(n-1)$

$\Rightarrow (n-1) + (n-2) \cdots$

$= n \times (n-1)/2 = n^2 - n/2$

$\therefore f(x) = O(n^2)$

Teacher Signature

Q:- Write C-program that accept 10 numbers from user and sorts the numbers using quick sort.

Sol^n:-

```
#include <stdio.h>
#include <conio.h>
void quicksort(int [], int, int);
int partition(int [], int, int);
void main()
{
int A[10];
int i;
clrscr();
printf("Enter any 10 nos \n");
for(i=0; i<=9; i++)
{
printf("Value %d:", i+1);
scanf("%d", &A[i]);
}
printf("List before sorting \n");
for(i=0; i<=9; i++)
{
printf("%4d", A[i]);
}
quicksort(A, 0, 9);
printf("\nList after sorting \n");
for(i=0; i<=9; i++)
{
printf("%4d", A[i]);
}
getch();
}
```

Teacher Signature

```
void quicksort(int A[], int lower, int upper)
{
int j;
if(lower < upper)
{
j = partition(A, lower, upper);
quicksort(A, lower, j-1);
quicksort(A, j+1, upper);
}
}
int partition(int A[], int lower, int upper)
{
int left, right, temp;
int a;
left = lower;
right = upper;
a = A[lower];
while(right > left)
{
while(A[left] < a)
{
left++;
}
while(A[right] > a)
{
right--;
}
if(left < right)
{
temp = A[left];
}
```

Teacher Signature

```
while(A[right]>a)
{
 right--;
}
if(left < right)
{
 temp = A[left];
A[left] = A[right];
A[right] = temp,
}
}
A[left] = A[right];
A[right] = a;
return(right);
}
```

(v) Merge sort :- Merge sort is based on "Divide and conquer" philosophy. The basic action is to split the initial unsorted array A[lower : upper] into its middle element "mid" where:

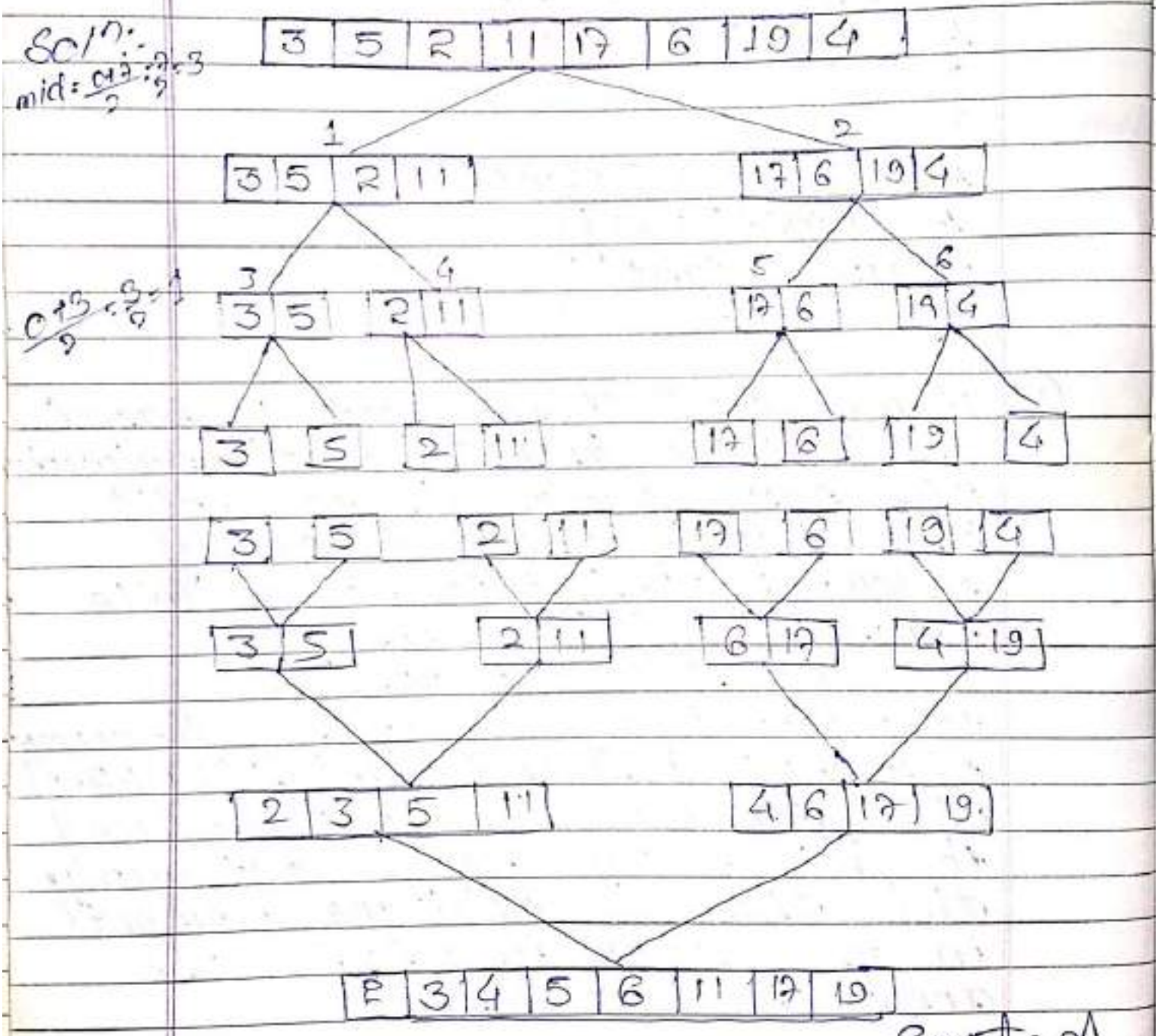
mid = (lower + upper) / 2

to have two unsorted sub-arrays A[lower : mid] and A[mid + 1 : upper]. The algorithm is repeatedly used to divide the arrays continuously till there is only one element in the array in which case array is sorted.

After splitting, the sorted arrays are merged based on sort orders.

Q:- Sort the following nos. using merge sort.
3, 5, 2, 11, 17, 6, 19, 4

Soln:-
mid: $\frac{0+7}{2} = \frac{7}{2} = 3$



Sorted

Teacher Signature

u- Algorithm (Merge Sort)

```
void mergesort (int A[], int lower, int upper)
{
/* A[] is the array to sort */
/* lower holds lower bond */
/* upper holds upper bond */
      int mid;
   if (upper > lower)
   {
      mid = (lower + upper)/2;
   mergesort (A, lower, mid);
   mergesort (A, mid+1, upper);
   merge (A, lower, mid, mid+1, upper);
   }
}
void merge (int A[], int lower1, int upper1,
            int lower2, int upper2)
  {
   int temp[100];
   int p, q, j, n;
   p = lower1; q = lower2; n = 0;
while ((p <= upper1) && (q <= upper2))
   {
   temp[n] = A[p] < A[q] ? A[p] : A[q];
      n++;
      p++;
      q++;
   }
   while (p <= upper1)
   {
```

Teacher Signature

```
temp[n] = A[p];
  n++;
  p++;
}
while (q <= upper2)
{
  temp[n] = A[q];
  q++;
  n++;
}
for(q=lower1, n=0; q<=upper1; q++, n++)
{
  A[q] = temp[n];
}
for(q=lower2, j=n; q<=upper2; q++, j++)
{
  A[q] = temp[j];
}
}
```

* Time Complexity

Total complexity = $2T(N/2) + C \cdot N$

$= \frac{N}{2^k} + C \cdot N$

$\therefore f(x) = O(n \log_2 n)$

Teacher Signature

Date 24/09/17

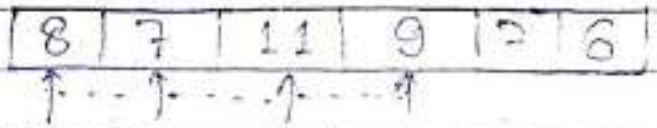
PAGE NO.: 121
DATE: / /

Searching :- The process of finding a specific data item in the list is called searching there are two popular search technique :-

1. Linear search / Sequential search
2. Binary search

1. Linear Search :- It is also called sequential sort. This searching technique start searching from the first element and move forward one by one element till the search is found or list is exhausted.

Key (searching)
9



(Matched)
Search Terminate

Time complexity = $f(x) = O(n)$

Algorithm (Linear Search)

```
void linearsearch(int A[], int n, int key)
{
/* A[] is the array to search in */
/* n is the no. of element */
/* key holds value of search */
```

Teacher Signature

```
        int i, found = 0;
    for(i=0; i<n; i++)
      {
       if(A[i] == key)
    {
    found = 1;
      break;
     }
}
    if(found == 1)
    printf("%d is found at %d position",
                    key, i+1)
    else
    printf("%d is not found", key);
     }
```

Q:- Write C program that accept 10-nos from user and search a number using linear search.

Sol:-

```
#include <stdio.h>
# include <conio.h>
# define size 10
void linearsearch(int[], int, int);
   void main()
{
   int A[size];
    int key, i;
   clrscr();
printf("Enter any 10 nos \n");
scanf("%d",
```

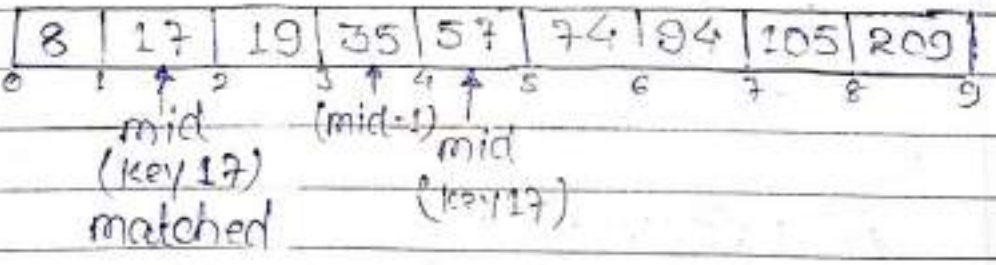
```
for(i=0; i<size; i++)
 {
printf("valued %d:", i+1);
scanf("%d", &A[i]);
 }
printf("Enter value to search:");
scanf("%d", &key);
 linearsearch(A, size, key);
 getch();
}
void linearsearch(int A[], int n, int key)
 {
    int i, found=0;
 for(i=0; i<n; i++)
   {
      if(A[i]==key)
       {
     found=1;
       break;
      }
 }
   if(found==1)
printf("%d is found at %d position",
                key, i+1);
 else
printf("%d is not found", key);
  }
```

(11) Binary Search :- This is very fast searching method which searches a value in sorted list. It start searching from the middle element "mid" where

mid = (lower + upper) / 2

If the search value is matched middle element then the search terminate otherwise, searching will take place either in right side of middle element or in the left side of middle element depending upon the search value is larger or smaller than middle element. The process of dividing and searching continuous till search value is found or list is exhausted.

Search
p = 17
mid = 0+8/2
= 4
mid = 0+3/2
= 1



∴ Time complexity

$f(x) = T(N/2)$

$= N/2^k$

$\therefore f(x) = O(\log_2 n)$

Teacher Signature

Algorithm (Binary search)

```
void Binarysearch (int A[], int n, int key)
{
/* A[] is the array to search in */
/* n is the no: of element */
/* key holds value of search */
    int i, found = 0;
   int mid, lower, upper;
      lower = 0;
      upper = n - 1;
   while (lower + upper) / 2;
   if (A[mid] == key)
     {
     found = 1;
       break;
     }
 else if (key > A[mid])
  lower = mid + 1;
  }
  if (found == 1)
printf("%d is found at %d position",
          key, mid + 1);
 else
printf("%d is not found", key);
  }
```

Teacher Signature

PAGE NO.: 126
DATE: / /

Q:- Write C-program that accept 10-nos. from user and search a number using binary search?

Sol^n:-

```
#include <stdio.h>
#include <conio.h>
#define size 10
void binarysearch(int [], int, int);
void sort (int[], int);
void main()
{
 int a[size];
int key, i;
clrscr();
printf("Enter any 10 nos \n");
for(i=0; i<size; i++)
{
printf ("value %d:", i+1);
scanf ("%d", &a[i]);
}
sort (a, size);
printf(" Enter value to search:");
scanf("%d", &key);
binarysearch (a, size, key);
getch();
}
void Binarysearch(int a[], int 0, int key)
{
int i, foend = 0;
int mid, lower, upper;
 lower = 0;
```

Teacher Signature

```
    upper = n-1;
while(lower <= upper)
{
mid = (lower + upper)/2;
if(a[mid] == key)
{
found = 1;
break;
}
else if (key > a[mid])
lower = mid+1;
else
upper = mid-1;
}
if(found == 1)
printf("%d is position at %d position",
                    key, mid+1);
else
printf("%d is not found");
}
void sort(int A[], int n)
{
int i, j, temp;
for(i=0; i<(n-1); i++)
{
for(j=0; j<(n-1)-i; j++)
{
if (A[j]>A[j+1])
{
temp = A[j];
A[j] = A[j+1];
```

Teacher Signature

```
A[j+1] = temp;
}
}
}
}
```

Date 25/09/19

TREE

→ Tree is a non-linear data structure which organises the data items hierarchical form to establish one-to-many relationship among the data items.

following tree shows the parent-child relationship of a family.

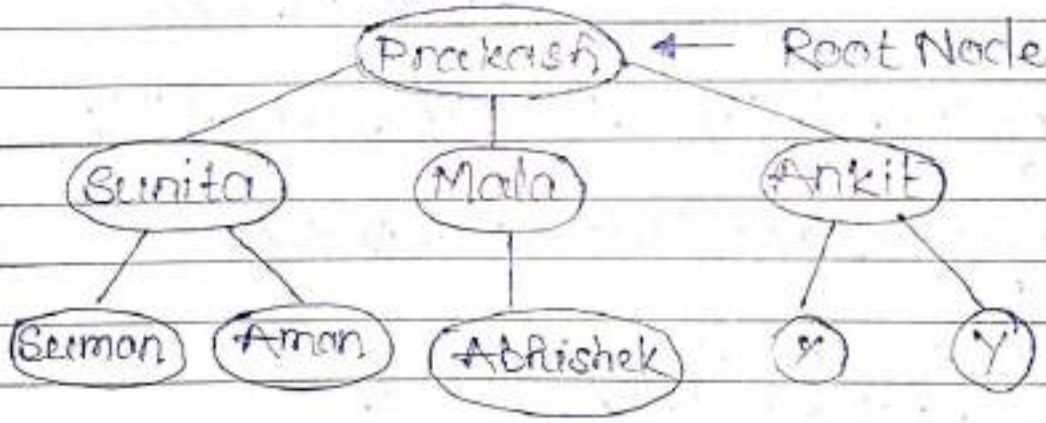


Fig :- Family Tree

If two nodes (X) and (Y) are connected with a line such that 'X' is above 'Y' then 'X' is said to be child of 'X'

Teacher Signature

A top most node having no parent is designated as "root" node. In the above family tree, a top-most node labelled "Prakash" is root node.

