Arrays in Data structures

data types → int, float, char, double, string

int a = 5

data type — name — value

Memory - Long tape of bytes (8 bits)

For int - 4 bytes = 8×4 → 32 bits

→ At one instance we are able to store one value only.

→ some data will stored in array

int a[60]; char b[10]; float b[5]

data example - [5, 10, 0, -2, 11]

['a', 'e', 'i', 'o', 'u']

[5, 'a', 'k', 2, 3] X (Not valid)

→ For arrays in run time we can't exceed the size.

int a[5] = {6, 2, 4, 3, 0}

Address formulae

a[0] = 6

a[1] = 2

a[2] = 4

a[3] = 3

a[4] = 0

i = 0 - (n-1)

Drawback of Array:

Waste of memory occupation.

```
char b[10] = {'s','a','t','i','s','h'}
```

s	a	t	i	s	h	\0	\0	\0	\0

For displaying:

```
int b[10];
    Printf("Enter the elements-");
      for (i=0; i<9; i++)
      scanf("%d", &b);
```

→ Array is a fixed size sequential collection of data which are in same type.

→ Access a particular element = arrayname[index]

→ data items are stored in continuous locations

→ Random access taking constant time.

In arrays we can't insert more than size of its own.

Eg:- int a[3];

Above array there is no possibility to insert more than 3 elements.

Otherwise we assigned a[100]; and used for 5 elements only then remaing memory is wasted

TO reduce memory wastage linked list will introduced.

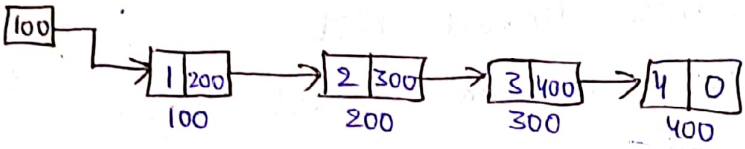
Linked List

→ linked list is a collection of different data types.

→ data is not stored in consecutive locations. it may be stored anywhere in the memory.

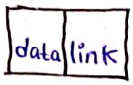
→ To find the address of a particular value we have to traverse to that node by using head pointer.

→ int data type occupies 8 bytes.



→ we have to store pointer also. For suppose we want to store extra data then memory manager can allocate space where 8 bytes are present.

→ No need to specify the size.



Logical representation of a node

```
struct node
  {
     int data;
     struct node *next; // creating pointer
  };
```

→ here data type is struct node

→ The node storing first node address is called as head/base, using head we can print the whole nodes.

→ Linked list is a linear data structure.

→ Time complexity is order of n O(n)

→ Extra space will be required to store address of next node.

→ Binary search is not possible.

→ It is present in dynamic size, memory allocation happens in run time.

Linked list is of 4 types.

→ single linked list

→ double linked list

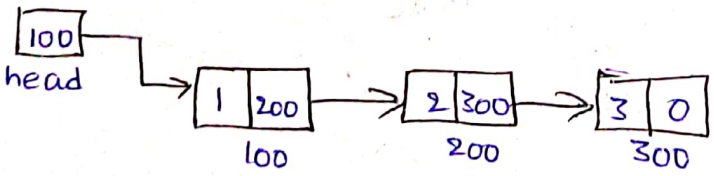
→ circular linked list

→ Doubly circular linked list.

Single Linked List

Single linked list is a type of linked list in which having link in one side with the next node.

Node = Data part + Address part.



→ head represents first node Address.

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

→ For SLL creation.

creation of SLL:

```
#include <stdio.h>
# include <stdlib.h>
struct node{
        int data;
        struct node *next;
      };
Void main()
 {
    struct node *head, *temp, *n1;
     head=0;
   int choice;
  Printf ("enter 1 to create SLL else enter 0-")
  scanf ("%d", &choice);
 while (choice==1)
 {
  n1=(struct node *) malloc (sizeof (struct node));
   Printf ("enter data-");
   scanf ("%d", &n1->data);
     n1->next=0;
     // For creating first node
        if (head==0)
          {
            head = temp = n1;
          }
        else{
            temp->next=n1;
              temp=n1;
            }
        Printf("enter 1 to create another else 0-"
 }       scanf ("%d" &choice);
```

// in this area we done our all operations - - - ->

// For displaying

```
temp = head;
while (temp != 0)
  {
    printf ("%d  ", temp->data);
     temp=temp->next;
  }

} // main closed.
```

In single linked there are many functions They are:

Insertion - At beginning
At ending
At a specified Place (index)

Deletion - At beginning
At ending
At a specified Place. (index)

Concatenation - Adding two linked lists

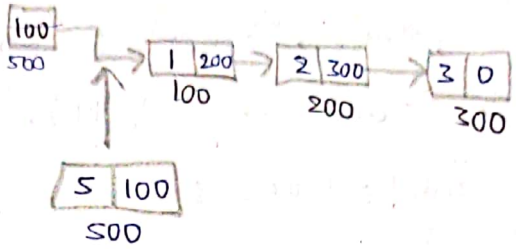
searching - To search specified element. finding its index position.

Insertion

→ Insertion is used to insert data in beginning, ending (or) at specified position.

→ we are now writing code for these all - - - - ->

Insertion at beginning:



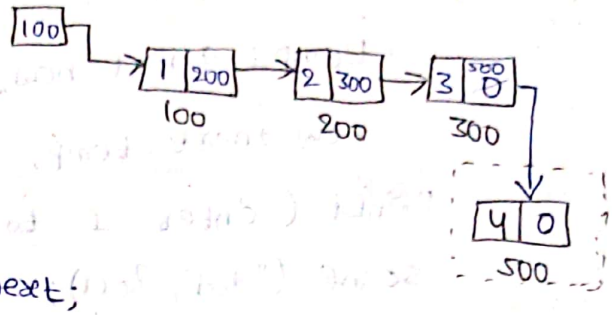

```
#include <stdio.h>
#include <stdlib.h>
struct node {
        int data;
        struct node *next;
      };
void main()
 {
   struct node *head, *temp, *n1, *begin;
    head=0;
 int choice;
 Printf ("Enter 1 to create else enter 0-");
 scanf ("%d", &choice);
while (choice == 1)
{
 n1 = (struct node *) malloc (sizeof (struct node));
 Printf ("Enter data-");
// scanf ("%d", &choice);
  scanf ("%d", &n1->data);
    n1->next=0;
      if (head==0)
          temp=head=n1;
      else {
          temp->next=n1;
           temp=n1;
          }
    Printf ("Enter 1 to create another else 0-");
    scanf ("%d", &choice);
}

// Asking user to insert at beginning or not
```

```
int opt;
printf ("enter 1 to insert at begin else 0-");
scanf ("%d", &opt);
while (opt = 1)
 {
  begin = (struct node *) malloc (size of(struct node));
   printf ("enter data-");
   scanf ("%d", & begin->data);
// Logic
begin->next = head;
 head = begin;
printf ("enter 1 to insert another else 0-");
scanf ("%d", &opt);
// For displaying
   temp = head;
   while (temp != 0)
    {
      printf ("%d ", &temp->data);
       temp = temp->next;
    }
```

Inserting at ending

```
// creation is same for all
 struct node *end;
int opt;
printf ("enter 1 to insert at end else 0-");
scanf ("%d", &opt);
while (opt == 1)
 {
  end = (struct node *) malloc (sizeof(struct node));
   printf ("enter data-");
   scanf ("%d", & end->data);
   end->next = 0;
```

```
// For traversing
  temp = head;
   while (temp1 = 0)
      {
        temp1 = temp;
        temp = temp -> next;
      }
// Logic
     temp1 -> next = end;
Printf ("Enter 1 or 0 - ");
 scanf ("%d", &opt);
}
```



Inserting At a specified position

```
// creation is same
struct node *new;
int a;key;
Printf ("Enter 1 to insert at specified else 0-");
scanf ("%d", &a);
while (a == 1) {
  Printf ("Enter position to insert - ");
   scanf ("%d", &key);
new = (struct node *) malloc (sizeof (struct node));
Printf ("Enter data - ");
 scanf ("%d", &new -> data);
// For traverssing
    int i = 0;
   temp = head;
      while (i < key) {
          temp1 = temp;
           temp = temp -> next;
            i++;
         }
```

```
// Logic
    temp1 -> next = new;
    new -> next = temp;
Printf ("Enter 1 to insert another else 0-");
scanf ("%d", &a);
```

Deletion

→ Deletion is generally used to delete unnecessary data in the LL.

→ This can be done in three ways: Deletion at beginning, ending, At specified Place.

→ creation is same as before.

deletion at beginning

```
// Asking user to delete or not
int k;
Printf ("Enter 1 to delete at begin else 0-");
scanf ("%d", &k)
while (k == 1) {
    // Logic
    temp = head;
    head = temp -> next;
    free(temp);
    // For one more to delete
    Printf ("Enter 1 to delete else 0-");
    scanf ("%d", &k);
}
```

deletion at ending

```
// creation is same as before
// Asking user to delete at end or not.
int a;
Printf ("Enter 1 to delete at end else 0-")
 scanf ("%d", &a);
while (a==1)
 {
   // For traversing
   temp=head;
    while (temp!=0){
        temp1=temp;
         temp=temp->next;
        }
  // Logic
    temp1->next=0;
    free (temp);
    Printf ("Enter 1 or 0-");
     scanf ("%d", &a);
}
```

deleting at specified position

```
// Ask user to key value
  // For traversing
      int i=0;
       while (i<key){
            temp1=temp;
            temp=temp->next; i++;
              }
```

```
// Logic
    temp1->next = temp->next;
     free(temp);
     temp->next = 0;
printf("enter 1 or 0-");
 scanf("%d", &a);
}
```

concatenation

```
// input two single linked lists
// For concatenation
 // For traverse
  temp=head;
   while (temp->next!=0){
        temp=temp->next;
         }
        temp->next = head1;
 // For printing
      temp=head;
     while(temp!=0){
       {
         printf("%d", temp->data);
           temp = temp->next;
       }
```

searching For element

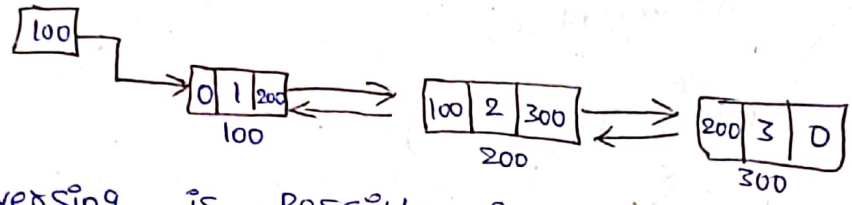
→ Also finding its index

```
// input single linked list creation
int key, index=0, flag=0;
Printf ("Enter data to search-");
scanf ("%d", &key);
// For traversing
  temp=head;
  while (temp!=0) {
      temp=temp→next;
        if (key==temp→data)
          {
            flag=1;
            Printf("In position is %d", index+1);
            break;
          }
          index++;
      }
if (flag==1)
  Printf ("searched element is found");
else
    Printf ("searched element is not found);
```

Double Linked List

→ Double liked list is a type of linked list in which is having data, 2 address parts.

one is for previous node address, another is for next node address to store.



→ Traversing is possible in both front and reverse.

→ Generally music apps like wynk, spotify etc., are used to play previous song or next song by using this double linked list.

→ Here also head represents first node's address.

→ Insertion, deletion, searching, concatenation, reverse

Creation of DLL

```
#include <stdio.h>
#include <stdlib.h>
struct node{
    int data;
    struct node *prev, *next;
    }; // semi-colon is must

Void main()
 {
   struct node *head, *temp, *n1;
    head=0; int choice;
    Printf ("Enter 1 to create dLL else 0-");
    scanf ("%d", &choice);
```

```
while (choice==1)
{
n1=(struct node *) malloc(sizeof(struct node));
Printf("Enter data-");
Scanf("%d", &n1->data);
  n1->prev=0;
  n1->next=0;
  if
 while (head==0)
    {
      temp=head=n1;
    }
   else{
    // n1->next
      temp->next=n1;
        n1->prev=temp;
       temp=n1;
    }
 Printf("Enter 1 to create another node else 0-");
 Scanf("%d", &choice);
}

// For printing
  temp=head;
 while (temp->next!=0)   // while (temp!=0)
      {
        Printf("%d", temp->data);
        temp=temp->next;
      }

} //main closed.
```

Insertion at beginning

```
// create the DLL
// Insertion at beginning
int a;
Printf ("Enter 1 to insert at begin else 0-");
scanf ("%d", &a);
while (a==1) {
  begin = (struct node *) malloc (sizeof (struct node));
  Printf ("Enter data-");
  scanf ("%d", &begin->data);
    new->prev=0;
    new->next=0;
 // Logic
  temp = head;
  new->next = temp;        // new=begin
  temp->prev=new;
  head=new;
 Printf ("Enter 1 to insert another else 0-");
 scanf ("%d", &a);
}
```

Insertion at ending

```
while (a==1) {
  end= (struct node *) malloc (sizeof(struct node));
  Printf ("Enter data-");
  scanf ("%d", &end->data);
      // For traversing to last node
```

```
temp = head;
while (temp->next != 0) {
    temp1 = temp;
    temp = temp->next;
}

// Logic
    temp->nex = new;
    new -> prev = temp;
    new -> next = 0;
    Printf ("Enter 1 or 0 - ");
    scanf ("%d", &a);
}
```

Insertion at specified Position

```
// Input DLL creation
// Allocate memory to new node first
// Ask user to insert at which position.

// traversing
    int i=0;
    temp = head;
    while (i<key) {
        temp1 = temp;
        temp = temp->next; i++;
    }

// Logic
    temp1 ->next = new;
    new -> prev = temp1;
    new -> next = temp;
    temp -> prev = new;
```

```
temp = head;
while (temp->next != 0) {
        temp1 = temp;
        temp = temp->next;
      }

// Logic
    temp->nex = new;
    new -> prev = temp;
    new -> next = 0;
    Printf ("Enter 1 or 0 - ");
    Scanf ("%d", &a);
}
```

Insertion at Specified Position

```
// Input DLL creation
// Allocate memory to new node first
// Ask user to insert at which position.

// traversing
    int i=0;
    temp = head;
    while (i<key) {
            temp1 = temp;
            temp = temp->next;  i++;
            }

// Logic
        temp1 ->next = new;
        new -> prev = temp1;
        new -> next = temp;
        temp -> prev = new;
```

Deletion at beginning

```
// input DLL creation.
// Asking user to delete at begin or not.
int k;
Printf ("Enter 1 to delete at begin or 0-")
scanf ("%d", &k);
while (k==1) {
    temp=head;
    head=temp→next;
    free (temp);
```

Deletion at ending

```
// For traversing
    temp=head;
    while (temp→next!=0) {
        temp1=temp;
        temp=temp→next;
        }
// Logic
    temp1→next=0;
    free (temp);
```

Deletion at specified position

```
// Ask position i.e., key to delete
// For Traversing
    int i=0;
    temp=head;
    while (temp→next!=0) {
```

```
        temp1 =temp;
        temp= temp→next;  i++;
    }

temp1 → next= temp→next;
temp →next →prev= temp1;
 temp→ prev=0;
 temp→ next=0;
   free (temp);
```

concatenation

head, head 1

```
// input  two  double  linked  lists
// concatenation
// For  traversing
    temp=head;
     while (temp→next!=0){
          temp=temp→next;
          }
          temp →next=head1;
Print  the  new  double  linked  list.
```

searching for an element

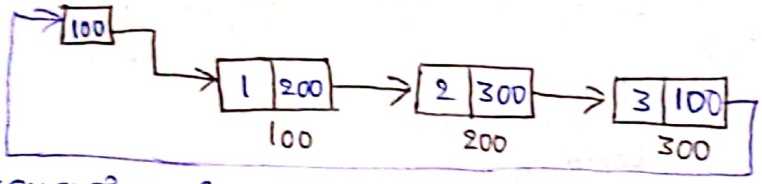
```
// input double linked list
// For searching
int k, index=0, flag=0;
printf ("enter element to search-");
scanf ("%d", &k);
// For Traversing
   temp=head;
     while (temp->next!=0)
           {
             temp=temp->next;
           if (k==temp->data) {
                flag=1;
             printf("\n position is %d", index+1),
               break;
           }
           index++;
         }
if (flag==1)
  printf ("searched element is found");
else
    printf ("searched element is not found");
```

Circular Linked List

Circular linked list is a type of linked in which having data part, address part.

→ It is similar to single linked list but it have link, last node to first node.

→ It appears in circular shape, so it is called as circular linked list.



→ Traversing is possible in only one way, but it have ability to reach to the first node.

→ Used mostly in playlist, they will repeat after completion of last song in the list, it means plays first song after completion of last song.

→ here head and last also contains first node's address.

Creation of circular linked list

```
#include<stdio.h>
#include<stdlib.h>
Struct node{
        int data;
        struct node *next;
        };
Void main()
 {
   struct node *head, *temp, *n1;
    head=0;
   int choice;
  Printf ("Enter 1 to create CLL else 0-");
   scanf ("%d", &choice);
```

```
while (choice=1){
n1= (struct node *) malloc (sizeof (struct node));
Printf ("Enter data - ");
scanf ("%d", &n1->data);
 n1->next=0;
  if (head==0)
      temp=head=n1;
  else{
      temp->next=n1;
        temp=n1;
      }
    n1->next=head;
Printf ("Enter 1 to create another else 0-");
scanf ("%d", &choice);
// For Printing
  temp=head;
  while (temp->next!=head){
      Printf ("%d", temp->data);
       temp = temp->next;
      }
      Printf ("%d", temp->data);
} // main closed.
```

Insertion at beginning

```
// input CLL creation
// Asking user to insert at beginning or not
int k;
Printf ("Enter 1 to insert at begin else 0 - ");
scanf ("%d", &k);
while (k==1) {
 new = (struct node *) malloc (sizeof (struct node));
 Printf ("Enter data");
 scanf ("%d", &new->data);
```

~~// Logic~~
~~temp=head;~~
~~new->next=temp;~~
~~temp->prev=new;~~
~~head=new;~~

// Logic
~~begin->next=head;~~ new->next=head;
// For traversing
temp=head;
while (temp->next!=head){
temp=temp->next;
}
head=begin;
temp->next=head;

```
 Printf ("Enter 1 to insert another else 0 - ");
 scanf ("%d", &k);
}
```

Insertion at ending

```
// input CLL creation
int choice;
Printf ("Enter 1 to insert at ending else 0 - ");
scanf ("%d", &choice);
while (choice=1) {
 end = (struct node *) malloc (sizeof (struct node));
 Printf ("Enter data - ");
 scanf ("%d", &end->data);

 // For traversing
```

```
temp = head;
while (temp→next != head){
        temp1 = temp;
        temp = temp→next;
    }
```

// Logic
temp → next = new;
new → prev = temp;
~~temp = temp → next;~~
new → next = 0;

// Logic
temp → next = end;
end → next = head;

```
Print ("Enter 1 or 0 - ");
scanf ("%d", &k);
}
```

Insertion at Specified Place

```
// input CLL creation
k // Ask user to insert at key value or not
// Ask key value to insert.
while (k == 1) {
  Print ("Enter key value to insert");
  int key;
  scanf ("%d", &key);
new = (struct node *) malloc (sizeof(struct node));
  Print ("Enter data - ");
  scanf ("%d", &new→data);
// For traversing
  int i=0;
    temp = head;
      while (i < key) {
          temp1 = temp;
          temp = temp→next; i++;
      }
```

// Logic

```
temp1->next = new;
new->prev = temp1;
new->next = temp;
temp->prev = new;
```

// Logic

```
temp1->next = key;
key->next = temp;
```

```
printf("enter 1 to insert one more else 0-");
scanf("%d", &k);
}
```

deletion at beginning

```
// input CLL creation
int choice;
printf("enter 1 to delete node at begin else 0:");
scanf("%d", &choice);
while (choice==1){
    temp=head;
    head=temp->next;
    free(temp);
printf("enter 1 or 0-");
scanf("%d", &choice);
}
```

```
temp=head;
while(temp->next=head){
    temp=temp->next;
}
d=head;
head=head->next;
temp->next=head;
free(d);
}
```

deletion at ending

```
// input CLL creation
// Ask user to delete node at end or not.
// For traversing
    temp=head;
        while (temp->next!=head){
            temp1=temp;
            temp=temp->next;
```

```
// Logic
    temp1 -> next = 0;
    free (temp);
}
```

```
// Logic
    temp1 -> next = head;
    free (temp);
}
```

```
printf ("enter 1 or 0 to delete - ");
scanf ("%d", &choice);
}
```

deletion At specified Position

```
// input CLL creation
// Ask user to delete at specified position
// Ask key value to delete
```

```
// For traversing
    int i=0;
    while (i<key) {
        temp1 = temp;
        temp = temp -> next;
        i++;
    }

// Logic
    temp1 -> next = temp -> next;
    temp -> next -> prev = temp1;
    temp -> prev = 0;
    temp -> next = 0;
    free (temp);
```

```
// For traversing
int i=0;
temp=head;
while (i<k) {
    temp1 = temp;
    temp = temp -> next;
    i++;
}

// Logic
temp1 -> next = temp -> next;
free (temp);
```

```
printf ("enter 1 or 0 to delete - ");
scanf ("%d", &choice);
}
```

concatenation

```
// input two circular linked lists say
   head, head1
    (i)    (ii)
                        temp->next=head1;
                    // 2nd list end
// concatenation                n2->next=head;
// For traversing
        temp=head;
        while (temp->next!=head){
                temp=temp->next;
              }
             temp->next=head1;   // Logic
```

searching

```
// input CLL creation
// For searching
 int key, index=0, flag=0;
 Printf ("enter element to search-");
 scanf ("%d", &key);
// For traversing
 temp=head;
  while (temp->next!=head){
                temp=temp->next;
              if (key==temp->data)
                 {
                   flag=1;
              Printf ("Inposition of searched
                 }break; element is %d", index+1);
               index++;
 }
```

```
if (flag==1)
    {
      printf ("searched element is found \n");
    }
else
    printf ("\n searched element not found");
```

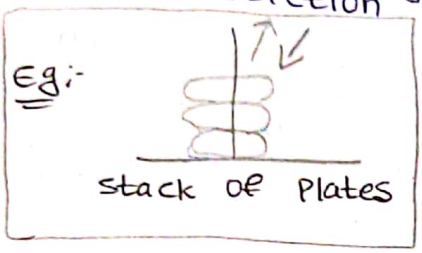
STACK

Introduction to stack:

→ stack is a linear data structure, it is a container which follows insertion, deletion follow some rules.

