:- Introduction :-

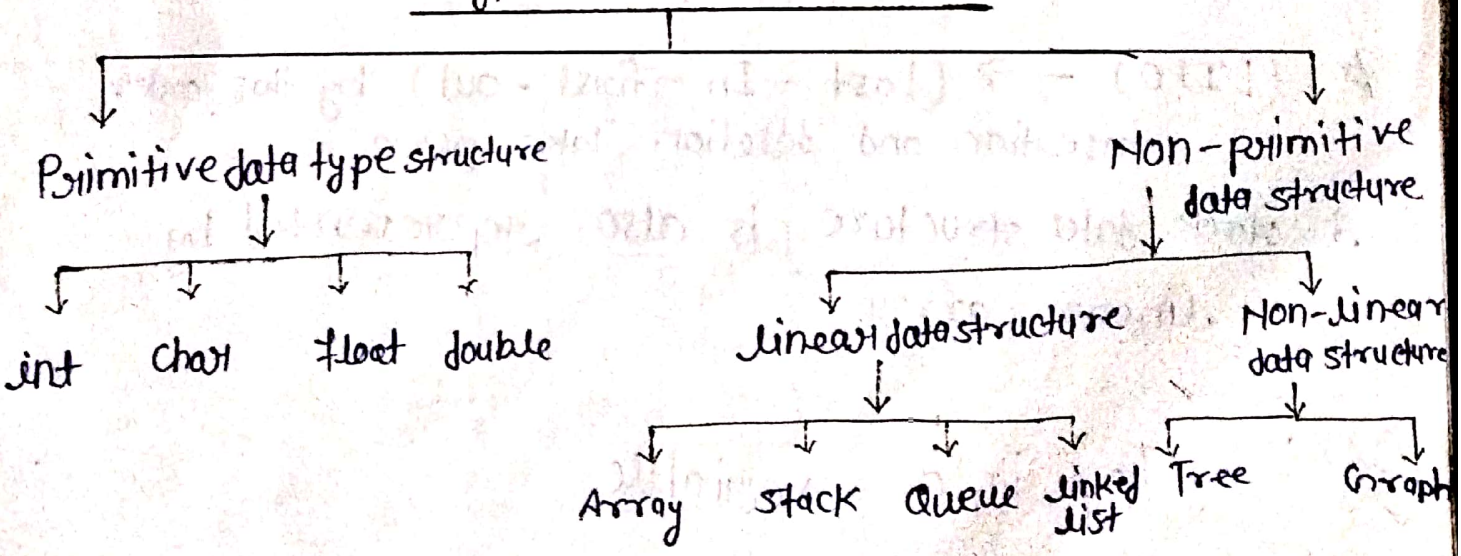
Data structure using - C

- Raw facts or message
- Way of Representation
  way of organization

Data structure defined as :-

1. Organization of data.
2. Representation of data.
3. Define the relationship between different data types
   1. logical relationship.
   2. Mathematical relationship.

:- Types of Data structure :-



Scanned with CamScanner

Array :- Array is the collection of similar and finite number of data items or data types.

Application :- Array represent static memory allocation.

Types of Array :-

① Single dimensional Array (Linear array).

② Multidimensional Array.

There are two types of multidimensional array.

① Two dimensional array.

② Three dimensional array.

Stack :- stack is a non primitive linear type of data structure in which insertion (Push) and deletion (POP) are both operation perform at the same end. and that end are called by Top of stack

★ (LIFO) ⟶ (Last - In - first - out). by this order insertion and deletion take place.

A stack data structure is also represented by linear array.

★ Top is a variable.

Scanned with CamScanner

Condition :-

① Initially when stack is empty then, set

top = null

or

top = -1

if (top == -1), then

write stack is initially empty.

② Before insertion first increase top by 1. by using assignment.

set top = top + 1

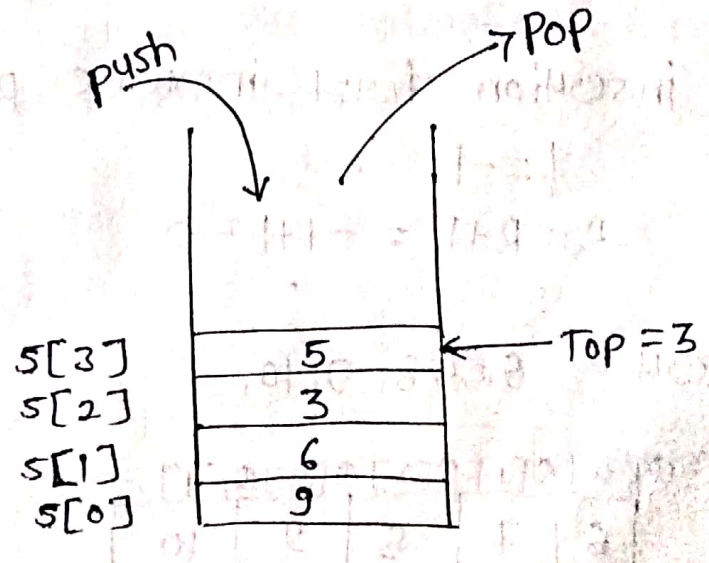
top = -1 + 1

top = 0

Ex:- 9, 6, 3, 5

S[top] = item

S[0] = 9



Scanned with CamScanner

Queue :- Queue is a non primitive linear type of data structure in which two variable invalve which is front and Rear.

★ (FIFO) ⟶ (First-In-first-out) by this order insertion and deletion take place. from front and rear.

Queue data structure is also represented by array.

Condition :-

① Initially queue is empty then

set front = Null

or front = -1

and set Rear = Null

or Rear = -1

(front = Rear = -1)

Insertion ⟶ Rear

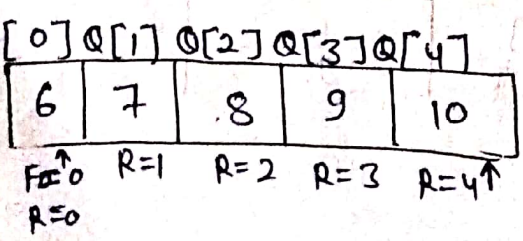
Deletion ⟶ front.

② Before insertion first increase Rear by 1.

f = -1

R = R+1 = -1+1 = 0

Ex :- insert 6, 7, 8, 9, 10,

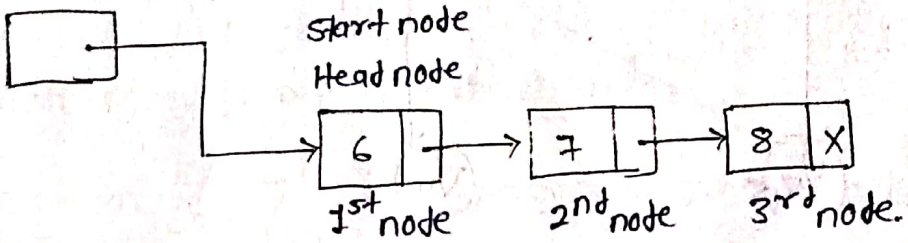


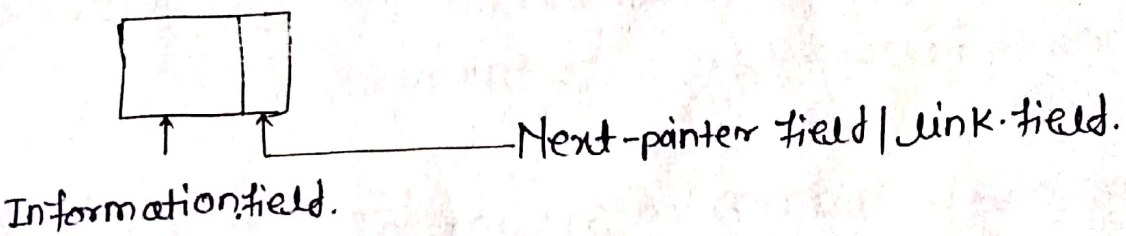
Scanned with CamScanner

Linked list :- it is represented by pointer.

int * ptr

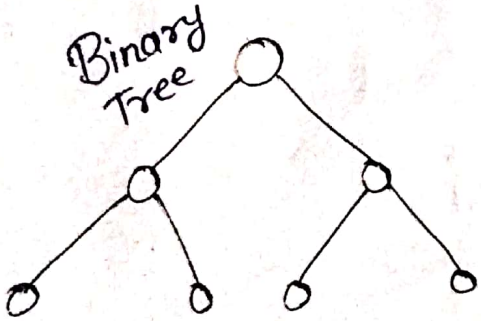
ex :- singly linked list.

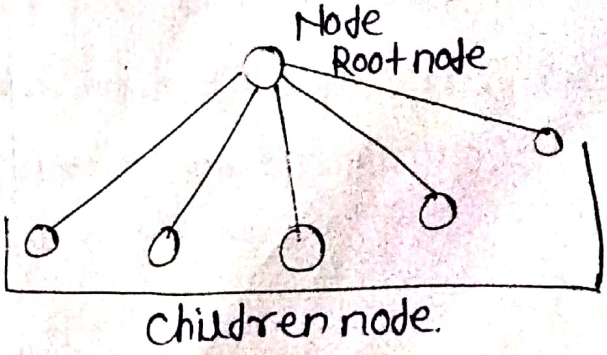




Non-linear data structure :-

① Tree :- it is the collection of nodes and edges is called tree.

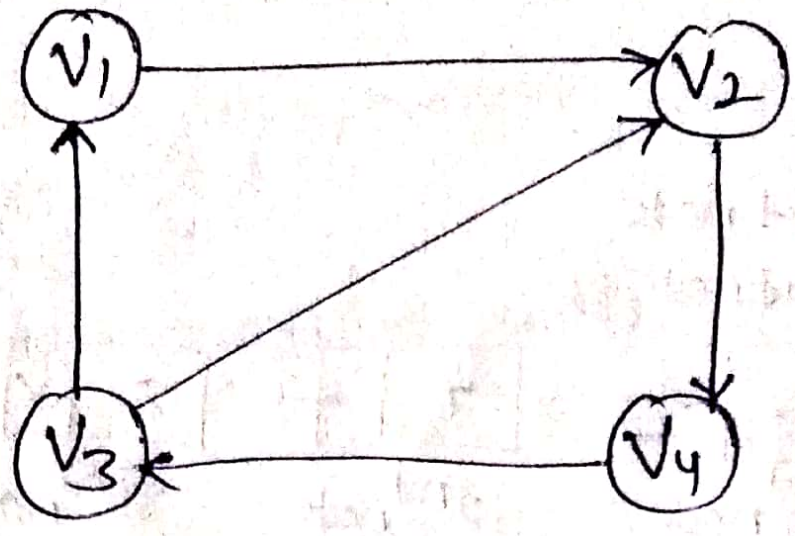




Scanned with CamScanner

Graph :- A collection of nodes and vertices is called graph.

$G = (V, E)$



Scanned with CamScanner

Data structure :- Data structure is a way of organizing all data items that consider their relationship to eachother.

Data structure operation :-

① Insertion :- To add an element in a given list.

② Deletion :- To delete an element from a given list.

③ Traversing :- To access and visit each and every element of a list exactly at one time

④ Merging :- To merge or combined two sorted list in a given element.

⑤ Searching :- To find the location of searched item.

⑥ Sorting :- To arranged the linked list item in a given order. (order may be increasing or decreasing).

Scanned with CamScanner

Algorithm :- The world algorithm originates from arabic world 'algarism' which is link to the name of arabic mathematician.

"An algorithm may be defined as a finite sequence of instructions each of which has clear meaning can be performed with a finite amount of effort in a finite amount of time.

Structure of algorithm :-

step-1 :- Input step.

step-2 :- Assignment step.

step-3 :- Decision making step.

step-4 :- Repetitive step:

step-5 :- output step.

To find the sum of two no. write structure of algorithm.

Begin

① To select two number A and B.

② To assign value of given number

A = 10

B = 20

③ To find sum of two number

sum = A+B

④ To display result sum || sum = 30

Scanned with CamScanner

Properties of an algorithm :-

① Finiteness :- An algorithm must terminate after a finite number of step.

② Definiteness :- The step of an algorithm must be precisely defined.

③ Effectiveness :- The step of an algorithm should be effective to solve a given problem, they should not be too complex.

④ Correctness :- Each and every step of an algorithm should be correct.

⑤ Meaning should be clear and easy to understand

⑥ Input|output :- An algorithm should have certain initial and precise input, after applying algorithm there is a final output.

Scanned with CamScanner

Analysis of an Algorithm

Analysis of algorithm is measured by the complexcity of an algorithm.

complexcity :- Number of key operation for any input size.

The complexcity of an algorithm is a function which gives the running time and space in terms of the input size.

Input size n, f(n).

Two types of complexcity :-

① Running time complexcity.

② space complexcity.

The analysis of an algorithm is the major task in the field of computer science. we must have some criteria to measure the efficiency of our algorithm.

Q :- find out running time complexcity of linear searching algorithm.

Scanned with CamScanner

Asymptotic Notation

1. Θ (Theta) notation
2. O (Big-oh) notation
3. Ω (omega) notation

Asymptotic Notation are used to measure the performance of an algorithm.

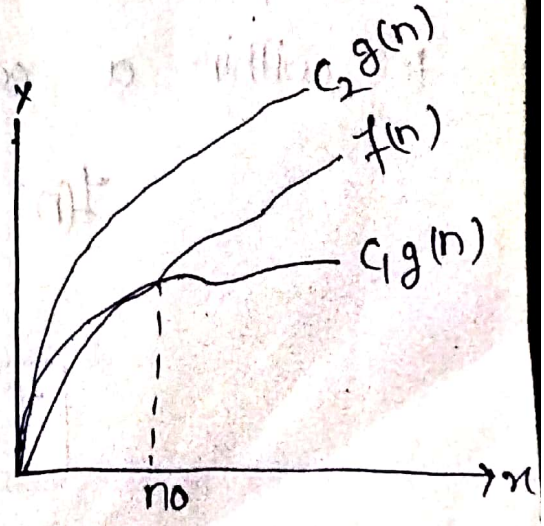
These notation is used to describe the asymptotic running time of an algorithm are defined in terms of function.

1. **Θ (Theta) Notation** :- These notation bounds a function to within constant some factors

$$c_1 |(g(n))| \leq \underbrace{f(n)}_{\text{Bounds}} \leq c_2 |g(n)|$$

for all $n > no$

Then $\boxed{f(n) = \Theta(g(n))}$



graph of Θ(Theta) notation.

Scanned with CamScanner

Suppose :-

$c_1 g(n) = 9n$

$c_2 g(n) = 9n + 16$

$f(n) = 9n + 1$

ex :- $f(n) = 18n + 9$

↓ constant: ↑ constant value.