Definition of Tree :- A tree can be defined as finite set "T" of one or more nodes such that there is a designated node called "root" node and remaining nodes are partitioned into $n \geq 0$ disjoints set of nodes $T_1, T_2, \ldots T_n$. Each of these sets are tree once again. Hence, the definition is recursive. The disjoint set $T_1, T_2 \ldots T_n$ are called sub-trees of Root.

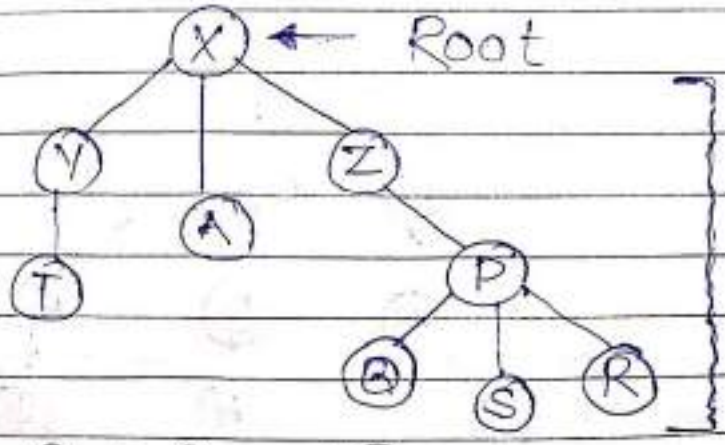


Fig:- General Tree

Teacher Signature

Terminologies of Tree - data structure

(1) Edge/Branch:- A line connecting two nodes is called an "edge" or "Branch".

(2) Leaf Node:- The nodes having no children are called leaf nodes. It is also called external or terminal nodes.

(3) Non-leaf Node:- The nodes having one or more children are called non-left nodes or internal nodes or non-terminal nodes.

(4) Sibling:- The nodes having same parent are called siblings.

(5) Degree:- The degree of a node is nothing but the numbers of children of that node.

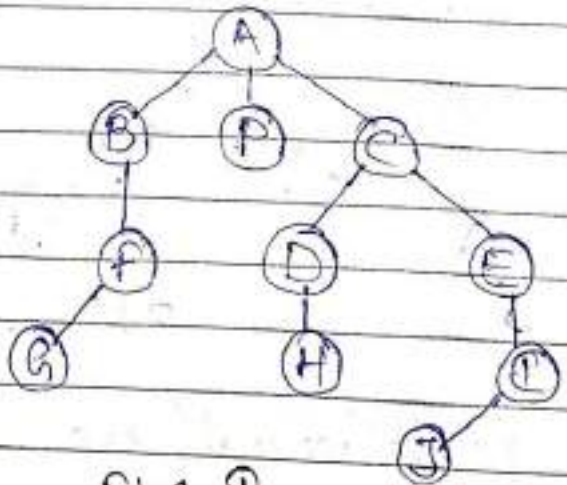


Fig1:- Tree

Teacher Signature_____

List of leaf Nodes - G, P, H, J
List of Non-leaf nodes - A, B, C, D, E, F, I
List of siblings :- B, P, C (sibling 1)
D, E (sibling 2)
Degree of Node A :- 3
Degree of Node H :- 0
Degree of Node I :- 1

(6) Height of a Node :- The longest path from a node to leaf node.
Height of node C = 3
Height of node B = 2

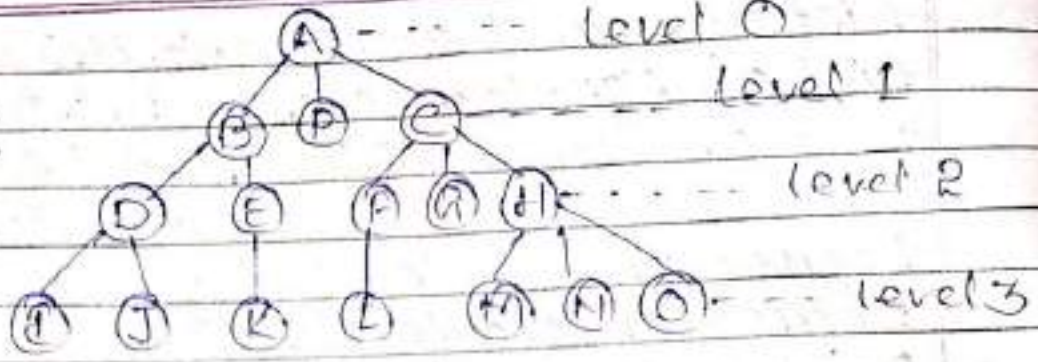
(7) Height of Tree :- The height of root node is called height of tree.
Height of Tree in fig 1 = 4

(8) Path :- The nodes occuring in traversal, from one node to another is called path.
eg:- A path from node C to J = C, E, I, J

(9) Path length :- The number of branch/edges occuring in a path is called path length.
eg:- Path length of path from C to J = 3

(10) Depth / level :- The level of root node is 0. The level of any other node is 1 more than the level of its parent.

Teacher Signature



* Type of Tree Data Structure
1. Binary Tree
2. Binary Search Tree (BST)
3. AVL - Tree
4. B - Tree
5. Heap - Tree.

1. Binary Tree :- Binary tree is a kind of tree data structure in which a node can have almost two children. One child is called left child or left sub-tree and another child is called right child or right sub-tree.

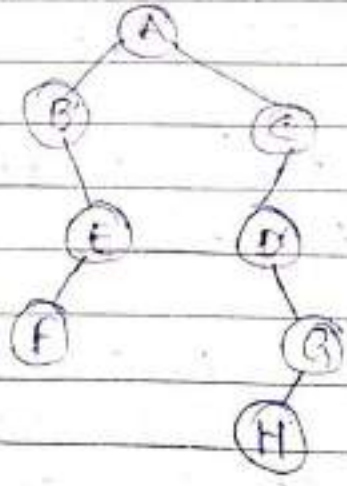


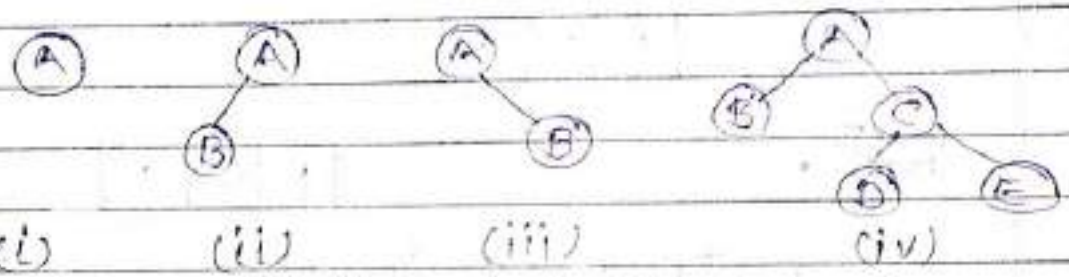
Fig:- Binary Tree

Teacher Signature

Recursive Definition of Binary Tree

A binary tree can be defined recursively as follows:

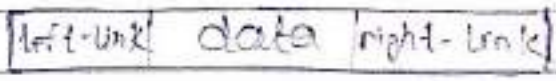
a) An empty is a binary tree
b) A binary tree consists of a node called "root", a left sub-tree and a right sub-tree, both of which are binary tree once again. Hence, the definition is recursive



(i) (ii) (iii) (iv)

Fig: List of Binary trees

Node of Binary Tree



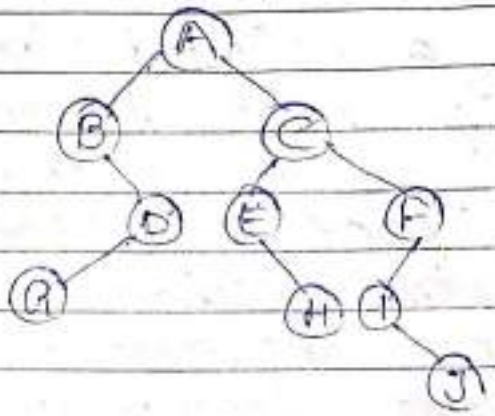
Representation of Binary Tree

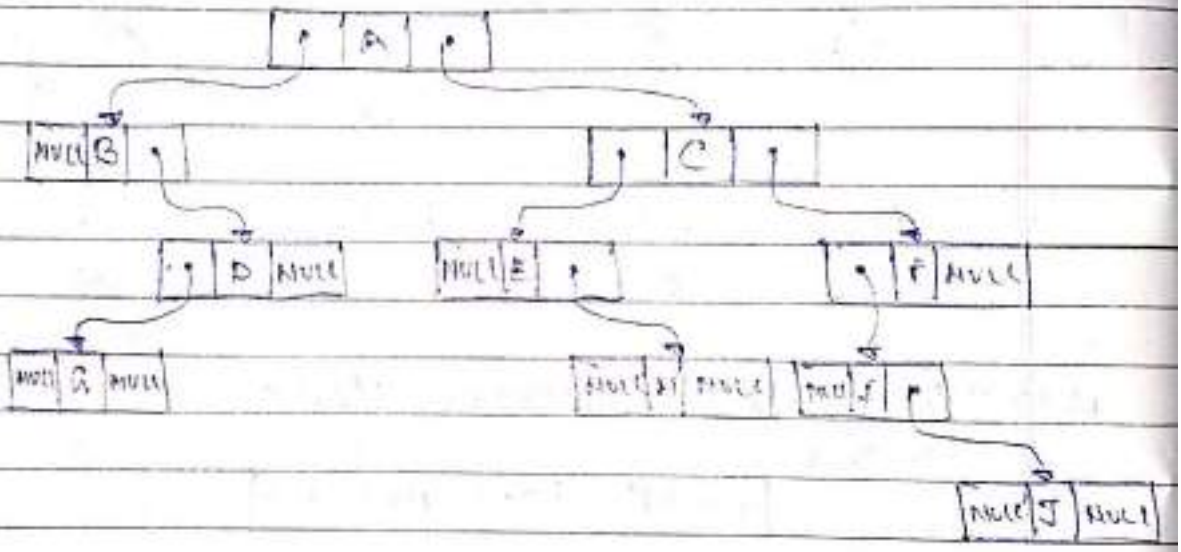
A binary tree can be represented either using

a) Linked - List
b) Array

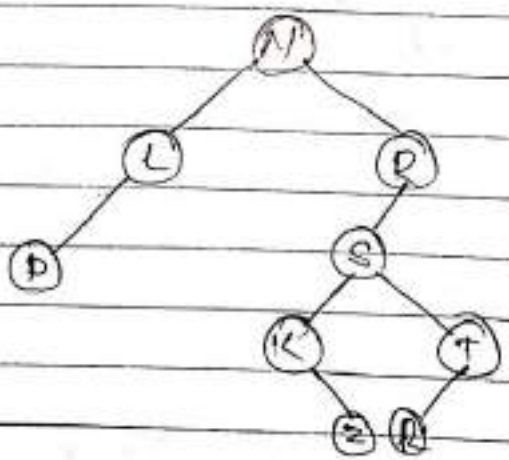
Teacher Signature

Linked - Representation of Binary Tree

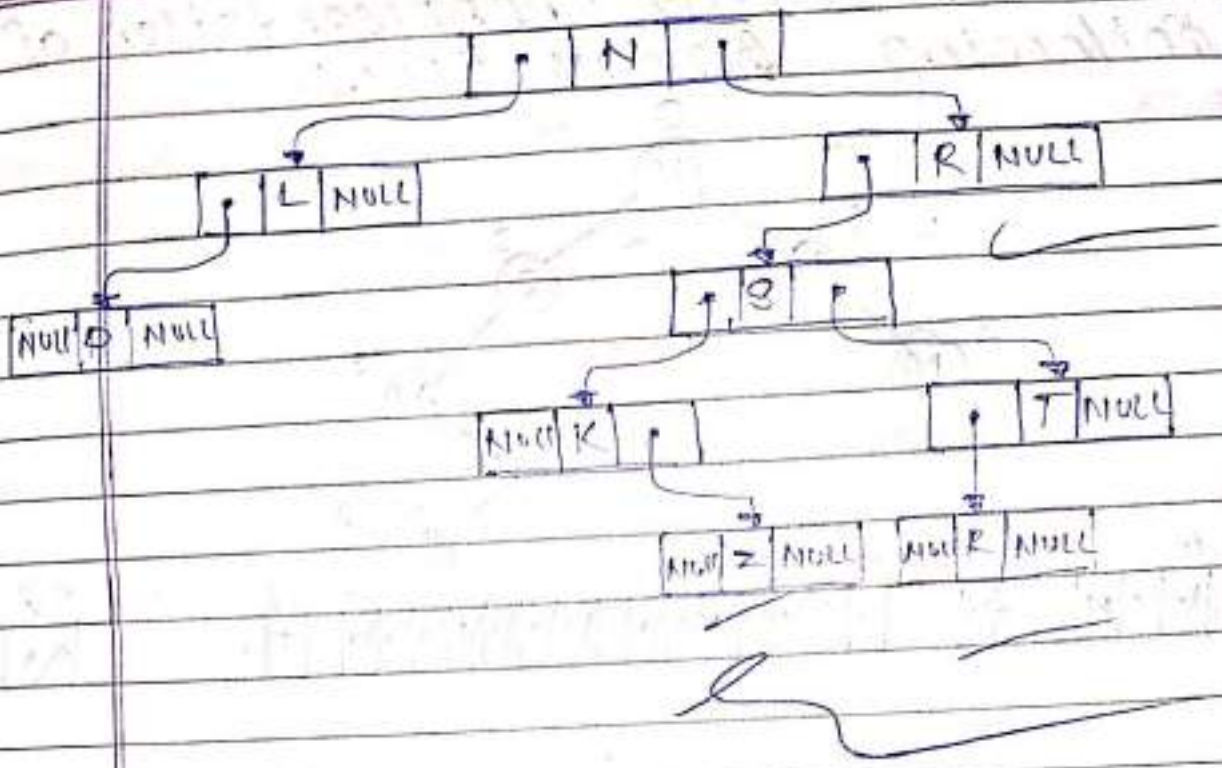




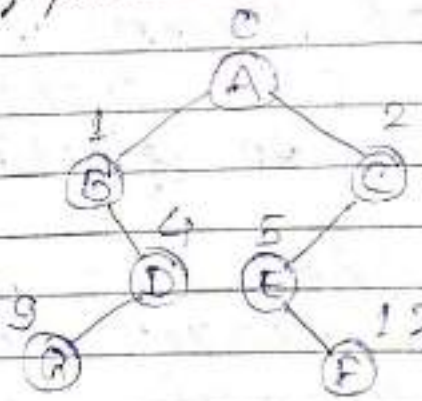
Q:- Give the Linked-representation of following Binary tree