Rule: insertion, deletion } possible from one end only

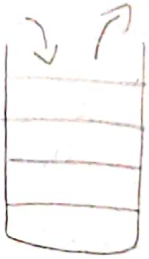
Eg:



LIFO - Last in first out

FILO - first in last out

Insertion - push(x) } possible at top of stack
deletion - pop()



logical representation of stack

→ Fundamental operations are Push, POP. STACK

Operations:

Push(x) → insertion of data in stack
Pop() → has no arguements
Peek() / top() - returns top most element in stack
Isempty() - checks empty or not returns True (or) False
Isfull() - check stack is full or not returns T (or) F

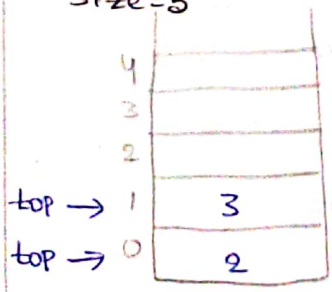
Implementation of stack we have two methods:

(i) static - Array
(ii) Dynamic - Linked list

At initial top = -1

→ when stack is empty, it returns "underflow" condition

size=5



top = -1 Stack

"Unflow"

Push & POP

```
Push(2)
  top++;
Push(3)
   top++;
Pop()
  top--;
Pop()
   top--;
Push(4)
```

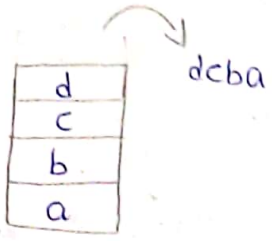
→ If stack is filled its maximum size (top = n-1) it returns "overflow" condition.

isFull() } returns true or False
isEmpty() }

Applications:

1) Reverse a string:

abcd = dcba



2) Undo: (ctrl+z)

This mechanism is performed using stack in our Text editor.

3) Recursion/Function call:

↓

Function calling itself is recursion.

4) To check the balance of Paranthesis.

{ } { { { } { { } } }

5) Infix to Postfix/Prefix.

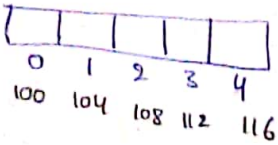
6) Evaluation of Postfix expression.

Implementation of stack using Array:-

→ using static memory allocation

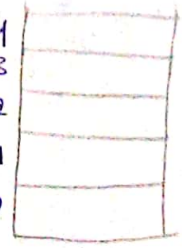
int a[5];

5×4 = 20 bytes



int stack[5];

→ suppose we have to insert more than 5 elements, then create new array with that size.



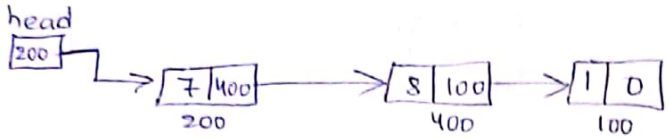
Code:

```
#include<stdio.h>          → else use #define N 5
int stack[100], choice, n, top, x, i;

void push()
  {
    if (top>=n-1)
      {
        printf("\n STACK IS over flow");
      }
    else{
        printf("enter element to push-");
        scanf("%d", &x);
        top++;
        stack[top] = x;
      }
  }

void POP()
  {
    if (top<=-1)
      {
        printf("\n STACK IS UNDER FLOW");
      }
    else{
        printf("\nIt the Popped element is %d", stack[top]);
        top--;
      }
  }

Void display()
  {
    if (top>=0)
      {
        printf("\n The elements in STACK is-\n");
          for(i=top; i>=0; i--)
              printf("\n %d", stack[i]);
          printf("\n Press next choice");
      }
    else { printf("The stack is empty\n"); }
```

```
Void Peek()
{
   if (top==-1)
     {
       Printf("\n The stack is underelow");
     }
   else{
       Printf(" Peek value: %d", stack[top]);
       }
}

int main()
{
   top= -1;
Printf("\n Enter size of the stack [MAX=100] : ");
 scanf ("%d", &n);
Printf("\nIt STACK OPERATIONS USING ARRAY");
Printf ("\nIt ----------");
Printf ("\nIt 1.PUSH\nIt 2. POP\nIt 3. DISPLAY\nIt 4. Peek\nIt
                                        5. EXIT");
do
 {
    int choice;
       Printf ("Enter the choice-");
       scanf ("%d", &choice);
       switch (choice)
         {
            case 1:
              {
                 Push();
                 break;
              }
            case 2:
              {
                 Pop();
                 break;
              }
```

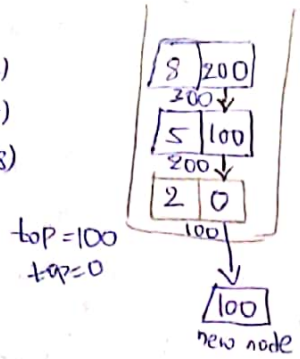
```
        case 3:
          {
            display();
            break;
          }
        case 4:
          {
            Peek();
            break;
          }
        case 5:
          {
            Printf("\n\t EXIT POINT");
            break;
          }
        default:
          {
            Printf ("\n\t Please enter valid choice
                                    (1/2/3/4)");
          }
      } //switch
  } // do loop closed
while (choice!=5);
    return 0;
} //main closed.....
```

Implementation of stack using linked list:-

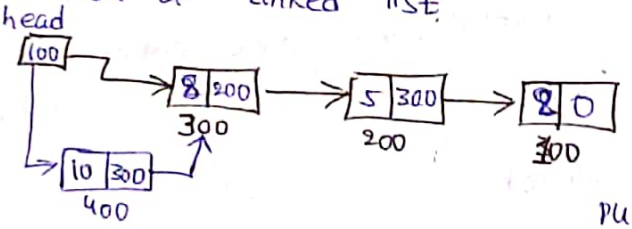


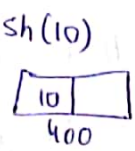
Rule: LIFO Principle

→ In linked list we follow for insertion or deletion O(n)
But in stack it is O(1) only for any operation
i.e., Push()
Pop()



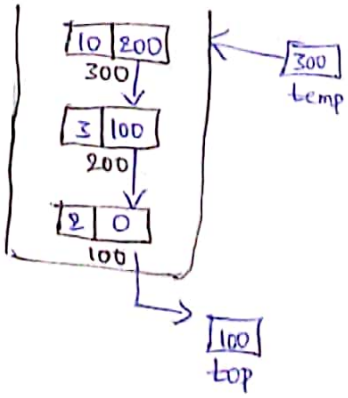
above as in form of linked list





Code:

```
Struct node {
        int data;
        struct node *next;
    };
    struct node *top = 0;
Void Push(int x)
{
        // creating node
        struct node *newnode;
    newnode = (struct node *) malloc (sizeof (struct node));
        newnode→data = x;
        newnode→next = top;
            top = newnode;
```



```
Void display()
  {
     struct node *temp;
          temp=top;
       if (top==0){
          printf("STACK is empty");
        }
       else
         {
           while (temp!=0); // doubt for semicolon
             {
              printf("%d", temp->data);
                temp=temp->next;
```

```
Void Peek()
  {
     if (top==0)
        Printe ("The STACK IS EMPTY");
     else{
        Printf ("TOP element is %d", top->data);
        }
  }
```

```
Void Pop()
  {
     struct node *temp;
          temp=top;
      if (top==0)
         {
           Printf ("STACK IS UNDERFLOW");
         }
       else{
          Printe ("The Popped element is %d", top->data);
            top=top->next;
            free(temp);
         }
  }
```

```
Void main()
{
  int c=1, n;
    while(c==1)
   {
     Printf("\nIt 1. Push \nIt 2. display \nIt 3. POP \nIt 4. Peek
     Scanf("%d",&n);
     Printf("Enter value");
     switch(n)
    {
      case 1:
        {
          Push();
          break;
        }
      case 2:
        {
          display();
          break;
        }
      case 3:
        {
          POP();
          break;
        }
      case 4:
        {
          Peek();
          break;
        }
    } // switch closed
  Printf("Press 1 to continue-");
  Scanf("%d",&c);
} // Loop closed

} // main closed....
```

INFIX, PREFIX AND POSTFIX EXPRESSIONS

Expression:

-> It contains operators, operands, constants, symbols, paranth

Eg:- 5+1, P+2, (a+b)*c

Binary operator :- INFIX :-

<operand> <operator> <operand> -> infix notation

Precedence order:

1. ( ) { } [ ]
2. ^ -> R-L
3. * / % -> L-R
4. + - -> L-R

Eg:- 5+1*6 → (5+1)*6 → 6*6 = 36
5+1*6 → 5+(1*6) → 5+6 = 11

Q. Evaluate 1+2*5+30/5

we Associtivity -> L-R

1+2*5+30/5
= 1+10+30/5
= 1+10+6
= 11+6 ⇒ 17

2^2^3
R->L
2^8
=256 ✓
4^3 = 64 X L->R

PREFIX: (Polish notation)

<operator> <operand> <operand>

Eg: 5+1 ⇒ +51
(infix) (prefix)

a*b+c ⇒ *ab+c
(infix)
+*abc -> prefix

Postfix: (reverse polish notation)

<operand> <operand> <operator>

5+1 ⇒ 51+

-> This may be complex expression

a*b+c ⇒ ab*+c
ab*c+

Infix to postfix conversion:-

1) Print the operands (a, b, c), as they arrive to postfix
2) If stack is empty (or) contains a left Paranthesis on top, Push the incoming operator onto the stack.
3) If incoming symbol is '(', Push it onto stack
4) If incoming symbol is ')', Pop the stack & Print the operators until '(' is found.
5) If the symbol is an operator, then check the stack is empty or not.
   (i) If the stack is empty then Push the operator onto the top of the stack.
   (ii) If the stack is not empty, then we have to check the Priority of the operator.
   a) If the Priority of Incoming operator is having higher priority of the top of the stack then Push that operator onto the top of the stack.
   b) If the Priority of Incoming operator is having lesser priority then top of the stack, Pop & Print the top. Then test the incoming operator with new top of the stack.
   c) If the Priority of Incoming operator having equal Precedence with top of stack, use Associativity rule.
6) After reading entire infix expression, if stack is not empty, then Pop all the symbols in the stack and add it to postfix.

Associativity:

If it is L→R then Pop & Print the top of stack and Push the incoming operator

If it is R→L then Push the incoming operator

Ex: convert this infix to postfix expression.

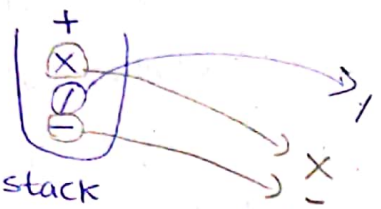
A+B/C

Stack

Postfix

ABC/+

→ A-B/C xD+E



Postfix

ABC/DX-E+

→ A+(B*C)

A+ BC*

ABC*+ → Postfix

1. Convert infix to postfix k+L-M*N+(O^P)*W/U/V*T+Q

Input Exp	stack	Postfix Exp
k		k
+	+	k
L		kL
-	-	kL+
M		kL+M
*	* / -	kL+M
N	- *	kL+MN
+	+	kL+MN*-
(	+(	kL+MN*-
O	+(	kL+MN*-O
=> ^	+(^	kL+MN*-O
P	+(^ →	kL+MN*-OP
)	+	kL+MN*OP^
*	+*	kL+MN*OP^
W	+* →	kL+MN*OP^W
/	+/	kL+MN*OP^W*

U	+/ →	kL+mN*-OP^W*
/	+/	kL+mN*-OP^W*/
V	+/	kL+mN*-OP^W*/V
U	+/	kL+mN*-OP^W*U
/	+/	kL+mN*-OP^W*U/
V	+/ →	kL+mN*-OP^W*U/V
*	+*	kL+mN*-OP^W*U/V/
T	+* →	kL+mN*-OP^W*U/V/T
+	+	kL+mN*-OP^W*U/V/T*+
Q	+	kL+mN*-OP^W*U/V/T*+Q

Postfix = kL+mN*-OP^W*U/V/T*+Q

If it have some R→L condition ^J^A

^	+^	- - - - - - -
J	+^	- - - - - - J
^	+^^	- - - - - - J
A	+^^	- - - - - J+^^

2. Convert this infix expression to postfix.

A-B+(m^N)x(o+p)-Q/R^S xT+Z

input exp	stack	postfix
A		A
-	-	A
B	-	AB
+	+	AB-
(	+(	AB-
M	+(	AB-M
^	+(^	AB
N	+(^	AB
)	+ → x	AB
x	+x	AB
(	+x(	AB
o	+x(	AB
+	+x(+	AB
P	+x(+	AB
)	+x	A
-	-	A
Q	-	AB
/	-/	A
R	-/	A
^	-/^	A
S	-/^	AB
x	-x	A
T	-x	A
+	+	
Z	+	

Finally, the postfix expr

AB-

^	+(^	AB-M
N	+(^	AB-MN
)	+)	AB-MN^
x	+x	AB-MN^
(	+x(	AB-MN^
O	+x(	AB-MN^O
+	+x(+	AB-MN^O
P	+x(+	AB-MN^OP
)	+x	AB-MN^OP+
-	-	AB-MN^OP+*
Q	-	AB-MN^OP+*+
/	-/	AB-MN^OP+*+Q
R	-/	AB-MN^OP+*+QR
^	-/^	AB-MN^OP+*+QR
S	-/^	AB-MN^OP+*+QRS
x	-x	AB-MN^OP+*+QRS^/
T	-x	AB-MN^OP+*+QRS^/T
+	+	AB-MN^OP+*+QRS^/T*-
Z	+	AB-MN^OP+*+QRS^/T*-Z

Finally, The postfix expression=

AB-MN^OP+*+QRS^/T*-Z+

Infix to Prefix using stack:-

Prefix → Pehle <operator> <operand> <operand> me operator hai

Eg:

A + B * C

A + (B * C)

A + * BC

+ A * BC

1. Convert infix expression to prefix expression

k + L - M * N + (O ^ P) * W / U / V * T + Q

write it in reverse order firstly to solve

Q + T * V / U / W *) P ^ O (+ N * M - L + K

infix	stack	Prefix
Q		Q
+	+	Q
T	+	QT
*	+*	QT
V	+*	QTV
/	+/	QTV*
U	+/	QTV*U
/	+//	QTV*U/
W	+//	QTV*U/W
*	+//*	QTVUW
)	+//*)	QTVUW
P	+//*)	QTVUWP
^	+//*)^	QTVUWP
O	+//*)^	QTVUWPO

(	+*//*	QTVUWPO^
+	++	QTVUWPO^*//*
N	++	QTVUWPO^*//*
*	++*	QTVUWPO^*//*
M	++*	QTVUWPO^*//*m
−	++-	QTVUWPO^*//*m*
L	++-	QTVUWPO^*//*m*L
+	++-+	QTVUWPO^*//*m*L
k	++-+	QTVUWPO^*//*m*Lk

we get, QTVUWPO^*//*m*Lk+-++

⇓

Prefix = ++-+kL*m*//*^OPWUVTQ

Algorithm:-

1) Reverse the infix expression and write it.
2) If operand is there simply put into Prefix.
3) If stack is empty (or) contains a right paranthesis ')', Push the incoming operator onto the stack
4) If incoming symbol is ')', Push it onto the stack.
5) If incoming symbol is '(', Pop the stack & Print the operators until we get ')'.
6) If the symbol is an operator check the stack is empty (or) not.

(i) If stack is empty then push that onto the stack.

(ii) If the stack is not empty, then check the priority of the operator.

a) If the priority of incoming operator having higher precidence than top of the stack, then push that operator onto top of the stack.

b) If the priority of incoming operator having lesser precidence than top then pop & print the top. Then test the incoming operator with new top of the stack.

c) If the priority of incoming operator is equals to top of stack, use Associativity rule.

6) After reading entire infix expression, if stack is not empty, then pop all the operators in the stack using LIFO method and add it to prefix.

7) Then reverse the expression to get prefix expression.

Associativity:

If its precedence is L→R then push the incoming operator.

If its precedence is R→L then pop & print the top of the stack and push the incoming operator.

Evaluation of Prefix Expression

infix - a + b * c - d / e ^ f

a=2, b=3, c=4, d=16
c=2, f=3

Prefix = - + a * b c / d ^ e f

⇒ - + 2 * 3 4 / 16 ^ 2 3

<operator> <operand> <operand> → Binary operator

Prefix → Read exp from Right to left

Postfix → Read exp from left to right.

- + 2 * 3 4 / 16 ^ 2 3 ←

- + 2 * 3 4 / 16 8

- + 2 * 3 4 / 2 (16/8 = 2)

- + 2 * 3 4 2

- + 2 12 2

- 14 2

Prefix = 12

Not getting <oper> <operand> <operand> that expression may be not valid expression.

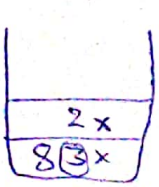
To get prefix expression in only one scan, then have to use stack for this.

Using stack:

- + 2 * 3 4 / 16 ^ 2 3

→ If the incoming is operand so, then push it onto stack.

→ If the incoming is operator. then pop the 2 top most elements from stack, do that operation and result will be pushed onto the top of the stack.



2 ^ 3 = 8

op1 = 2
op2 = 3

Algorithm:

1) scan the Prefix expression from right to left.

2) for each char in Prefix exp do

ie operand is these, push it onto stack
else if operator is there, pop 2 elements

OP1 = top element.
OP2 = next to top element
result = OP1 operator OP2
Push result onto stack
return stack[top]

Evaluation of Postfix Expression

infix - a+b*c-d/e^e

a=2, b=3, c=4, d=16
e=2, f=3

Postfix - abc*+def^/-

234*+16 23^/-

Postfix - Read the expression from L→R
<operand> <operand> <operator>

→ 234*+16 23^/-

2 12 + 16 2 3 ^/-

OP2 = 4
OP1 = 3
3*4 = 12

14 16 2 3 ^/-

14 16 8 /-

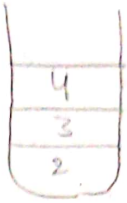
OP1 = 16
OP2 = 8
16/8 = 2

14 2 -

Postfix = 12

top value = OP 2
next top = OP 1
} in postfix

top value = OP 1
next top value = OP 2
} in prefix



OP 2 = 4
OP 1 = 3

OP 1 OP OP 2
3 * 4 = 12

Algorithm:

1) scan the ~~infix~~ postfix expression from left to right.

2) for each char in postfix expression
do

if operand is there, push it onto stack
else if operator is there, pop 2 elements from top

OP 1 = next to top element
OP 2 = top element

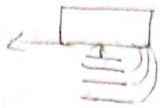
result = op1 operator op2

Push result onto stack
return stack [top]

4.1

Introduction to Queue

-> Queue is a linear data structure
-> ADT - Abstract Data type.

Eg:- Buying a cinema ticket 

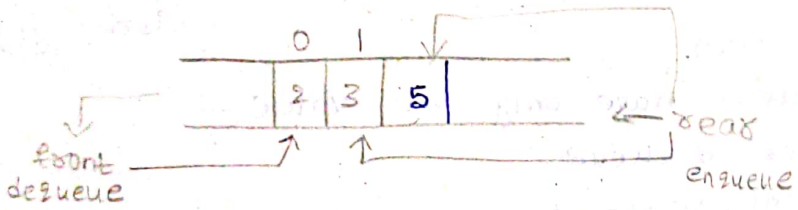
which follows FIFO (First In First out)

Rule:

Insertion - Rear/Tail → enqueue()
deletion - Front/head → dequeue()

Queue is a linear data structure in which Insertion (enqueue), deletion (dequeue) are possible different ends.

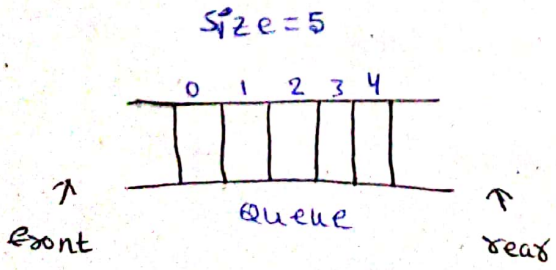
enqueue at Rear
dequeue at front



for enqueue we have to rear++
we can't directly dequeue 5, it is possible when firstly dequeue 2,3 after that it is possible.
also called as LILO (Last In Last Out)

Operations:-

Enqueue(2)
dequeue()
front()/Peek()
isFull()
isEmpty()



front=rear=-1 (queue is empty)

// for inserting in 0'th index

front++, rear++;

// For enqueue in 1st index

rear++;

enqueue (10)

// for enqueue in 2nd index

rear++;

enqueue(-1)

// For dequeue

front++;

rear= size-1

{ Print("overflow"); }

Time complexity - O(n)

front=rear=-1;

Print ("Underflow");

front==rear

// queue have only one value after dequeue

front=rear=-1

(or)

front++;

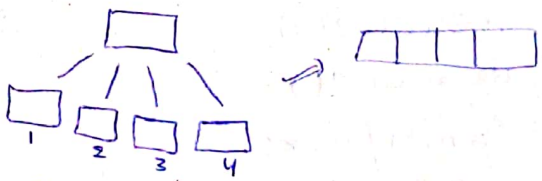
→ isempty()

Time complexity- O(1)

→ Queue have wastage of memory

Applications:-

→ Printer



→ customer case, they put on hold until representive is free.