$f(n) = 18n + 9$

c_2 $g(n)$

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

$\boxed{f(n) = O(n)}$

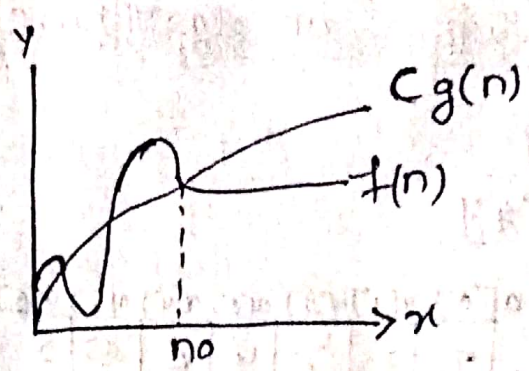
(2) O (Big-oh) notation :- This notation represent an upper bound for a function to within a constant factor.

$f(n) \leq c\, g(n)$

for all $n > n_0$

$$\boxed{f(n) = O(g(n))}$$

Scanned with CamScanner



graph of Big-notation

ex:- $f(n) = 16n^2 + 6n + 5$

$\downarrow$

n^2

$cg(n)$

$$\boxed{f(n) = O(n^2)}$$

③ Ω (omega) notation :- (lower bound) This notation represent a lower bound for a function to within a constant factor.

$$cg(n) \leq f(n)$$

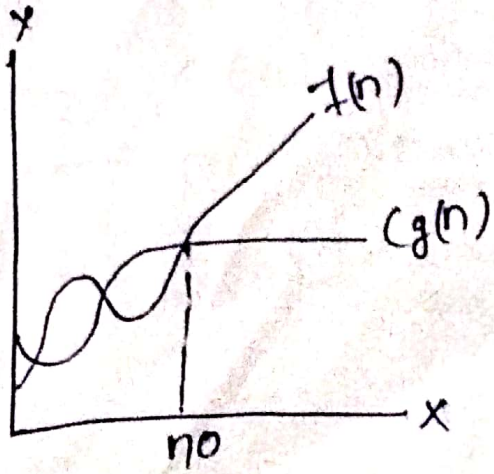
$$f(n) = \Omega(g(n))$$

ex:- $f(n) = 18n + 9$

$\uparrow$

$cg(n)$

$$\boxed{f(n) = \Omega(n)}$$

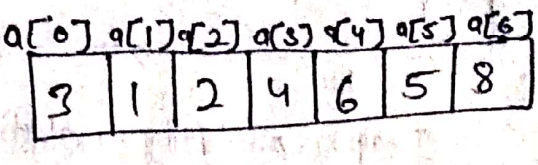


graph of Ω (omega) notation.

Scanned with CamScanner

Analysis of an algorithm :-

ex :- int a[7]



search item = 8

Linear search :-

Algorithm :-

Begin ① for i = 0 to n-1

② if (a[i] == item), then

③ write search is successful and exist.
location = i

④ if (i == n), then

⑤ write item is not found in the list and search is unsuccessful.

end.

Scanned with CamScanner

Q:- Find the running time complexcity of linear search.

$1+2+3+4+\_\_\_\_+n$

$T_n = a+(n-1)d$

$= 1+(n-1)1$

$= 1+n-1$

$T_n = n$

$\boxed{T_n = O(n)}$ complexcity.

The complexcity of an algorithm (A) is the function $f(n)$ which represent the running time and storage space requirement of an algorithm in terms of the size (n) of input data.

" Running time complexcity the no. of key operation is performed (in searching and sorting algorithm)" in any input size"

1. worst case complexcity $\longrightarrow O(n)$ - Maximum value
2. Average case complexcity $\longrightarrow O(n/2)$ - Expected value
3. Best case complexcity. $\longrightarrow O(n)$ - Minimum value.

Scanned with CamScanner

1) worst case :- The maximum value of $f(n)$ for any possible input.

2) Average case :- The expected value of $f(n)$

3) Best case :- Minimum possible value of $f(n)$ is called best case

Scanned with CamScanner

Problem solving concept :-

Basically there are two types for salving a problem.

① Top down approach

② Bottom up approach.

① Top down approach :- The top down basically devides a complex problem into multiple smaller parts (modules).

These modules are further partison (decomposed) until the resulting module is cannot be further decomposed.

" The top down approach is a process of breaking the large program into smaller program (sub-program) in an efficient way."

The flow of control is from top to bottom. (down ward direction).

Top down approach is basically used in structured programming language.

Obal, portrand, and C etc.

Scanned with CamScanner

2) Bottom up design approach :- In bottom up design approach the integration of some modules into a single module.
(sub program into a program).
at the higher level this is repeat.

bottom up approach is testing as each fundamental module is first tested before merging it to the bigger one.

" In bottom up design approach we solve smaller problem and integrate it as whole and complete the solution."

direction is from bottom to up.

this approach is mainly used in ~~mainly~~ objected oriented programming language.

like C++, Java, C-sharp, C#, python

Scanned with CamScanner

Date 6-4-8-22

1) WAP in to linear search

```
#include<stdio.h>
#include<conio.h>
int main()
{
int i, a[10], n, item;
printf("Enter the element of array\n");
scanf("%d", &n);
printf("Enter the element\n");
for(i=0; i<n; i++)
{
scanf("%d", &a[i]);
}
printf("Enter item to search in list\n");
scanf("%d", &item);
for(i=0; i<=10; i++)
{
if(a[i]==item)
{
printf("item found at location %d\n", i+1);
break;
}
}
if(i==n)
{printf("item does not found\n");
}return 0;
}
```

Scanned with CamScanner

Array :-

An array is the collection of finite and homogeneous numbers of data items. each data items are called elements of an array.

There are two types of Array :-

(1) Linear Array (single dimensional Array)
one-way list | Array

(2) Multidimensional Array

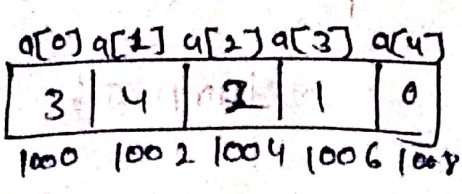
(I) Two dimensional Array — ex:- matrix

(II) Three dimensional Array — ex:- a[2][3][4]

(1) Linear Array :- it is a list of finite number, collection of homogeneous data elements. (similar types of data elements).

Linear array is also named by sequential Array in which index number in a sequential order (consecutive) ~~onfugative~~ order. the elements of an array are stored respectively in successive memory location.

it also locate contiguous memory address.



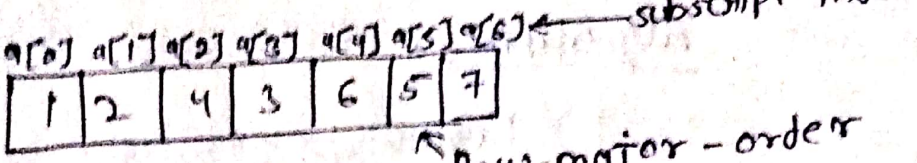
Scanned with CamScanner

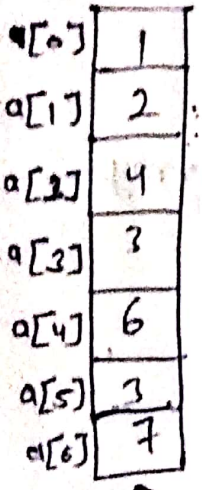
Size of or length of a linear Array =

upper bound − lower bound + 1

= UB − LB + 1

ex:-





Note:-

Application of linear array is used to represent computer memory where elements are stored in successive memory location. (memory cells)

Declaration of linear Array :-

Syntax :- data-type array-name[size];

ex:-

```
int marks[10];
float salary[10];
int a[20];
```

Scanned with CamScanner

Traversing operation :-

Traversing a linear array :-

Traverse Linear array (a, n)

Algorithm :-

Begin

① for j = 0 to n-1

② process a[j] and
display the given array.

exit.

end

Processing of linear array :-

① Input an array

```
for(j=0 ; j<n; j++)
  {
   scanf("%d", &a[j]);
  }
```

② To display the input array

```
for (j=0 ; j<n; j++)
  { printf ("%d", a[j]);
  }
```

Scanned with CamScanner

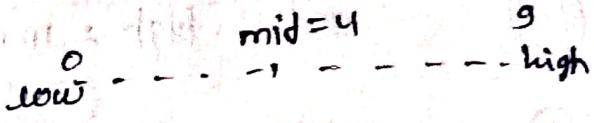
8/08/22

Binary Searching :-

① In case of binary searching given list/array must be in sorted order.

$$mid = (low + high) / 2$$

if (a[mid] == item)



low = 0

high = n-1 = 10-1

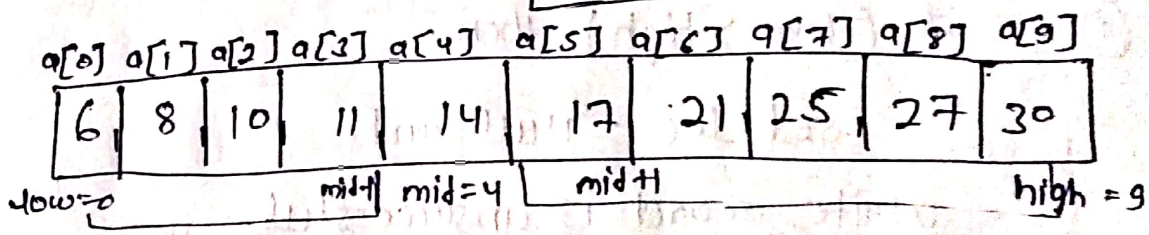
mid = int (low+high) / 2

= int (0+9)/2

= int (4.5)

mid = 4

ex :-



Algorithm :-

Binary search (a, n, item, low, mid, high)

Begin

① set low = 0

② set high = n-1 || 10-1 = 9

② calculate mid = (low+high)/2

mid = 0+9 / 2

mid = 4

a[mid] = a[4] = 4

Scanned with CamScanner

Search item (Assume).

item = 10, 17

③ while ((low ≤ high) and (a(mid) ≠ item) do

④ if (item < a[mid]), then

set high = mid - 1

⑤ set high = mid - 1

⑥ else

set low = mid + 1

end if

⑦ Again calculate mid

set mid = (low + high)/2

end while

⑧ if (low > high), then

set loc = Null and

write search is unsuccessful.

⑨ else

set loc = mid and

write search is successful.

end if

end

Scanned with CamScanner

Problem :-

Step 1 :-

low = 0
high = 9
find mid = (0+9)/2
mid = 4

Step-2

while (low ≤ high) and (a[mid] ≠ item)
if (item < a[mid])
set high = mid-1
= 4-1
= 3

Step-3

Again find
mid = (low+high)/2
mid = (0+3)/2
mid = 1

Step-4

Again compare if (item > a[mid])
if (10>8)
set low = mid+1
= 1+1
= 2

Scanned with CamScanner

Step-5 Again calculate

mid = (low+high)/2

=(2+3)/2

mid = 2

a[mid] = 10 = item, search is successful and loc = mid

loc = 2

Step-1

low = 0

high = 9

mid = (0+9)/2

= 4

mid = 4

Step-2 while (low ≤ high) and (a[mid] ≠ item)

if (item > mid)

17 > 14

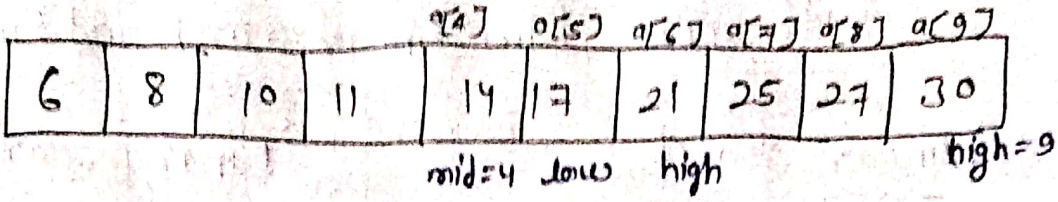
else

set low = mid+1

low = 4+1

low = 5

Scanned with CamScanner



step-3 Again find mid

$$mid = (low + high)/2$$

$$= (5+9)/2$$

$$= 14/2$$

$$\boxed{mid = 7}$$

step-4 :- Again compare

if (item < mid)

if (17 < 25)

set high = mid-1

$$= 7-1$$

$$= 6$$

step-5 :- Again calculate mid

$$mid = 5+6$$

$$= 11/2$$

$$= 5$$

a[mid] = 17 = item search is successful.

Scanned with CamScanner

Difference b/w linear search and Binary search

Linear search	Binary search
① complexcity of linear search is $O(n)$ in worst case.	① complexcity of binary search is $O(\log_2^n)$ in worst case.
② In case of linear search list may be in sorted or not.	② In case of binary search list always in a sorted order.
③ linear search is a based on sequential searching technique	③ Binary search is based on divide and conquor technique.
④ linear search take more amount of time for searching an item.	④ Binary search is more efficient and it takes less amount of time for searching an item.
⑤ In best case linear search time complexcity $O(1)$	⑤ similar for binary search.

Scanned with CamScanner

Stack :- A stack is a list of element in which an element may be inserted or deleted only one end called the top of the stack.

stack are also called last-in, first-out (LIFO) type of data structure.

Stack's operation :- There are two operation.

① PUSH (Insert an element)

② POP (Delete an element)

Stack Implementation :-

A stack data structure can be implemented by an array. (linear array).

Representation of a stack :-

when initially stack is empty

Top = -1

Scanned with CamScanner

-: Algorithm of stack :-

Begin PUSH (S, max.size, item, Top).

① if (Top == max.size - 1), then

② write stack is full and show overflow condition

exit.

③ set Top = Top+1 || Increase Top by 1.

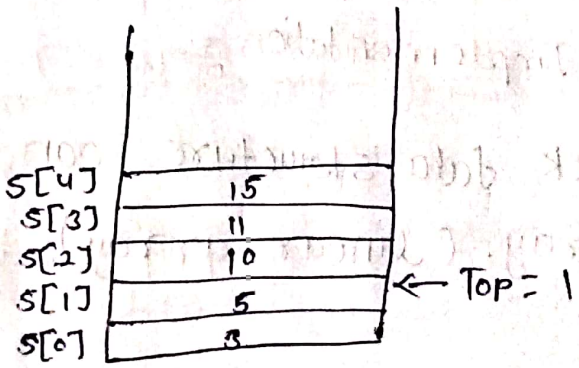
④ set S[Top] = item. || insert given item at top

end.

ex:-

let us assume.

item = 10, 11, 15



before insertion first increase top by 1, by using assignment

set Top = Top+1

Top = 1+1

Top = 2

S[2] = 10

Scanned with CamScanner

Algorithm of stack :-

POP(S, TOP, item, max.size).

Begin

① if (TOP = = -1), then

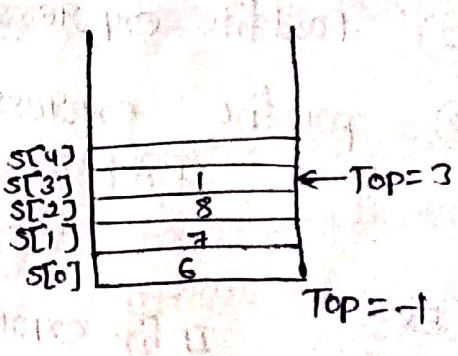
② write stack is initially empty and show underflow condition exit.