Teacher Signature



Array Representation of Binary tree



Formeela

Left Child index = $2.i + 1$

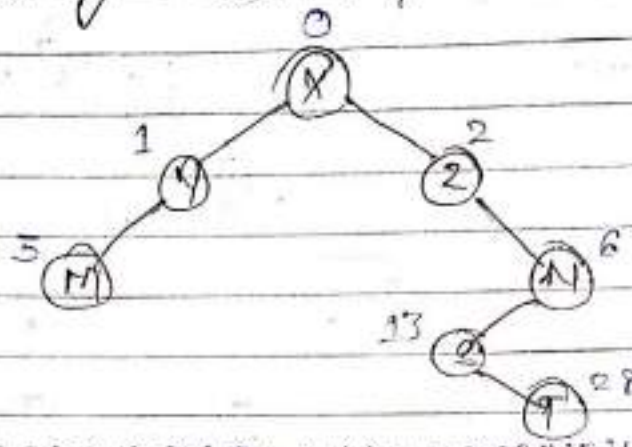
Right Child index = $2.i + 2$

where i is the index of parent

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26
A	B	C	\0	D	E	\0	\0	\0	G	\0	\0	F	\0	\0	\0	\0	\0	\0	\0	\0	\0	\0	\0	\0	\0	\0

Teacher Signature

Q:- Give the array representation of following binary tree:-



0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57	58
X	Y	Z	M	^	^	N	^	^	^	^	^	^	^	^	^	^	^	^	^	^	^	^	^	^	^	^	^	^	^	^	^	^	^	^	^	^	^	^	^	^	^	^	^	^	^	^	^	^	^	^	^	^	^	^	^	^	^	^	^	T	.....	^

Traversal of Binary Tree

Traversal refers to visiting each node exactly once.

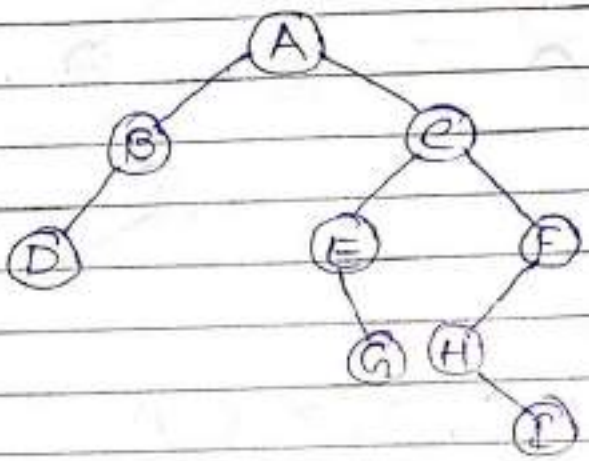
There are three techniques to traverse Binary tree. These are:-

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

(1) Inorder Traversal (Left-Root-Right) In this traversal technique left sub-tree is visited first then root node is visited and finally right sub-tree is traversed. Since left sub-tree and right sub-tree are another binary

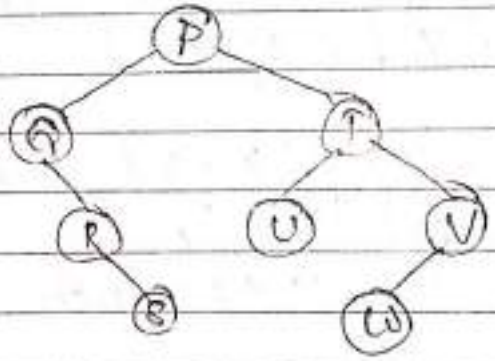
Teacher Signature

tree and therefore traversal from left sub-tree and right sub-tree is initiated in similar manner



Inorder Traversal output
D, B, A, E, G, C, H, I, F

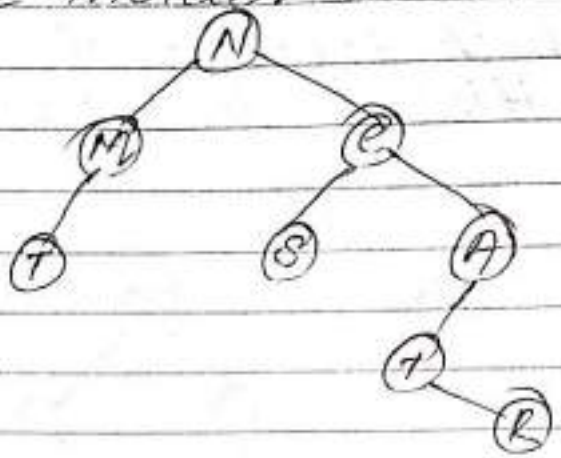
Q:- Give the inorder traversal output



Inorder Traversal Output
Q, R, S, P, U, T, W, V

Teacher Signature

Q7 Give the inorder traversal output



Output
T, M, N, S, C, T, R, A

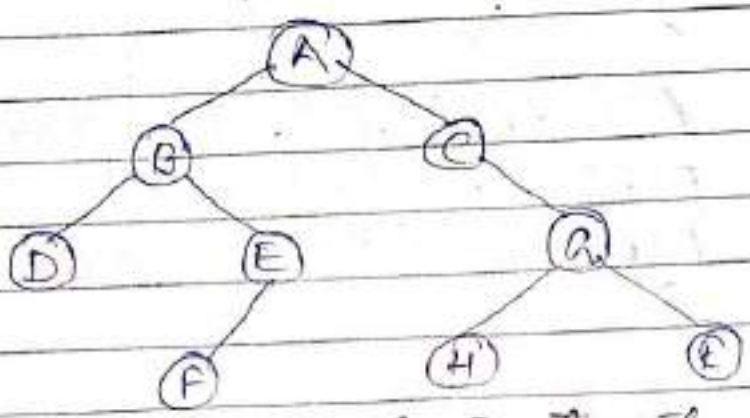
* Algorithm (Inorder Traversal)

```
void Inorder (Bnode * T)
{
/* T is a pointer to root node */
    if (T != NULL)
    {
     Inorder (T -> left);
    printf ("%c", T -> data);
     Inorder (T -> right);
     }
}
```

(2) Preorder Traversal (Root → left → RIGHT)
In this traversal technique root node is visited first then the left sub-tree is traversed and finally right sub-tree is visited. Since the left sub-tree

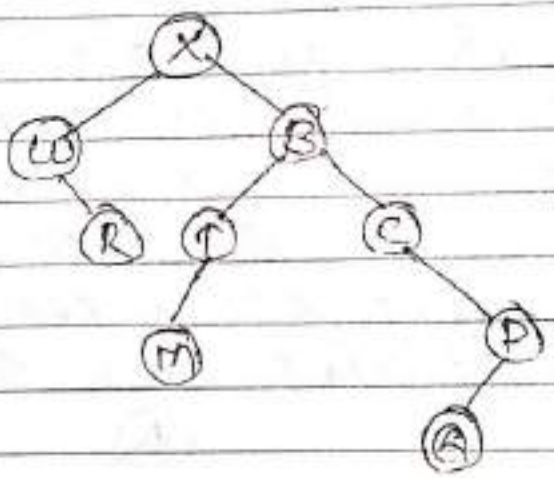
Teacher Signature

and right sub-tree are another binary tree and therefore traversal from left sub-tree and right sub-tree is initiated in similar manner.



Preorder Traversal Output
A, B, D, E, F, C, G, H, I

Q) Give the preorder traversal output



Output
X, W, R, B, T, M, C, P, Q

Teacher Signature

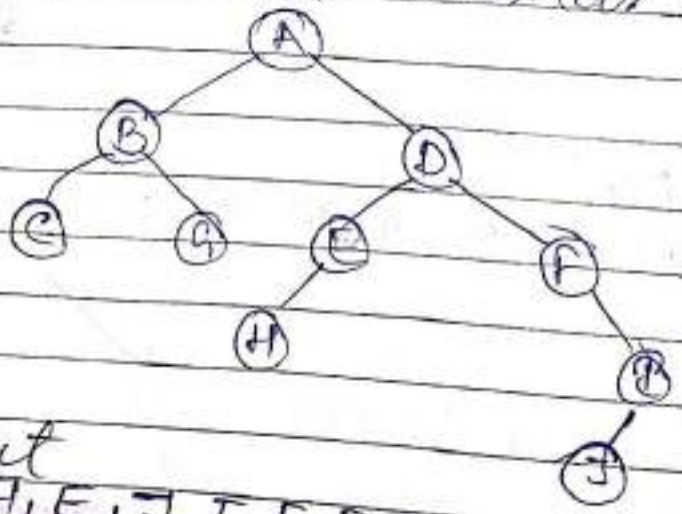
PAGE NO.: 140
DATE: / /

Algorithm (Pre-order Traversal)

```
void preorder(Bnode * T)
{
/* T is a pointer to root node */
    if (T != NULL)
    {
    printf("%c", T -> data);
    preorder(T -> left);
    preorder(T -> right);
    }
}
```

(3) Postorder Traversal (Left - Right - Root)

In this traversal technique left sub-tree is visited first then the right sub-tree is traversed and finally root node is visited. Since the left sub-tree and right sub-tree are another binary tree and therefore, traversal from left sub-tree and right sub-tree is initiated in similar manner.

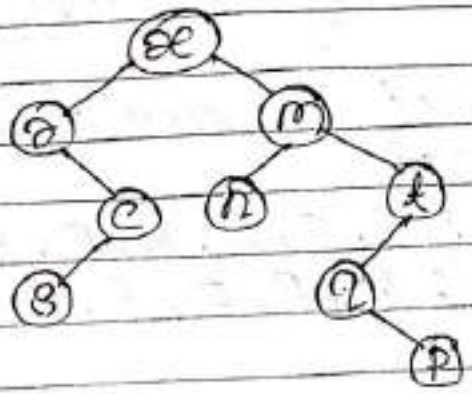


Output
C, G, B, H, E, J, I, F, D, A

Teacher Signature

PAGE NO.: 121
DATE: / /

Q?) Give the postorder traversal output



Output
S, C, a, n, p, q, t, m, x

Algorithm (Postorder)

```
void postorder (Bnode * T)
{
/* T is a pointer to root node */
        if (T != NULL)
        {
        postorder(T -> left);
        postorder(T -> right);
        printf("%c", T -> data);
        }
}
```

Teacher Signature

Date 03/10/19

PAGE NO.: 142
DATE: / /

Construction of Binary Tree from Traversal Output

Construct Binary Tree from the following traversal

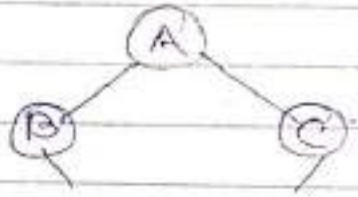
Inorder: B, F, E, A, D, H, G, C

preorder: A, B, E, F, C, D, G, H

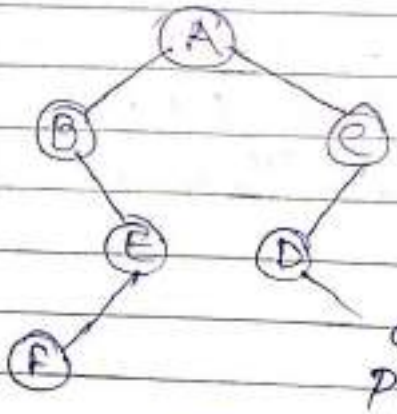
Sol^n:-

In: B, F, E
Pre: B, E, F

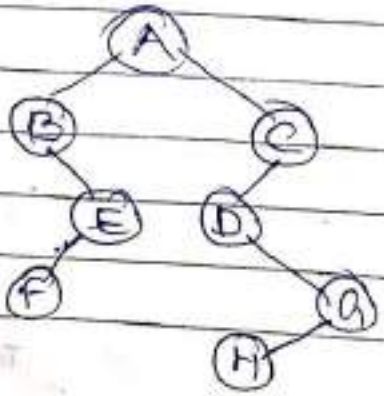
In: D, H, G, C
Pre: C, D, G, H



In: F, E In: D, H, G
Pre: E, F Pre: D, G, H



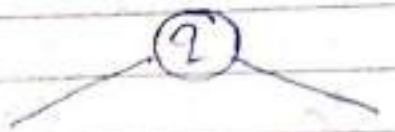
In: H, G
Pre: G, H



Teacher Signature_____

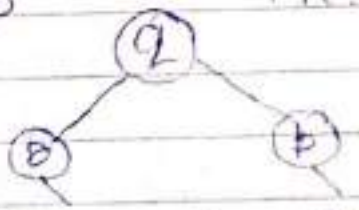
Q:- Construct Binary Tree from the following traversal
Inorder: r, a, v, b, q, p, s, m, t
preorder: q, r, v, a, b, p, m, s, t

Soln:-



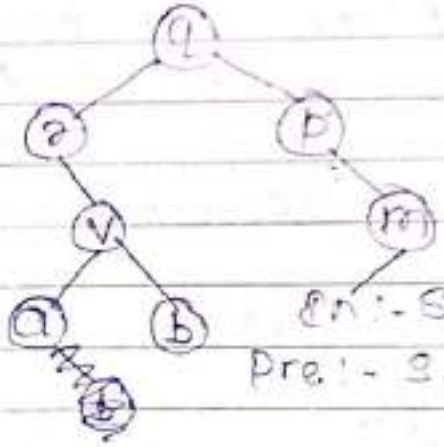
In :- r, a, v, b
Pre :- r, v, a, b

In :- p, s, m, t
Pre :- p, m, s, t

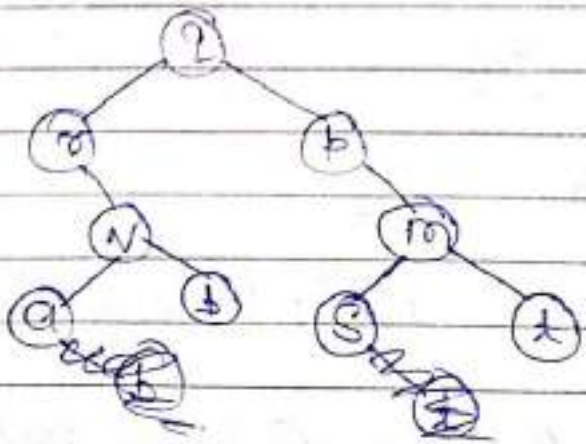


In :- a, v, b
Pre :- v, a, b

In :- s, m, t
pre :- m, s, t



In :- s, t
Pre :- s, t



Teacher Signature

Binary Search Tree (BST)

BST is a kind of binary tree in which the information in the node is maintained in some order. The information in the node is simply referred to as key.