→ Processor

Implementation of Queue using Array

Queue

enqueue - rear
dequeue - front

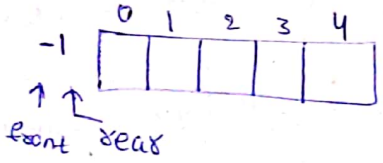
→ By static memory Allocation

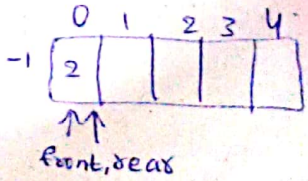
FIFO O(1)

```
int a[5];
int queue[5];
```

Code:

```
#include<stdio.h>
#include <stdlib.h>
#define N 5
int queue[N];
int front=-1;
int rear=-1;

Void enqueue (int x)
{
    if (rear==N-1)
      {
        Printf ("Queue is full");
      }
    else if (front==-1 && rear==-1)
          {
              front=rear=0;
                  queue[rear]=x;
    else {
    //For inserting at 1st index
        rear++;
          queue[rear]=x;
        }
}
```





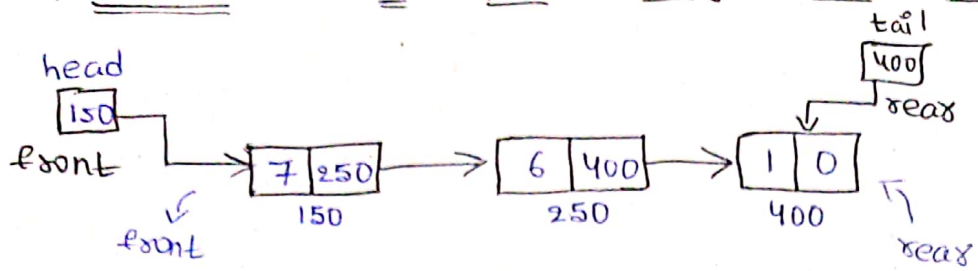
```
Void dequeue
{
    if (front == -1 && rear == -1)
      {
        Printf("Que is empty");

    else if (front == rear)
      {
        front = rear = -1;
      }

    else{
        Printf("The dequeued element is %d", queue[front]);
        front++;
      }
}

Void display()
{
    int i;
    if (front == -1 && rear == -1)
      {
        Printf("Queue is empty");

    else{
                                  i<=rear
                                  (or)
        for(i=front; i<rear+1; i++)
          {
            Printf("%d", &queue[i]);
          }
      }
}

Void Peek()
  {
    if(front == -1 && rear == -1)
      {
        Printf("Queue is empty");
      }
    else{
        Printf("%d", queue[front]);
      }
  }
```

→ Although we have space after dequeue but we never fill that space. using circular it possible.

```
Void main()
{
    int c=1, n;
    while(c==1)
    {
      Printf ("\n1t 1enQueue\n1t 2. dequeue.\n1t 3. Peek\n 4. display
                \n enter number :");
      Scanf ("%d", &n);
      switch(n)
       {
          case 1 :
            {
              enqueue();
              break;
            }
         case 2:
          {
            dequeue();
             break;
          }

         case 3:
          {
            Peek();
            break;
          }

         case 4:
           {
            display();
            break;
           }

      deeault:
              Printf(" check the entered value");
      } //switch
     Printf("enter 1 to continue:");
     scanf ("%d", &c);
   } //Loop closed

} //main closed
```

4.3

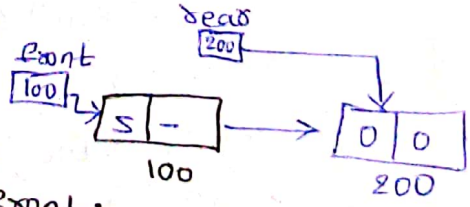
Implementation of Queue using linked list



→ Dynamic memory Allocation.

FIFO rule, Time complexity O(1)


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

struct node *front=0;
struct node *rear=0;

Void enqueue (int x)
  {
     struct node *newnode;
newnode = (struct node *) malloc (sizeof (struct node));
     newnode → data = x;
     newnode → next = 0;
  if (front == 0 && rear == 0)
     {
        front = rear = newnode;
     }
else
   {
     rear → next = newnode;
      rear = newnode;
   }
}
```

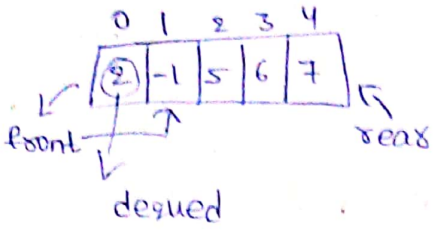


rear → next = front;

```
Void dequeue()
{
    struct node *temp;
      temp = front;
    if (front == 0 && rear == 0) //only one node is there
        Printf("Queue is empty");
    else { Printe("%d deleted data", front->data);
        front = front->next;
        free(temp);
        rear->next = front;
}     }
Void display()
{
   struct node *temp;
    if (front == 0 && rear == 0)
        Printe("Queue is empty");
    else
      {
        temp = front;
        while (temp! = 0)   while (temp! = front)
          {
            Printf("%d", temp->data);
              temp = temp->next;
          }
      }
Void Peek()
{
    if (front == 0 && rear == 0)
          Printe("Queue is empty");
     else
       {
        Printe("%d", front->data);   // = 0
       }
}

Time complexity is O(n)
```

4.4

Circular Queue



front=0
rear=4
} Queue is full

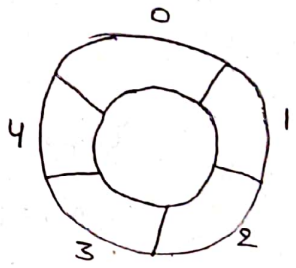
Again calling enqueue(), it shows full but we have some space at 0th index in this case we didn't allow to insert any value.

→ This is drawback of Queue.

To over come this drawback we have introduced circular Queue.

Implementation of circular Queue using Arrays

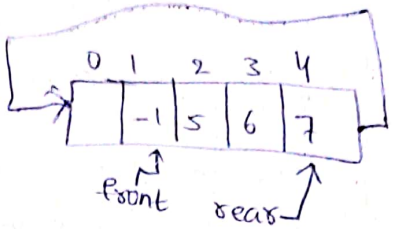
```
# define N 5
   int Queue[N];
   int front=-1;
   int rear=-1;

void enqueue(int x)
{
   if (front==-1 && rear==-1)
      {
         front=rear=0;
         queue[rear]=x;
      }
   else if ((rear+1)%N == front)        // Not rear==N-1
      {
         printf("Queue is full");
      }
   else
      {
   rear=(rear+1) %N                     // Not rear++
      } queue[rear]=x;
}
```



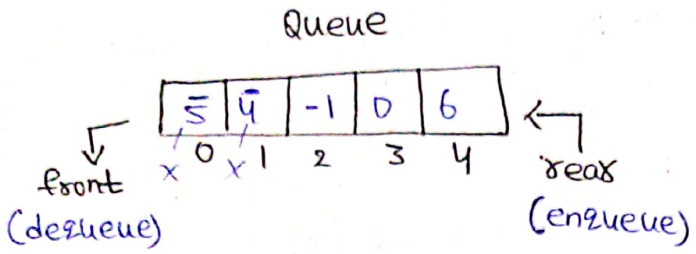
// some elements present

// Queue is full

```
void dequeue()
{
    if (front==-1 && rear==-1)
        Printf("Queue is empty");
    else if (front==rear)
    {
        front=rear=-1;
    }
    else{
        Printf("%d the dequeued element", Queue[front]);
        // Not front++;
        front=(front+1)%N;
    }
}

void display()
{
    if (front==-1 && rear==-1)
        Printf("Queue is empty");
    else{
        int i=front;
        Printf("Queue is :");
        while(i!=rear)
        {
            Printf("%d", Queue[i]);
            i=(i+1)%N;
        } Printf("%d", Queue[rear]);
    }

void Peek()
{
    if (front==-1 && rear==-1)
        Printf("Queue is empty");
    else{
        Printf("%d", Queue[front]); } }
```



Implementation of circular Queue using LL :-

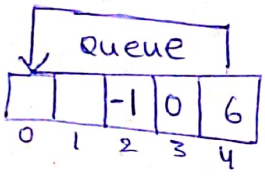


FIFO

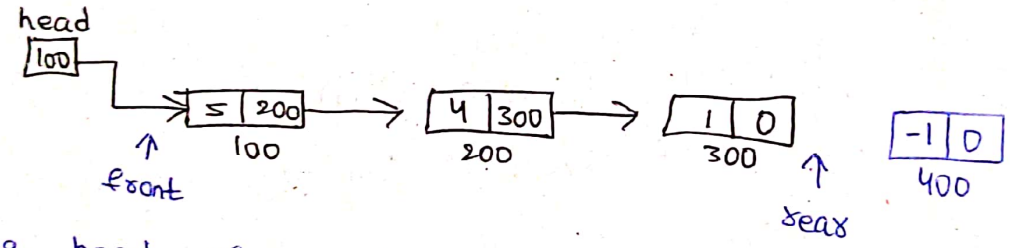
Although we have space in linear queue we can't enqueue elements, circular Queue it is possible

circular queue

No wastage of space

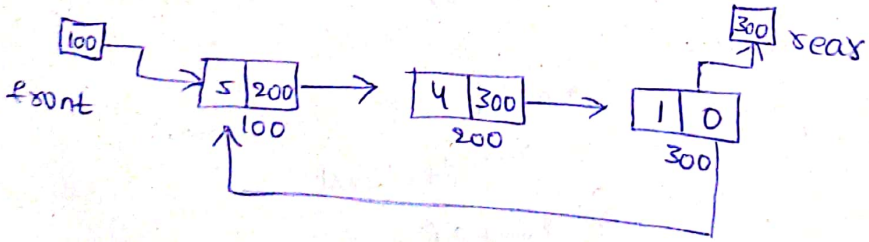


Using linked list:-



Using head Time complexity is O(n) ×

→ we are using *tail, *head it may be O(1) ✓



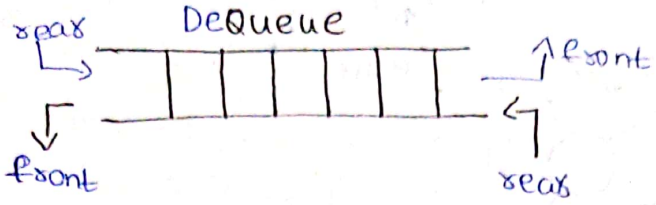
-

code:

```
#include <stdio.h>
#include<stdlib.h>
Struct node
{
   int data;
   Struct node *next;
};
Struct node *front=0;
Struct node *rear=0;
struct node *new;
typedef struct node n;
Void enqueue()
{
   new= (n *) malloc (sizeof (n));
      Printf ("Enter data");
      Scanf ("%d", &new->data);
   if ( front = =0 && rear= =0)
         front =rear=new;
   else
     {
        rear-> next=new;
          rear=new;
     }
    rear-> next= front;
}
Void dequeue()
 {
      n * temp = front;
        if (front= =0 && rear= =0)
              Printf ("Queue is empty");
```

```
    else if (front==rear)
      {
        front=rear=0;
          free(temp);
      }
    else{
      printf("The deleted element = %d\n", front->data);
          front=front->next;
          rear->next=front;
          free(temp);
        }
}
Void Peek()
{
    if (front==0&&rear==0)
        printf("Queue is empty \n");
    else
        printf("peek value = %d \n", front->data);
}
void display()
 {
     n* temp= front;
      if (front==0)
          printf("Queue is empty");
      else{
          while(temp!=rear)
            {
              printf("%d \t", temp->data);
              temp=temp->next;
            }
          Printf("%d \n", temp->data);
        }
 }
```

Dequeue (Double ended Queue)

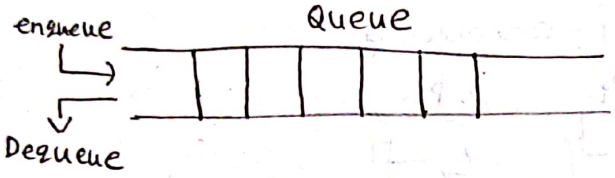


→ Insertion and deletion is possible at both ends

Properties:

Stack - LIFO

Queue - FIFO

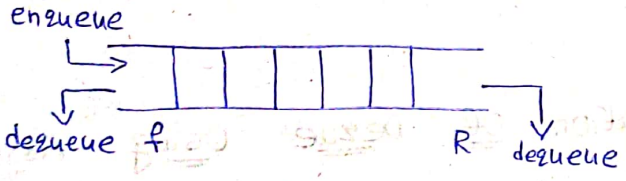


Stack:

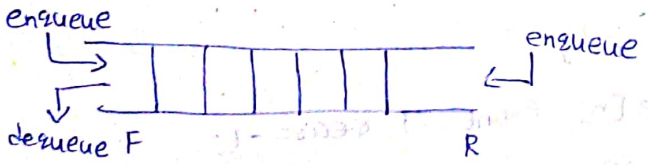
(i) Input - restricted

(ii) Output - restricted

(i)



(ii)



Operations in Deque:

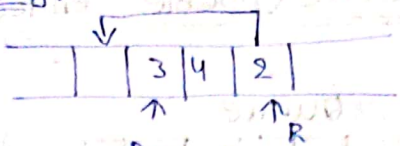
- insert at front
- delete at front
- insert at rear
- delete at rear

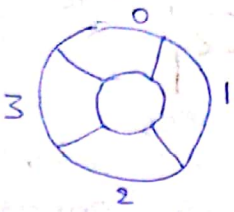
- get front
- get rear

is Full() → by dequeue

is empty() → by dequeue

Circular Array:-





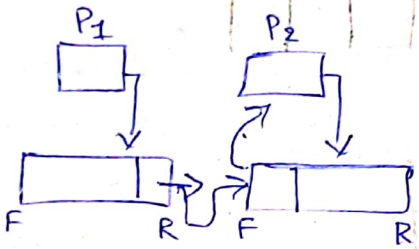
Time complexity - O(1)

Applications:-

→ Used for Redo & Undo operations

→ checking for Palindrome → radar ←

→ multi Processor-scheduling



P2 take threat from P1's rear

→ A-steel Algorithm for Job scheduling

Implementation of Deque using Array:-

```
#include<stdio.h>
# define n 5
int deque[n], front=-1, rear=-1;
void enqueefront()
{
   int a;
   Printf("enter data-");
   scanf("%d", &a);
       if((rear+1)%n==front)
              Printf("overflowln");
       else if (front==-1 && rear==-1)
              {
                 front=rear=0;
```

```
            deque[front]=a;
        }
    else
        {
            front=(front+(n-1))%n;
            deque[front]=a;
        }
}
void enqueuerear()
{
    int a;
    Printf("Enter data-");
    Scanf("%d",&a);
    if((rear+1)%n==front)
        Printf("overflow\n");
    else if(front==-1&&rear==-1)
        {
            front=rear=0;
            deque[rear]=a;
        }
    else{
        rear=(rear+1)%n
        deque[rear]=a;
        }
}
Void dequefront()
{
    if(front==-1&&rear==-1)
        Printf("UnderFlow\n");
    else if(front==rear)
        {
        Printf("Deleted element at front=%d\n",deque[front]);
        front=rear=-1;
        }
    else{
Printf("Deleted element at at front=%d\n",deque[front]);
        front=(front+1)%n;
```

```
Void  dequerear()
 {
    if(front==-1&&rear==-1)
          Printf("Underflow\n");
    else if(front==rear)
     {
      Printf("Deleted element at rear=%d\n", deque[rear]);
          front=rear=-1;
      }
     else{
      Printf("Deleted element at rear=%d\n", deque[rear]);
          rear=(rear+(n-1))%n;
        }
}

Void  display()
 {
    int i=front;
      if (front==-1&&rear==-1)
            Printf("underflow\n");
      else{
          while(i!=rear)
            {
              Printf("%d\t", deque[i]);
                i=(i+1)%n;
            }
          Printf("%d\n", deque[i]);
        }
}
Void  getfront()
 {
    if(front==-1&&rear==-1)
          Printf("Underflow\n");
    else
        Printf("Front=%d\n", deque[front]);
}
```

```
Void   getrear()
 {
    if (front==-1 && rear==-1)
          Printf("Underflow\n");
    else
          Printf ("Rear = %d \n", deque[rear]);
 }

Void   main()
 {
     enqueefront();
     enqueerear();
     enqueefront();
     enqueerear();
     enqueefront();
     dequeefront();
     dequeerear();
      display();
     getfront();
      getrear();
     dequeefront();
     dequeerear();
     dequeefront();
     dequeerear();
      display();
 }
```

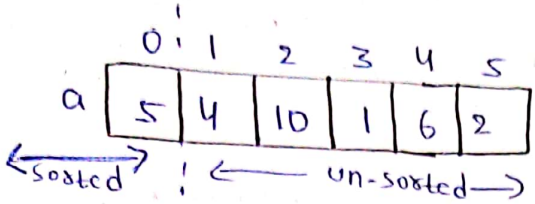
Deque using linked list

```
#include <stdio.h>
#include <stdlib.h>
struct node
{
    struct node *prev;
    int data;
    struct node *next;
} *front=0, *rear=0, *new, *temp;

void create()
{
    int c=1;
    while (c==1)
    {
    new= (struct node *)malloc (sizeof (struct node));
      Printf ("Enter data-");
      scanf ("%d", &new->data);
       new->prev=0;
       new->next=0;
        if (front==0)
              front=rear=new;
        else{
             rear->next=new;
             new->prev=rear;
             rear=new;
            }
     Printf ("Enter 1 to continue-");
       scanf ("%d", &c);
    }
}
void enqueefront()
{
   new=(struct node *) malloc (sizeof(struct node));
       Printf("Enter data=");
        scanf ("%d", &new->data);
```

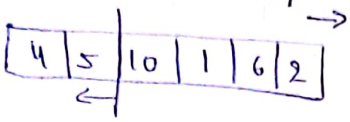
```
        new->prev=0;
        new->next=front;
        front->prev=new;
        front=new;
}

void enqueuerear()
{
    new=(struct node *)malloc(sizeof(struct node));
        Printf("enter data=");
        Scanf("%d",&new->data);
        new->next=0;
        rear->next=new;
        new->prev=rear;
        rear=new;
}

Void dequeefront()
{
    if (front==0)
        Printf("Queue is empty");
    else
    {
        Printf("dequeued element=%d\n", front->data);
        temp=front;
        front=front->next;
        front->prev=0;
        free(temp);
    }
}
void dequerear()
{
    if (front==0)
        Printf("Queue is empty\n");
    else{
        Printf("dequeued element=%d\n", rear->data);
        temp=rear;
        rear=rear->prev;
        rear->next=0;
        free(temp)
```

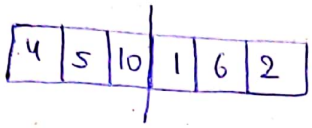
```
Void  display()
{
    temp = front;
      while (temp != 0)
        {
          printf ("%d \t", temp->data);
           temp = temp->next;
        }
}

Void  main()
{
   create();
   enqueefront();
   enqueerear();
   dequeefront();
    dequeerear();
   dequeefront();
   dequeerear();
    display();
}
```

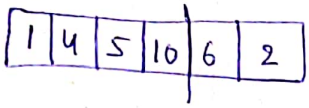
Insertion sort

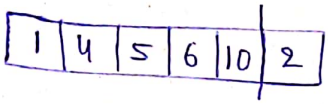


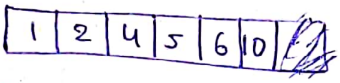
sorted sublist / un-sorted sublist →





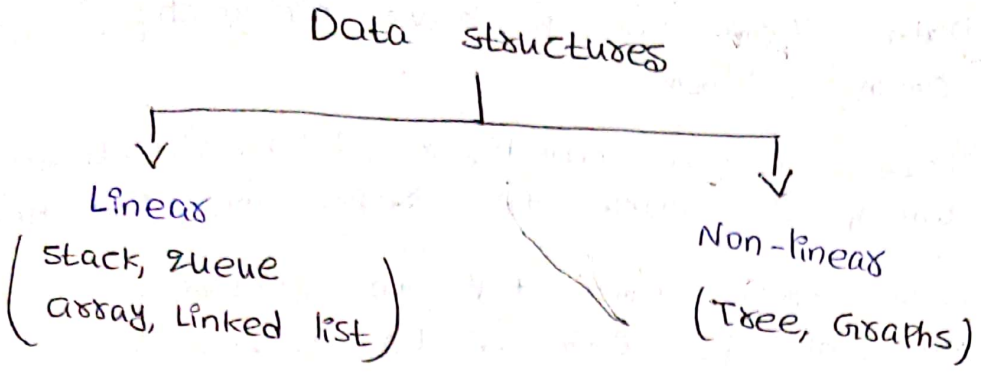


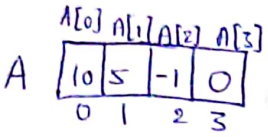




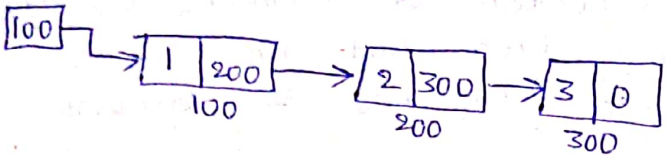
why this is known as Insertion sort?
we are taken one value from un-sorted sublist and we have inserted that value in sorted sublist, that's why this is insertion sort.