③ set item = S[TOP] || select item to be deleted from top of stack

set top = top - 1 || After deletion decrease top by 1

end.

ex:-



before deletion first decrease top by 1

set Top = top-1

Top = 3-1

Top = 2

Scanned with CamScanner

:- Application of stack :-

1. function calling
2. Evaluation of expression
3. Parsing
4. Reversing a list.
5. Implementation of recursion.

Evaluation of expression :-

1. Infix expression (Infix notation) ⟶ A⊕B
2. Postfix expression (postfix notation) or Reverse polish notation. AB⊕
3. prefix expression (prefix notation) or polish notation ⊕AB

Ex :- Infix expression ⟶ postfix expression

A+B*C

A+BC*

= ABC*+ postfix expression.

Read symbol	stack	postfix expr.
	(	
A	(	A
+	(+	A
B	(+	AB
*	(+*	ABC
(	(+*	ABC*+
)		

Scanned with CamScanner

Polish notation or prefix notation :- For most common Arithemetic operations. The operator symbal is placed between two operant. this is called Infix - notation.

ex:- A⊕B
A+B , A-B , A*B , A|B,

The polish notation named after the name of mathematician, refers to the notation in which the operator symbal is placed before its two operants.

ex:- +AB , −AB , *AB , /AB

The polish notation is also called by prefix notation.

The fundamental property of polish notation is that the order in which the operation are to be performed is completely determined by the position of operator and operants in the expression.

There is no need of paranthesis ( ) when writing expression in polish notation.

Scanned with CamScanner

Evaluation of postfix expression :-

ex:- $I = 5*(6+2) - 12/4$

sol by direct method

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

$= 5,6,2,+,* - 12/4$

$= 5,6,2,+,* - 12,4,/$

$= 5,6,2,+,*,12,4,/,-$

postfix form

Symbol read	Stack	Result
5	5	
6	5,6	
2	5,6,2	
+	5,8	
*	40	
12	40,12	
4	40,12,4	
/	40,3	
-	37	37

Scanned with CamScanner

② consider the following postfix expr.

12, 7, 3, -, 1, 2, 1, 5, +, *, +

Symbol read	stack	result.
12	12	
7	12, 7	
3	12, 7, 3	
-	12, 4	
1	3	
2	3, 2	
1	3, 2, 1	
5	3, 2, 1, 5	
+	3, 2, 6	
*	3, 12	
+	15	15

Scanned with CamScanner

Transforming Infix exp. into postfix exp.

To transform the infix expression into its equivalent postfix expression we use the following algorithm, by using a stack.

Algorithm

Polish (I, P)

Suppose (I) is an arithemetic expression written in infix notation. this algorithm finds the equivalent postfix expression (P).

Begin

Step-1 :- push "(" onto stack and add ~~to~~ ")" to end of expression I.

Step-2 :- Scan expression I from left to right and repeat step 3 to 6 for each element of expression I until the stack is empty.

Step-3 :- If an operand is encountered then add it to expression P.

Step-4 :- If a left paranthesis "(" is encountered push it onto stack.

Step-5 :- If an operator ⊕ is encountered then
(a) repeat POP from stack and add it to exp. P. each ⊕

Scanned with CamScanner

which has the same presidency as or higher presidency than operator.

(b) Add operator to stack.

Step-6:- If a right ")" is encountered then.

(a) Repeat Pop from stack and add to expression P. (each operator) until left paranthesis is "(" encountered.

(b) Remove the left "(" from stack (do not add the left paranthesis to expression P.)

Step-7:- exist.

end

(1) consider the following Infix expression I

I = (7-5) * (9/2)) translate or transform into postfix expression P.

Sol

Scan Symbol	stack	Postfix expression P
	(	
(	((	
7	((	7
-	((-	7
5	((-	7,5
)	(	7,5,-
*	(*	7,5,-
(	(*(	7,5,-
9	(*(	7,5,-,9
/	(*(/	7,5,-,9
2	(*(/	7,5,-,9,2
)	(*	7,5,-,9,2,/
)		7,5,-,9,2,/,*

Scanned with CamScanner

(2) Transform expression I into postfix expression.

$I = A+(B*C-(D/E\wedge F)*G)*H$

Solu:- $I = A+(B*C-(D/E\wedge F)*G)*H)$

Symbol Read	stack	Postfix expression P.
	(	
A	(	A
+	(+	A
(	(+(	A
B	(+(	AB
*	(+(*	AB
C	(+(*	ABC
-	(+(-	ABC*
(	(+(-(	ABC*
D	(+(-(	ABC*D
/	(+(-(/	ABC*D
E	(+(-(/	ABC*DE
$\wedge$	(+(-(/$\wedge$	ABC*DE
F	(+(-(/$\wedge$	ABC*DEF
)	(+(-	ABC*DEF$\wedge$/
*	(+(-*	ABC*DEF$\wedge$/
G	(+(-*	ABC*DEF$\wedge$/G
)	(+	ABC*DEF$\wedge$/G*-
*	(+*	ABC*DEF$\wedge$/G*-
H	(+*	ABC*DEF$\wedge$/G*-H**
)		

Scanned with CamScanner

③ $I = ((A+B)*D) \wedge (E-F)$

Sol $I = ((A+B)*D) \wedge (E-F))$

Symbol scan	stack	Postfix Expression P
	(	
(	((	
(	(((	
A	(((	A
+	(((+	A
B	(((+	AB
)	((	AB+
*	((*	AB+
D	((*	AB+D
)	(	AB+D*
^	(^	AB+D*
(	(^(	AB+D*
E	(^(	AB+D*E
-	(^(-	AB+D*E
F	(^(-	AB+D*EF
)	(^	AB+D*EF-
)		AB+D*EF-^

Note:- Presidency according to → $(), \$, \wedge, /, *, +, -$

Scanned with CamScanner

Recursion :-

Recursion is defined as defining anything in terms of itself.

Recursion is used to salve problems involving iterations (A single line of code executed multiple times).

Recursion is an alternative to iteration in making a function execute again and again.

"A recursion is a function call by itself during execution of a program".

Syntax :-

```
datatype func. name (data-type var_name)
```

ex:-

```
int factorial (int n)
{
  if (n==0 || n==1)
    return 1;
  else
    return (n*factorial(n-1));
}
```

Scanned with CamScanner

(1) Write down an algorithm to find factorial of any number. and to create a program in C to find the factorial of any number.

Sol

Algorithm :-

factorial(factorial, n)

Begin

step (1) Declare valid variable.

step (2) Take input ~~from~~ user.

step-(3) factorial (n)

step-(4) {
if (n==0||n==1)
then it will
return 1;

step (5) exit.

step (6) else
return * fact(n-1);

End.

Scanned with CamScanner

① WAP in C to find factorial of any number.

```
#include<stdio.h>
int fact(int n);
void main()
{
  int n, factorial;
printf("find the factorial of any number\n");
printf("enter the number:\n");
scanf("%d", &n);
    f = fact(n);
printf("factorial = %d", factorial);
      getch();
}
   int fact(int n)
   {
     if(n==0)
     return 1;
     else
       return(n * fact(n-1));
   }
```

Scanned with CamScanner

Ackermann function

The Ackermann function is a function with two arguments each of which can be assigned any non-negative (positive) integer 0, 1, 2 - - - - this function is defined as follows.

Definition :-

$A(m,n)$

case-1 :- if $m=0$, and $n \neq 0$, then $A(m,n) = n+1$

case-2 :- if $m \neq 0$ and $n=0$, then $A(m,n) = A(m-1,1)$

case-3 :- if $m \neq 0$ and $n \neq 0$, then $A(m,n) = A(m-1, A(m,n$

ex:- find the value of $A(0,1)$, $A(1,0)$, $A(1,1)$ by using Ackermaun function

sol ① if $m=0$, $n \neq 0$ then Apply case 1

$$A(m,n) = n+1$$

$$A(0,1) = 1+1$$

$$A(0,1) = 2$$

Similarly

$$A(1,0) = 1$$

② if $m \neq 0$, $n=0$ then apply case 2

$$A(m,n) = A(m-1,1)$$

$$A(1,0) = A(0,1)$$

$$A(0,1) = 1+1$$

$A(0,1) = 2$ then $A(1,0) = 2$

Scanned with CamScanner

(3) $A(1,1)$, if $m \neq 0, n \neq 0$ then apply case 3

$$A(m,n) = A(m-1, A(m,n-1))$$

$$A(1,1) = A(0, A(1,0))$$

$$A(1,0) = A(1-1,1)$$

$$A(1,0) = A(0,1)$$

$$A(0,1) = 2$$

$$A(1,0) = A(0,1) = 2$$

put value of $A(1,0) = 2$

$$A(1,1) = A(0,2)$$

$$A(0,2) = 2+1$$

$$A(0,2) = 3$$

$$A(1,1) = A(0,2) = 2$$

finally

$$A(1,1) = 3$$

(2) find the value of $A(1,3)$

Sol

if $m \neq 0$, $n \neq 0$

$$A(m,n) = A(m-1, A(m,n-1))$$

$$A(1,3) = A(0, A(1,2))$$

$$= A(0, A(0, A(1,1)))$$

$$= A(0, A(0, A(0, A(1,0))))$$

$$= A(0, A(0, A(0, A(0,1))))$$

$$= A(0, A(0, A(0,2))$$

$$= A(0, A(0,3)$$

$$= A(0,4) \Rightarrow A(1,3) = 5$$ Ans

Scanned with CamScanner

Ackermann function Algorithm

```
int Ack (int m, int n)
{
```

Step-1 `if (m==0)`

Step-2 `return (n+1);`

Step-3 `else if ((m>0) && (n==0))`

Step-4 `return (Ack (m-1, 1));`

Step-5 `else if ((m>0) && (n>0))`

Step-6 `return (Ack (m-1, A(m, n-1));`

① WAP in C to find the value of Ackermann function.

```
#include<stdio.h>
#include<conio.h>
 int Ack(int m,int n);
  void main ()
 {
  int m,n;
printf(" Find the value of Ackermann function\n");
printf (" enter the value of m: and n: \n");
   scanf(" %d %d ", &m, &n);
printf(" Ackermann value = %d\n", Ack (m,n));
   getch ();
}
int Ack (int m, int n)
{ if (m==0)
   return (n+1);
```

Scanned with CamScanner

```
else if ((m>0)&&(n==0))
    return (Ack(m-1, 1));

else if ((m>0)&&(n>0))
    return (Ack(m-1, A(m,n-1)));
}
```

Output :-

```
Enter the value of m: and n:
Ack(0,1)

Ackermann value = 2
```

Scanned with CamScanner

Tower of Hanai

The tower of Hanai is famous recursive problem which is based on 3 pegs and set of discs of graduated size.

In tower of hanai problem move all given discs from begining and to end by using Auxillaryend

Rules to solve this problem is as follows:-

Rule:- only one disc may be moved at a time.

Rule:- At no time can a larger discs be placed on a smaller disc.

The solution to tower of hanai problem for a number of disc n appears (n=3)

observe that it consist of the following no. of steps.

to find movement by using formula $= 2^n - 1$

ex:- $(n = 3)$

No. of step or movement $= 2^n - 1$

$= 2^3 - 1$

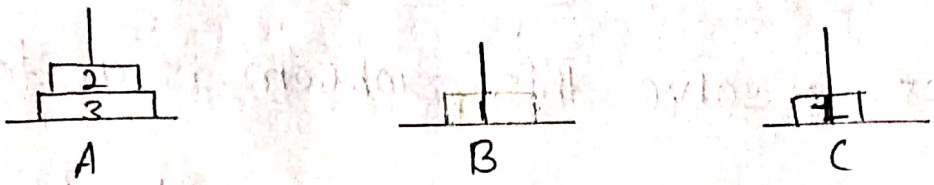
$= 7$

Scanned with CamScanner

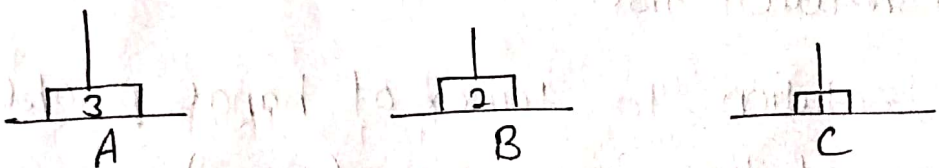
ex:-



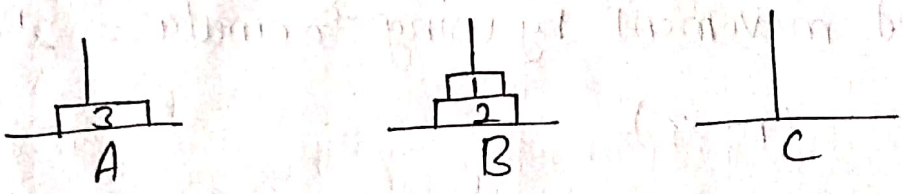
Step-1 move disc 1 from A to B i.e A → B



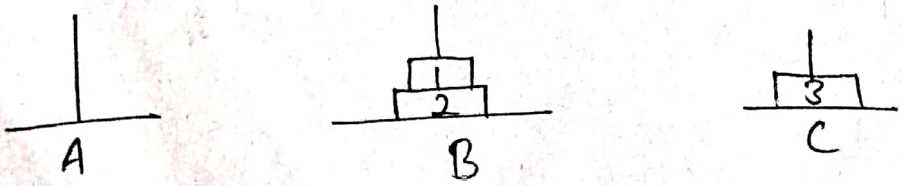
Step-2 move disc 2 from A to B i.e. A → B



Step-3 move disc 1 from C to B i.e C → B

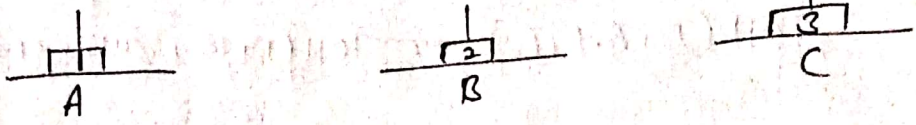


step-4 move disc 3 from A to C i.e A → C

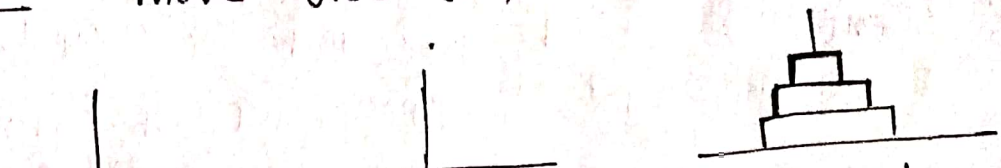


Scanned with CamScanner

Step-5 : move disc 1 from B to A i.e B → A



Step-6 move disc 1 from A to C



movement so-

A → C
A → B
C → B
A → C
B → A
B → C
A → C Solution.

Tower of Hanoi :-

Algorithm

TOH (n, A, B, C) or TOH (n, Beg, aux, end)

Begin

Step-1 :- if n=1, then move disc from Beg to end
A to C and exit.

Step-2 :- call TOH (n-1, Beg, end, aux) then move disc from Beg ⟶ end.

Step-3 :- Again call TOH (n-1, aux, Beg, end

end.