A binary search tree is a binary tree which is either empty or following criteria are satisfied

1. All keys of the left sub-trees are smaller than the key in the root.
2. All keys of the right sub-trees are greater than the key in the root.
3. The left sub-trees & right sub-trees are binary search tree. once again. Hence, the definition is recursive.

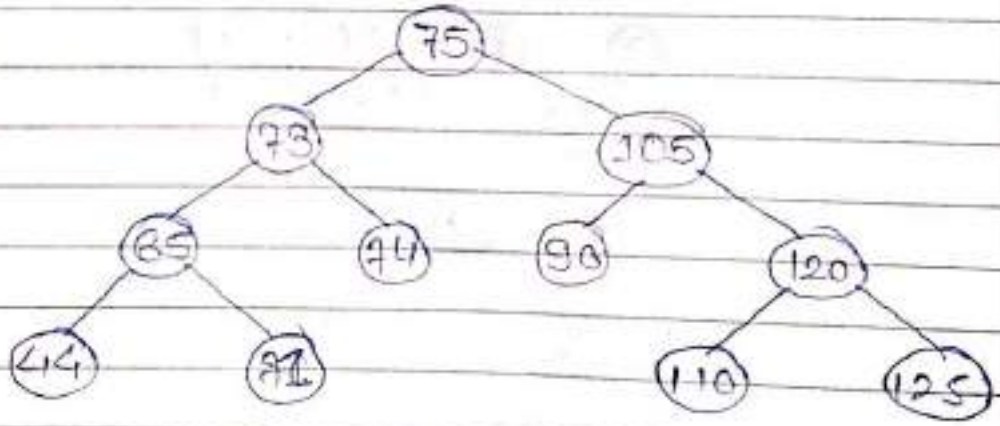


Fig:- Binary Search Tree

The main advantage of BST is that it makes the search process very fast. In each step of searching

Teacher Signature_____

Date 11/10/19

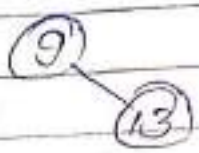
PAGE NO.: 145
DATE: / /

the list will reduced by almost half

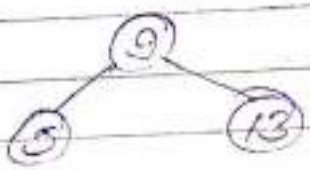
Q:- Construct BST with following keys:
[9, 13, 5, 32, 11, 6, 12, 2, 95, 65, 3]

Sol:- Add: 9

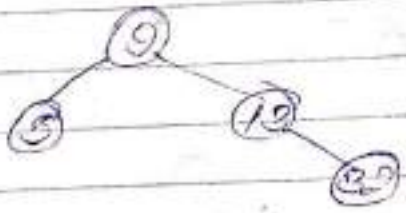
Add: 13



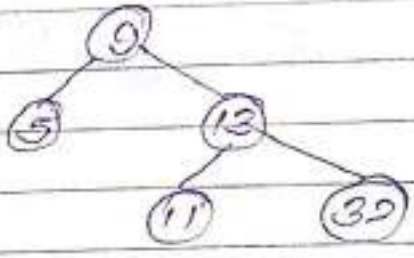
Add: 5



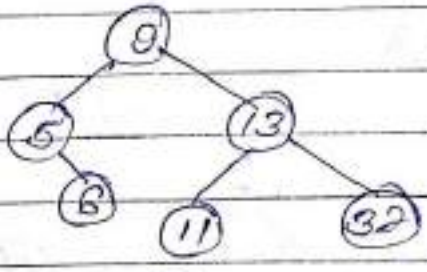
Add: 32



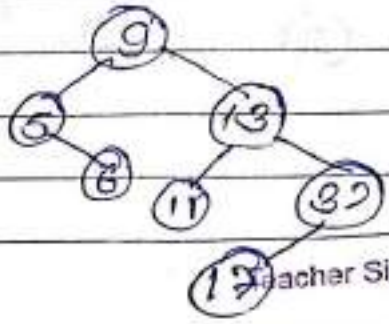
Add: 11



Add: 6

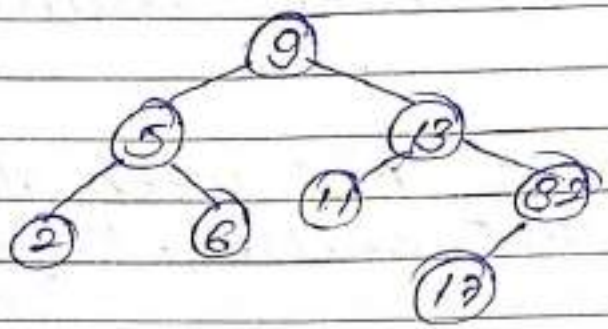


Add: 12

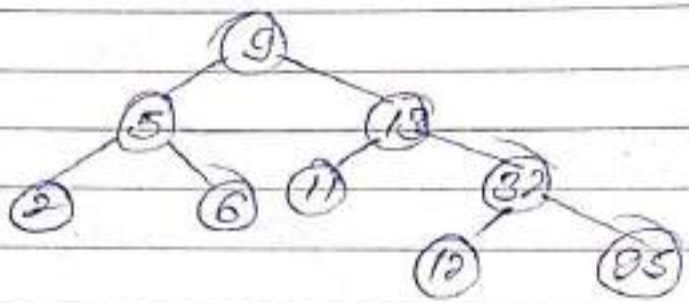


Teacher Signature

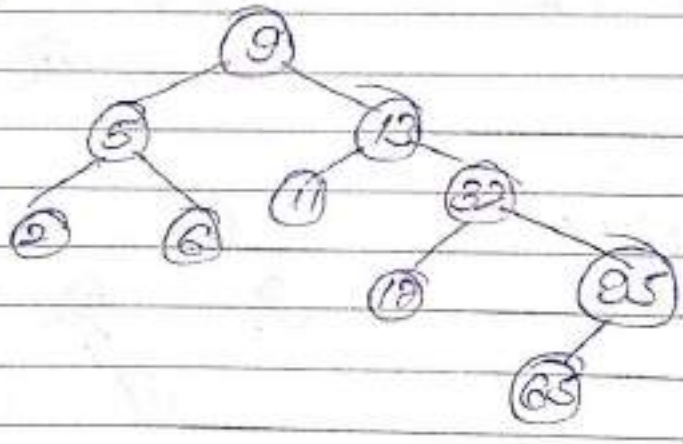
Add: 2



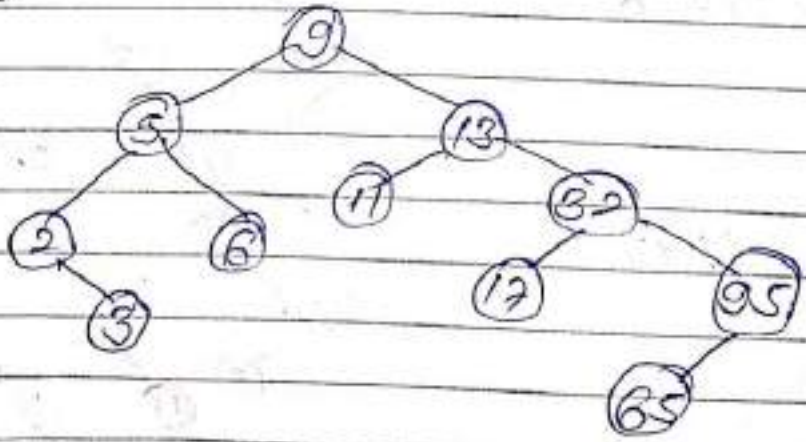
Add: 95



Add: 65



Add: 3



Teacher Signature_____

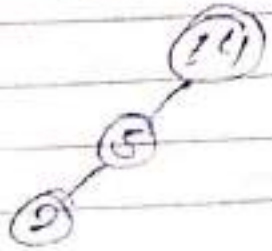
PAGE NO.: 147
DATE: / /

Q?: Construct BST with following keys:
14, 5, 2, 9, 75, 65, 2, 64, 11, 6, 19, 9572

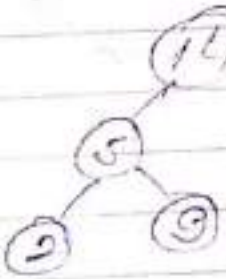
Sol?: Add: 14

Add: 5

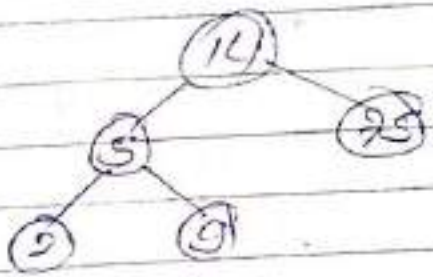
Add: 2



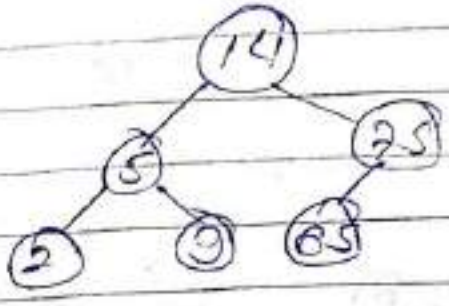
Add: 9



Add: 75

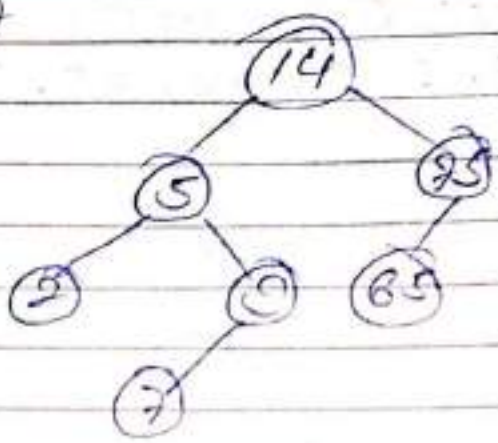


Add: 65

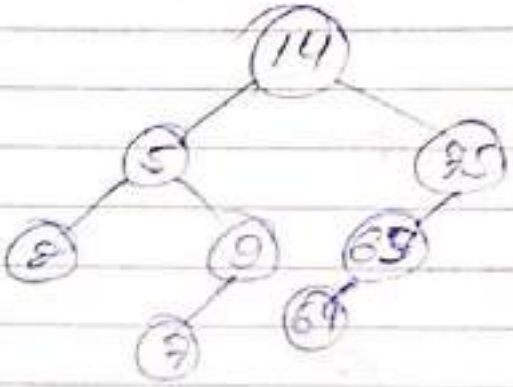


Teacher Signature

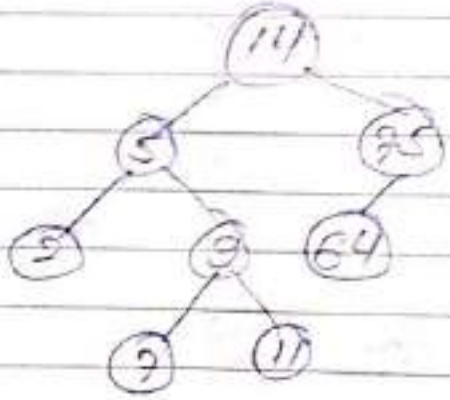
Add: 7



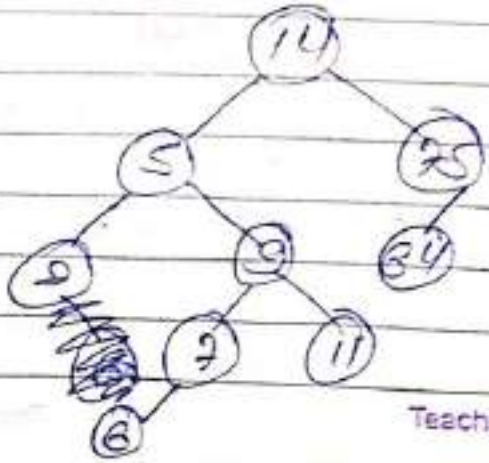
Add: 64



Add: 11

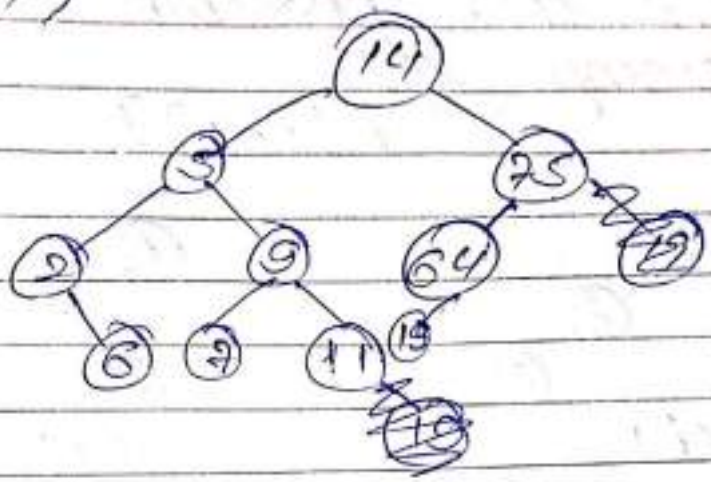


Add: 6

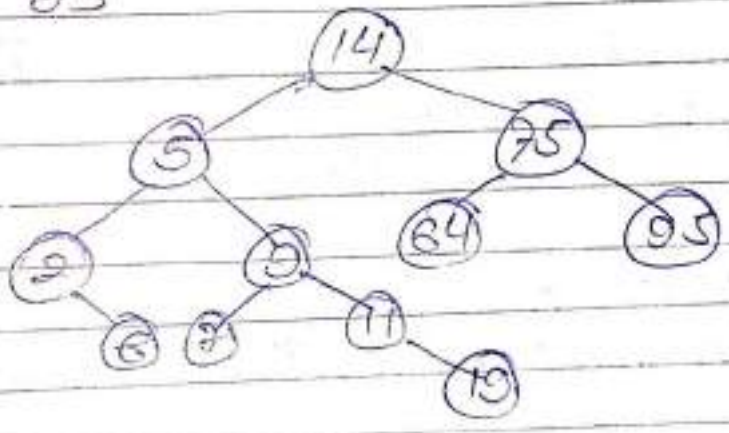


Teacher Signature_____

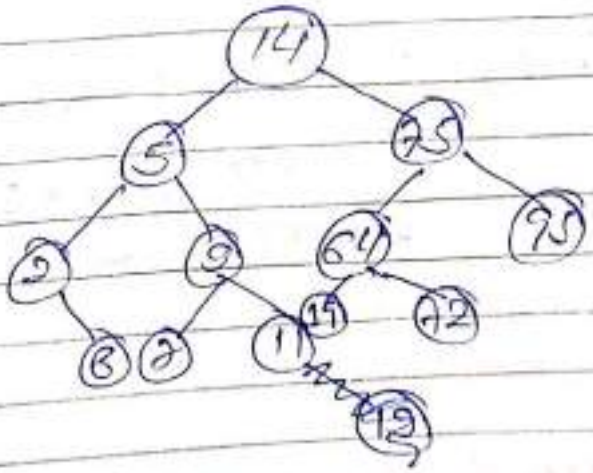
Add: 19



Add: 95



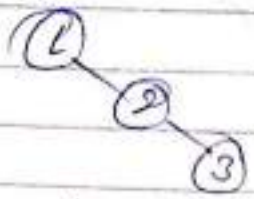
Add: 72



Teacher Signature

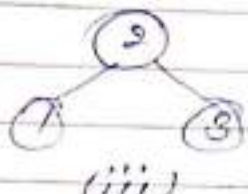
Q1:- Draw all possible BST with following keys:
[1, 2, 3]