```
for (i=1; i<n; i++)
  {
    temp = a[i];
     j = i-1;
    // creating loop to reach 0th index
     while (j>=0 && a[j] > temp)
        {
           a[j+1] = a[j]
            j--;
        }
     a[j+1] = temp;
  }
```

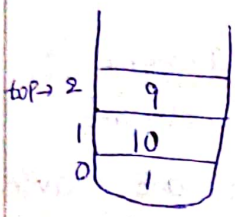




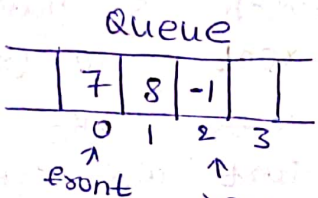
→ Array



→ Linked list



stack
LIFO



FIFO

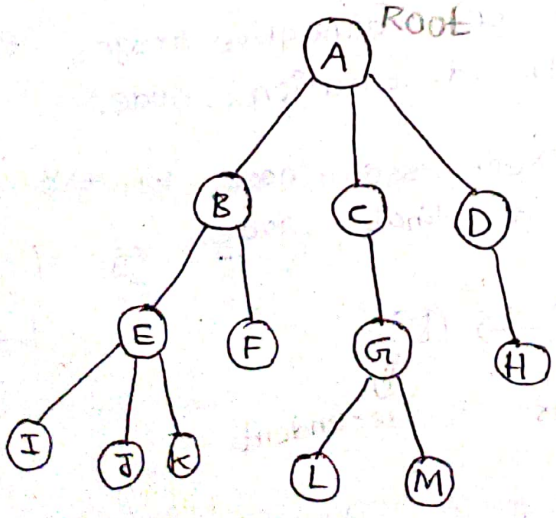
Non-Linear data structure:-

→ Trees, Graphs are considered as Non-linear data structure.

→ It doesn't have and sequence, it have multiple levels.

→ Data is in the form of ~~variety~~. Hierarchy.

eg:- college Management.



→ Data items which have hierarchy relations among them.

* → Tree is a non-linear data structure which is going to represent this hierarchy.

→ Tree grows from top to bottom.
We can go from B to A.

Root = A

nodes = BCDEFG (Having information)

⇒ Tree is a collection of elements (or) nodes (bcz it having data and link so, it called as node.) linked together to stimulate a hierarchy.

Terminology:-

The node which is having any Parent is called as root node. (or) Top most element

Node - which stores the information and having data part and link to next node.

Parent - Immediate predecessor of any node x is called Parent.
'B' is the Parent of E & F.

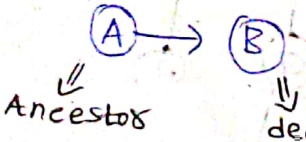
child - Immediate succesor is called child node.
B, C, D are children of A.

Leaf node - The node which doesn't have child
I, J, K, L, M, F are leaf nodes.
also called as external node.

Non-leaf node - which is having atleast one child.
A, B, C, D, E, G
Internal nodes.

Path - sequence of connective edges from source node to destination node.

Ancestor - any predecessor node on the path from root to that node.



Eg:- L = A, C, G

H = A, D

Internal node - A node with children

Descendent - Any successor node on the path from that node to the leaf node.

Eg: C = G, L, M

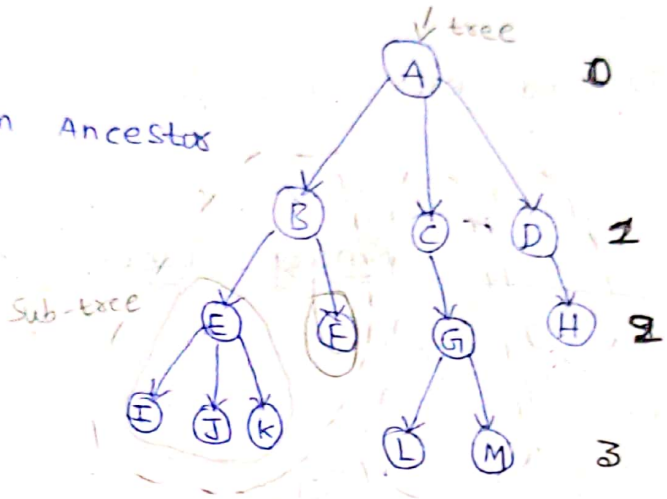
B = E, F, I, J, K

Q. what is the common ancestor of G and H?

G - C, A

H - D, A

'A' is common Ancestor



(0, 1, 2, 3 → Level)

Q. common descendent of C and D.

C - G, L, M

D - H

so, there is no common descendent for C and D.

Sub-tree - which is contains an node of this tree T and all its descendents is called as sub-tree.

Sibling - All children of same parent.

E, F are siblings

F, G are cousins not siblings.

Degree - The no. of children of that node

degree of A = 3, E = 3

" " M = 0

For all leaf nodes it is '0'.

Tree degree = highest degree of any node.

Depth of node - Length of path from root to that node.

F = 2

J = 3

D = 1

Root node

A = 0

height of node - no. of edges in the longest path from that node to leaf node

B = (2), not 1

→ Height and depth may be same.

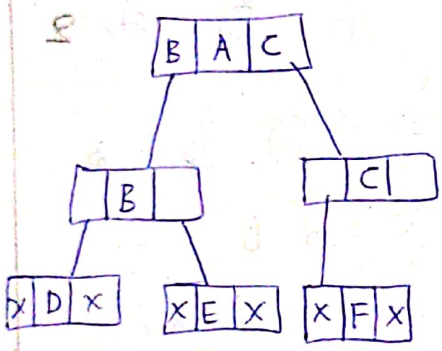
height of root node = height of tree

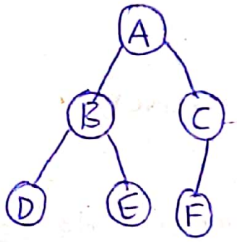
Level of node - No. of edges from root to given node

Level of tree = Level height of tree

To find eges: (n-1) edges

Binary Tree





Memory Allocation:

```
struct node
{
// int, float  char data;
    struct node *left;
    struct node *right;
};
```

Disaddv of Array
Static memory

Applications:

→ It stores Hierarchial data, implements file system.

→ Routing Protocols.

→ It organizes data for quick search, insertion and deletion i.e., Binary tree, heap tree etc.,

depth of root node, height of leaf node is always "0".

→ Height and depth of node may or may not be same.

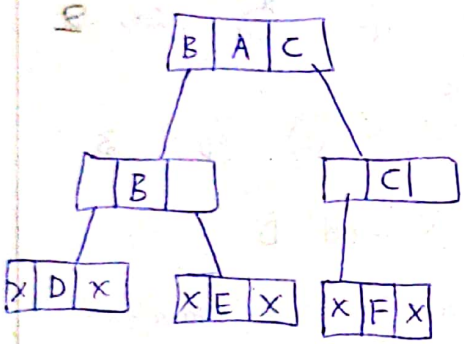
height of root node = height of tree

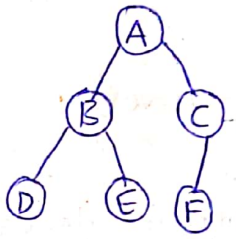
Level of node - No. of edges from root to given node

Level of tree = Level height of tree

To find eges: (n-1) edges

Binary Tree





Memory Allocation:

```
struct node
{
// int, float  char data;
    struct node *left;
    struct node *right;
};
```

Disadv of Array
Static memory

Applications:

→ It stores Hierarchial data, implements file system.

→ Routing Protocols.

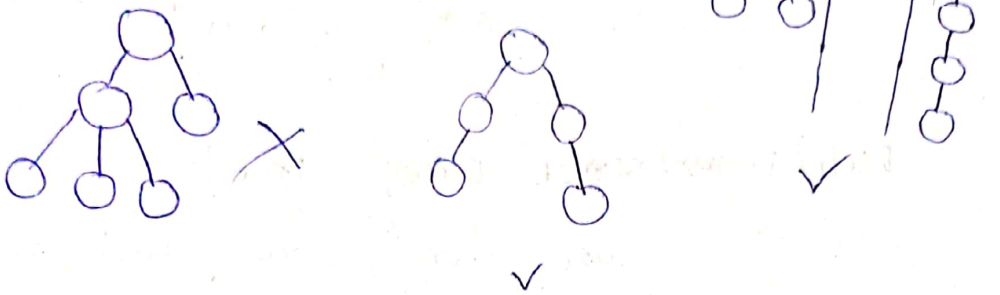
→ It organizes data for quick search, insertion and deletion i.e., Binary tree, heap tree etc.,

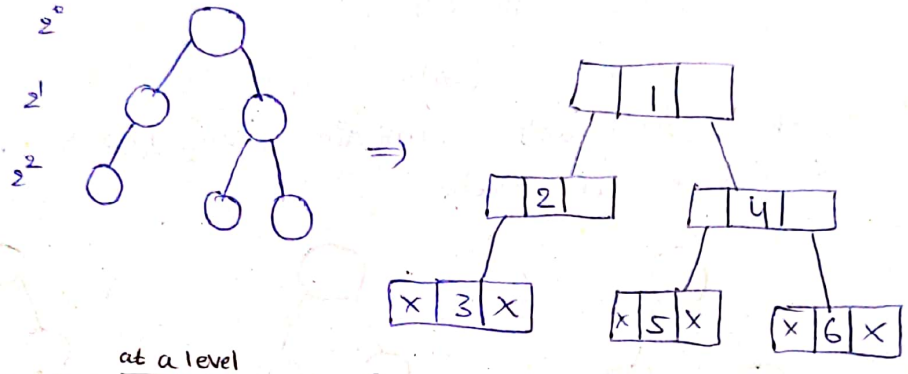
depth of root node, height of leaf node is always "0".

5.2

Binary tree & its Types

→ It is a tree in which having each node max (atmost) of 2 children.
i.e., having 0, 1, 2 children





Max possible nodes at a level = 2^i

Max no. of nodes of height $h = 2^{h+1} - 1$

min no. of nodes of height $h = h+1$

Q. Find the max height of tree?

max height =

min height =

$$n = 2^{h+1} - 1$$

$$n+1 = 2^{h+1}$$

$$\log_2(n+1) = \log_2 2^{h+1}$$

$$\log_2(n+1) = h+1$$

$$h = [\log_2(n+1) - 1] \rightarrow \text{min}$$

$$n = h+1$$

$$h = n-1 \rightarrow \text{max}$$

Types of Binary trees :-

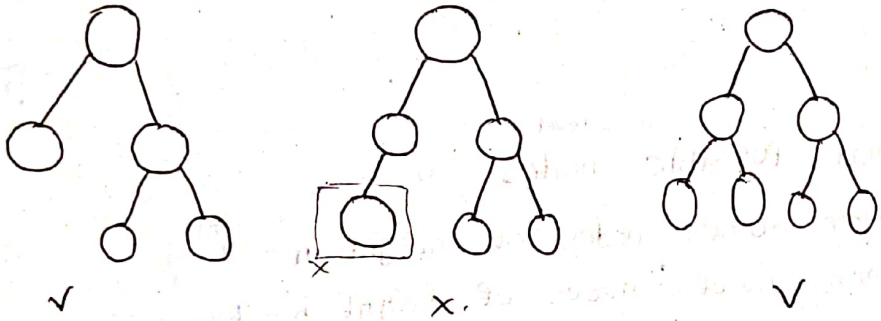
- Binary Tree
  - Full/proper/strict
  - comple Binary tree
  - perfect Binary tree
  - Degenerate Binary tree

Full/proper/strict Binary Tree :-

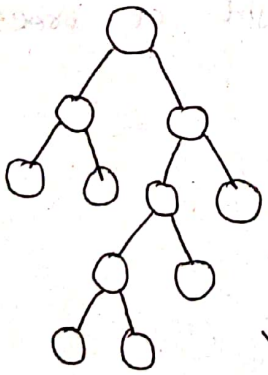
→ It is Binary tree where each node conta
0 (or) 2 children.

(or)

each node will contain exactly 2 children except leaf node.



✓ ✗ ✓



✓

no.of leaf node = no.of internal nodes +1

$6 = 5+1$

$6 = 6$

Max node = same as binary tree

$2^{h+1} - 1$

Min nodes = 2^{h+1}

Min height = $\lceil \log_2 (n+1) \rceil - 1$

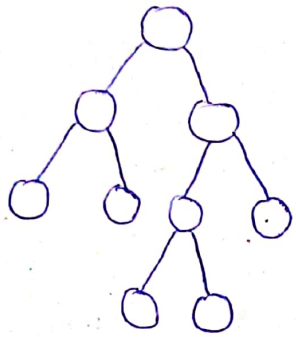
Max height $= \left(\frac{n-1}{2}\right)$

$n = 2h+1$

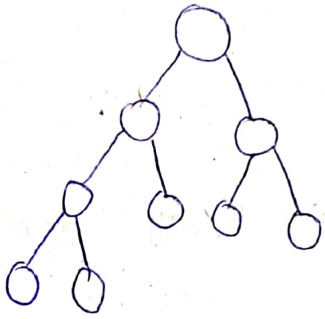
$h = \frac{n-1}{2}$

Complete Binary Tree :-

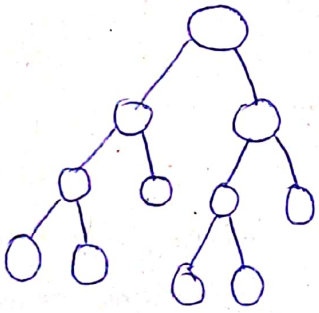
→ If a tree is a complete binary tree if all the levels are completely filled. (except possibly the last level has nodes as left as possible)



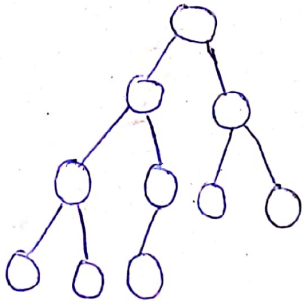
×



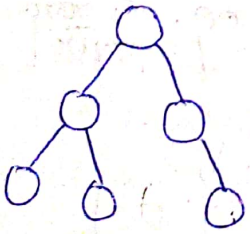
✓



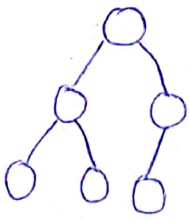
×



✓



×



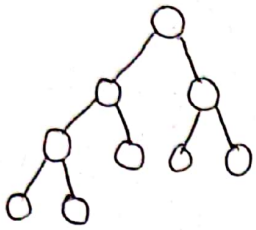
✓

Max nodes $= 2^{h+1} - 1$ min nodes $= 2^h$

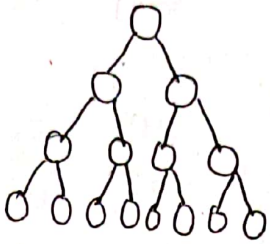
Max height $= \lceil \log_2(n+1) \rceil - 1$ max height $= \log n$

Perfect Binary Tree:-

A Tree can be a perfect Binary Tree if
→ all the internal nodes having 2 children
→ all the leaf nodes are at same level.



×

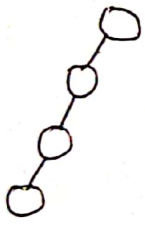


✓

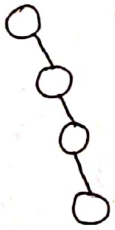
Every perfect Binary tree can be Full, complete Binary tree. vice versa not possible.

Degenerate Binary Tree:-

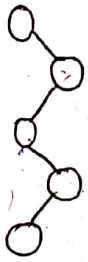
All the internal nodes have 1 children & leaves are at same level.



Left-scewed BT

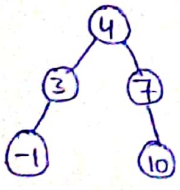


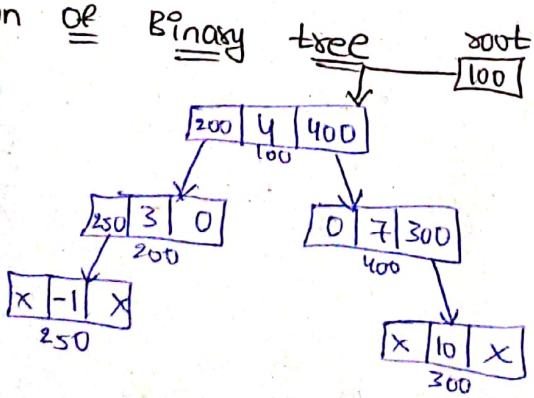
right-scewed BT



✓ mixture of left, right scwed BT

5.3 Implementation of Binary tree



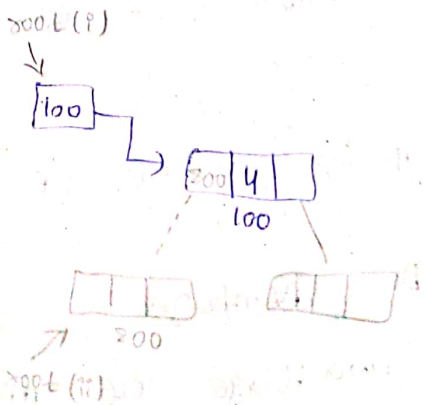


Contains 0, 1, 2 any can possible.

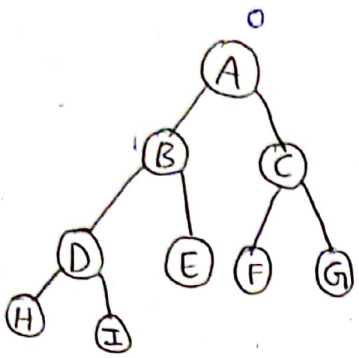
code:

```
#include <stdio.h>
#include <stdlib.h>
Struct node
{
    int data;
    struct node *left, *right;
};
// int create()
// {
struct node *create()
{
    int x;
    struct node *newnode;
newnode=(struct node *) malloc (size of (struct node));
Printf("enter data");  or("enter data (-1 for no node):");
                                    //for empty node
scanf ("%d", &x);
    if (x==-1)
    {
        return 0;
    }
newnode→data = x;
Printf("enter left child of %d", x);
newnode→left = create();
Printf("enter right child of %d", x);
newnode→right = create();
    return newnode;
```

```
Void main() {
    struct node *root;
    root = 0;
    root = create();
```



5.4 Array Representation of Binary Tree
(sequential representation)



⟹

A	B	C	D	E	F	G	H	I
0	1	2	3	4	5	6	7	8

case-I

A	B	C	D	E	F	G	H	I
1	2	3	4	5	6	7	8	9

case-II

case-(i)

ie a node is at i^{th} index:-

→ left child would be at: $[(2*i)+1]$

→ right child would be at: $[(2*i)+2]$

→ Parent would be at: $\left[\frac{(i-1)}{2}\right]$ → floor value

Casee-(II)

ie a node is at i^{th} index:

left child at: $(2*i)$

right child at = $[(2*i)+1]$

Parent at: $\left[\frac{i}{2}\right]$

→ Binary Tree must be complete Binary tree nodes must fill from left to right.

Having atmost of two children

Dis-advantage:

→ wastage of memory

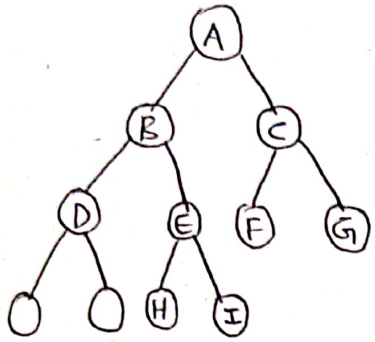
→ static memory Allocation

Advantage:

Easy to implement

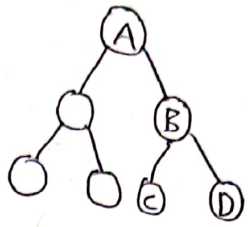
Easy to traverse

Q. Represent the following Binary Tree in Array form.



A	B	C	D	E	F	G	-	-	H	I
0	1	2	3	4	5	6	7	8	9	10

A	B	C	D	E	F	G	-	-	H	I
1	2	3	4	5	6	7	8	9	10	11

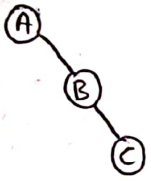


A	-	B	-	-	C	D
0	1	2	3	4	5	6

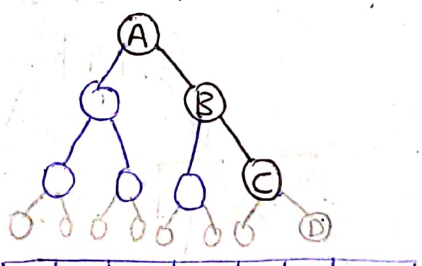
A	-	B	-	-	C	D
1	2	3	4	5	6	7

If H, I can placed at 7,8 (or) 8,9 positions then above formulas will not be applicable.

Q. Write array form:



Not complete B.T
Not possible



A	-	B	-	-	-	C
0	1	2	3	4	5	6

If the given tree is not complete B.T then, you have to make it as CBT using empty nodes.

A	-	B	-	-	-	C	-	-	-	-	-	-	D

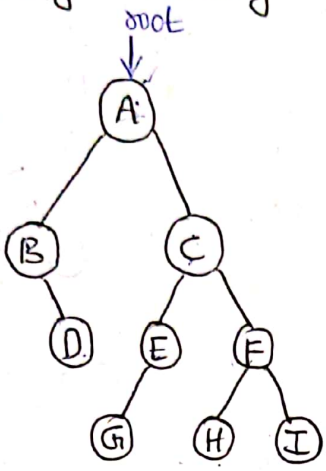
5.5/ Binary Tree Traversal

Inorder - Left Root Right

Pre-order - Root Left Right

Post-order - Left Right Root

Q. Write Inorder, Preorder and Post-order to the following Binary Tree.



left, root, right

In-order = BDAGECHFI

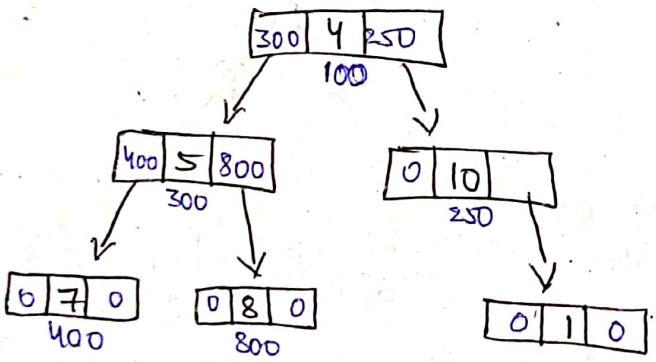
root, left, right

Pre-order = ABDCEGFHI

left, ~~root~~, right, root

Post-order = ~~EDBA~~ DBGEHIFCA

Q. Write Program for shown Tree. 5.6



In-order = left root right

7 5 8 4 10 1

Pre-order = root left right

4 5 7 8 10 1

Post-order = left right root

7 8 5 1 10 4

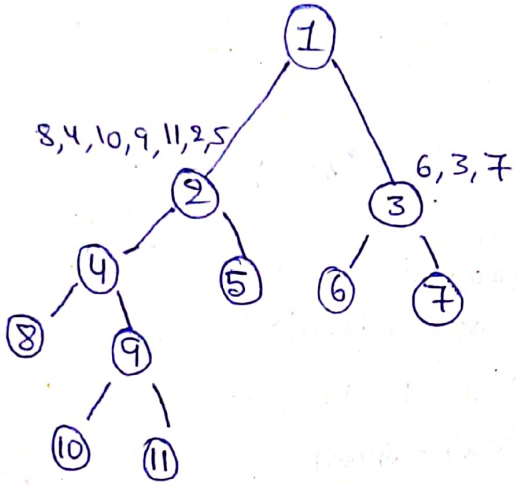
```
void main()
{
   struct node *root,
    Printf ("Pre-order is-");
     Preorder (root);