① WAP of TOH.

```
#include<stdio.h>
    void TOH(int n, char A, char B, char C);
    int main()
    { int n;
printf("enter the no.of disc\n");
  scanf("%d", &n)
    TOH(n, 'A', 'B', 'C');
     return 0;
    }
    void TOH (int n, char A, char B, char C)
```

Scanned with CamScanner

```
{
    if (n>0)
    {
      TOH (n-1, A, C, B)
    printf("\n %c -> %c ", A, B);
       TOH (n-1, C, B, A)
    }
}
```

Output :-

n=3

A → C
A → B
C → B
A → C
B → A
B → C
A → C

Scanned with CamScanner

Queues :- A queue is a linear data structure in which all insertion operations are performed at one end that end is know as Rear of the queue, and all deletions can take place only at one end is known as front of the ~~new~~ queue.

Queue is a fifo type of data structure.
(First in first out).

since the first element in a queue will be the first element out from the queue.

An important example of a queue or application of a queue in computer science occurs in a time sharing system, in which program with the same priority from a queue while waiting to be executed.

front ⟶ Deletion

Rear ⟶ Insertion

if (front == Rear ≠ Null)

Q[0]	Q[1]	Q[2]	Q[3]	Q[4]
3	4	2	1	5

Scanned with CamScanner

Types of Queue -

1) Linear Queue
2) Circular Queue
3) Double ended Queue (Deque)
4) Priority Queue

Linear Queue

Algo
Insert Linear Queue (Q, maxsize, front, rear, item)
Begin
① if (Rear == maxsize-1), then
write Queue is full and show overflow condition
exit
② Set Rear = Rear+1 // Increase rear by 1
③ Set Q[Rear] = item // Insert item at rear
end

Algo
Deletion Linear Queue (Q, item, maxsize, front, rear)
Begin
① if (Front == -1), then
initially Queue is empty and show underflow condition
exit

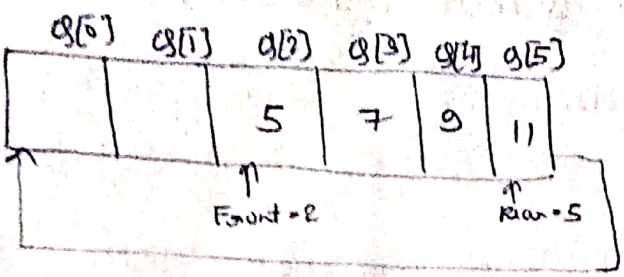
Scanned with CamScanner

② item = Q[front] // Delete item from front

③ set front = front + 1

end

Circular Queue



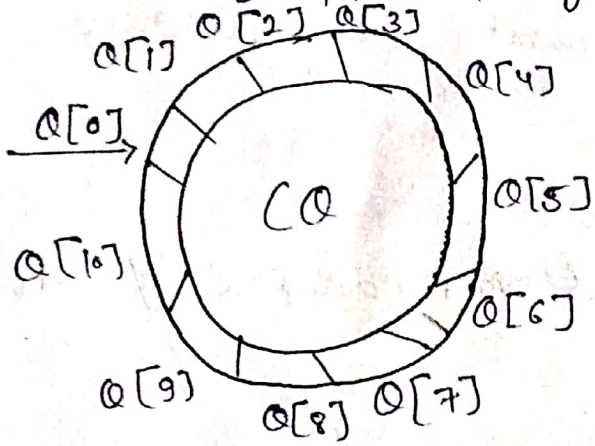
Rear = (Rear + 1) % Maxsize

= (5+1) % 6

= 6 % 6

Rear = 0

In case of circular queue Q[0] terms after Q[n-1] in the array.



maxsize = ~~10~~ 11

n = 11

n-1 = 10

Reset Rear = 0

Q[Rear] = item

Scanned with CamScanner

Linked list

(1) Initially when CQ is empty
then front = rear = -1

(2) Before insertion in circular queue.
first increase rear by 1
set rear = rear+1

(3) Similarly at the time of deletion.
delete the selected / given item, then
increase front by 1.
set front = front+1

(4) when front = 0 and rear = maxsize - 1 then CQ is full. and show overflow condition.

* With this assumption, we insert item into the Queue by assigning the item to Q[0] instead of increasing rear. we reset rear = 0 and then assigned item using this Q[rear] = item.

Circular Queue is used to overcome the limitation of linear Queue.

Scanned with CamScanner

Write down an algorithm to insert and delete an item in circular queue.

Sol

Algorithm :-

Insert circular Queue (Q, maxsize, front, Rear, item)

Begin.

① if (front = =((Rear+1) % maxsize)

then write queue is full and shows overflow condition.

exit.

② else. take the value

if (front = = -1)

set front = 0

Rear = 0

③ set Q[Rear] = item // insert item at rear.

end.

Algorithm :-

Deletion circular Queue (Q, maxsize, front, Rear, item)

Begin

① if (front = = -1)

then write queue is empty and shows underflow condition.

end.

② if (front = = Rear)

set front = -1

Rear = -1

③ front = ((front +1) % maxsize)

④ set ~~Q[front] = item~~ item = Q[front]

end

Scanned with CamScanner

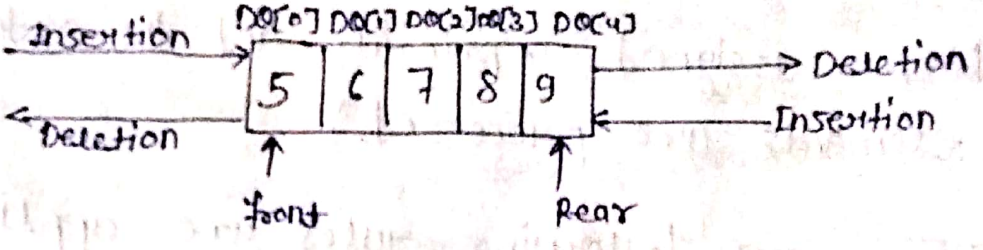
Deque (Double ended Queue) :- Deque is a short name of double ended queue. it is a homogeneous list of element in which insertion and deletions operation are performed from both end.

we can insert front end and from the Rear end both.

Similarly we can delete items from both ends

Representation of Deque :-

ex:-



There are two variation of Deque :-

① Input - restricted Deque

② Output- " "

Input - restricted Deque :- An input restricted deque restric the insertion of elements at one end only, but deletion of element can be done at both end of a queue.

Scanned with CamScanner

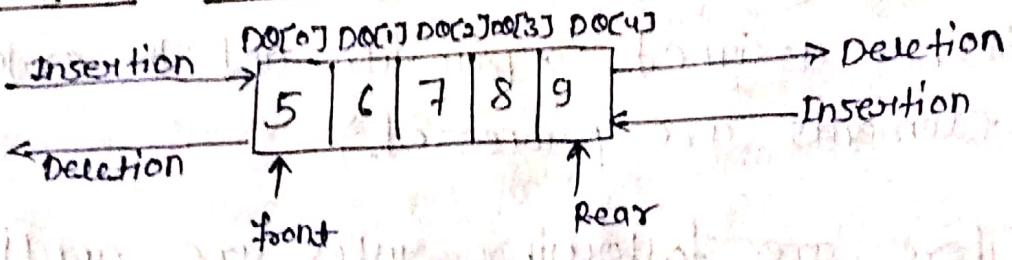
Deque (Double ended Queue) :- Deque is a short name of double ended queue. it is a homogeneous list of element in which insertion and deletions operation are performed from both end.

we can insert front end and from the Rear end both.

Similarly we can delete items from both ends

Representation of Deque :-

ex:-



There are two variation of Deque :-

1. Input - restricted Deque
2. Output- " "

Input - restricted Deque :- An input restricted deque restric the insertion of elements at one end only, but deletion of element can be done at both end of a queue.

Scanned with CamScanner

Output-restricted Deque :- An output restricted deque, restricts the deletion of an elements at only one end, but allows insertion at both ends.

Priority Queue :- The priority Queue is a collection of elements where the elements are stored according their priority level. (each element has been assigned a priority and the order in which the elements are processed.)

There are following rules are applied to maintain a priority queue.

Rule-1 :- An element of higher priority is processed before any element of lower priority.

Rule-2 :- Two element with the same priority are processed according to the order in which they are added.

Application of Queue :- There are various application of Queue as given.

1. Priority queue are used for implementing job scheduling (process scheduling) by the operating system. with higher priority processed are to be processed first.

Scanned with CamScanner

② Queue are also used in different scheduling technique. like. shortest job first, Round-Robin technique (In time sharing system) for processor scheduling.

③ Printer service routine are designed using queue.

Q① Consider a dequeue where deque is a circular array which is allocated six memory cells the given position of front and rear is front = 2 and rear = 4. Describe the following operations.

① f is added to rear of the deque.

② Two letter on the rear end are deleted

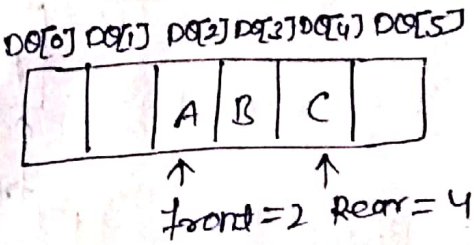
③ K and L are added from front end

④ one letter is deleted from front end

⑤ R is added from the front end.

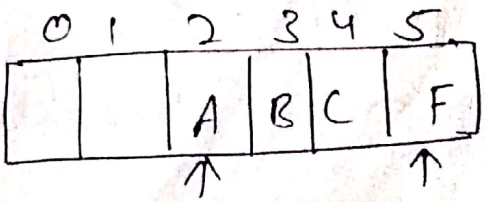
⑥ 5 is added from the right end.

Sol

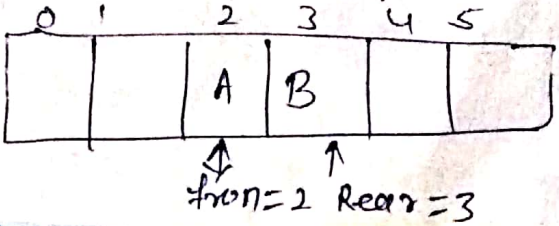


find out front and rear location after processing all applying insertion and deletion operation.

Ans ①

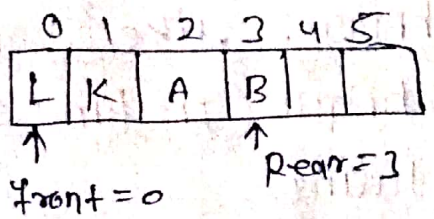


② Two item deleted from rear end

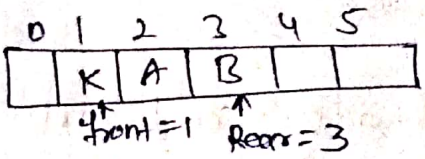


Scanned with CamScanner

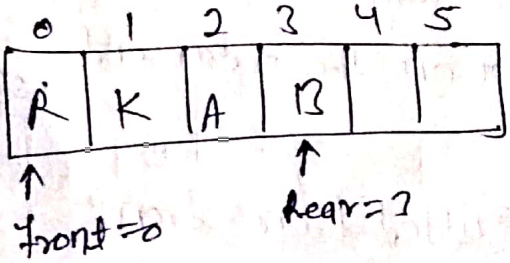
(3) K and L are added front end.



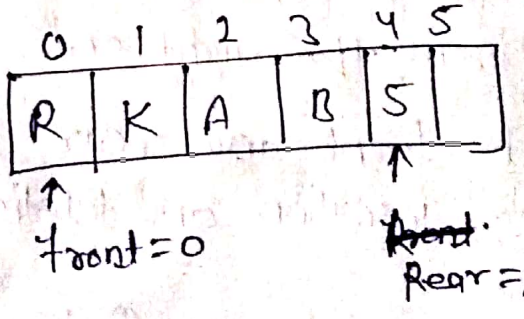
(4) one letter is deleted from front end.



(5) R is added from front



(6) S is added from rear end.



Scanned with CamScanner

-: Sorting :-

Sorting :- Sorting is the process of arranging the elements in some order. (logical order).
The logical order may be ascending or descending order. in case of numeric value

Types of sorting :- There are various types of sorting

1. Bubble sorting
2. Selection sorting
3. Insertion sorting
4. Quick sorting
5. Merge sorting
6. Heap sorting.

Internal sorting :- Internal sorting deals with sorting the data held in computer memory.

External sorting :- it deals with sorting the data stored in data files. this method is used when the volume of data is very large and cannot be held in computer memory.
external sorting takes more time to sort the given

Scanned with CamScanner

item which is stored in a file.

① Bubble sorting :-

Bubble sorting require (n-1) passes to sort n number of elements.

Bubble sort method require (n-1) pass to sort an array.

in each pass every elements a[i] is compaired with a[i+1]

for i=0 to n-k

where (K) is the number of pass.

If a[i] > a[i+1] then they are swapped, thus.

after the end of first pass the largest element of the list is placed at the location of the array. In each successive pass, the next largest element is placed at position 0, 1, ---- (n-3), (n-2), (n-1) respectively.

Scanned with CamScanner

Algorithm for Bubble sorting

Bubble sort (a, n)

a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]
7	3	5	9	8	2	4

Begin

Step-1 for i = 0 to n-1 by 1 do

Step-2 for j = 0 to (n-i-1) by 1 do

Step-3 if (a[j] > a[j+1])

set temp = a[j] // temp = 7

set a[j] = a[j+1] // a[j] = 3

set a[j+1] = temp // a[j+1] = 7

ex:- 3, 11, 1, 10, 6, 2, 4, 5

pass-1

3, (11), (1)

3, 1, (11), (10), 6, 2, 4, 5

3, 1, 10, (11), (6), 2, 4, 5

3, 1, 10, 6, (11), (2), 4, 5

3, 1, 10, 6, 2, (11), (4), 5

3, 1, 10, 6, 2, 4, (11), (5)

3, 1, 10, 6, 2, 4, 5, 11

Scanned with CamScanner

Pass-2 (3) (1) 10, 6, 2, 4, 5, 11

1, (3), (10), 6, 2, 4, 5, 11

Do not exchange

1, 3, (10), (6), 2, 4, 5, 11

1, 3, 6, (10), (2), 4, 5, 11

1, 3, 6, 2, (10), (4), 5, 11

1, 3, 6, 2, 4, (10), (5), 11

11, 3, 6, 2, 4, 5, 10, 11

Pass-3 (1) (3) 6, 2, 4, 5, 10, 11

Do not exchange.

1, 3, (6) (2) 4, 5, 10, 11

1, 3, 2, (6), (4), 5, 10, 11

1, 3, 2, 4, (6), (5) 10, 11

1, 3, 2, 4, 5, 6, 10, 11

Pass-4 1, 3, 2, 4, 5, 6, 10, 11

Do not exchange

1, (3), (2), 4, 5, 6, 10, 11

1, 2, 3, 4, 5, 6, 10, 11

Scanned with CamScanner

Analysis of Bubble sort

Size of list = n

$$T(n) = n-1 + (n-2) + (n-3) + \cdots + 2+1$$

$$= \frac{n}{2}[2a + (n-1)d]$$

$$= \frac{(n-1)}{2}[2 \cdot 1 + (n-1-1) \cdot 1]$$

$$= \frac{(n-1)}{2}[2 + n - 2]$$

$$= \frac{n(n-1)}{2} = \frac{(n^2-n)}{2}$$

Complexity of Bubble sort. $T(n) = \Theta(n^2)$

Scanned with CamScanner

Selection sorting :-

The selection sort method also requires (n-1) pass to sort an array.