Soln:-

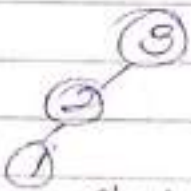


(i)

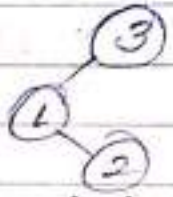
(ii)



(iii)



(iv)



(v)

Formula
Total diff. BST with n-keys

$$\boxed{{}^{2n}C_n - {}^{2n}C_{n-1}}$$

Let n = 3
Total BST - ${}^{6}C_3 - {}^{6}C_2$

$$= \frac{6 \times 5 \times 4}{3 \times 2 \times 1} - \frac{6 \times 5}{2 \times 1}$$

$$= 20 - 15$$

$$= 5$$

Q2:- Construct all possible BST with following keys.

Teacher Signature_____

[1, 2, 3, 4]

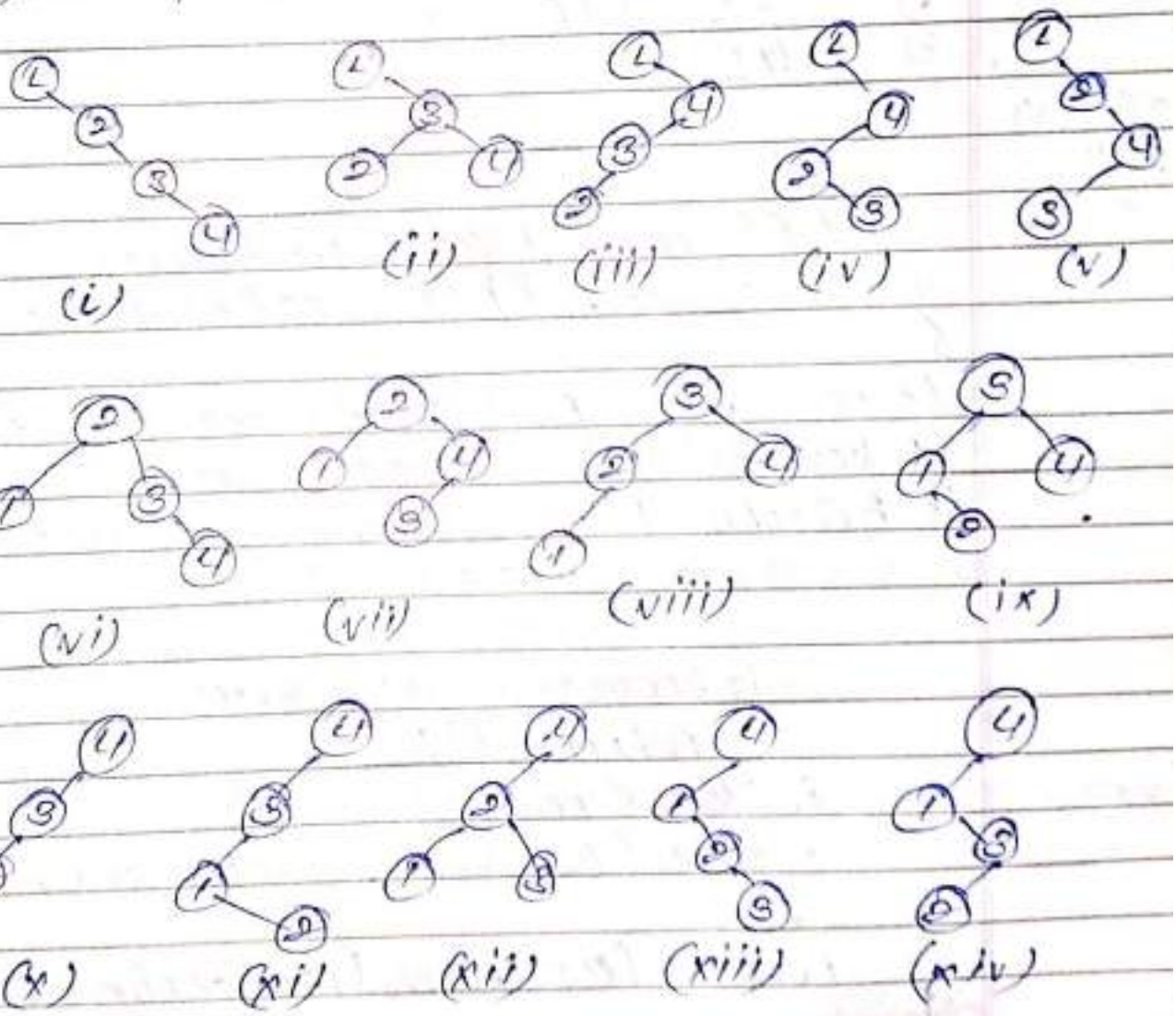
Soln:- $^{2n}C_n - {}^{2n}C_{n-1}$

$\Rightarrow {}^{8}C_4 - {}^{8}C_3$

$\Rightarrow \frac{8 \times 7 \times 6 \times 5}{4 \times 3 \times 2 \times 1} - \frac{8 \times 7 \times 6}{3 \times 2 \times 1}$

$\Rightarrow 70 - 56$

$\Rightarrow 14$



Teacher Signature

Q:- Construct all possible BST with following keys.
[9,3,5,2,4]

$$^{2n}C_n - {}^{2n}C_{n-1}$$

$$^{10}C_5 - {}^{10}C_4$$

$$\Rightarrow \frac{10\times9\times8\times7\times6}{5\times4\times3\times2\times1} - \frac{10\times9\times8\times7}{4\times3\times2\times1}$$

$$\Rightarrow 252 - 210$$

$$\Rightarrow 42$$

Date 12/10/19

BST Search Algorithm (Recursive)

```
int BSTSearch (BST *root, int key)
{
/* root is a pointer to root node */
/* key holds value to search */
/* Return 1 if found else returns 0 */
        if(root == NULL)
          return (0);
        if(root->data == key)
          return(1);
        if(key < root->data)
        return (BSTSearch (root->left, key));
         else
        return (BSTSearch (root->right, key));
}
```

Teacher Signature

BST Search Algorithm (Non-Recursive)

```
int BSTSearch(BST *root, int key)
{
/* root is a pointer to root node */
/* key holds value to search */
/* Return 1 if found else return 0 */
while(root != NULL && root->data != key)
{
    if(key < root->data)
    root = root->left;
    else
    root = root->right;
}
if( root == NULL)
  return (0);
  else
  return (1);
}
```

Algorithm to insert Node in BST
(or)
Create BST (Recursive)

```
void BSTInsert(BST *root, BST *newnode)
{
        if(root == NULL)
        {
        root->data = newnode->data;
        root->left = root->right = NULL;
        return;
        }
```

Teacher Signature

```
if(root->data> newnode->data)
{
    if(root->left==NULL)
     root->left = newnode;
    else
   BSTInsert(root->left, newnode));
}
else
{
   if(root->right==NULL)
   root->right = newnode;
   else
   BSTInsert(root->right, newnode));
}
}
```

Algorithm to insert Node in BST
(or)
Create BST (Non-Recursive)

```
void BSTInsert(BST *root, BST *newnode)
{
        if(root==NULL)
        {
    root->data = newnode->data;
    root->left = root->right = NULL;
     return;
        }
while(root!=NULL)
{
 if(root->data>newnode->data)
```

Teacher Signature

```
{
    if(root->left==NULL)
    {
    root->left=newnode;
     root=NULL;
    }
  else
  root=root->left;
}
else
{
   if(root->right==NULL)
   {
     root->right=newnode;
     root=NULL;
    }
 else
    root=root->right;
  }
 }
}
```

Teacher Signature

Date 14/10/19

PAGE NO.: 156
DATE: / /

BST Operation
Main Menu
1. Create BST
2. Insert in BST
3. Inorder
4. Pre-orders
5. Postorders
6. Exit
Choice (1-6):

```
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
#include <alloc.h>
typedef struct BSTNode
{
int data;
struct BSTNode *left, *right;
}BST;
BST* BSTCreate (BST*);
void BSTInsert (BST*);
void Inorder (BST*);
void Preorder (BST*);
void Postorder (BST*);

void main()
{
BST *root = NULL;
int n;
while (1)
{
```

Teacher Signature

```
clrscr();
printf("\t\t BST Operation \n");
printf("\t\t MAIN MENU \n");
printf("\t 1. Create BST \n");
printf("\t 2. Insert in BST \n");
printf("\t 3. Inorder Traversal \n");
printf("\t 4. Preorder Traversal \n");
printf("\t 5. Postorder Traversal \n");
printf("\t 6. Exit \n");
scanf("%d", &n);
switch(n)
{
case 1:
root = BSTCreate(root);
break;
case 2:
BSTInsert(root);
break;
case 3:
if(root == NULL)
{
printf("BST is empty...");
}
else
Inorder(root);
break;
case 4:
if(root == NULL)
{
```

Teacher Signature

```
printf("BST is empty...");
}
else
Preorder(root);
break;
case 5:
if(root == NULL)
{
printf("BST is empty...");
}
else
Postorder(root);
break;
case 6:
exit(0);
break;
default:
printf("Invalid choice...");
}
getch();
}
}

/* Function to create BST */
BST* BSTCreate(BST *r)
{
BST *newnode, *ptr;
char ch;
if(r != NULL)
{
printf("BST is already created. Do
insert now...");
```

Teacher Signature

```
return(r);
}
do
{
newnode = (BST*)malloc(sizeof(BST));
printf("Input Value:");
scanf("%d", &newnode->data);
newnode->left = newnode->right = NULL;
if(r==NULL)
{
r= newnode;
}
else
{
ptr=r;
while(ptr!=NULL)
{
if(ptr->data->newnode->data)
{
if(ptr->left==NULL)
{
ptr->left= newnode;
ptr= NULL;
}
else
ptr=ptr->left;
}
else
{
if(ptr->right==NULL)
{
```

```
ptr->right = newnode;
ptr = NULL;
}
else
ptr = ptr->right;
}
}
}
printf("Do u wish to enter more(y/n):");
fflush(stdin);
ch = getchar();
}while(ch == 'Y' || ch = 'y');
return(r);
}
/* Function to insert into BST */
/* void BSTInsert (BST *r)
{
BST *newnode, *ptr;
if (r == NULL)
{
printf("BST is not created. Plz create first...");
return;
}
newnode = (BST*)malloc(sizeof(BST));
printf("Input Value to Insert:");
scanf("%d", &newnode->data);
newnode->left = newnode->right = NULL;
ptr = r;
while(ptr != NULL)
{
if(ptr->data > newnode->data)
{
```

Teacher Signature_____

```
if(ptr->left==NULL)
{
ptr->left=newnode;
ptr=NULL;
}
else
ptr=ptr->left;
}
else
{
if(ptr->right==NULL)
{
ptr->right=newnode;
ptr=NULL;
}
else
ptr=ptr->right;
}
}
}
/* function for inorder traversal */
void inorder(BST *r)
{
BST *T;
T=r;
if(T!=NULL)
{
inorder(T->left);
printf("%4d",T->data);
inorder(T->right);
}
}
```

Teacher Signature

```
/* Function for Preorder Traversal */
void Preorder (BST *T)
{
if (T != NULL)
{
printf ("%4d", T->data);
Preorder (T->left);
Preorder (T->right);
}
}

/* Function for Postorder Traversal */
void Postorder (BST *T)
{
if (T != NULL)
{
Postorder (T->left);
Postorder (T->right);
printf ("%4d", T->data);
}
}
```

Teacher Signature_____

Date 15/10/19

PAGE NO.: 163
DATE: / /

AVL - Tree

It is a kind of BST whose height ~~Balance~~ is kept balanced. Hence it is also called Height Balanced tree. The name AVL has been given on the name of its inventor called "Adelson-Velsakii & Landi"

An AVL-Tree has an extra field in each node to store balance factor (BF) where BF = Height of left subtree - Height of Right subtree

A BST to be an AVL-Tree only when the balance factor is in the set {1, 0, -1}

Node of AVL

```
struct AVLTree
{
  int data;
  int bf;
  struct AVLTree *left, *right;
};
```