}
void Preorder( struct node *root)
      {    if (root==0)
              { return; }
        Printf ("%d", root->data);
         Preorder (root->left);
         Preorder (root->right);
      }

void Postorder ( struct node *root)
   {
      if (root==0)
        { return ; }
    Postorder (root->left)
     Postorder (root->right)
      Printf ("%d", root->data);
   }
void inorder (struct node *root)
   {
       if (root==0)
        { return ; }
      inorder (root->left);
      Printf ("%d", root->data);
      inOrder (root->right)
   }
```

5.7

Construct a Binary tree from Preorder & Inorder.

Preorder - 1, 2, 4, 8, 9, 10, 11, 5, 3, 6, 7 (R L right)

Inorder - 8, 4, 10, 9, 11, 2, 5, 1, 6, 3, 7 (L Root right)



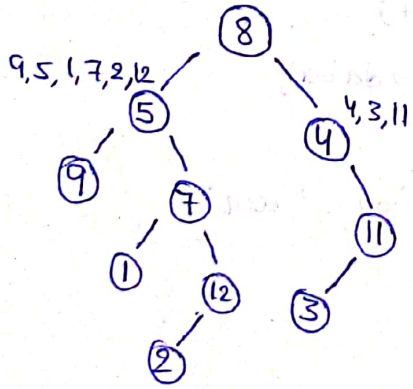
5.8

Construct a Binary tree from Postorder & Inorder

Postorder - 9, 1, 2, 12, 7, 5, 3, 11, 4, 8 (L right root)

Inorder - 9, 5, 1, 7, 2, 12, 8, 4, 3, 11 (L root right)

root



5.10

Binary search Tree

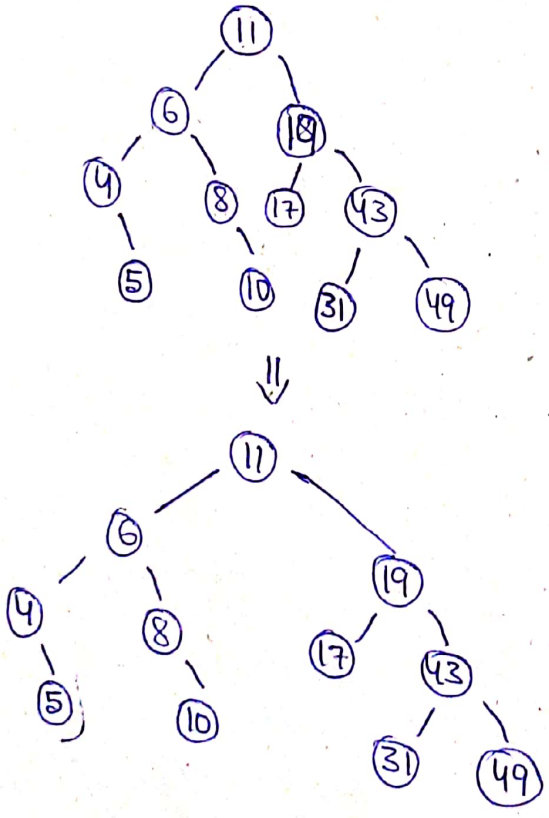
* It having atmost of two children

  Eg: having 0, 1 (or) 2 children.

* Left sub-tree of any node having less values whereas right sub-tree of high values. to th root node

Draw Binary search tree by inserting the follow numbers from left to right.

11, 6, 8, 19, 4, 10, 5, 17, 43, 49, 31



Deletion

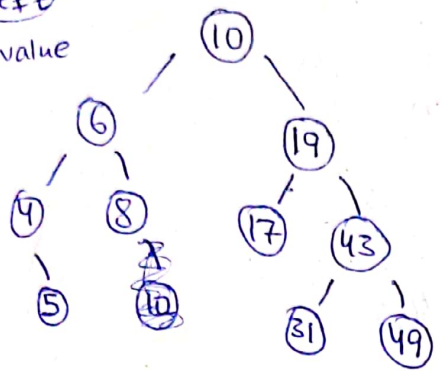
1. 0 child
2. 1 child
3. 2 children
   - (i) In-order Predecessor
   - (ii) In-order successor

→ In the left we have to swap the highest number

→ In the right we have to swap the lowest number

(3) In-order Predecessor

left
↑ value



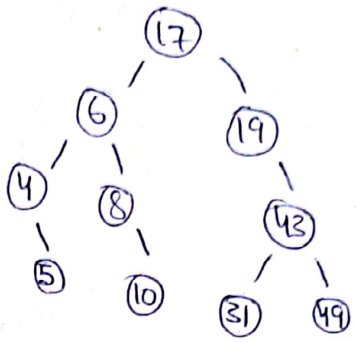
In-order

4, 5, 6, 8, 10, 11, 17, 19, 31, 43, 49

Pre (10) × succ (11)

In-order successor

right
↓ value

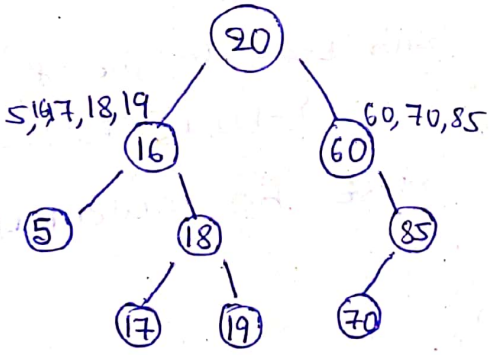


5.11

construct a Binary search tree ~~Pre~~-order.

Pre-order = 20, 16, 5, 18, 17, 19, 60, 85, 70 (Root-left-right)

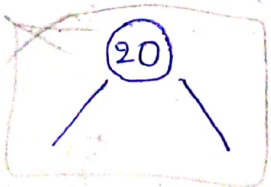
In-order = 5, 16, 17, 18, 19, 20, 60, 70, 85 (left-root-right)

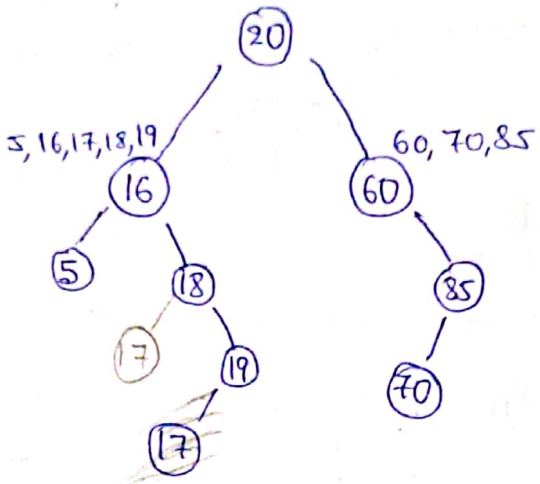


construct a Binary search tree from Post-order.

Post-order = 5, 17, 19, 18, 16, 70, 85, 60, 20 (left-right-root)

In-order = 5, 16, 17, 18, 19, 20, 60, 70, 85





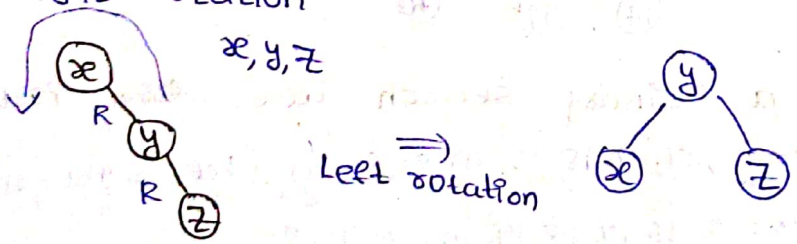
5.13

AVL Tree

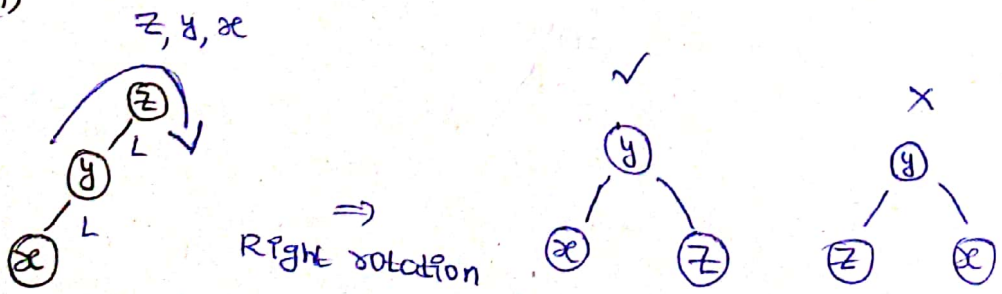
- Jim Oliver

* AVL means Adelson Velsky and Landis tree
  → self-branching BST
→ It is a BST, self balancing BST
→ For each node in the tree:
  height of left sub-tree - height of right sub-tree
  = {-1, 0, 1} → Balance Factor
→ Balance factor must be calculated for every level.

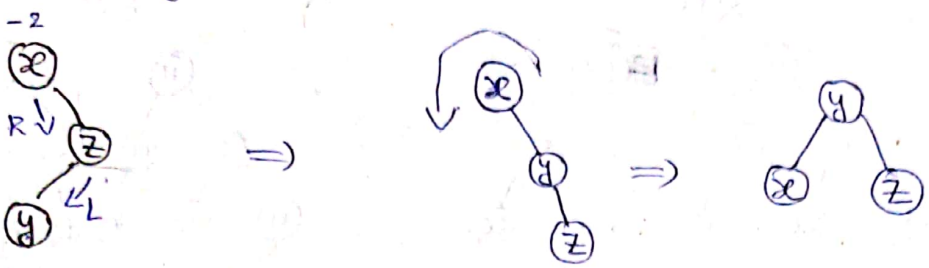
case-(i) Right rotation



case-(ii)



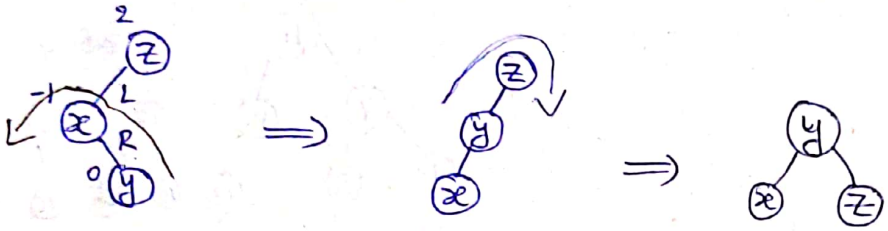
case-(iii) x, z, y Right-left rotation.



Un-balanced

case-(iv) Left-right rotation

z, x, y



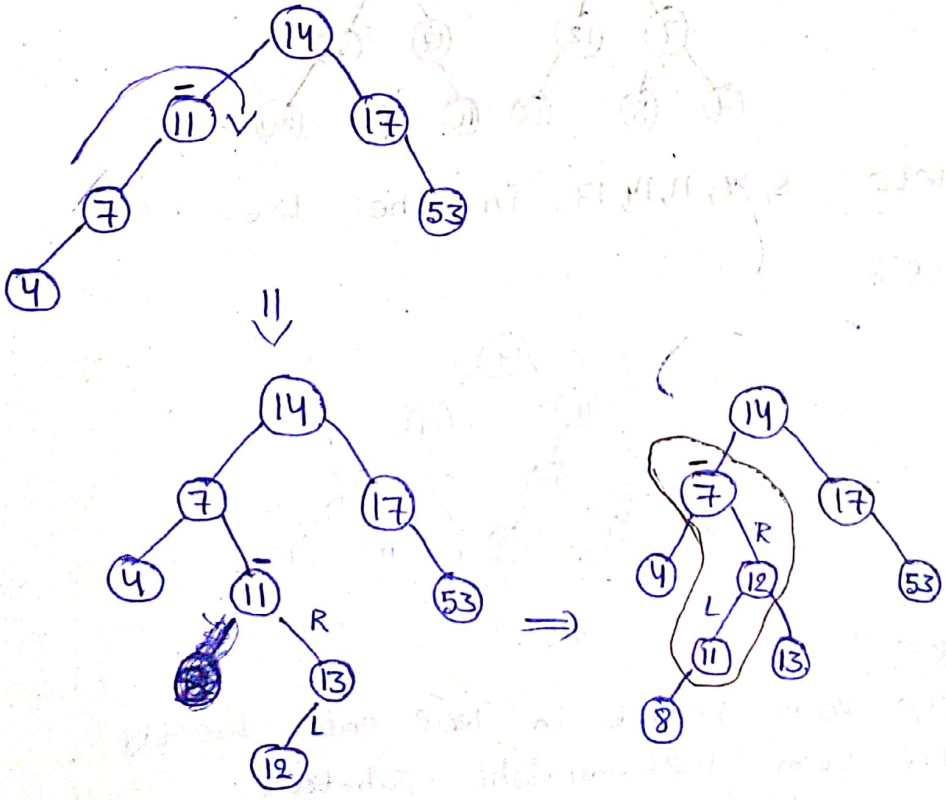
5.14

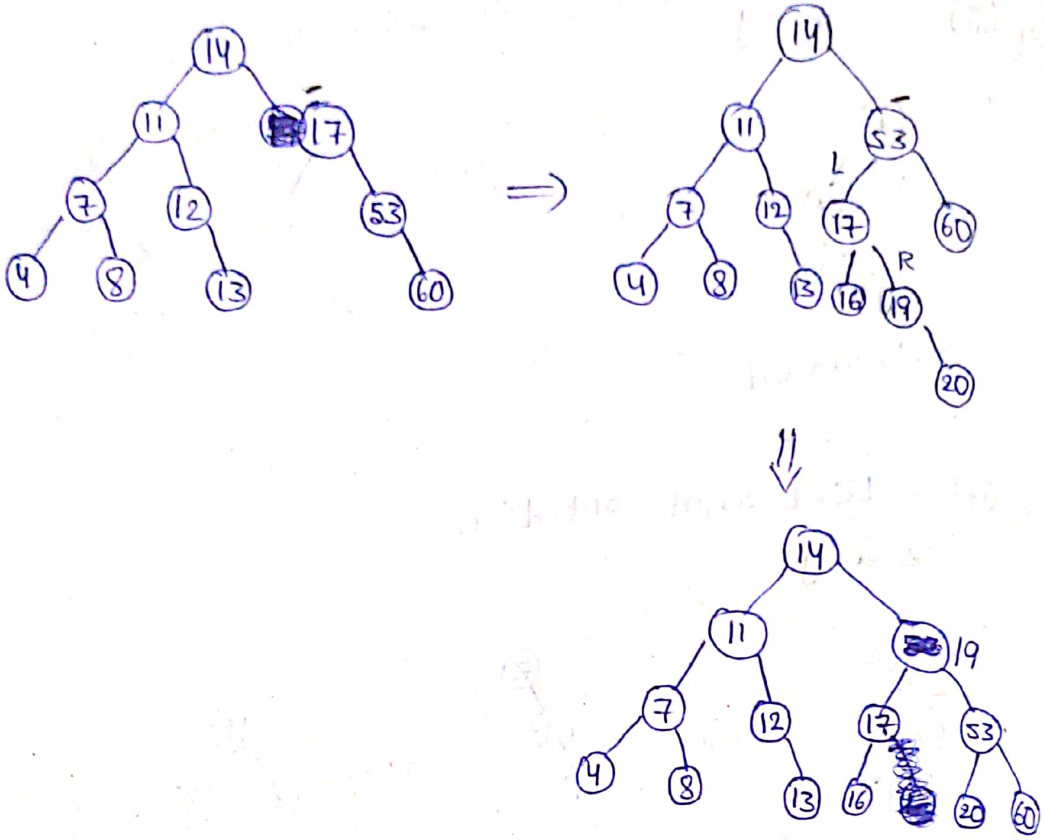
construct AVL tree by inserting the data:

14, 17, 11, 7, 53, 4, 13, 12, 8, 60, 19, 16, 20

→ It is BST

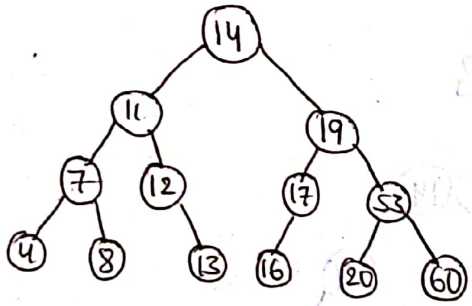
→ L-R = {-1, 0, 1}





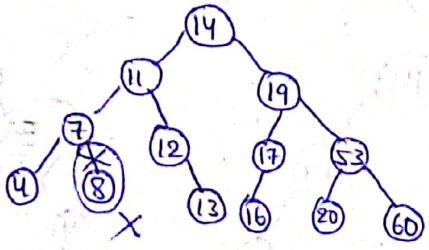
5.15

Deletion in AVL Tree



delete 8, 7, 11, 14, 17 in the tree.

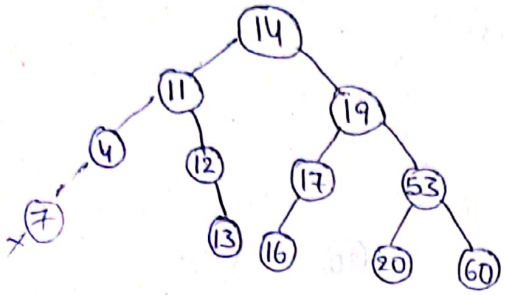
del of 8



Steps:

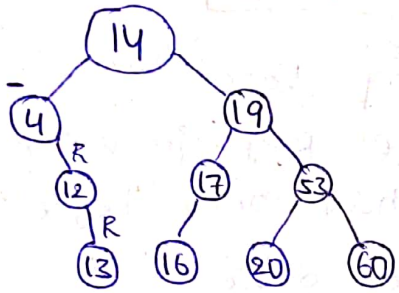
→ delete value present in leaf node directly
→ either from left (or) right sub-tree
left - replace with highest value
right - replace with lowest value
→ check with balance factor

del 7

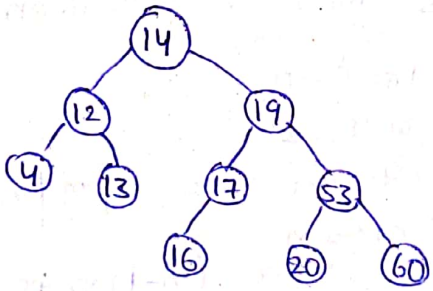


del 11

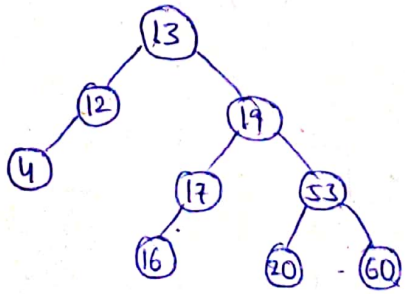
11 have two children you will delete either from left (or) right sub-tree.
i'm taking left.



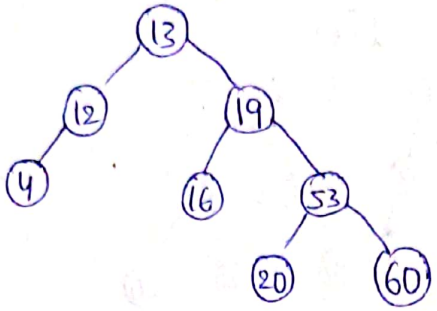
'4' is the critical node which is not balanced



del 14



del 17

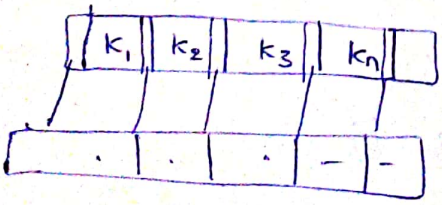


5.23

B - Trees

* balanced (m) order way tree
* Generalization of BST in which a node can have more than one key & more than 2 childre
* maintains sorted data.
* All leaf nodes must be at same level.
* B-tree of order m has following properties:
  - → every node has max m children.
  - → min children - leaf → 0
    root → 2
    internal nodes → $\lceil \frac{m}{2} \rceil$ → ceiling value
  - → every node has max children = m
    max (m-1) keys
  - → min keys - root node → 1
    all other nodes → $\lceil \frac{m}{2} \rceil - 1$

m=5



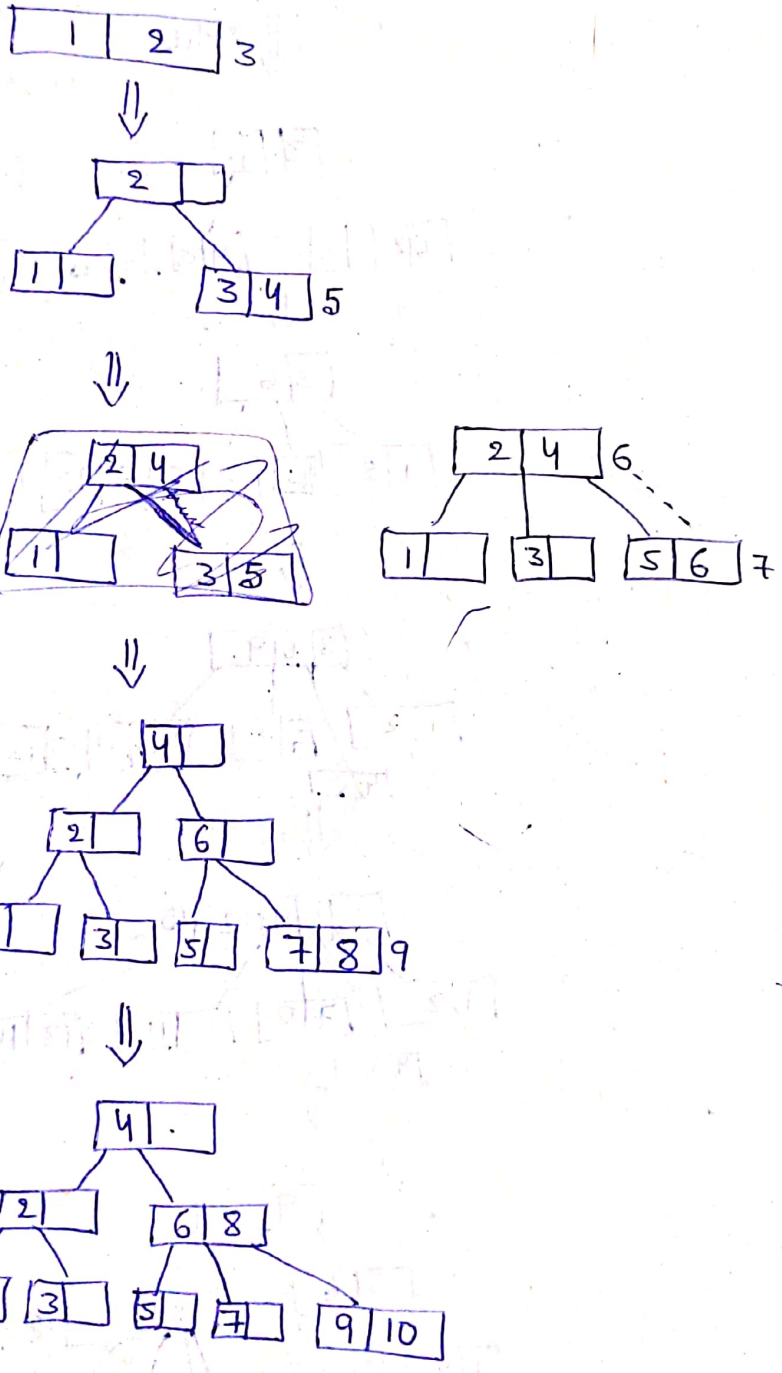
⇒ Insertion possible at only leaf node.