In the first pass, find the smallest element of the list from a[0], a[1], ---- ---- a[n-1] and swap with the smallest element with a[0].

In the second pass, find the next smallest element of the list from a[1] ---------- a[n-1] and swap with the desired element. and so on.

Algorith for selection sort :-

Selection sort (a, n)

Begin

① for i = 0 to n-1

set Min = i

② for j = i+1 to n-1

③ if (a[j] < a[min]) then

set min = j

④ do swapping.

set temp = a[i]

set a[i] = a[min]

set a[min] = temp.

Scanned with CamScanner

Analysis of selection sort :-

Size of list = n

$$T(n) = (n-1) + (n-2) + (n-3) + \ldots + 2 + 1$$

$$= \frac{n}{2}[2a + (n-1)d]$$

$$= \frac{n-1}{2}[2 \times 1 + (n-1-1) \cdot 1]$$

$$= \frac{n-1}{2}[2 + n - 2]$$

$$= \frac{n(n-1)}{2} = \frac{(n^2 - n)}{2}$$

complexcity of selection sort $T(n) = \Theta(n^2)$

Scanned with CamScanner

Insertion sorting

Insertion is an inplace compairision based sorting technique.

in insertion sort an element which is to be inserted in a sorted list, first find its appropriate place and then put it (inserted) at that place, so this technique is called insertion short.

The given array is searched sequentially and the selected item inserted at their appropriate position in the sorted sublist.

Algorithm

Insertion sort (a, n)

Begin

(1) for i=1 to n-1 do

(2) temp = a[i]

(3) set j = i-1 || j start at previous location i

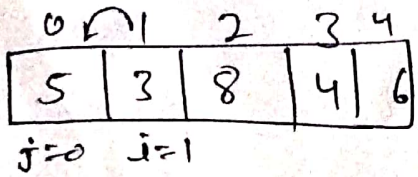
(4) while ((j>= 0) && (a[j] > temp))

(5) set a[j+1] = a[j]

(6) set j = j-1 || j = -1

(7) set a[j+1] = temp

ex:-



3, 5, 8, 4, 6

3, 4, 5, 8, 6

3, 4, 5, 6, 8

Scanned with CamScanner

* This sorting technique is not suitable for large data set. (larger size of list

* Insertion sort technique better for smaller size of list.

* Running time complexicity in worst case is $O(n^2)$

Seen
14/10/2022

```
#include <stdio.h>
int main ()
{ int a[50], i,j ,n , temp;
printf(" Enter the size of array \n");
scanf(" %d ", &n );
printf(" Enter the elements \n");
for (i=0 ; i<n; i++);
{
scanf ("%d", &a[i]);
}
for (i=1 ; i<n; i++)
{
temp = a[i];
j=i-1 ;
while ((j>=0 && (a[j] > temp))
{
a[j+1] = a[j];
j=j-1 ;
}
a[j+1] = temp;
}
```

Scanned with CamScanner

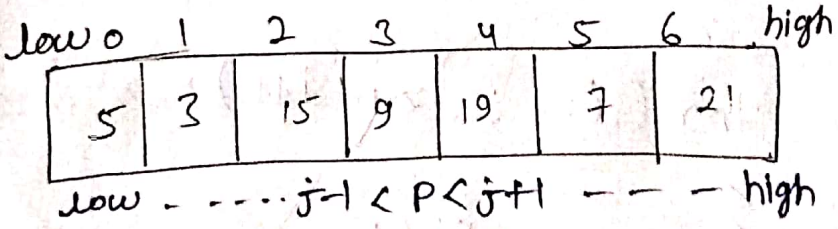
Quick Sorting

Quick sort is an algorithm of the divide and conquer type. quick sort is also known as partition exchange sort.

The basics of this algorithm is that to partition a single list into two sub-list. that to easy to sort elements and it is faster to sort element after sorting this two sublist we merge into a single list.

The basic strategy of quick sort is to divide and conquer.

to sort the element of the list by using quick sort first we select a pivat value or element. and fix this pivat value at a proper position where list is partitioned into two types sublist-1 and sublist-2, where all elements of sublist-1 is smallerthan pivat element and all elements of sublist-2 is greater than pivat element.



set low=i

low=P

high=j

Quick sort (a, low, j-1)

Quick sort (a, j+1, high)

Scanned with CamScanner

Quick sort Algorithm

Quick sort (a, low, high, pivot, n)

Begin

① if (low < high)
set pivot or p = low
set i = low
set j = high

② while (i < j)

③ while ((a[i] <= a[p]) && i < high), then
i = i+1 || i++

④ while (a[j] > a[p]) then
j = j-1 || j--

⑤ if (i < j) then swap a[i] → a[j]
set temp = a[i]
set a[i] = a[j]
a[j] = temp

⑥ swap temp = a[p] || swap a[p] → a[j]
set a[p] = a[j]
set a[j] = temp

⑦ Quick sort (a, low, j-1)
Quick sort (a, j+1, high).

end.

Quick Sort Running time complexity :-

$$\frac{n}{2} + \frac{n}{2}$$

$$T_n = O(n \cdot \log_2^n)$$

Scanned with CamScanner

ex:- 44, 33, 11, 55, 77, 90, 40, 60, 99, 22, 88, 66

low=0 (p/i) high=11 (j)

sol

step-1 (44), 33, 11, 55, 77, 90, 40, 60, 99, 22, 88, 66
P i j

step-2 (44), (33), (11), 55, 77, 90, 40, 60, 99, 22, 88, 66
P i j

step-3 (44), 33, (11), (55), 77, 90, 40, 60, 99, 22, 88, 66
P i j

stop increment i. condition false.

while (a[j] > a[p]) ; compare a[j] and a[p] if condition is true then j will decrease by 1.

step-4 (44), 33, 11, 55, 77, 90, 40, 60, 99, 22, (88), 66
P i j

Again decrease j by 1

step-5 (44), 33, 11, (55), 77, 90, 40, 60, 99, (22), 88, 66
P i j

above condition false, index value of i < j then swap. a[i] ⇔ a[j]

44, 33, 11, (22), 77, 90, 40, 60, 99, (55), 88, 66
P i j

step-6 (44), 33, 11, 22, (77), 90, 40, 60, 99, (55), 88, 66
P i j

step-7 (44), 33, 11, 22, (77), 90, 40, 60, (90), 55, 88, 66
P i j

step-8 (44), 33, 11, 22, (77), 90, 40, (60), 90, 55, 88, 66
P i j

step-9 44, 33, 11, 22, (77), 90, (40), 60, 90, 55, 88, 66
i j

swap a[i] ⇔ a[j]

44, 33, 11, 22, (40), 90, (77), 60, 90, 55, 88, 66
p i j

Scanned with CamScanner

Step-10 (44) 33, 11, 22, 40, (90) (77) 60, 99, 55 188, 66
P i j

Step-11 44, 33, 11, 22, 40, (99) 77, 60, 99, 55 188, 66
i/j

Step-12 44, 33, 11, 22, (40) (99) 77, 60, 99, 55, 88, 66
j i

If (i > j) then swap a[p] ↔ a[j]

low 40, 33, 11, 22, (44) 90, 77, 60, 99, 55, 88, 66 high
j-1 p/j j+1
sublist-1 sublist-2

Quick sort (a, low, j-1)

Quick sort (a, j+1, high).

Again apply algorithm. in sublist-1

40, 33, 11, 22
low = 0 high
p/i j

Step-1 i-condition is true then i = i+1

(40), (33), 11, (22)
p i j

Step-2 Again i-condition is true then i = i+1

(40), 33, (11), (22)
p i j

i-condition is true. i = i+1

Step-3 (40), 33, 11, (22)
p i/j

Step-4 Again i-condition is true. i = i+1

(40), 33, 11, (22) || i = 4
p j

if (i > j) then swap a[p] ↔ a[j]

Scanned with CamScanner

Step-5 low
22, 33, 11, 40
j-1

New sublist.

Step-6 low=0 high=2
22, 33, 11
P i j

1- condition is true. i=i+1

step-7 (22), (33), (11)
P i j

1 condition is false.
apply 2 condition it is also false.
swap a[i] ⟷ a[j]

step-8 (22), (11), (33)
P i j

Apply condition 1 i=i+1

step-9 (22), (11), (33) || i=2
P i|j

condition 2 is true, j=j-1

step-10 (22), (11), (33)
P j i

if (i<j) this condition is false. the swap
a[p] ⟷ a[j]

step-11 11, 22, 33

Sublist-1 complete.

11, 22 33, 44,

Scanned with CamScanner

for sublist-2

low=0 high
90, 77, 60, 99, 55, 88, 66
P i j

Step-1 1-condition is true, $i = i+1$

low
90, (77), 60, 99, 55, 88, (66)
P i j

Step-2 Again 1-condition is true, $i = i+1$

90, 77, (60), 99, 55, 88, (66)
P i j

Step-3 1-condition is true. $i = i+1$

(90), 77, 60, (99), 55, 88, (66)
P i j

Step-4 condition-1 is false and condition 2 is also false
condition-3 is true. $(i < j)$ then swap $a[i] \leftrightarrow a[j]$

(90), 77, 60, (66), ~~55~~, ~~88~~, (99)
P i j

Step-5 1-condition is true, $i = i+1$

(90), 77, 60, 66, (55), 88, (99)
P i j

1-condition is true, $i = i+1$

Step-6 (90), 77, 60, 66, 55, (88), (99)
P i j

Again 1-condition is true, $i = i+1$

Step-7 (90), 77, 60, 66, 55, 88, (99) $\|i = 6$
P i|j

2-condition is true, $j = j-1$

Step-8 (90), 77, 60, 66, 55, (88), (99)
P j i

if $(i > j)$ swap $a[p] \leftrightarrow a[j]$

Scanned with CamScanner

Step-9

low
88, 77, 60, 66, 55, (99), 99,
j-1 p j+1

New sublis-1

Step-10

low high
88, 77, 60, 66, 55
i|p j

1-condition is true. , i=i+1

Step-11

(88), (77), 60, 66, (55)
p i j

1-condition is true. , i=i+1

Step-12

(88), 77, (60), 66, (55)
p i j

1-condition is again true. , i=i+1

Step-13

(88), 77, 60, (66), (55)
p i j

Again 1-condition is true, i=i+1

Step-14

(88), 77, 60, 66, (55) || i=4
p i|j

Step-15

(88), 77, 60, 66, 55, || i=5
p j

if (i>j), swap a[p] ⇔ a[j]

Step-16

low
55, 77, 60, 66, (88)
j-1 p

New-sublist.

Scanned with CamScanner

Step-17

low high
55, 77, 60, 66
i|p j

Step-18

1-condition is true. , $i = i+1$

(55), (77), 60, (66)
p i j

Step-19

2-condition is true, $j = j-1$

(55), (77), (60), 66
p i j

Step-20

2-condition is again true. , $j = j-1$

(55), (77), 60, 66
p i|j

Step-21

Again 2-condition is true, $j = j-1$

(55), (77), 60, 66
p|j i
j<i high

New-sublist.

Step-22

low high
77, 60, 66
i|p j

1-condition is true, $i = i+1$

Step-23

(77), (60), (66)
p i j

1-condition is true, $i = i+1$

Step-24

(77), 60, (66)
p i|j

1-condition is true, $i = i+1$

Step-25

(77), 60, (66) // $i = 3$
p j

if ($i > j$) swap $a[p] \leftrightarrow a[j]$

Scanned with CamScanner

Step-26

low
66, 60, 77,
j-1 p

↓
New sublist.

Step-27

low
66, 60,
i|p j

1- condition is true, $i = i+1$

Step-28

(66), (60),
p i|j

1- condition again true, $i = i+1$

Step-29

(66), (60), || $i = 2$
p j

if ($i > j$) swap $a[p] \leftrightarrow a[j]$

Step-30

60, 66

final after sorting.

11, 22, 33, 44, 55, 60, 66, 77, 88, 90, 99 Ans

Very Good

Swap

Scanned with CamScanner

Which sort is fast and take less time to sort a list?
$n = 8$

Bubble sort $= O(n^2) = 64$

selection sort $= O(n^2) = 64$

Insertion sort $= O(n^2) = 64$

Quick sort $= O(n \cdot \log_2 n) = 24$

$= 8 \cdot \log 2^8$

$= 8 \cdot \log_2 2^3 =$

$= 8 \cdot 3$

$= 24$

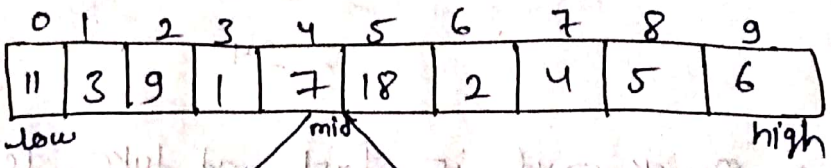
Note:- Quick sort is fast and take less time. because the no. of itrations are less as compared to all other sorts. in all case.

Scanned with CamScanner

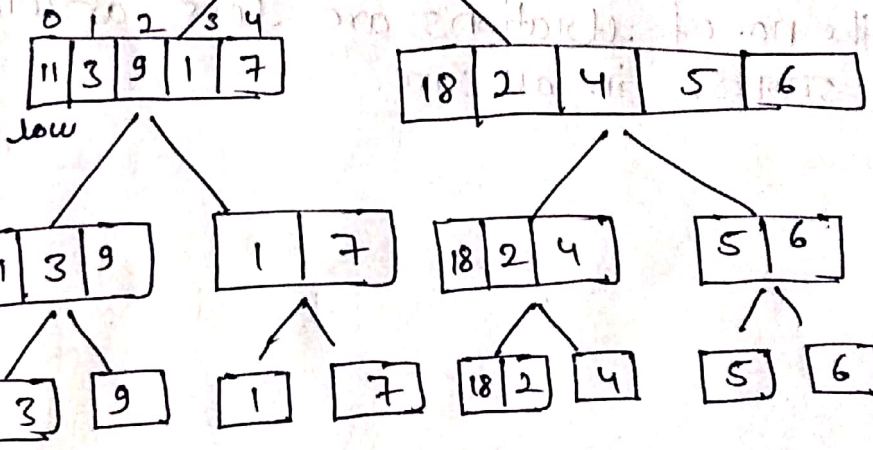
Merge sort

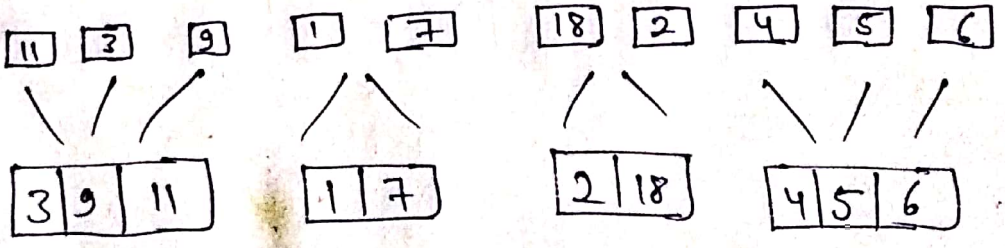
Merge sort is also recursing sorting technique. it is based on divide and conquer strategy, in which list is divided into sublist (sub array) till we get sublist of size-1.
After partition compare elements of one elements of sublist to merge them into a two elements of sorted sublist. this process continoas upto till we get a complete sorted list or array.