Construct AVL Tree from following keys

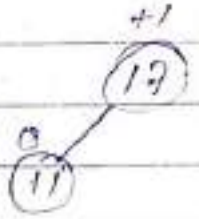
17, 11, 4, 9, 6, 15, 19, 35, 48, 2, 7

Teacher Signature

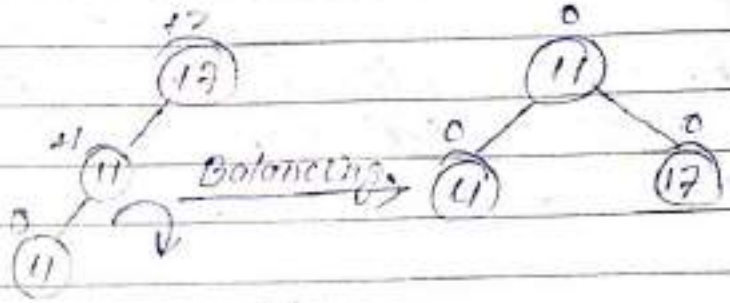
17, 11, 4, 9, 6, 15, 19, 35, 48, 2, 7

Sol:- Add: 17

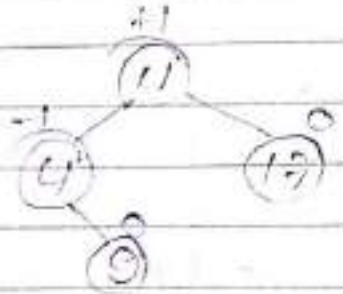
Add: 11



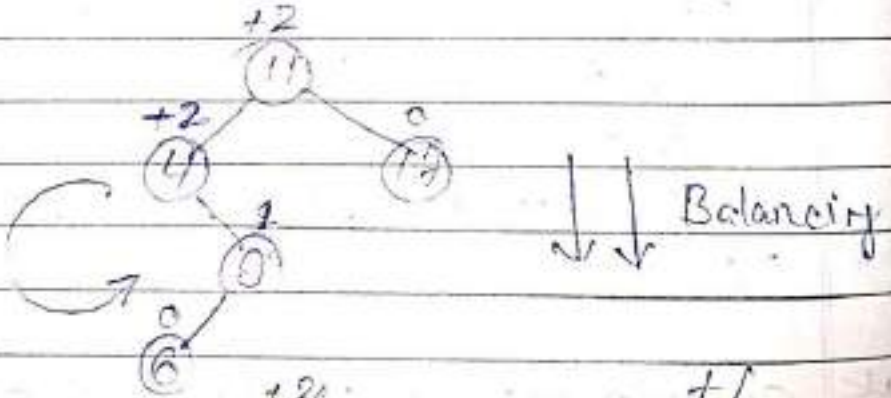
Add: 4



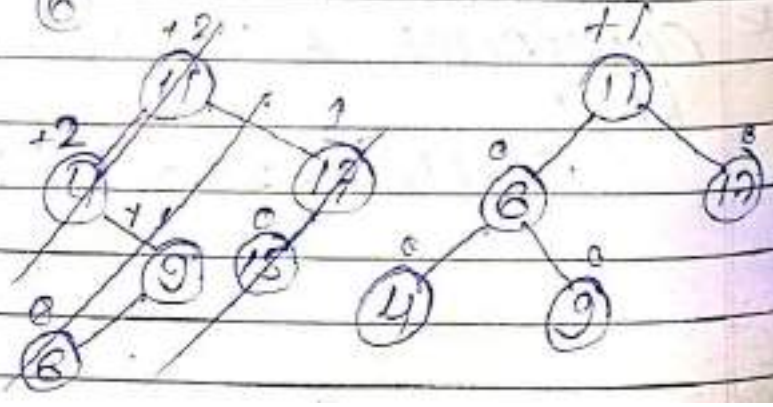
Add: 9



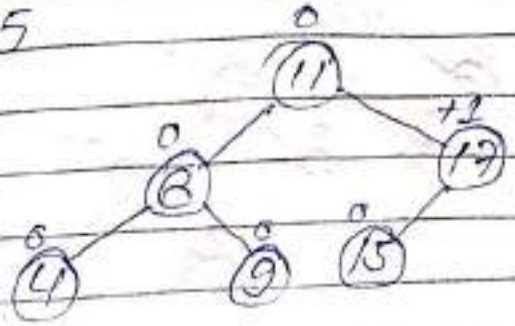
Add: 6



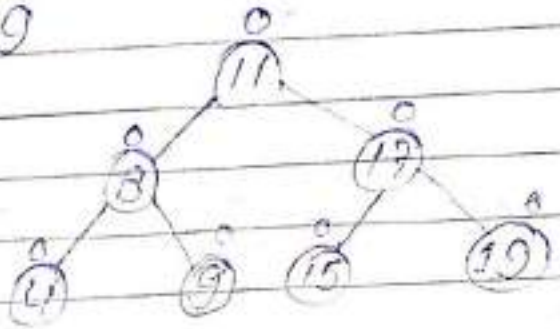
Add: 15



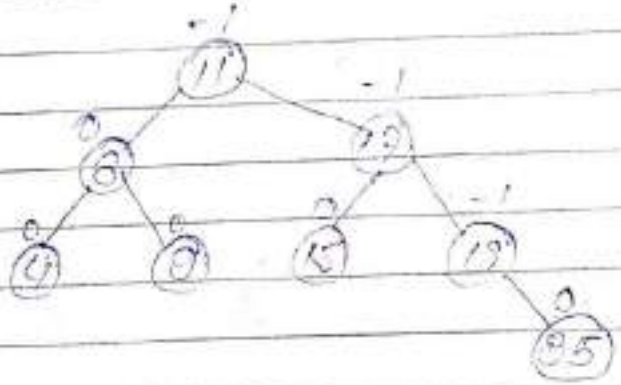
Add: 15



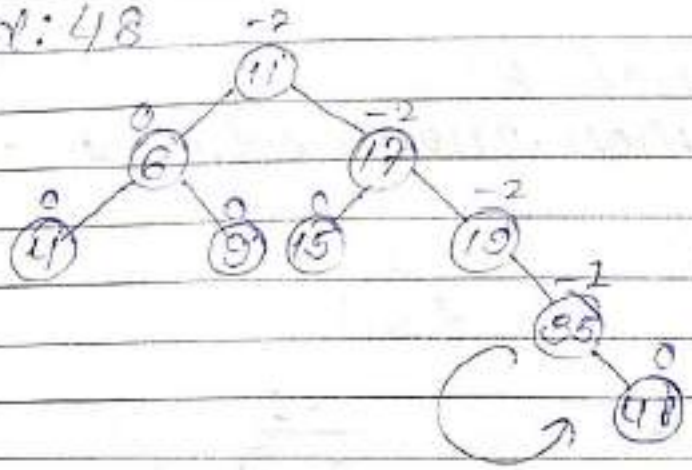
Add: 19



Add: 55

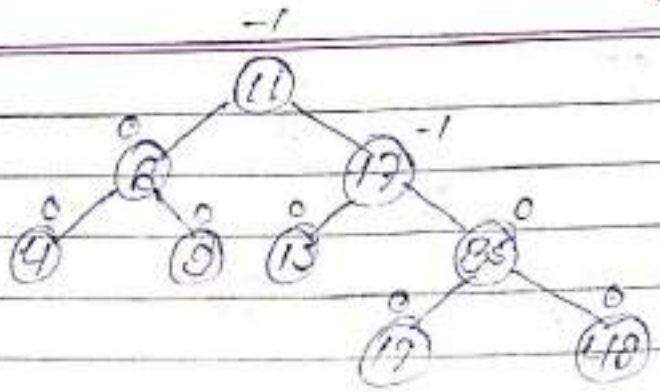


Add: 48

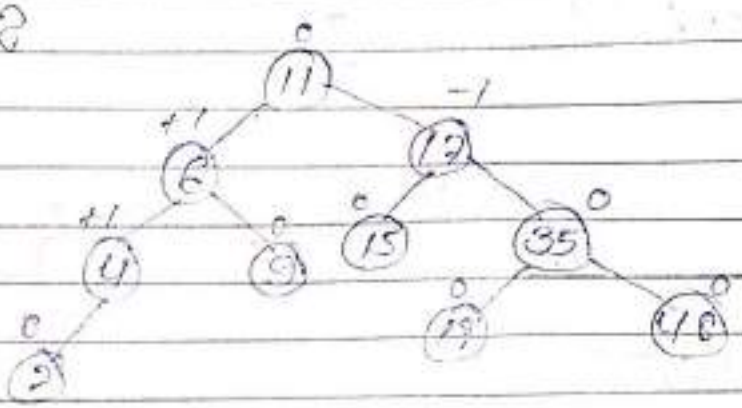


Balancing

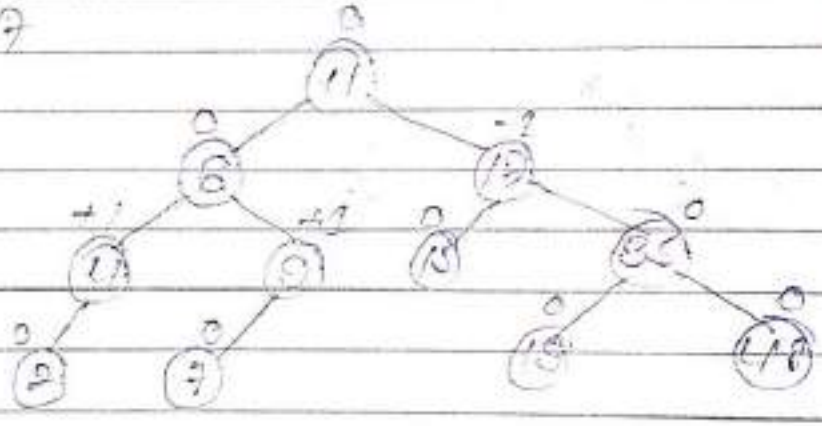
Teacher Signature



Add: 2



Add: 7



Date 16/10/19

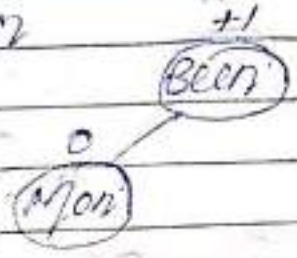
Q: Construct AVL
Sun, Mon, Tue, Wed, Thu, Fri, Sat

Add: Sun

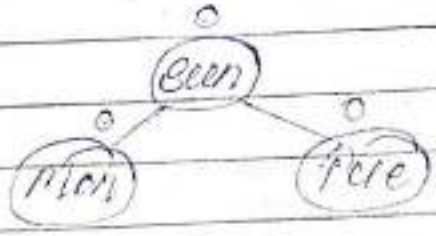
Add: Mon

Teacher Signature

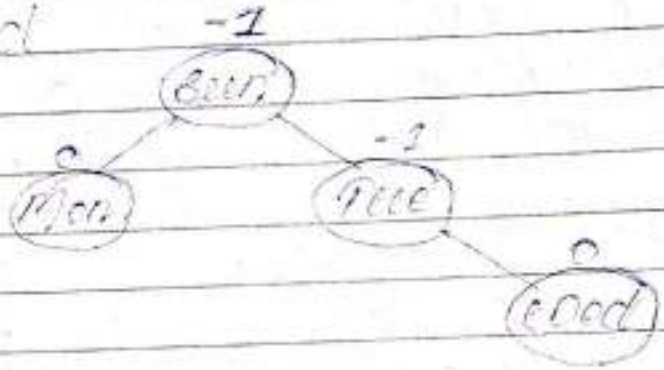
Add: Mon



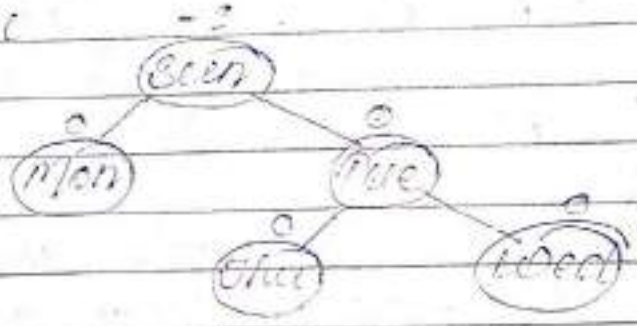
Add: Tue



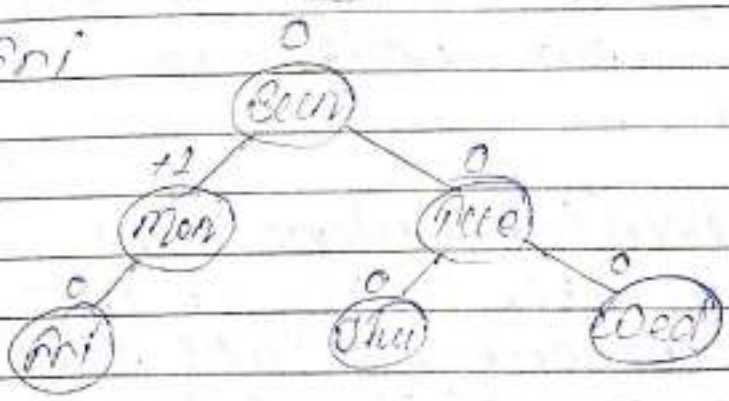
Add: Wed



Add: Thu

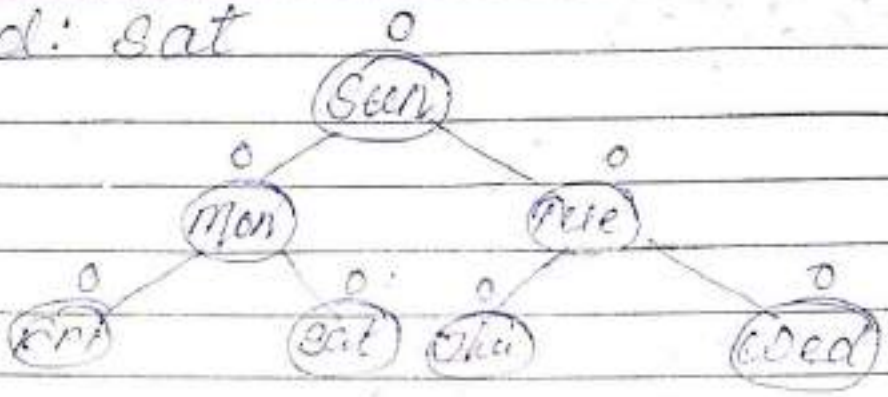


Add: Fri



Teacher Signature

Add: sat



B-Tree:- B-Tree is a kind of Binary tree in which a node can have more than one keys. It is suitable for organising data on secondary memory like disc.

A B-Tree of order "m" will have the following properties:

1. All non-leaf nodes except root node must have atleast $\lceil m/2 \rceil$ children and at most "m"-children.
2. If a node has "n"-number of children then it must have (n-1) keys.
3. Keys in the node are arranged in non-decreasing order.
4. All leaf nodes are at the same level.