5.24

create a B-tree of order-3 by inserting values from 1 to 10.

m = 3

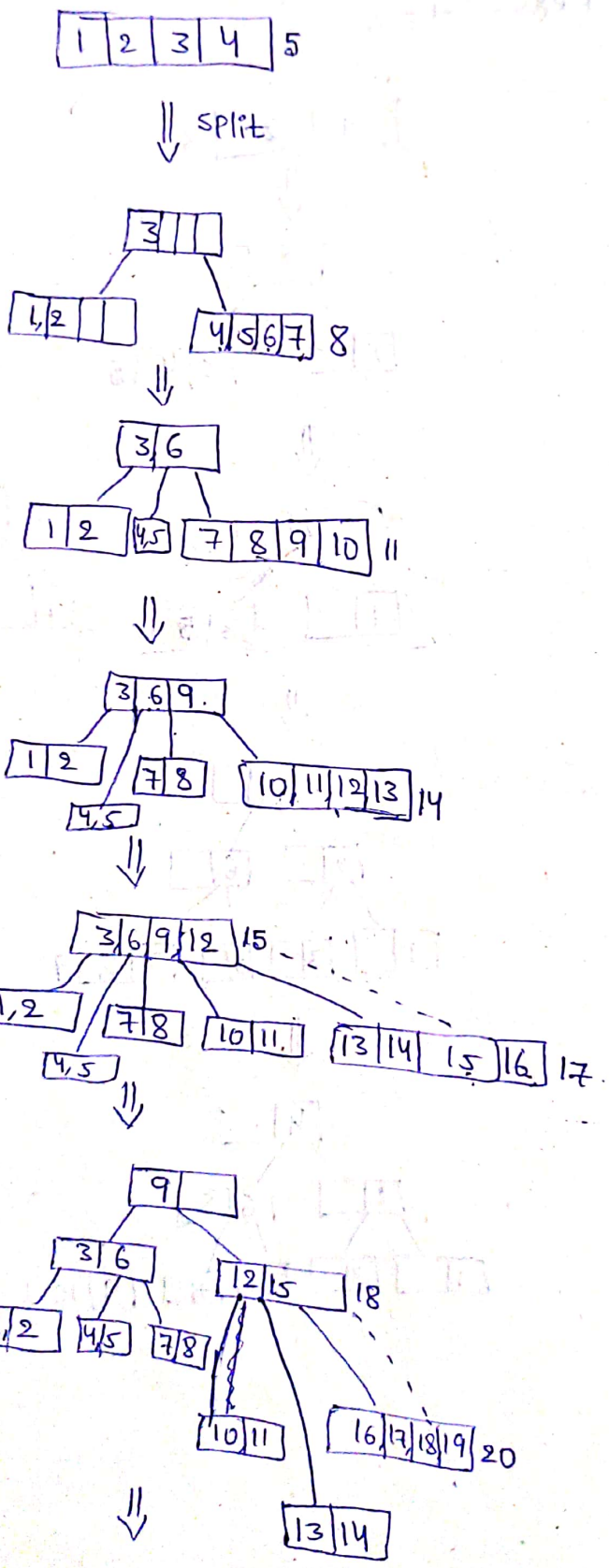
keys = 3-1 = 2

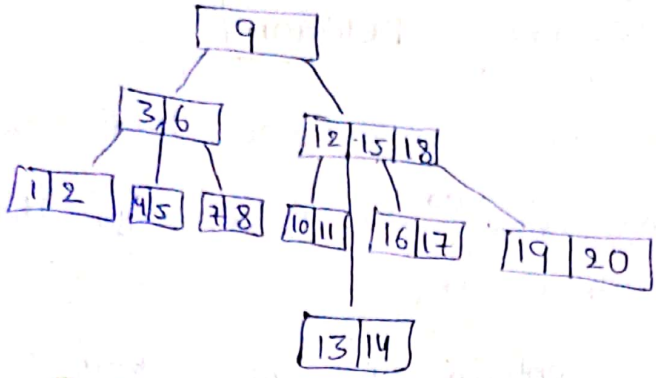


5.25

Create a B-tree of order 5 by inserting values from 1 to 20.

m = 5, keys = 5-1 = 4

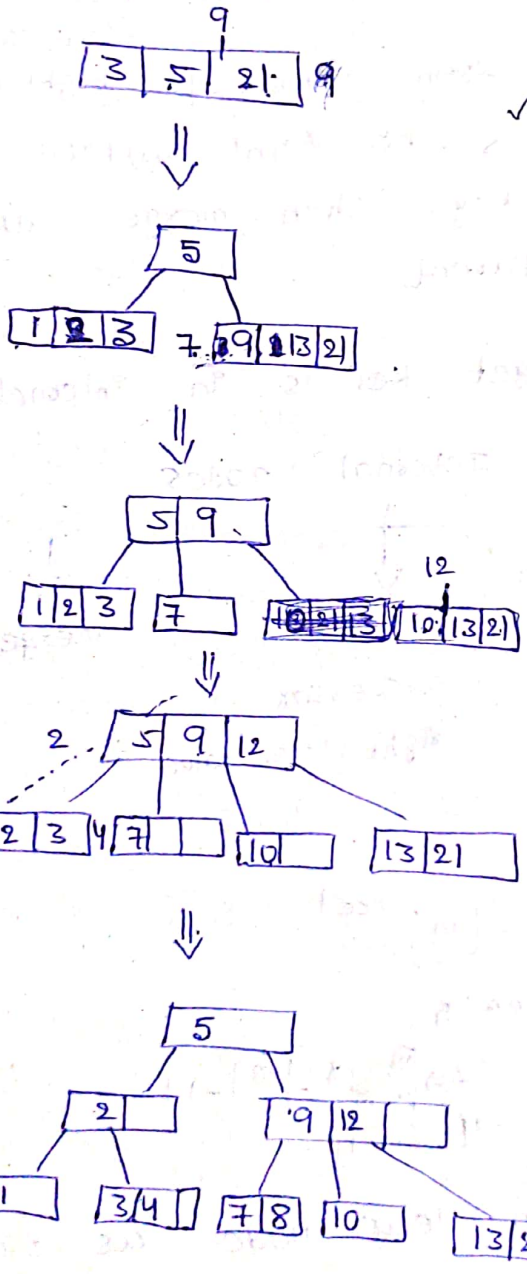




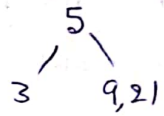
5.27

construct B-tree of order 4 by inserting 5, 3, 21, 9, 1, 13, 2, 7, 10, 12, 4, 8

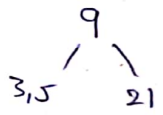
m=4, keys=3



✓ left-biased



right-biased



for ins of 8

even order → 2 possible trees

(i) left-biased

(ii) right-biased

B-Tree Deletion

① If target key is in the leaf node

(i) contains (more than max / min no.of) keys then directly delete it.

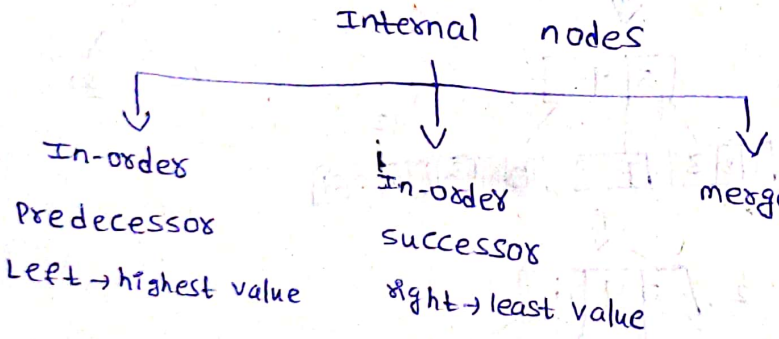
(ii) Leaf node contains min no.of keys

(a) borrow from immediate left sibling node. Then (highest value is replaced) take it from left and put it into the Parent and Parent value into that node.

(b) borrow it from immediate right node (sibling) (lowest value is replaced)

(c) If there is no right (or) left siblings contain min no.of keys then merge either with left (or) right Parent.

② If the target key is in internal nodes.



Order (m) = 5

min children = $\lceil \frac{m}{2} \rceil$ → ceil = 3

max children = 5

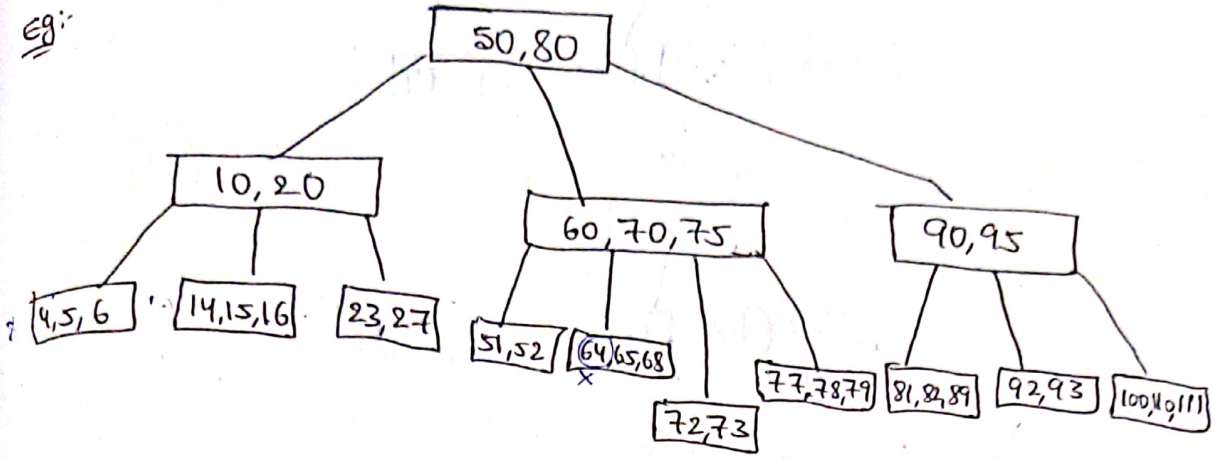
min keys = 2 → $\lceil \frac{m}{2} \rceil - 1$

max keys = 4 → (m-1)

→ Data stored in leaf node as well as internal node.

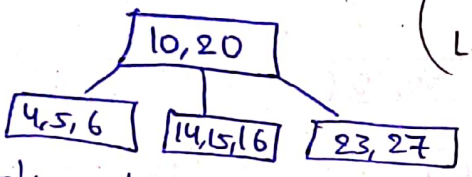
order = 5

Eg:



del 64: 64 is a leaf node, having max keys, then directly delete 64

del 23



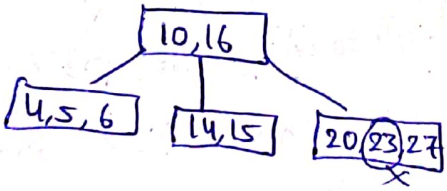
(Borrow from Left sibling)

max can replaced to Parent

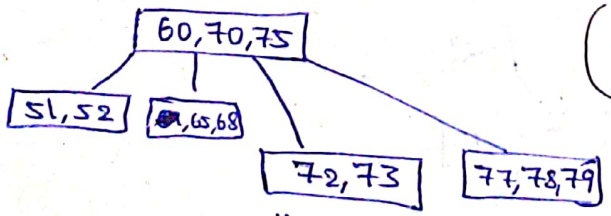
we can't directly delete 23, bcz it have min no.of nodes.

→ Borrow element from left sibling, cz it have more than min no.of keys

→ can't directly taken, max value is transferred to parent and common for both will come down.

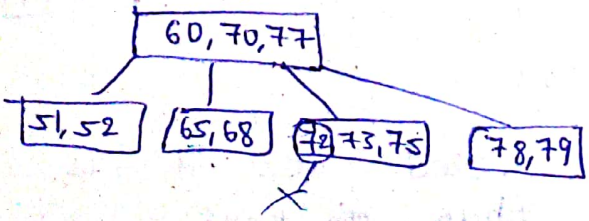


del 72

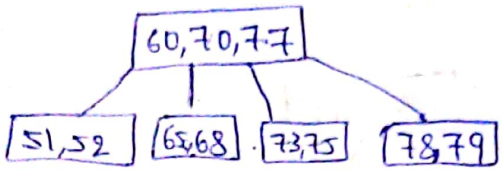


(Borrow from left sibling)

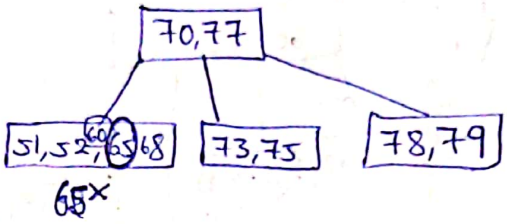
min can replaced to Parent



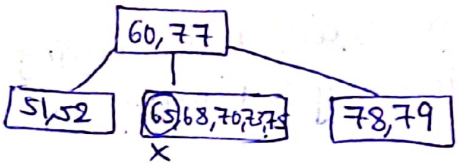
del 65



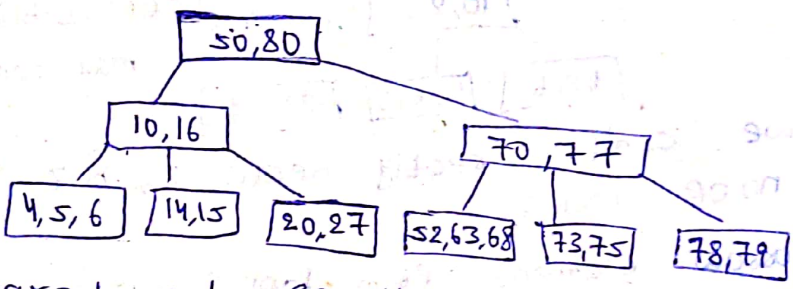
(merging)



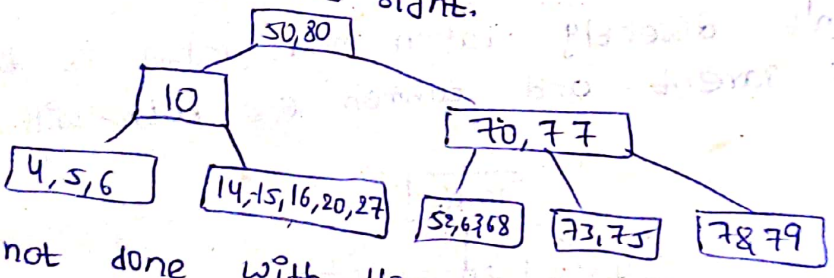
(or)



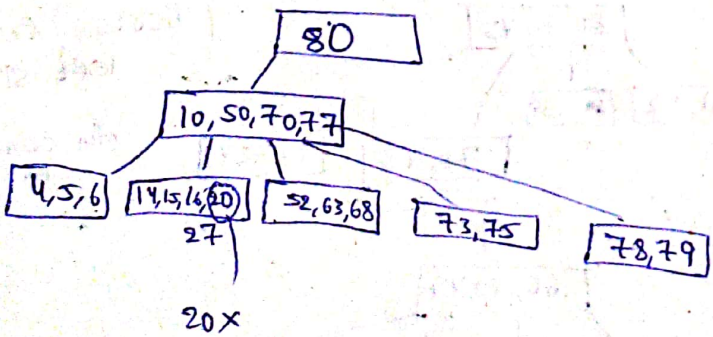
del 20



→ consider Parent node if there only min no.of nodes from left and right.

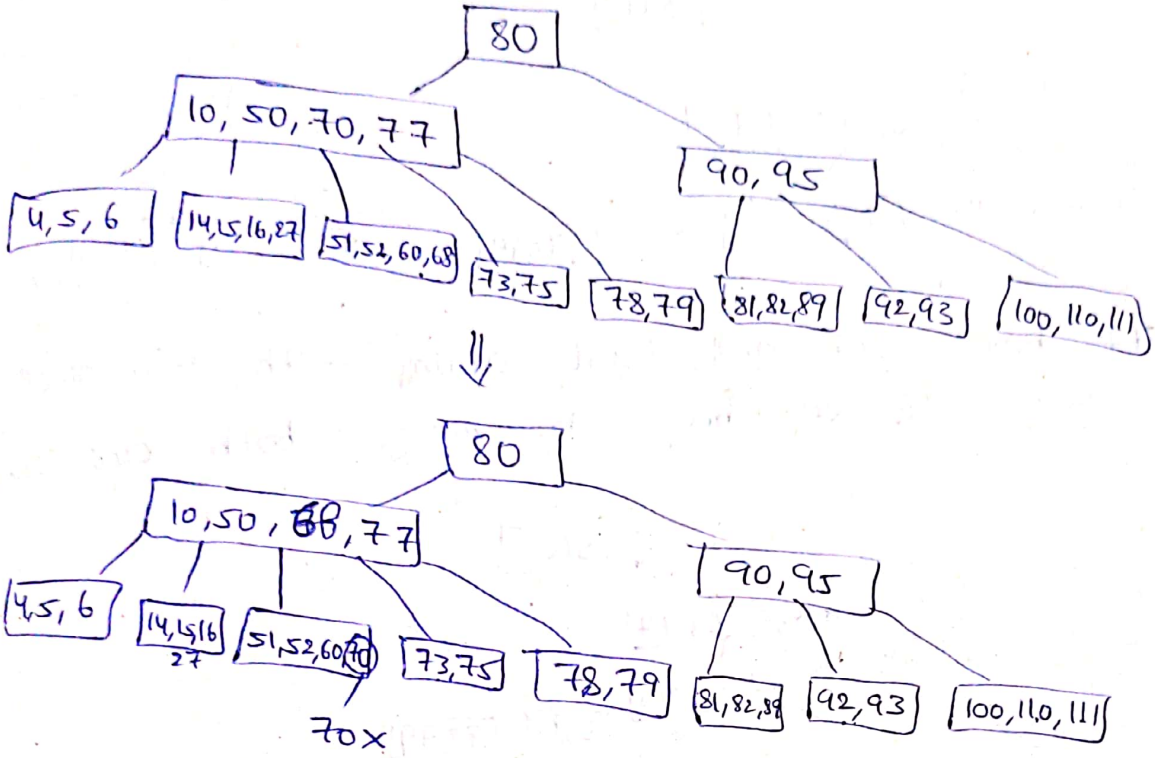


we're not done with this bcz every node must have min no.of nodes except root node.

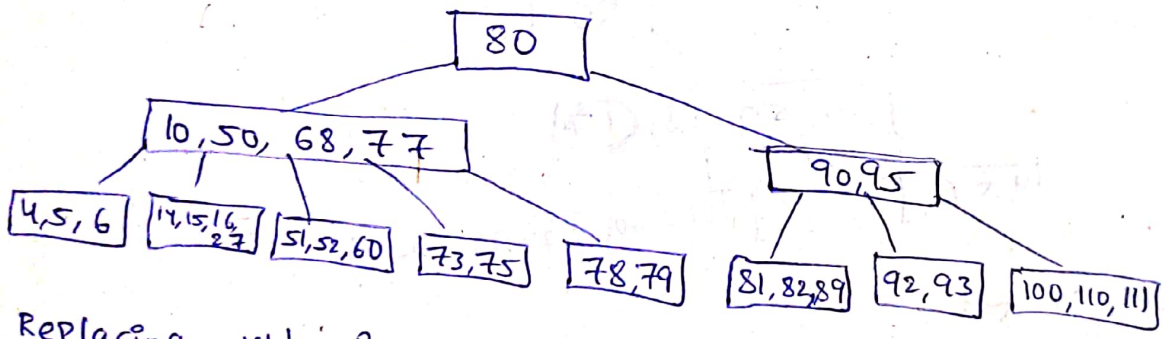


→ 20 can easily deleted bcz it is in leaf node having more than min keys.

del 70

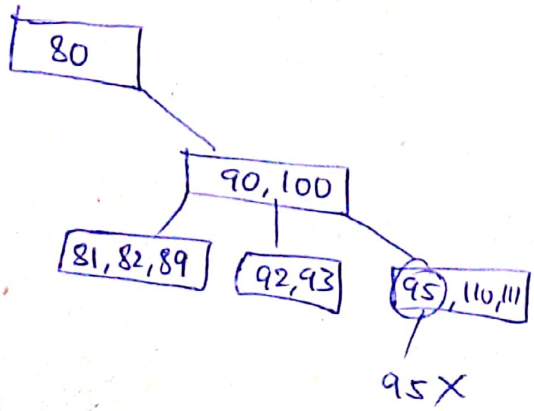


del 95

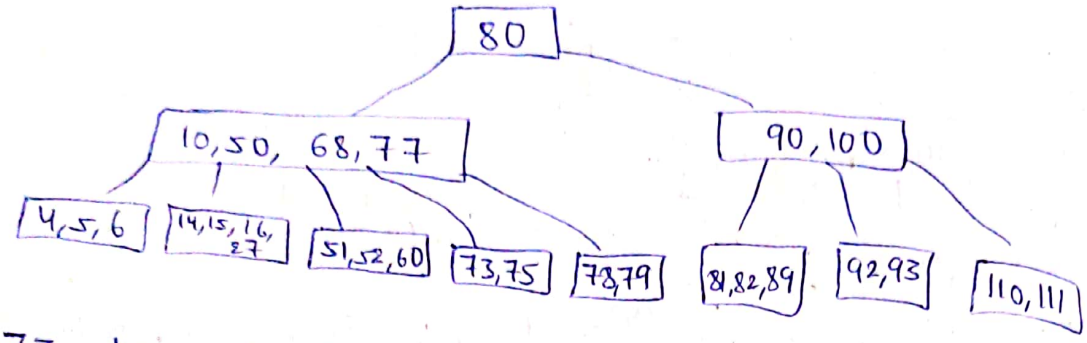


Replacing vth inorder predecessor → can't possible bez it having min no of nodes

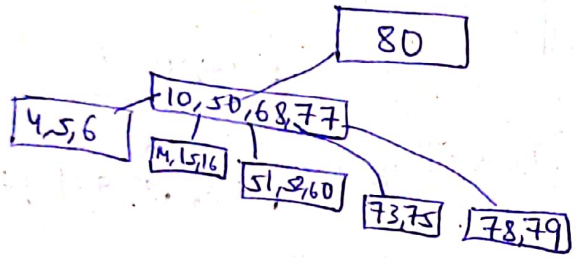
we are moving inorder successor

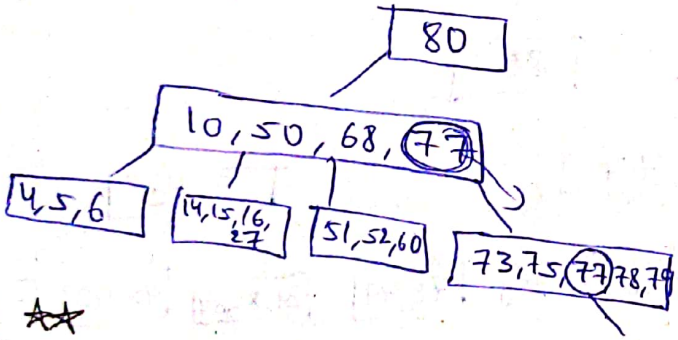


del 77



77 have left and right sibling with min no. of keys. so, we have to merge both our sibl





77x

del 90 ★★

5.29

B+ Tree

→ Data is stored only in leaf node.

→

Order (m) = 4

max children = 4

min children = $\lceil \frac{m}{2} \rceil = 2$

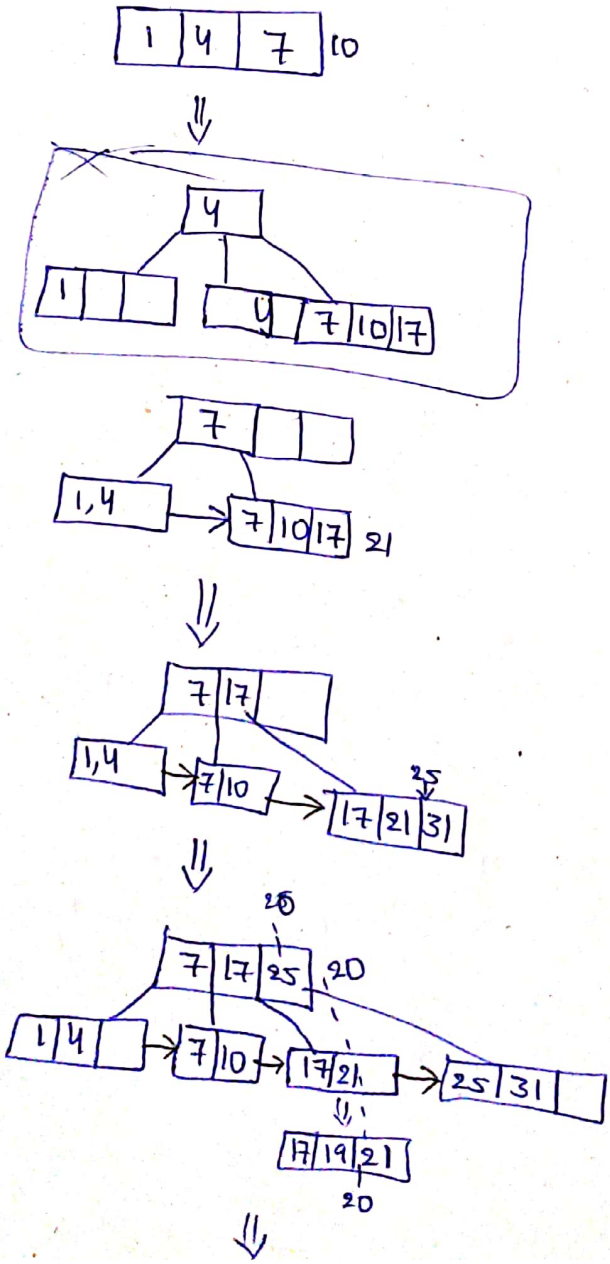
max keys = (m-1) = 3

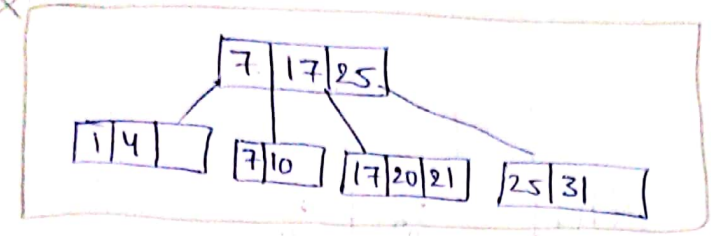
min keys = $\lceil \frac{m}{2} \rceil - 1 = 1$

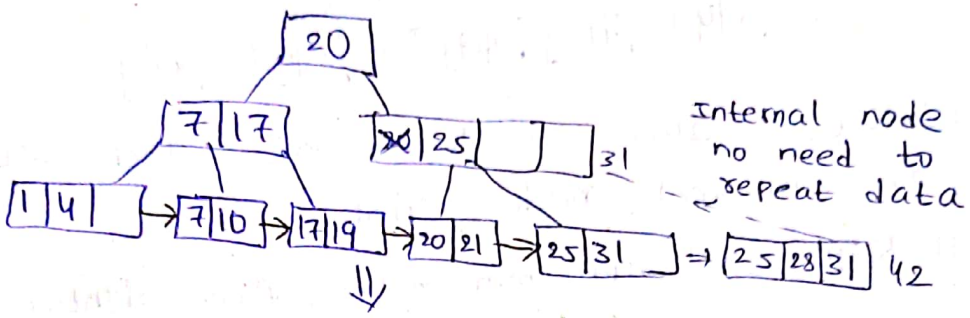
} except root node

Creation of B+ tree :-

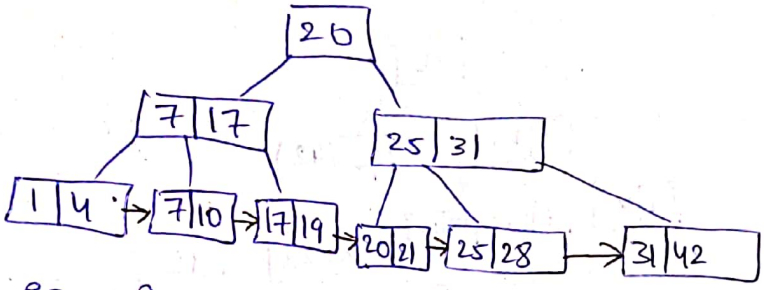
1, 4, 7, 10, 17, 21, 31, 25, 19, 20, 28, 42







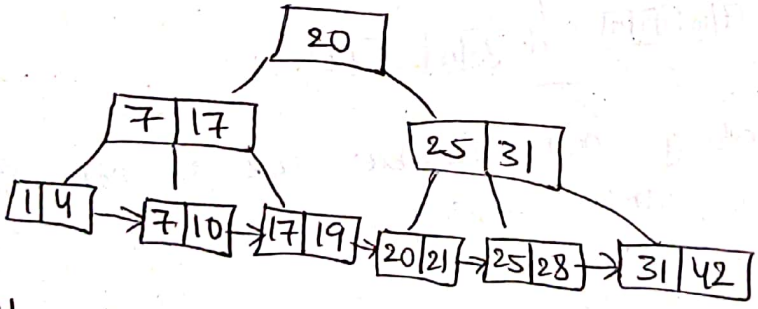
Internal node no need to repeat data



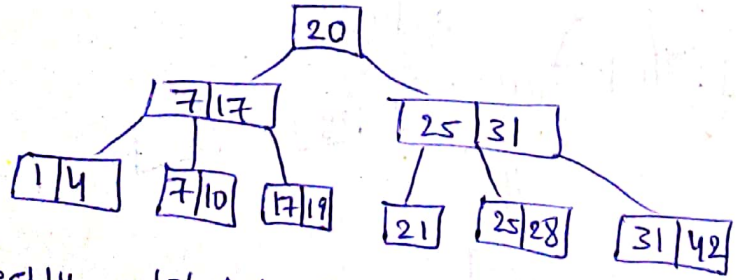
searching is in easy with B+ tree.

5.30

Deletion in B+ tree

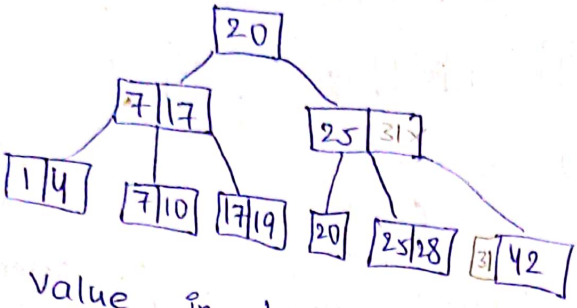


del 21



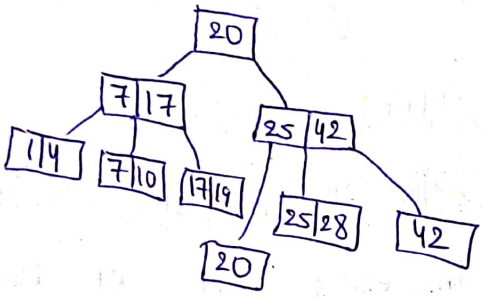
directly deleted BCZ it have more than min no.of keys.

del 31

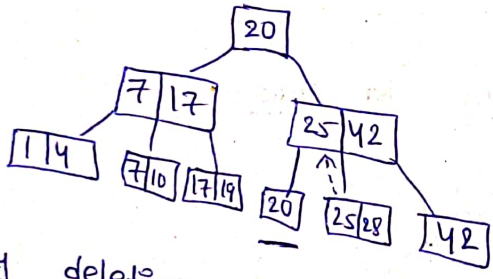


(i) delete value in leaf node then after delete in internal node also.

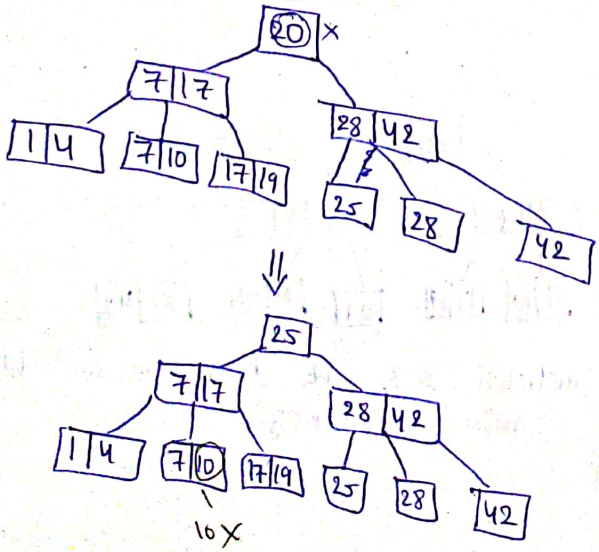
(ii) Replace with min value from right.



del 20

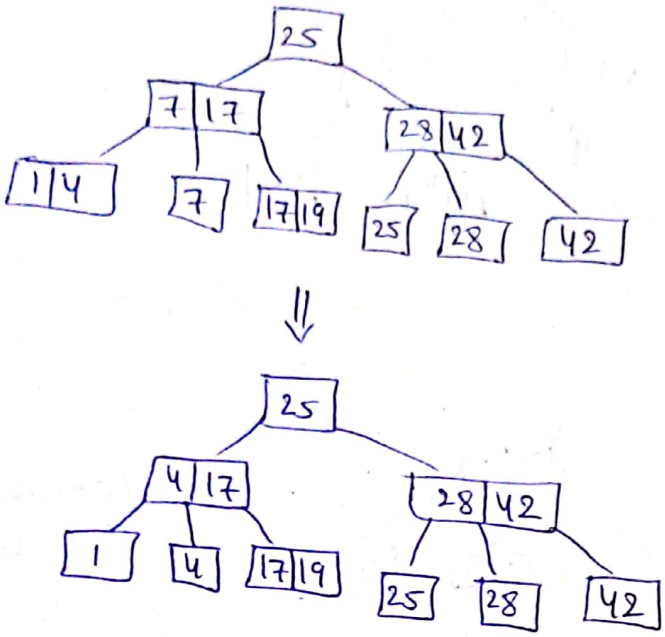


→ directly deleting not possible bcz it have min. Borrow from right sibling



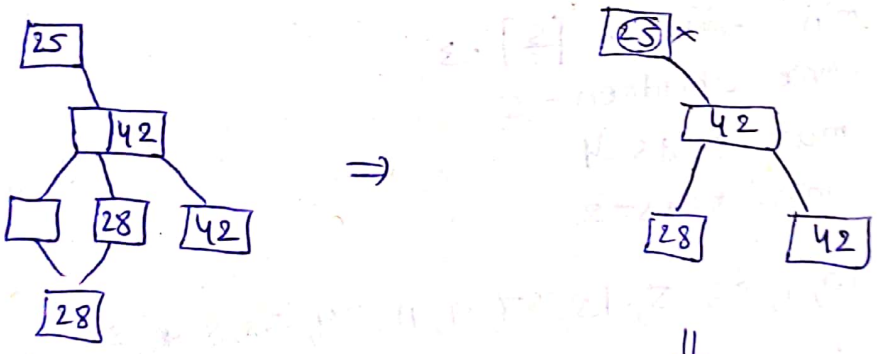
del 10 10x

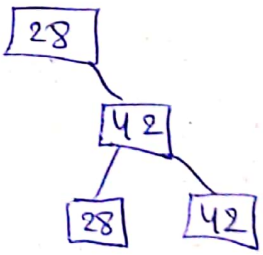
del 7



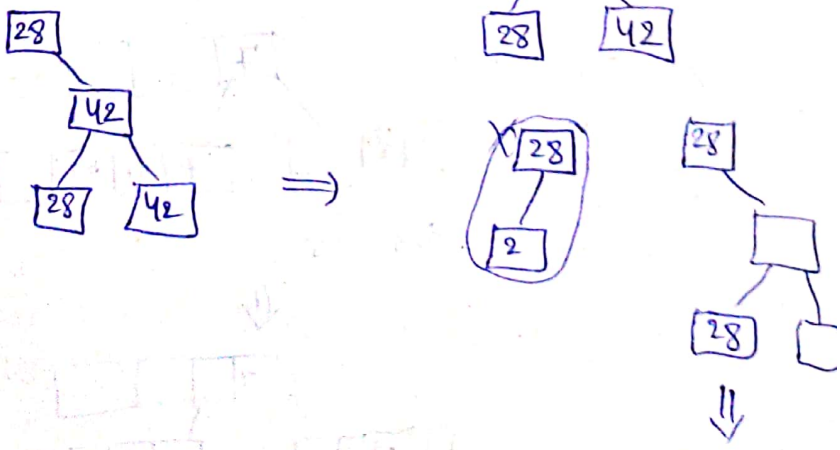
del 25