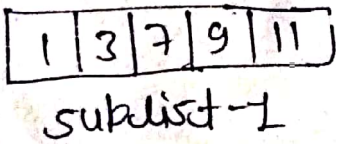
ex:-



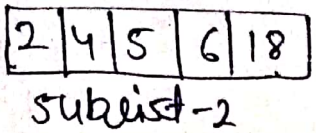
mid = (low+high)/2
= (0+9)/2
= 4



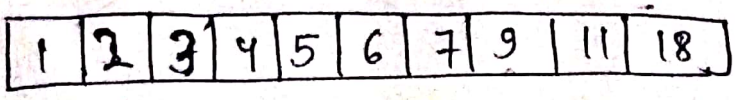




sublist-1



sublist-2



Scanned with CamScanner

Merge sorting Algorithm

Algorithm

Mergesort (a, low, mid, high)
Combine (a, low, mid, high)

Begin

Step-1 if (low < high), if it is true. then
calculate mid = (low+high)/2

Step-2 After partition the list
Call mergesort (a, low, mid)
Call mergesort (a, mid+1, high)

Call combine (a, low, mid, high)

Step-3 combine (a, low, mid, high)

(a) we select a temporary array temp[K]
set K = low
set i = low
set j = mid+1

(b) while (i <= mid && j <= high)

(c) if (a[i] < a[j]), then
set temp[K] = a[i]
set i++ and set K++ || increase by 1

(d) else
set temp[K] = a[j]
set j++ and K++ || increase by 1

(e) while (i <= mid), then
set temp[K] = a[i]
set i++ and set K++

Scanned with CamScanner

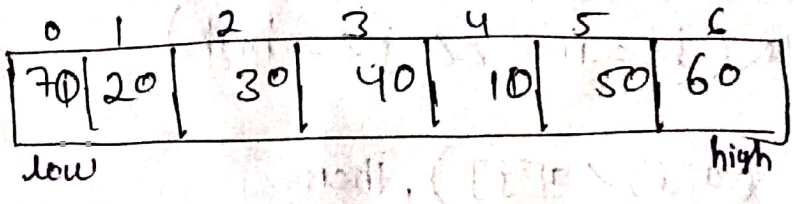
⑦ while (j<=high), then
set temp[k] = a[j] and
set j++, and set k++

⑧ for k=low to high do
a[k] = temp[k], then
print sorted array.

end.

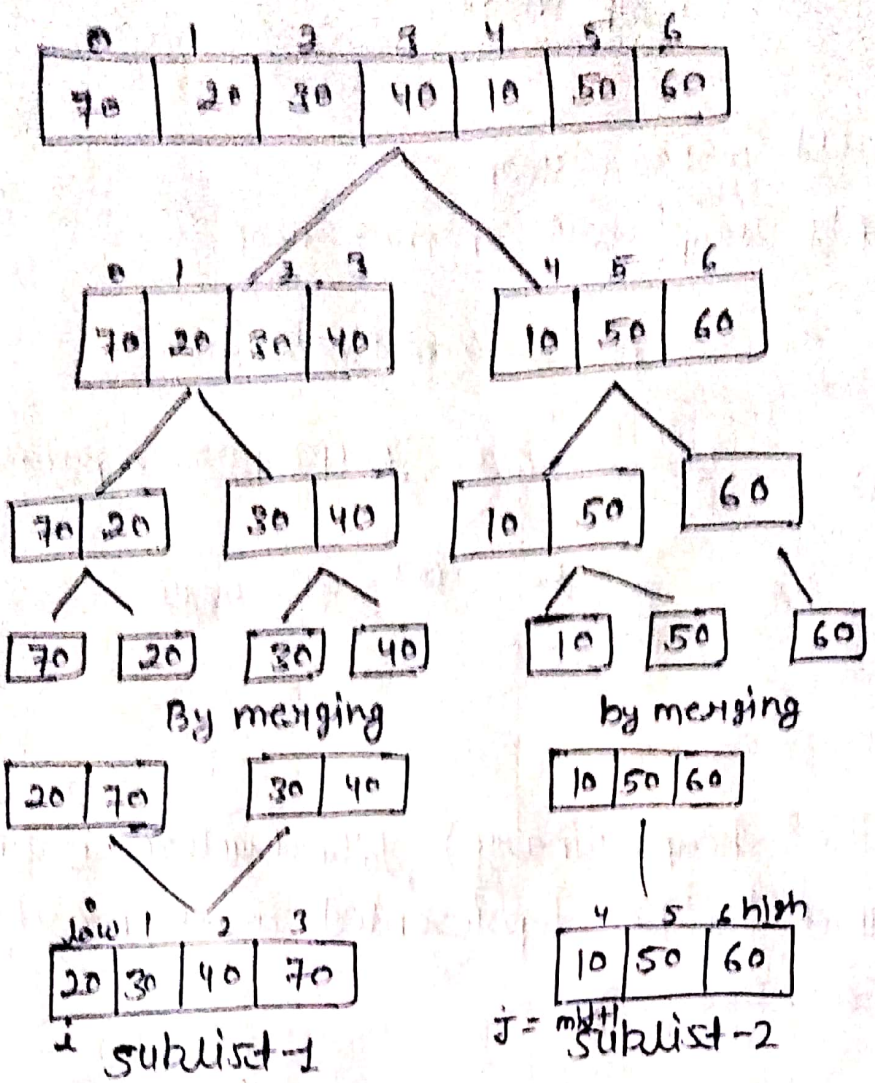
Complexity of merge sort $O(n \log_2^n)$ in all case.

Q sort the following list by using merge sort.



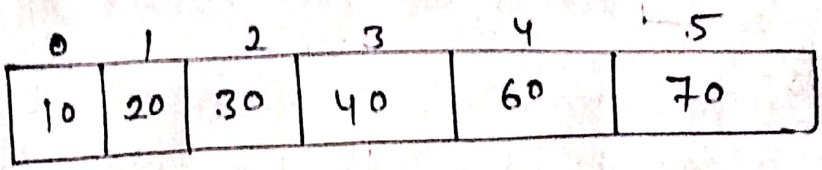
sol Calculate mid = (low+high)/2
= (0+6)/2
= 3

Scanned with CamScanner



Now we check condition. and take temp. array

temp[k]



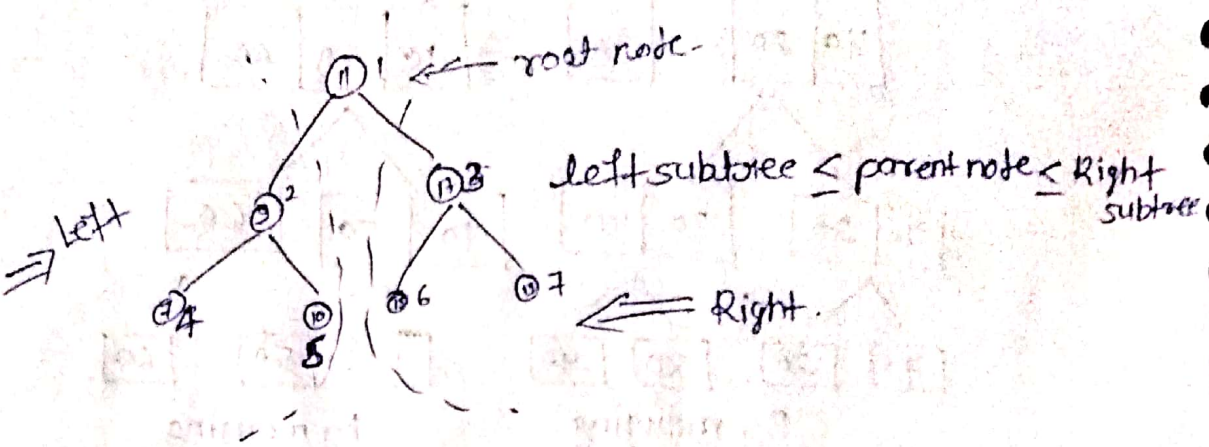
a[6] = temp[6]

list in sorted order.

Scanned with CamScanner

Heap sorting

1. Build a Max-Heap
2. Apply Heap sort on max-Heap.



Heaps :- The heap (Binary) data structure is an array object that can be represented as a nearly complete binary tree.

An array A that represent a heap is an object with two attributes.

Attributes :-

1. Length [A]
2. Heap-size[A]

1. **Length [A] :-** it is the number of elements in the array.

2. **Heap-size[A]** it is the numbers of element in the heap stored with in array [A].

Scanned with CamScanner

Heap_size[A] ≤ Length[A]

The root of tree is A[1], and given the index i of a node.

A[i]

For finding parent node.

① Parent[i]
return ⌊i/2⌋

parent[2]
return ⌊2/2⌋
= 1

For finding left and right node.

② Left[i]
return (2·i)

ex:- Left[1]
return (2·1)
= 2

③ Right[i]
return (2·i+1)

ex:- Right[1]
return (2·1+1)
= 3

There are two kinds of binary heaps.

① Max-Heap:- In a max-heap, the max-heap property is that for every node i other than the root node.

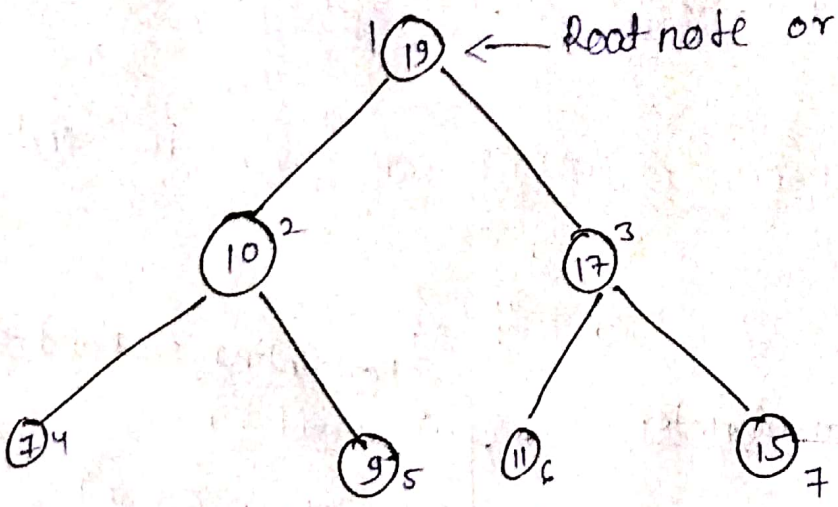
$$A[(\text{parent}[i])] \geq A[i]$$

* For every level of binary tree parent node is always greater or equal to it's child node.

"The largest element in the max-heap is sorted at the root" and the subtree rooted at a node contains values no larger than that contained at the node itself

Scanned with CamScanner

Ex:-



(2) Min-heap:- A min-heap is organised in the opposite way, the min_heap property is that for every node i other than root.

$$A[(parent[i])] \leq A[i]$$

Thus, the smallest element in a mean heap is at the root.

Building a Max-Heap:- we can use the procedure max-Heapify in a bottom up manner to convert an array $A[1 \ldots n]$, where (n) is the length of[A] or array.

Scanned with CamScanner

Algorithm

Build max-Heap [A]

Begin :-

(1) Heap_size [A] = length[A]

(2) for i = ⌊length [A]/2⌋ down to 1

(3) do max-Heapify (A,i)

end

* Compare the node value to its parent node and which one is greater that will shift to its parent node.

ex :-

1	2	3	4	5	6	7	8	9	10
9	11	6	45	22	10	12	90	67	17

build max-heap.

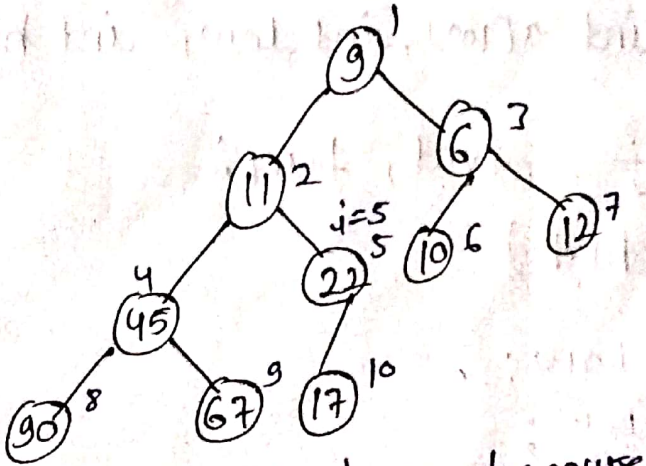
length [A] = 10 Heapsize [A] = 10

i = ⌊length[A]/2⌋ down to 1

i = 10/2

i = 5

step :-1



donot change because parent node is greater than child node.