Creation Algorithm

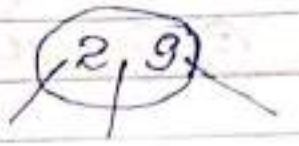
- Key are inserted in leaf node.
- If a node is full then the middle key get shifted to prevent node. If there is no parent then root is create

Date 18/10/19

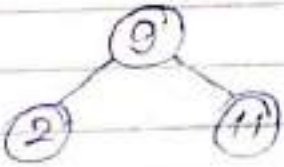
PAGE NO.: 123
DATE: / /

Q:- Construct B-Tree of order-3 following
9, 2, 11, 7, 32, 41, 6, 5, 19, 20, 15

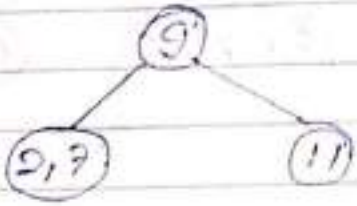
Add: 9, 2



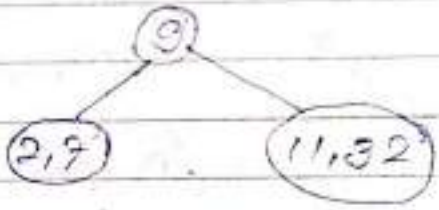
Add: 11



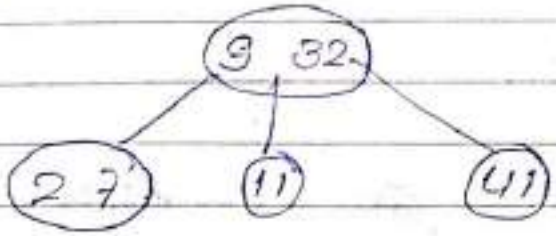
Add: 7



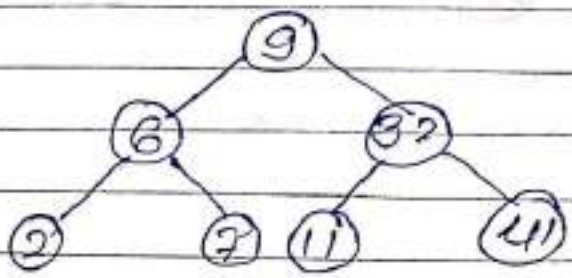
Add: 32



Add: 41

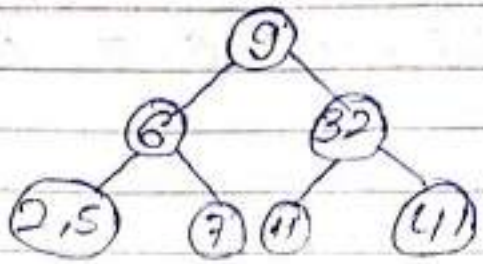


Add: 6

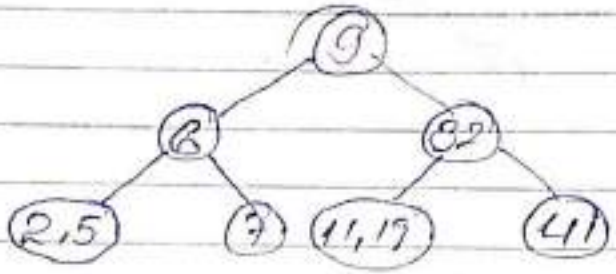


Teacher Signature

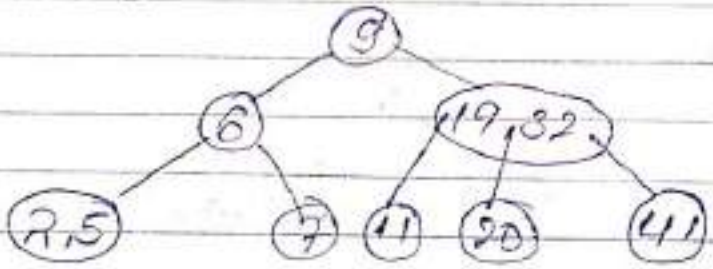
Add: 5



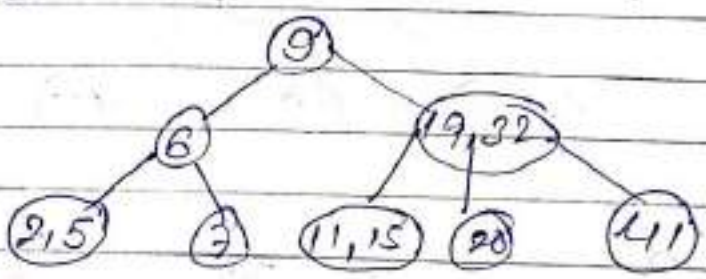
Add: 19



Add: 20



Add: 15

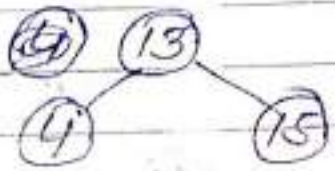


Teacher Signature

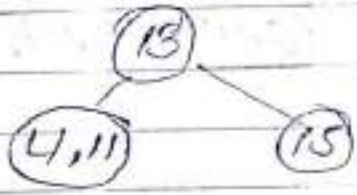
Q: Construct B-Tree of order-3 following keys:-
15, 4, 13, 11, 2, 6, 75, 71, 44, 90, 7, 1, 9

Add: 15, 4

Add: 13

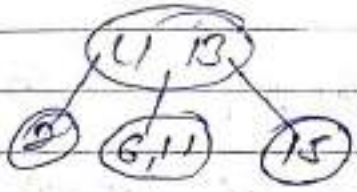


Add: 11

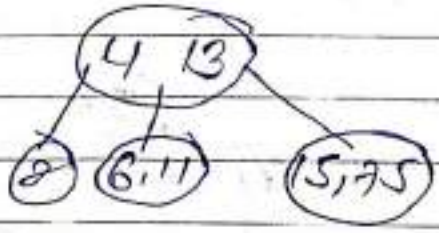


Add: 2

Add: 6

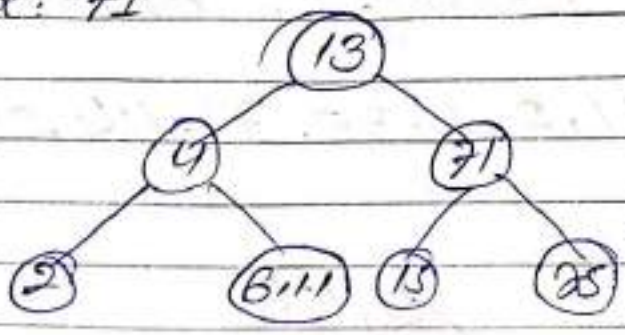


Add: 75

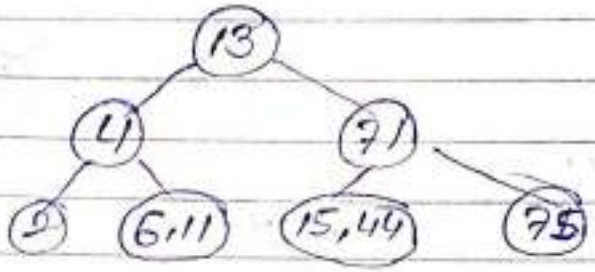


Teacher Signature

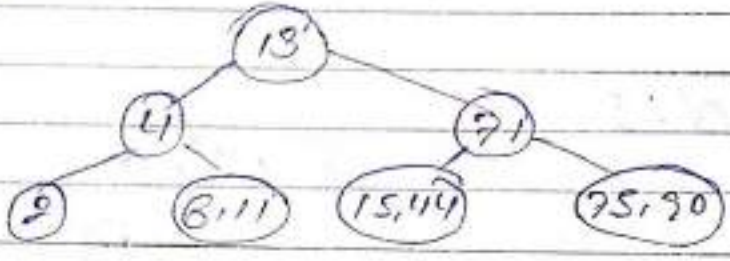
Add: 71



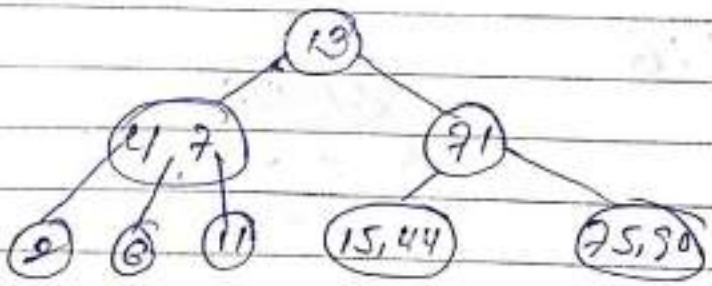
Add: 44



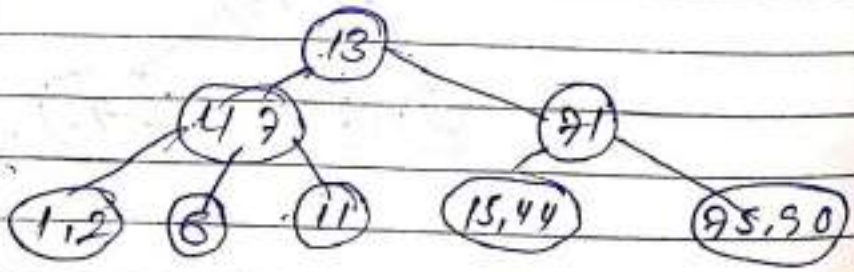
Add: 90



Add: 7

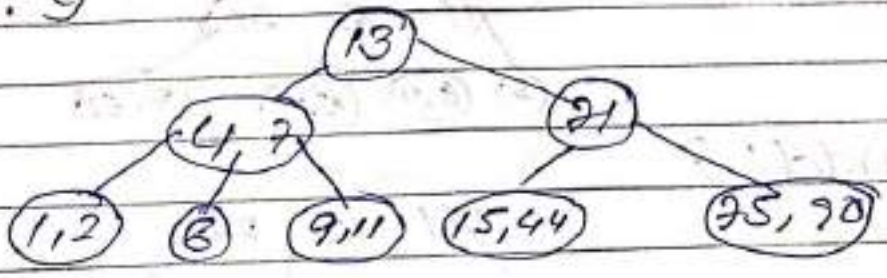


Add: 1



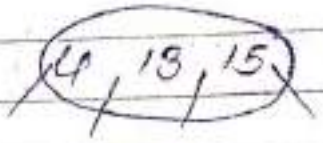
Teacher Signature

Add: 9

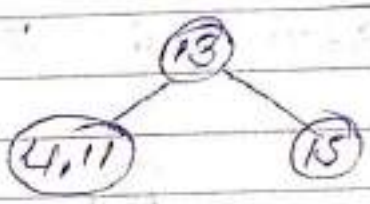


Q1: Construct B-Tree of order-4 following keys:
15, 4, 13, 11, 2, 6, 75, 71, 44, 90, 7, 1, 9

Soln:-

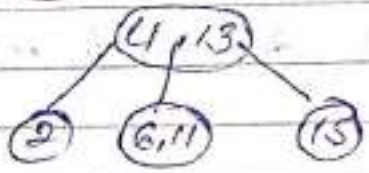


Add: 4

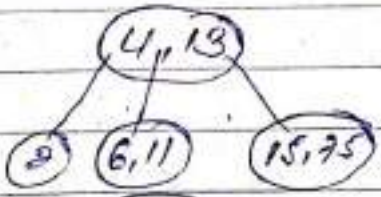


Add 2

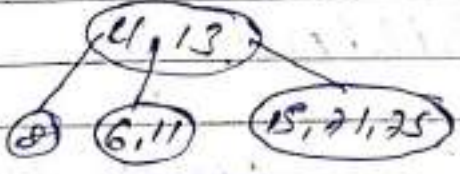
Add: 6



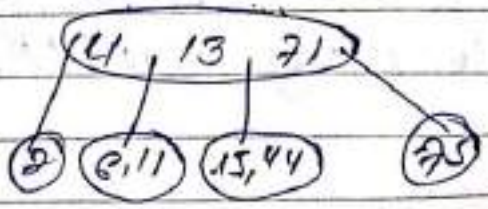
Add: 75



Add: 71

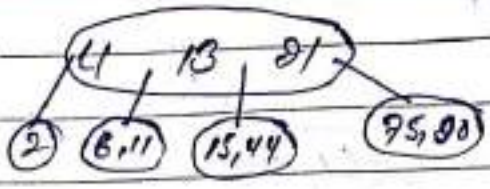


Add: 44

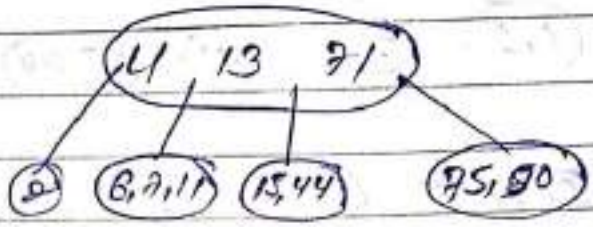


Teacher Signature

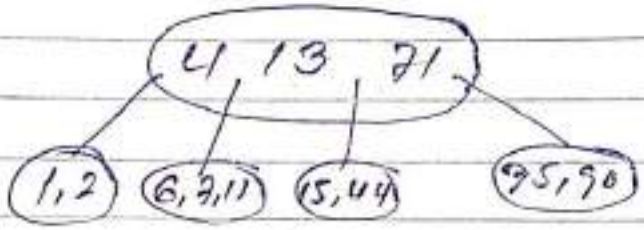
Add: 90



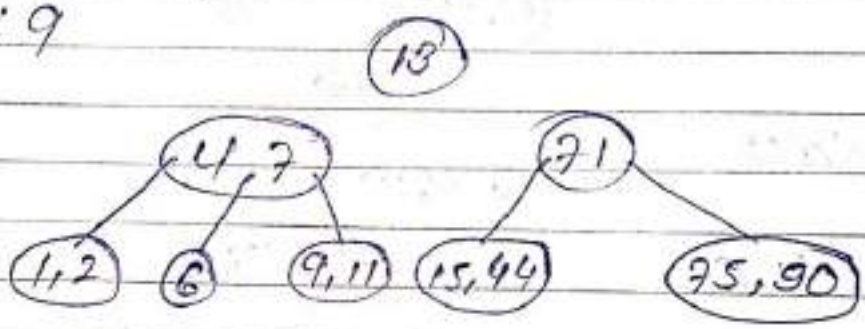
Add: 7



Add: 1



Add: 9



* Full Binary Tree (2-Tree)

A kind of binary tree in which a node can have either no children or 2-children.

This kind of binary tree is specially used for representing arithmetic expression.

Teacher Signature

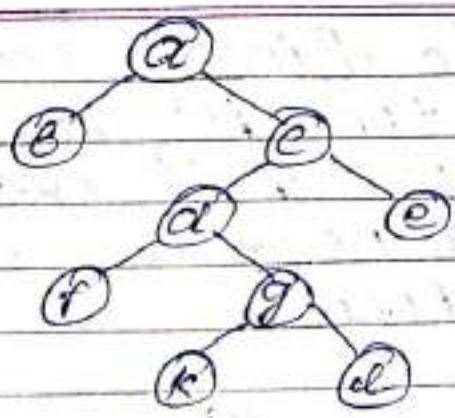


Fig:- 2-Tree.