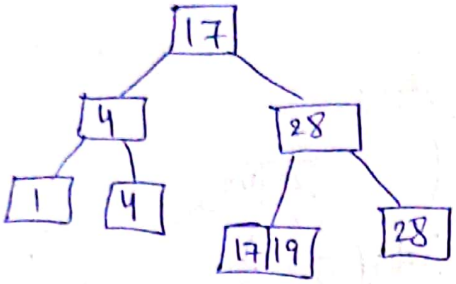
merging of nodes



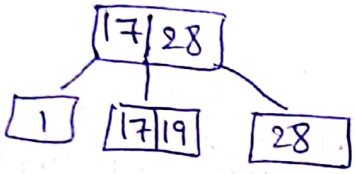


del 42





del 4



(shrinking) of the height

5.31

Insert below values in B+ tree of order = 5

order (m) = 5

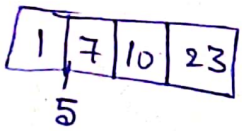
min children $\lceil \frac{5}{2} \rceil = 3$

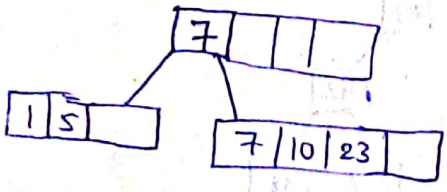
max children = 5

max keys = 4

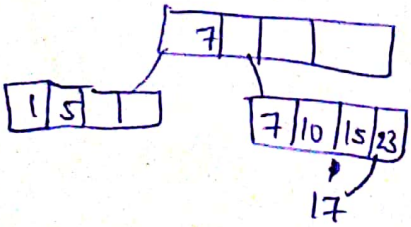
min keys = 2

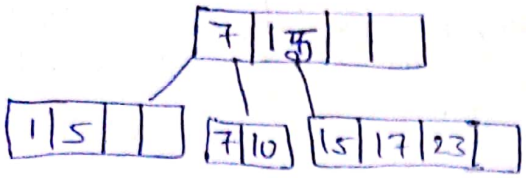
7, 10, 1, 23, 5, 15, 17, 9, 11, 39, 35, 8, 40, 25



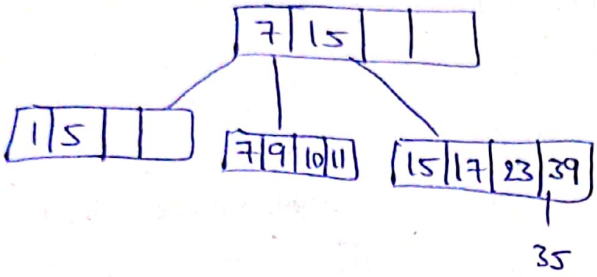


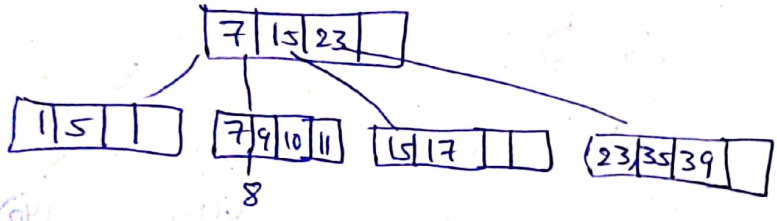
for 15

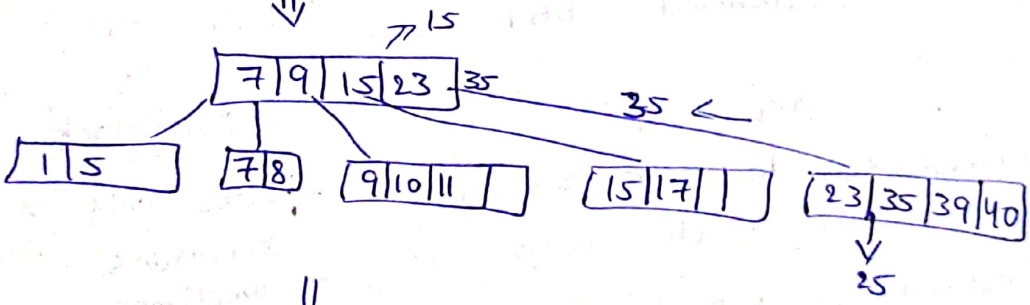


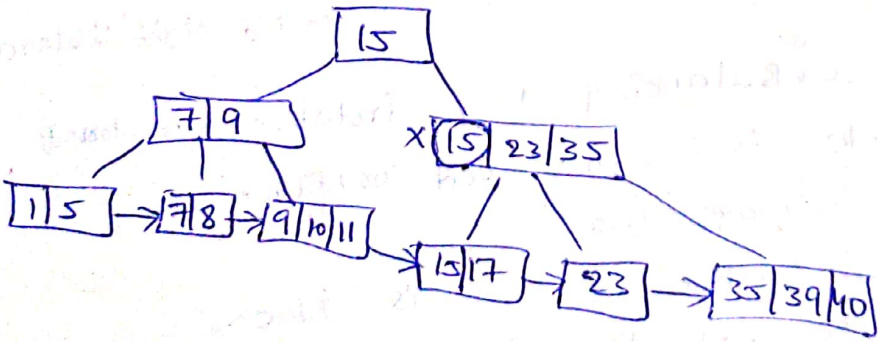


for 9, 11









Note: Root node will not follow the min keys rule.

5.16

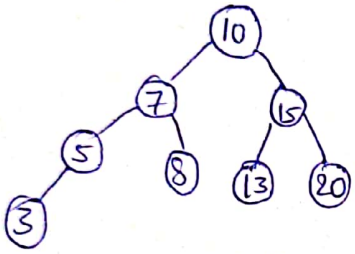
Red Black Tree

Introduction:

→ Tree is a data structure, it is a way of organizing data efficiently

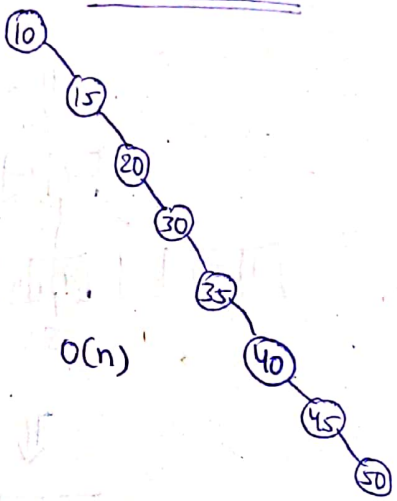
Average case - $O(\log_2 n)$
Best case - $O(1)$
Worst case - $O(n)$

BST



$O(h)$

right-skewed BST



$O(n)$

→ self Balanced BST

AVL

→ need to do more rotations to balance the tree
→ sub-set of R-B tree
→ strictly height balanced

Red-Black

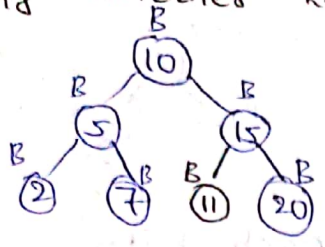
→ Here only 2 is max rotations.
→ sometimes no need to rotate, re-colouring only
→ Roughly hight balanced

Conditions:

→ It is a self Balancing BST (rotation (or) recoloring)
→ Every node is either red (or) Black.
→ Root is always black.
→ Every leaf which is Nil is Black.
→ If node is red then its children are Black
→ Every Path from a node to any of its descendent NIL node has same no. of Black nodes.

0 1
Black Red

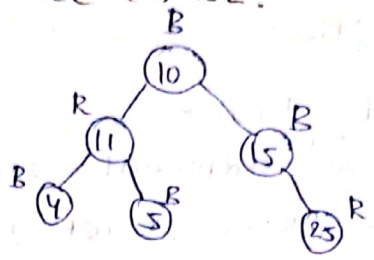
Q. Finding whether Red-Black tree (or) not?



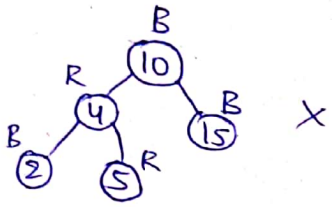
Not a R-B tree

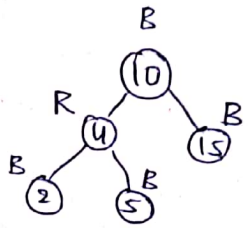
It is a R-B tree

→ Perfect BT is having only Black nodes is called Red-Black tree



→ NOT following BST so, it's not a Red-Black tree





→ BST, root is Black

→ Node Red → children Black

Red-Black tree

→ The farthest distance from root to leaf node is "not greater than twice of nearest path" from root to leaf node

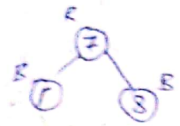
5.17

Insertion in Red-Black Tree:-

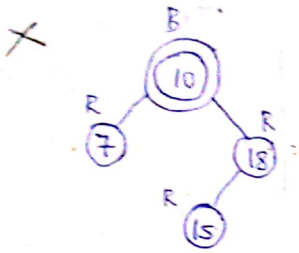
Rules:-

1) If there tree is empty, create newnode as root node with color Black.
2) If tree is not empty, create newnode as leaf node with color Red.
3) If Parent of newnode is Black then exit.
4) If Parent of newnode is Red, then check the color of Parent's sibling of newnode.
   (i) If color is Black (or) null then do suitable rotation & re-colour.
   (ii) If color is Red then recolour & also check, if Parent's Parent of newnode is not root node then recolor it and Re-check.

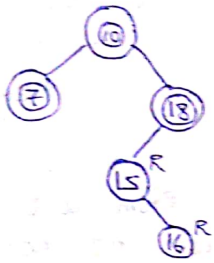
Properties:



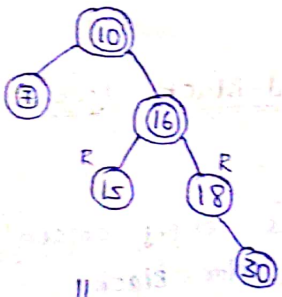
root = Black
no two adjacent red nodes
count no. of Black nodes in each Path



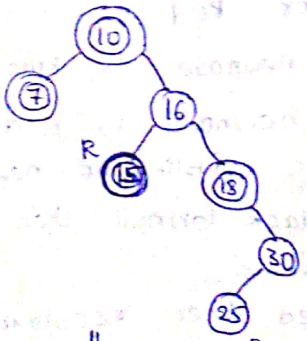
recolour



rotate & re-colour

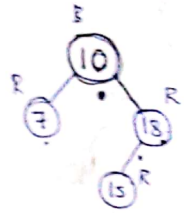


recolour, 16 also recolour

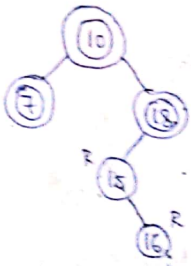


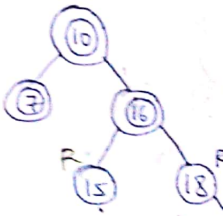
Parent's sibling Black (or) null
rotate & re-colour

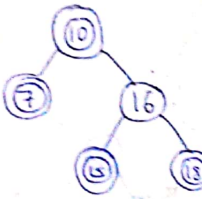
Insert the elements in
10, 18, 7, 15, 16, 30, 25, 4



R-R not
check

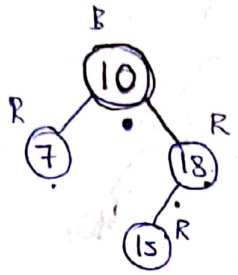




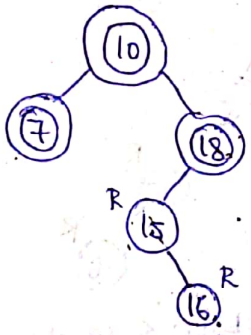


Insert the elements in Red-Black tree

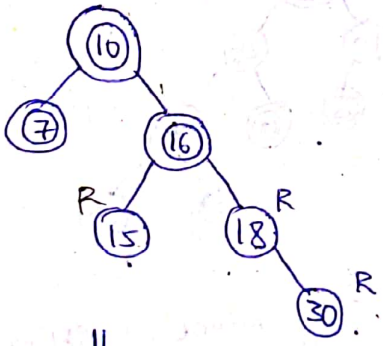
10, 18, 7, 15, 16, 30, 25, 40, 60, 2, 1, 70



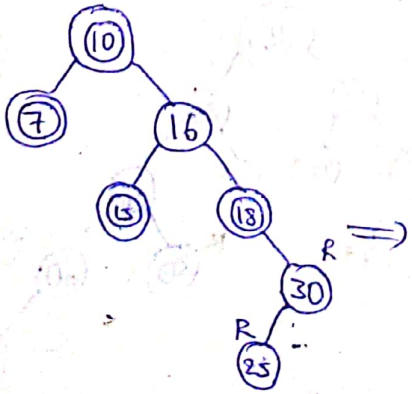
⇓ R-R not possible
Check Parent's sibling



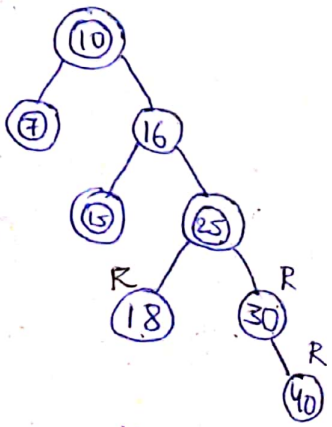
⇓



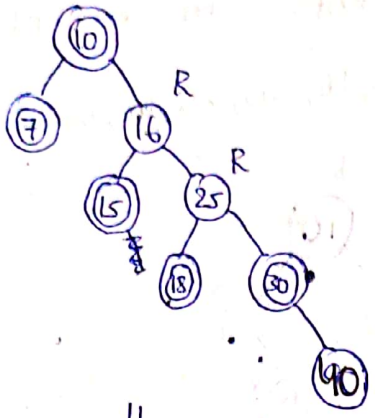
⇓



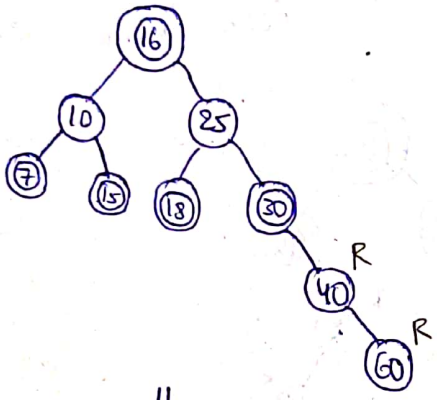
⟹



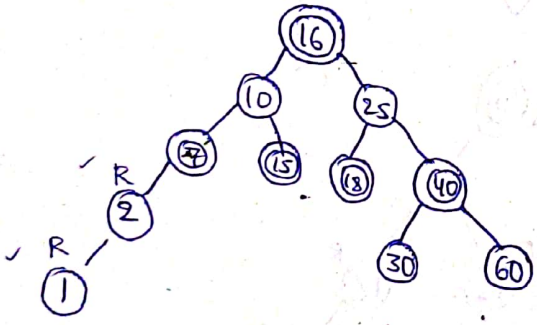
⇓ Recolour & re-check



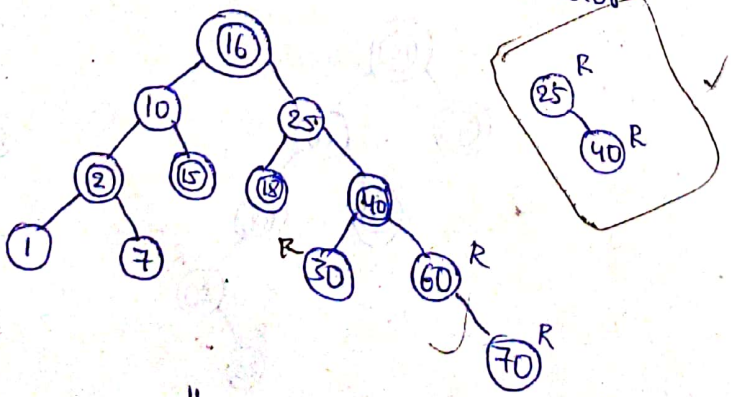
⇓ rotate and recolor



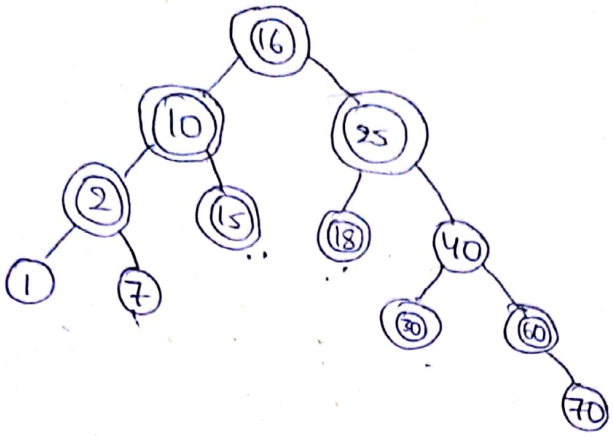
⇓ rotate & recolor



⇓ Parent's sibling is NULL
Rotate & re-color



⇓ Parent's sibling is R
then recolor & check



Note:

every Red-Black tree is a BST
But BST is need not to be Red-Black Tree.

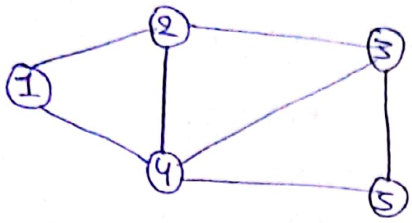
5.18

Deletion in Red-Black Tree

6. Graph Representation

6.1

Representation of a graph in computer:-



To represent graph in 2 methods

(i) Adjacency matrix

(ii) Adjacency list

(i) Adjacency matrix:- (dense graph)