Scanned with CamScanner

WAP of Quick sorting

```
#include<stdio.h>
#include<conio.h>
void Quicksort (int a[100], int low, int high);
   int main ()
{  int i, n, a[100];
printf (" enter the size of array\n ");
   scanf ("%d ", &n);
  printf(" enter the elements of array:\n ");
      for(i=0; i<n; i++)
        scanf(" %d ", &a[i]);
    Quicksort (a, 0, n-1);
printf (" sorted in asending order :-> ");
   for (i=0 ; i<n ; i++)
        printf (" %d", a[i]);
   return 0;
}
void Quicksort (int a[100], int low, int high)
    {  int, i, j, pivat, temp;
      if (low<high)
      {  pivat=low;
          i = low;
          j = high;
        while (i<j)
```

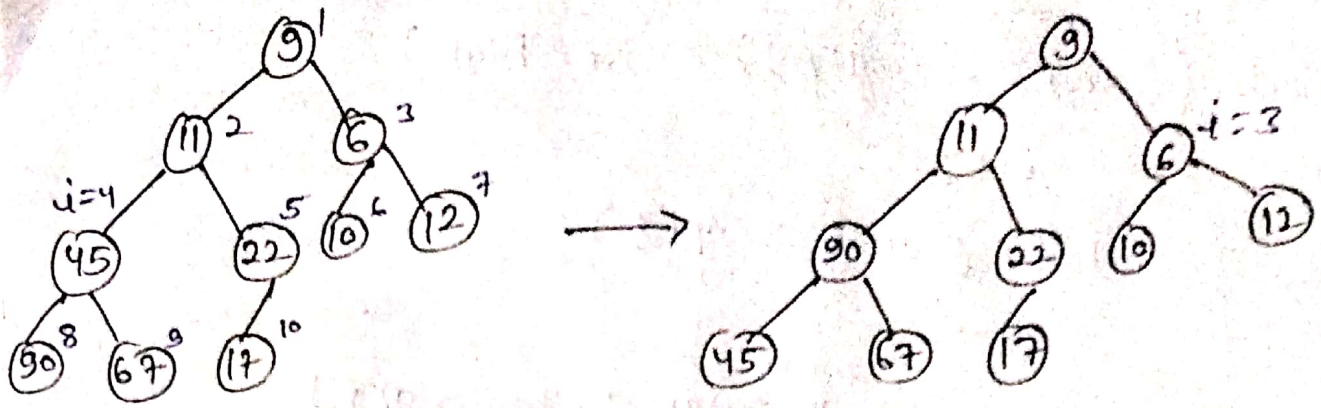
Scanned with CamScanner

```
{
while((a[i] <= a[p]) && i<high)
    i++;
  while (a[j] > a[p])
    j--;
  if (i<j) // swap a[i] <--> a[j]
  {
    temp = a[i];
    a[i]=a[j];
    a[j] = temp;
  }
}
  else // swap a[p] <--> a[j]
    temp= a[p];
    a[p] = a[j];
    a[j] = temp;

Quicksort (a, low, j-1);
Quicksort (a, j+1, high);

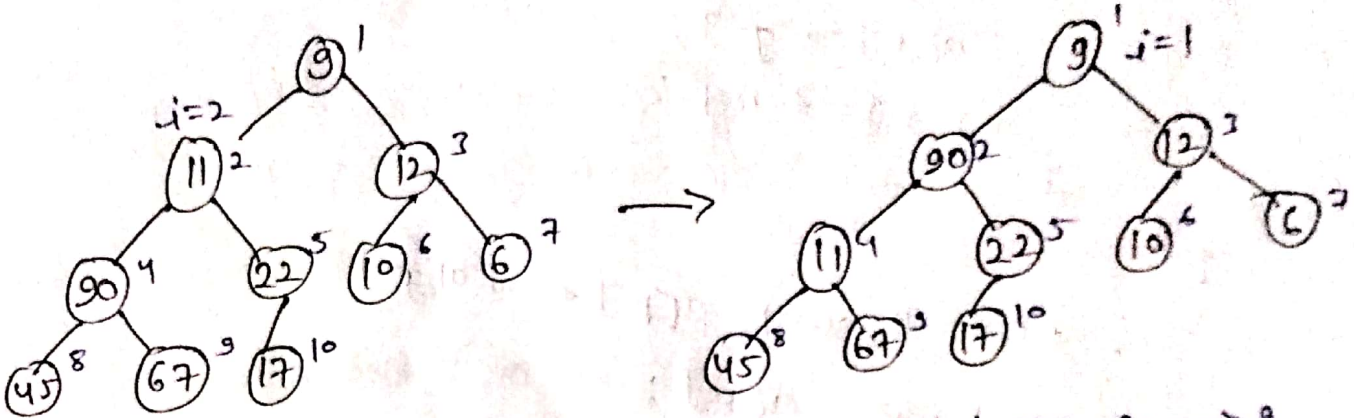
}
```

Scanned with CamScanner



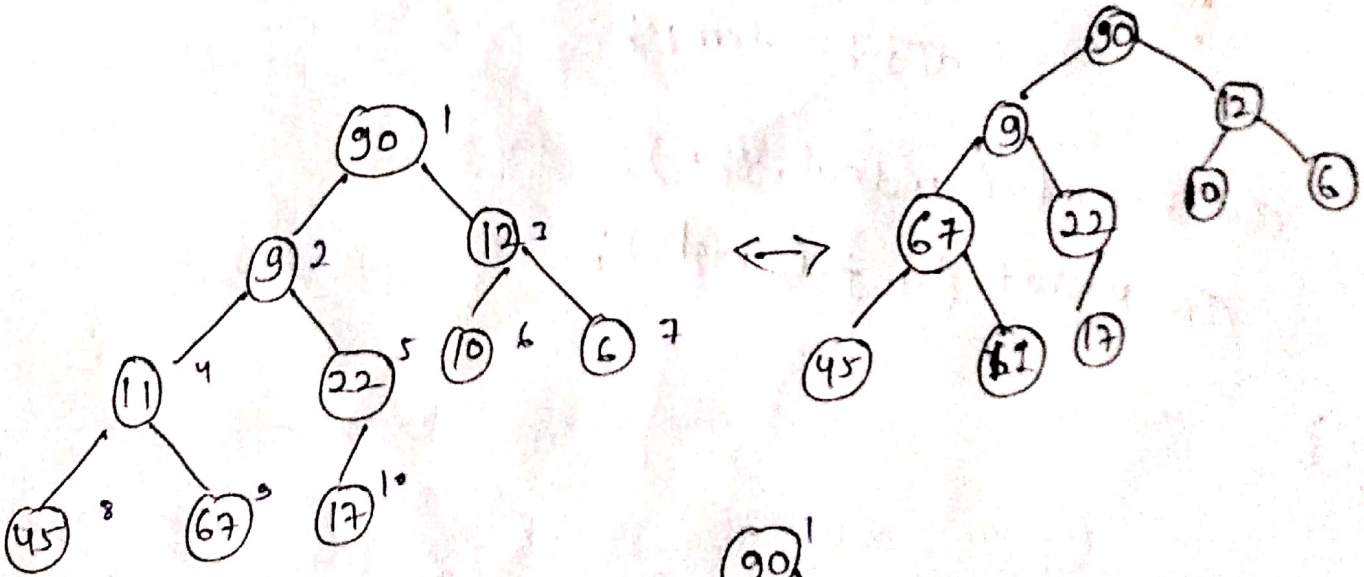
exchange 45 to 90 max.

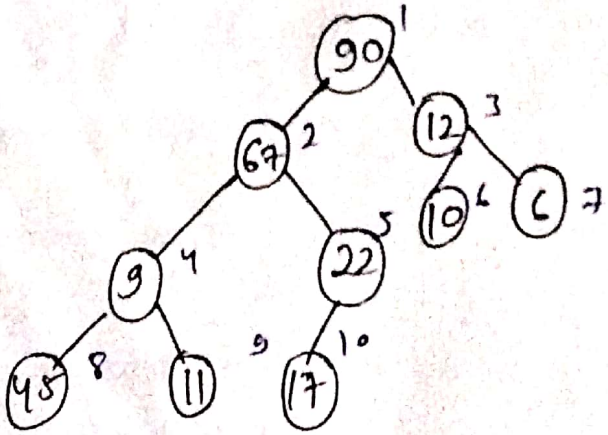
exchange 12 → 6 max.

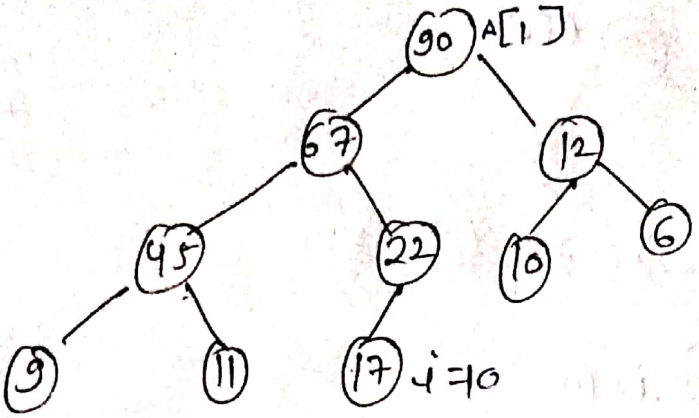


exchange 90 → 11

exchange 90 → 9







Now Apply sorting on max-Heap

A[1] = A[i]

Heap sort :- The heap sort algorithm starts by using build max-heap. build max-heap on the input array A[1, 2, - - - - n], where

n = length[A]

Since the maximum element of the array is shorted at the root A[1], it can be put into correct final position by exchanging it with A[n].

The heapsort algorithm then repeats this process for the max-heap of size (n-1) down to heapsize -2.

* The heapsort procedure takes running time = $O(n \cdot \log \frac{n}{2})$

Algorithm

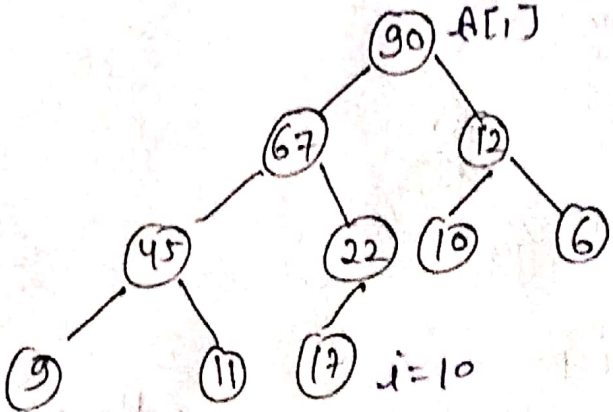
Heapsort[A]

Begin

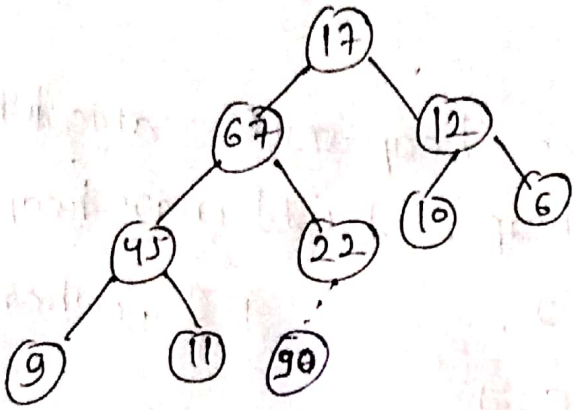
1. Build max-heap[A]
2. for i = length[A] down to 2
3. do exchange A[1] ↔ A[i]
4. Heapsize[A] = Heapsize[A]-1
5. do max-heapify (A, 1)

end

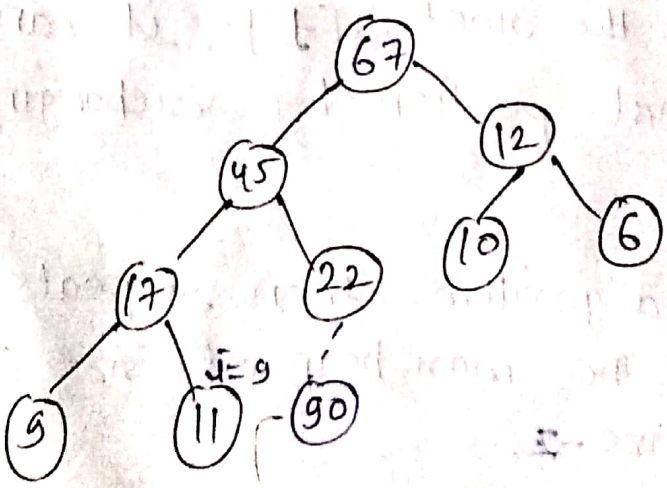
Scanned with CamScanner



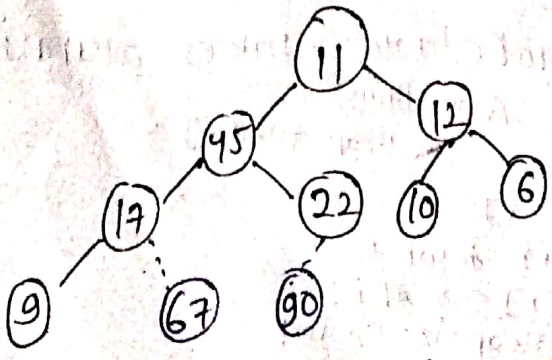
do exchange A[1] ↔ A[i]



do max-heapify.

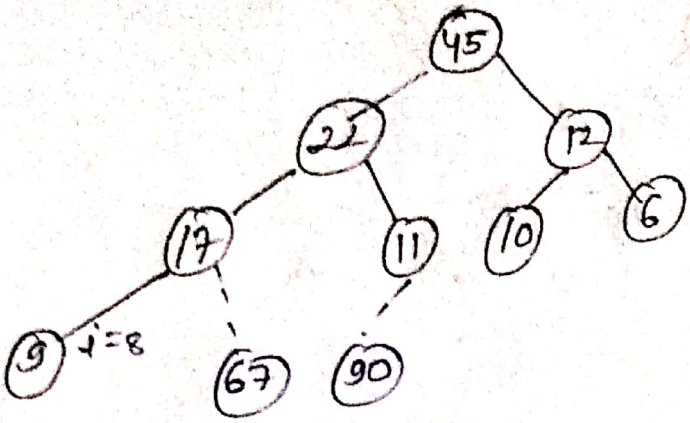


do exchange A[1] ↔ A[i]

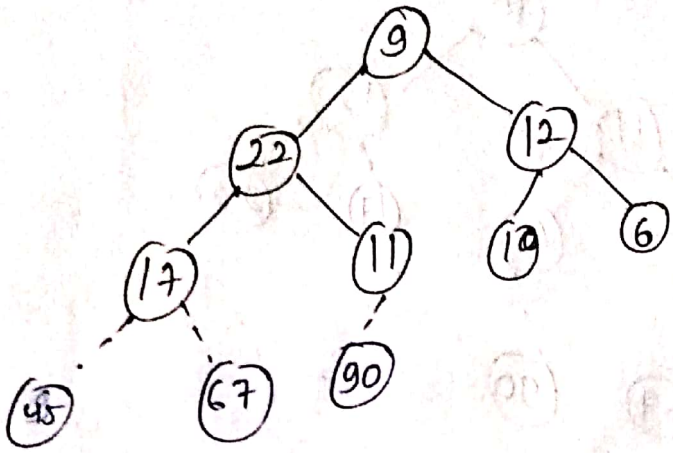


do max-heapify

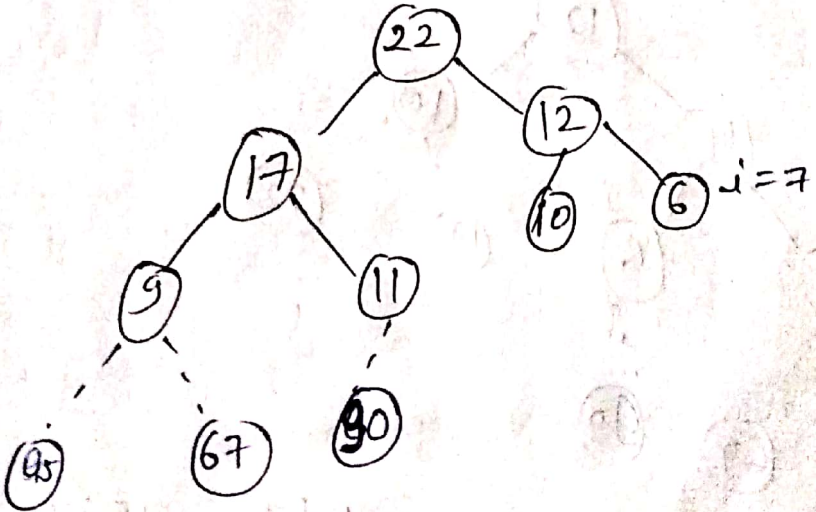
Scanned with CamScanner



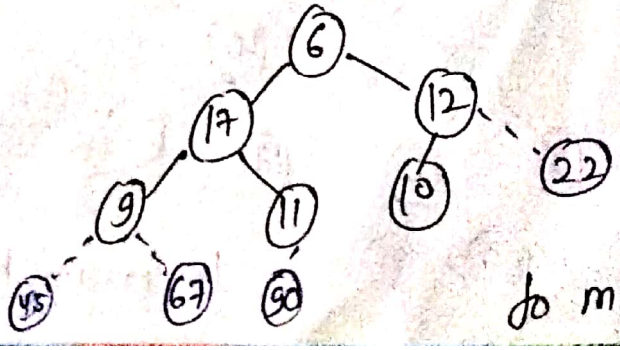
do exchang A[1] ⟷ A[i]



do max-heapify.