Q:- Represent following exp° using tree.
9+3*7-2

Sol:- 9+3*7-2
T1

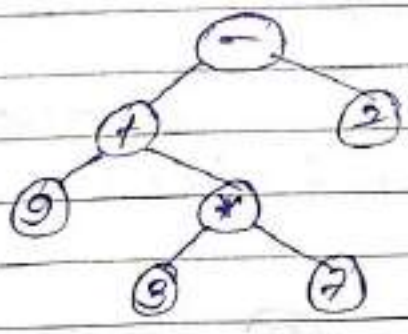
=) 9+T1-2
T2

=) T2-2

=) -, T2, 2

=) -, +, 9, T1, 2

=) -, +, 9, *, 3, 7, 2



Teacher Signature

✍ Complete Binary Tree:- A kind of binary tree in which every level is full except possibly the last node is called complete Binary tree.

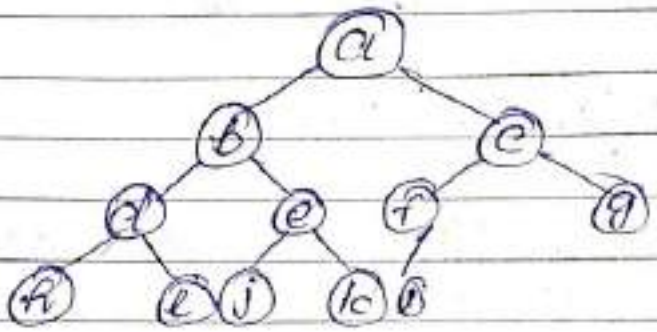


Fig:- Complete Binary Tree

Date 19/10/19

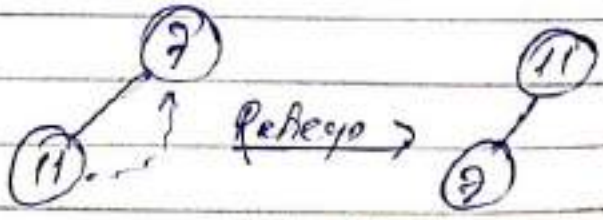
The nodes are added from "top-to-bottom" & "left-to-right".

✍ Heap:- Heap is a complete Binary tree in which a key of root is larges than the key of its children. Heap is specially used in Heapsort

Q?: Construct Heap with following keys
7, 11, 14, 2, 7, 15, 18, 35, 3, 12

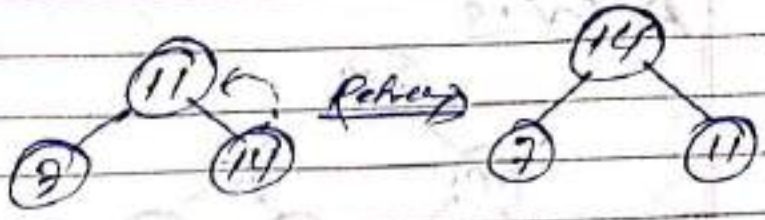
Add: 7

Add: 11

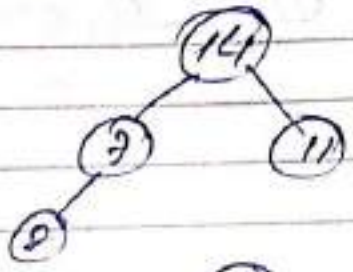


Teacher Signature

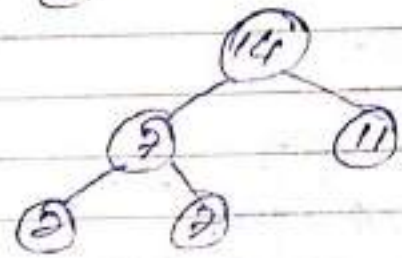
Add: 14



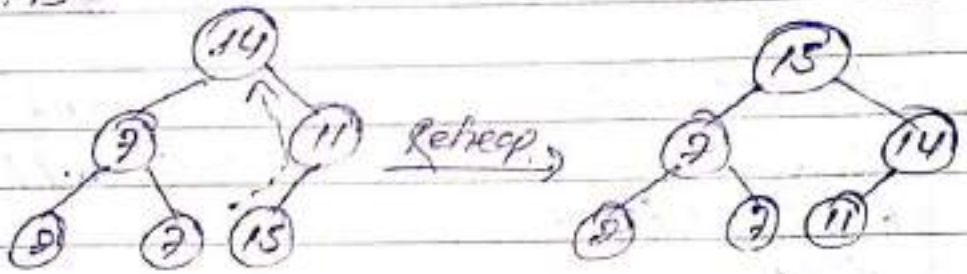
Add: 2



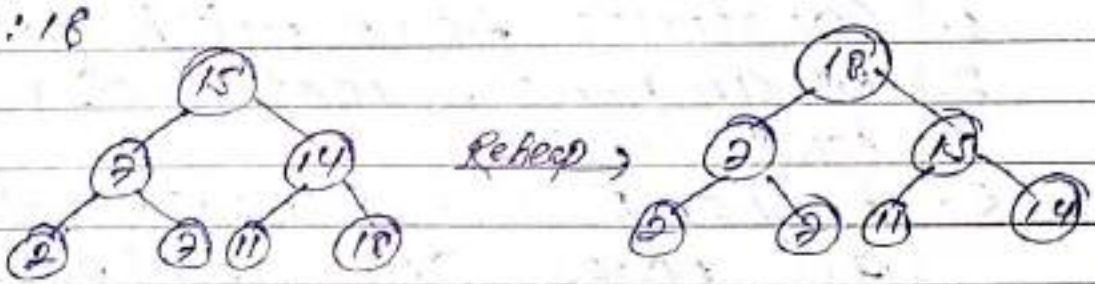
Add: 3



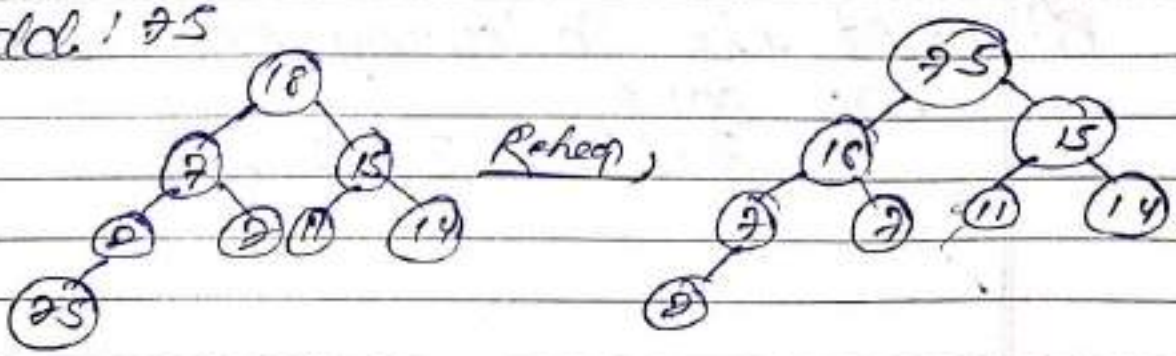
Add: 15



Add: 18

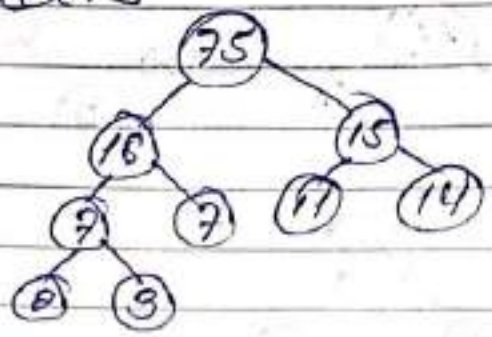


Add: 25

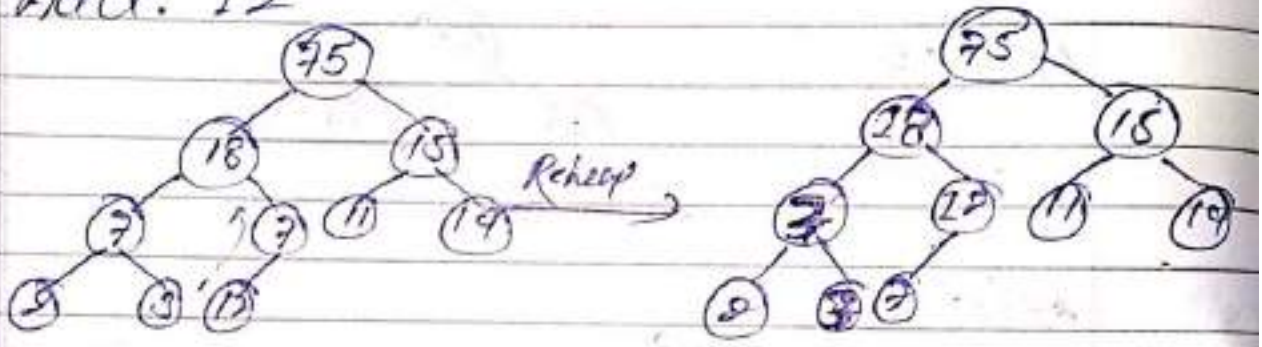


Teacher Signature

Add: 3



Add: 12



5. Heap sort :- This sorting technique uses heap to sort the data elements.

Steps:

1. Construct heap with given list
2. Renam Remove root and replace with last node
3. Re-heap and Repeate step-2 till all nodes are deleted.

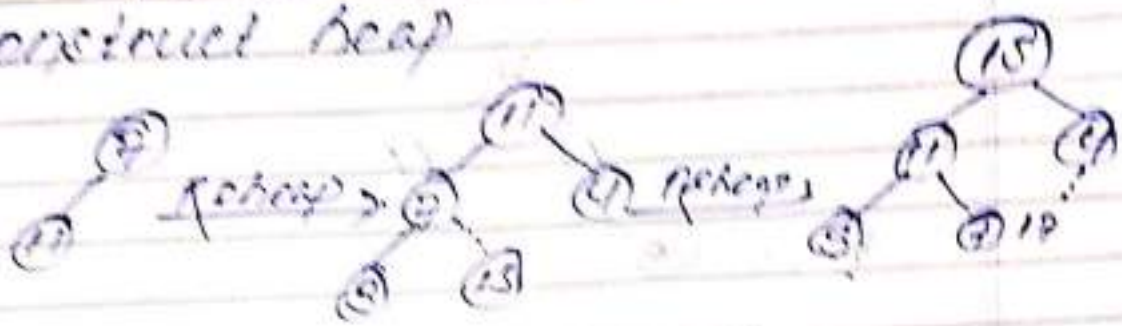
6. Sort the following nos using heap sort

7, 11, 4, 3, 15, 12

Teacher Signature

Sol: 2, 11, 4, 3, 15, 5, 12

Constructed heap



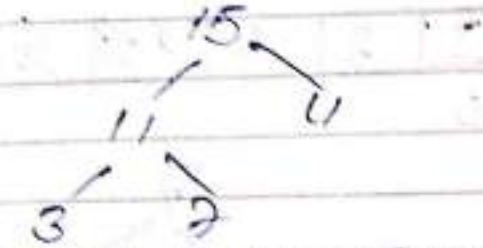
Reheap,

12	11	15	3	2	4

Step-1 Delete root & replace with last node

4	11	15	3	7	12

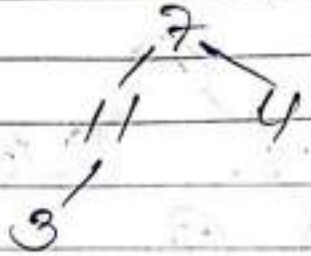
Re-heap



15	11	4	3	7	12

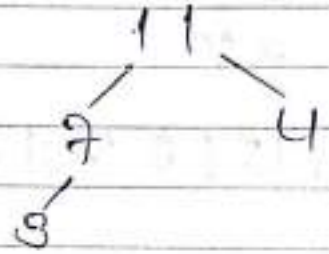
Teacher Signature

Step ② Delete root & replace with last node



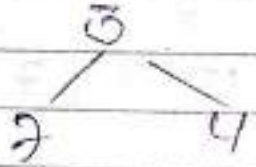
7	11	4	3	~~15~~	~~17~~

Reheap



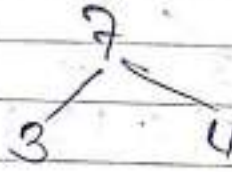
11	7	4	3	~~15~~	~~17~~

Step ③ Delete root & replace with last node



3	7	4	~~11~~	~~15~~	~~17~~

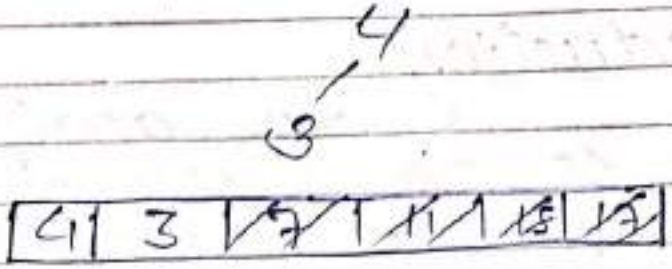
Reheap



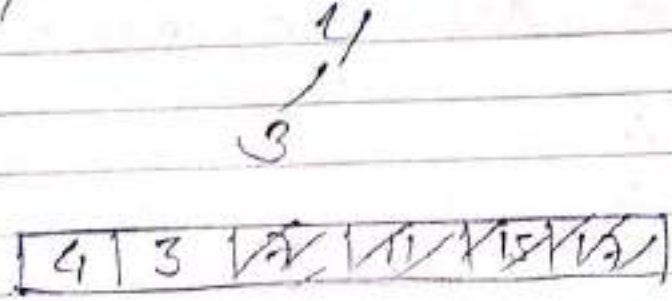
7	3	4	~~11~~	~~15~~	~~17~~

Teacher Signature

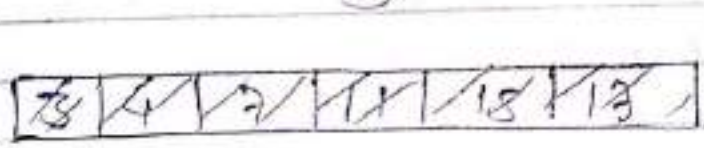
Step ④ Delete root & replace with last node



ReHeap



Step ⑤ Delete root & replace with last node



Sorted

Teacher Signature