every node is connected with each other.

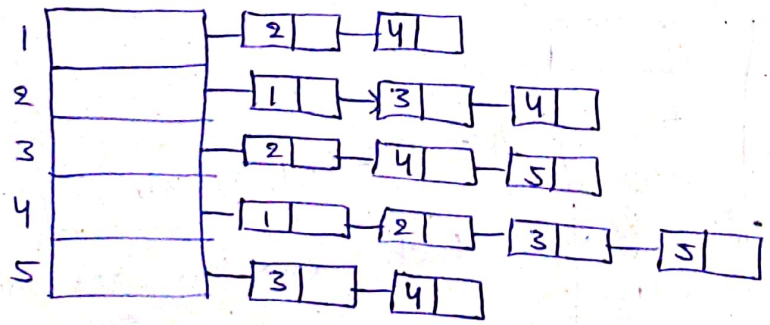
i↓ j→	1	2	3	4	5
1	0	1	0	1	0
2	1	0	1	1	0
3	0	1	0	1	1
4	1	1	1	0	1
5	0	0	1	1	0

5x5

Time complexity $\theta(n^2)$

{a[i][j] = 1 if i & j are adjacent
= 0 otherwise}

(ii) Adjacency list:- (sparse graph)



Time complexity $\theta(n+2e)$

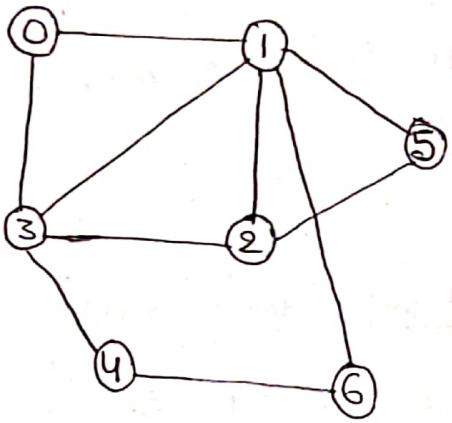
6.2

Graph Traversals

In Graph Traversal we have two methods:

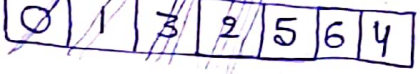
(i) BFS (Breadth first search)

(ii) DFS (Depth first search)



(i) BFS:

Data structure used in BFS is Queue

Queue:- 

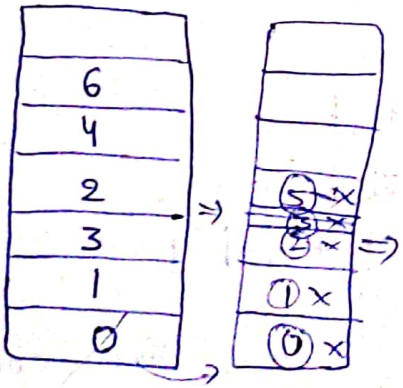
(FIFO)

Result = 0132564

(ii) DFS:

Data structure used in DFS is stack (LIFO)

Stack:-



Result = 0,1,3,2,4,6,5

Back track to get missed elements.

6.3

Types of edges in DFS

In Data structures we have 4 types of edges in DFS (Depth first search)

(i) Tree edge:-

member of DFS traversal

(ii) Forward edge:-

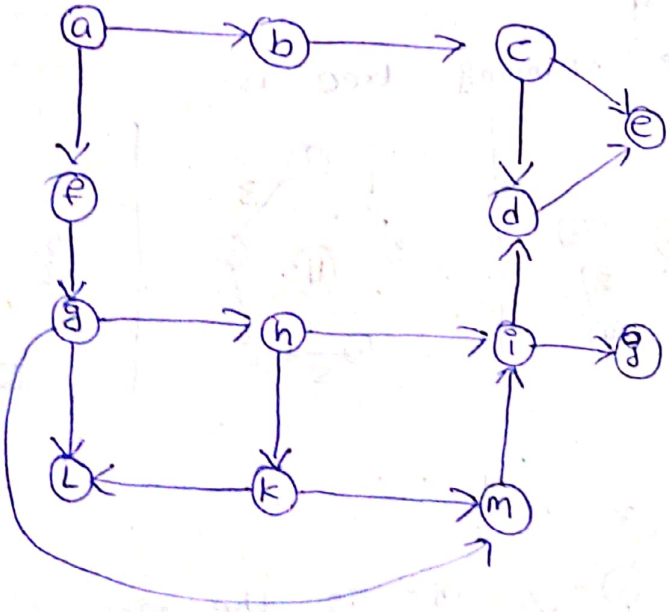
e: (x, y) where y appears after x and there is Path from x to y.

(iii) Back edge:-

e: (x, y) where y appears before x and there is Path from y to x.

(iv) Cross edge:-

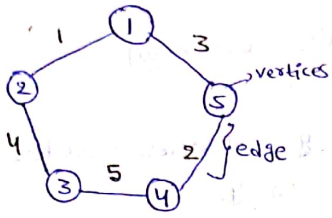
Any edge from x to y is a cross edge where there's no Path from y to x.



6.4

MST (minimum Spanning Tree)

Spanning Tree:-



$G_1 (V, \varepsilon)$ V = vertices

$G' (V', \varepsilon')$ ε = edges

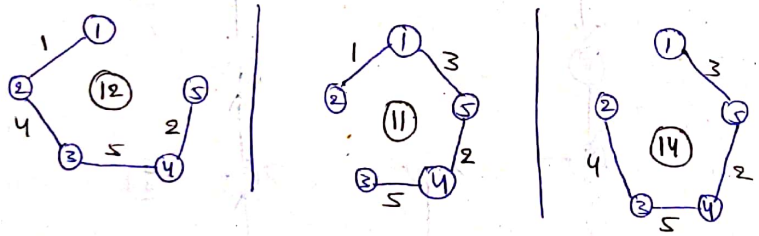
$V' = V$, $\varepsilon' \subset \varepsilon$, $\varepsilon' = |V| - 1$

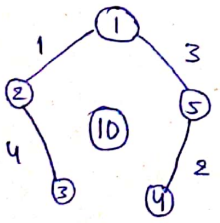
$\varepsilon' = |V| - 1$

$= 5 - 1$

$= 4$

resultant of spanning tree is:





✓ This is the minimum spanning tree (MST)

Note:

→ Spanning tree should not contain any cycle.

→ Spanning tree shouldn't be dis-continuous.

properties of MST:

→ Removing one edge from the will make it disconnected.

→ Adding one edge to the s

→ If each edge have distinct these will be only one & un

→ A complete undirected graph can h

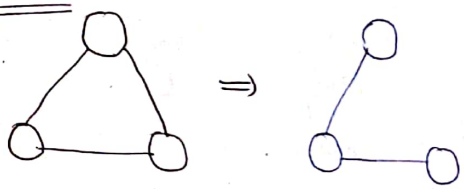
→ Every connected & undirected grap one ST

→ Disconnected graph doesn't have

→ From a complete graph by rem edges we can construct a

Q. From a complete graph by edges, we can construct a

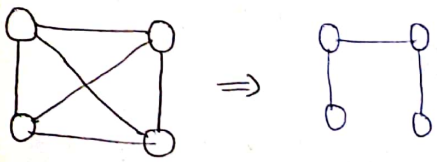
3 vertices



e - n + 1

3 - 3 + 1 = 1

4 vertices



e - n + 1

6 - 4 + 1 = (3)

max

properties of MST:

→ Removing one edge from the spanning tree will make it disconnected.

→ Adding one edge to the ST will create a loop.

→ If each edge have distinct weight then there will be only one & unique MST

→ A complete undirected graph can have n^{n-2} no.of ST

→ Every connected & undirected graph has atleast one ST

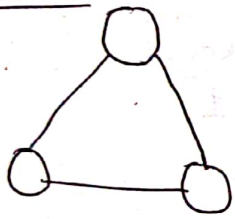
→ Disconnected graph doesn't have any ST

→ From a complete graph by removing max (e-n+1) edges we can construct a ST

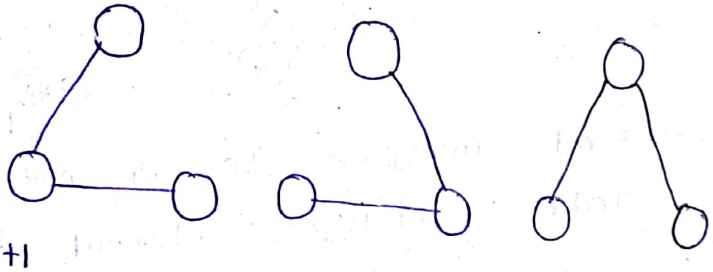
(e-n+1: e = edges, n = vertices)

Q. From a complete graph by removing max(e-n+1) edges, we can construct a spanning tree

3 vertices

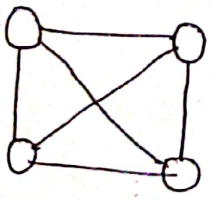


⇒

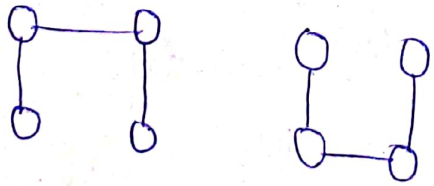


e-n+1
3-3+1 = 1

4 vertices



⇒



e-n+1
6-4+1 = (3)

max possibilities n^{n-2}

$= 4^{4-2}$

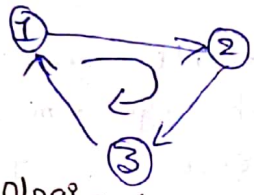
$= 16$

Topological sorting

→ It is a linear ordering of its vertices such that for every directed edge uv from vertex u to v, u comes before vertex v in the ordering. (u → v)

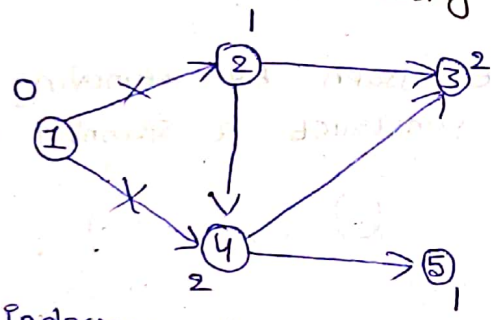
→ Graph should be Directed Acyclic Graph

→ Every DAG will have atleast one topological ordering.



Not possible to find Topological sorting

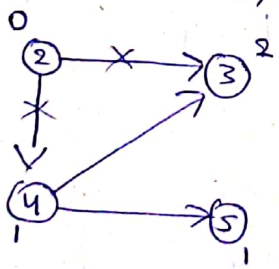
Q. Find Topological sorting for below Graph.



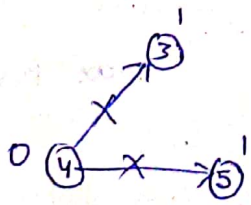
Result
1

→ Find indegree to all nodes.

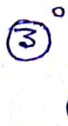
Take 0 in-degree element, delete it and print



Result
1, 2



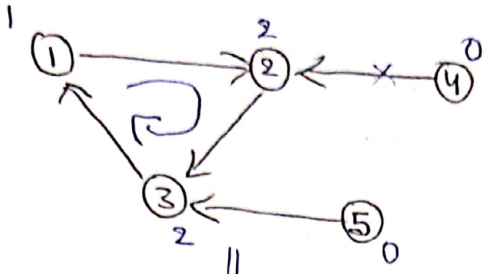
Result
1, 2, 4



Result
1, 2, 4, 3, 5
1, 2, 4, 5, 3

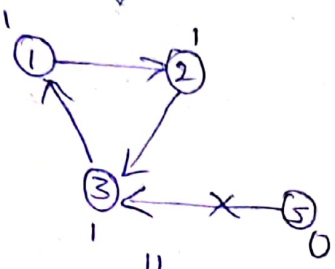
→ 2 TS is possible

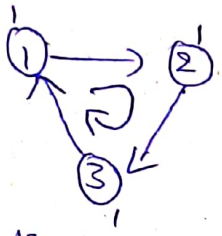
Q. Topological sorting for below graph



Case-(i) – 4

Case-(ii) – 5

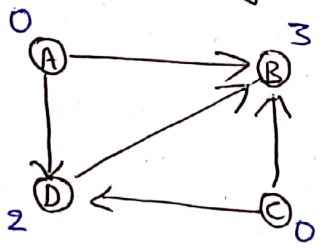




In this we didn't get any vertex having in-degree '0'.

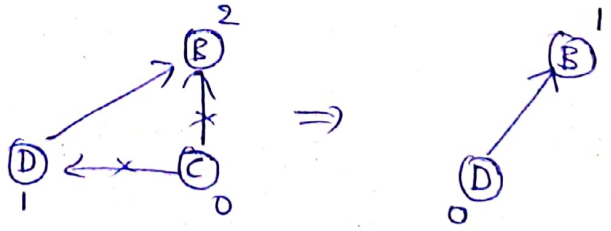
So, Topological sorting is not possible for this graph bcz it is a cyclic graph.

Q. Find Topological sorting for below graph.



Case-I

A, C, D, B



Case-II

CADB