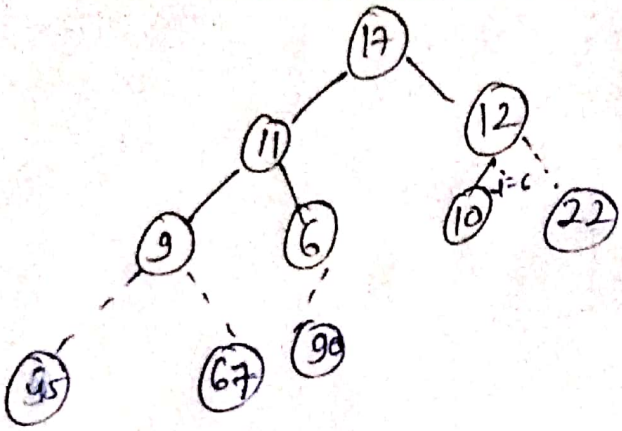


do exchange A[1] ⟷ A[i]

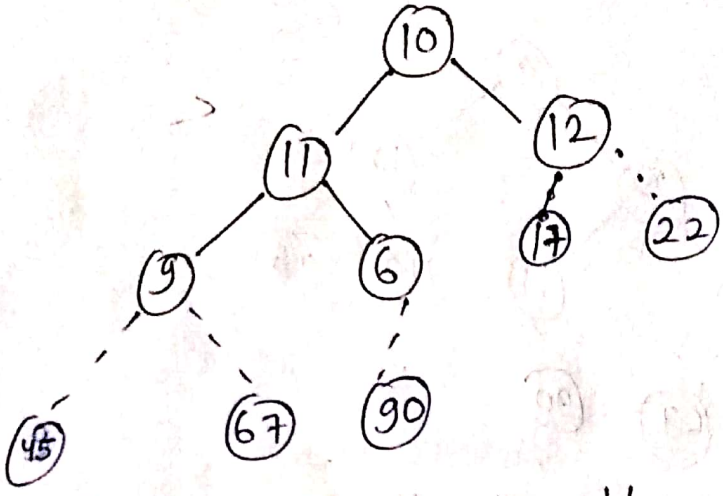


do max-heap.

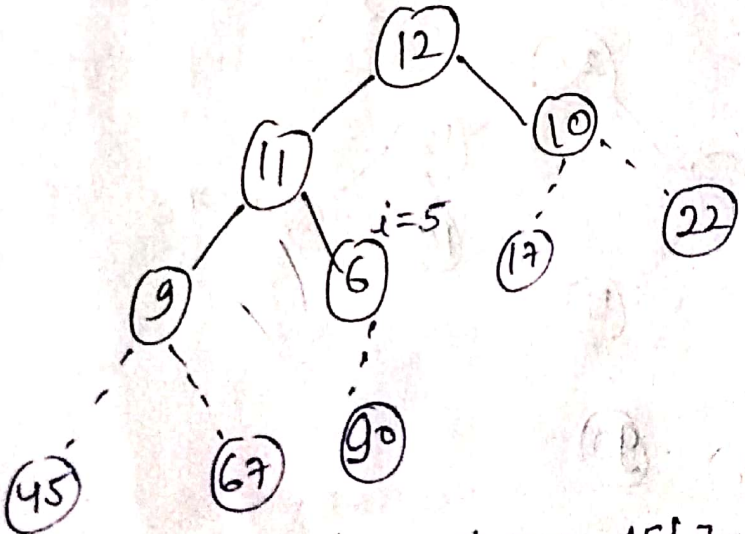
Scanned with CamScanner



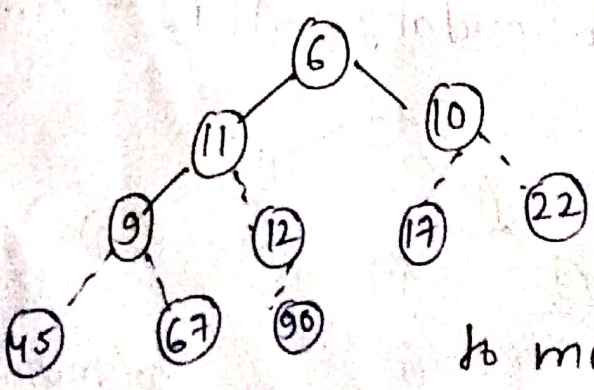
do exchange A[1] ↔ A[i]



do max-heapify

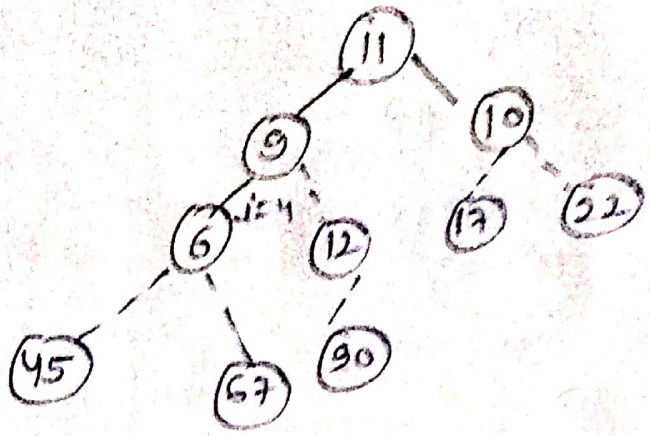


do exchange A[1] ↔ A[i]

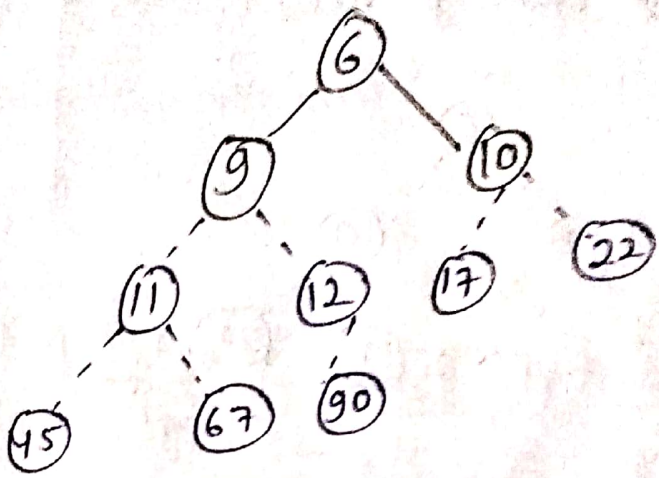


do max_heap.

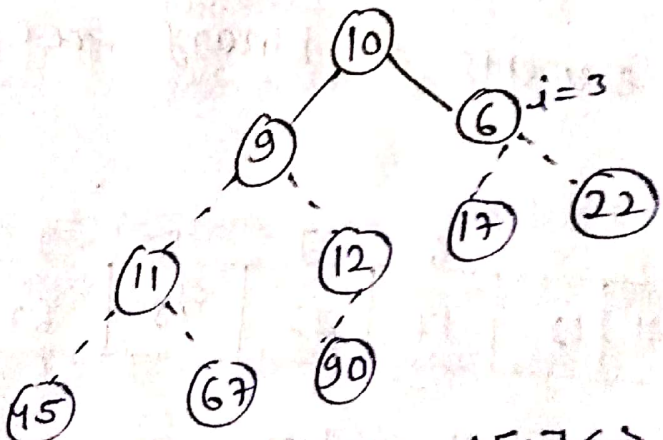
Scanned with CamScanner



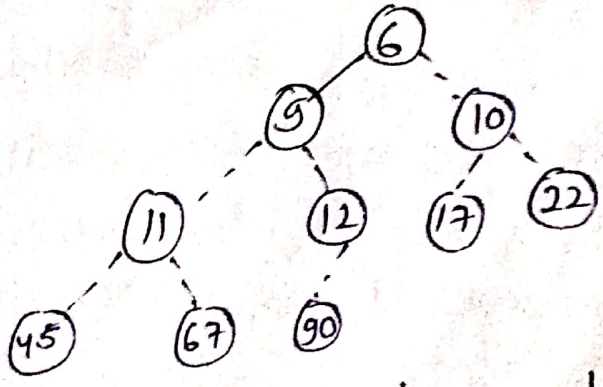
do exchange A[1] ↔ A[i]



do max-heap

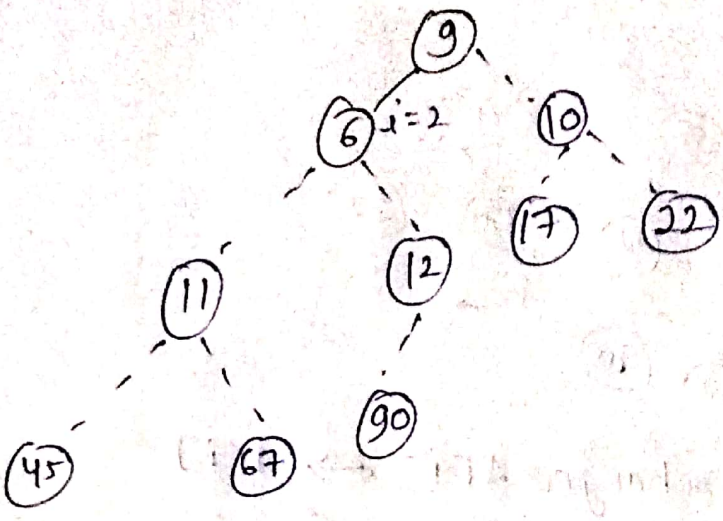


do exchange A[1] ↔ A[i]

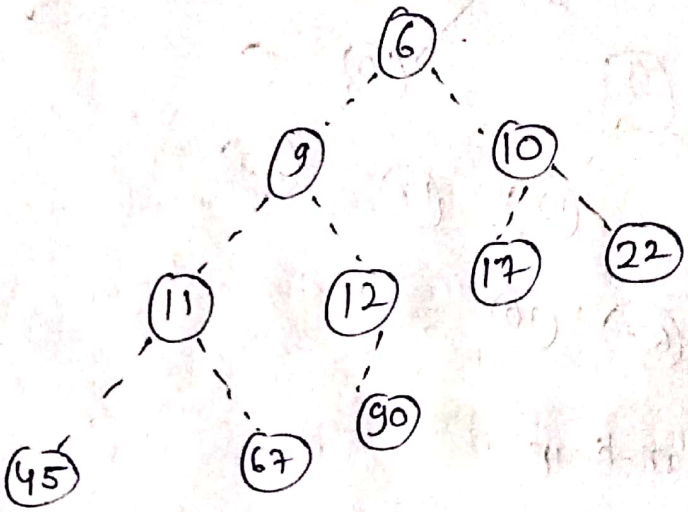


do max-heap

Scanned with CamScanner



do exchange A[1] ⟷ A[i]



put all elements of binary tree into array.

1	2	3	4	5	6	7	8	9	10
6	9	10	11	12	17	22	45	67	90

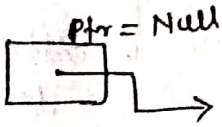
In a sorted order.

Scanned with CamScanner

Linked list

A linked list is a collection of elements, the elements of linked list is stored at a node (at information field).
Observe that the link list is collection of elements called nodes, each of which stores two items of information.
1-one is information field. and 2-one is a next pointer field.

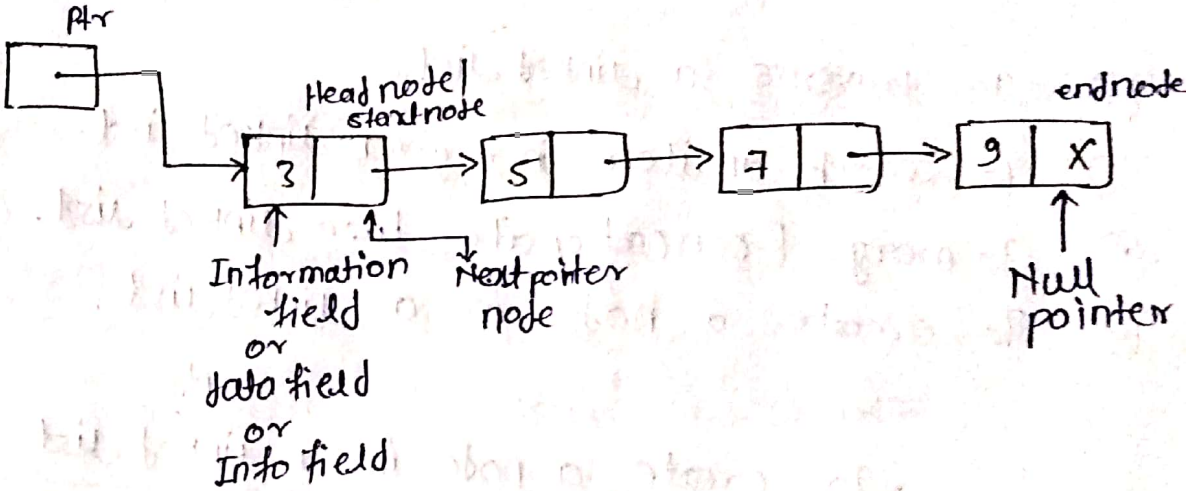
ex:-



Initially

if(ptr == Null) ,then

there is empty list.



* Linked list overcome the limitation of an array where an array size is fixed while linkedlist increased or decreased during execution (or run time).

Types of linked list :- Basically there are following types of linked list.

1. Singly linked list (One-way linked list)
2. Doubly linked list (Two-way linked list)
3. circular linked list (circular-way).

Scanned with CamScanner

① **Singly linked list :-** singly linked list is type of linked list in which contains nodes which have a data parts as well as an address part.

Operations of singly linked list :-

① To Insert an item In a list
- At Beginning
- At end
- After a given loc / node.

② To delete an item from a list.
- from beginning
- from end
- from a given loc.

③ To traverse a linked list.

④ To search an item in a given linked list.

⑤ To merge / concatenate two linked list.

⑥ To create a node in a linked list.

:- To create a node in a linked list or empty :-

Begin

step ① if (ptr == Null) then

step ② write list is empty.

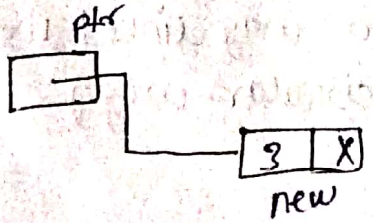
step ③ new = (node *) malloc (size of(node))

step ④ new → info = item

step ⑤ new → next = Null

end

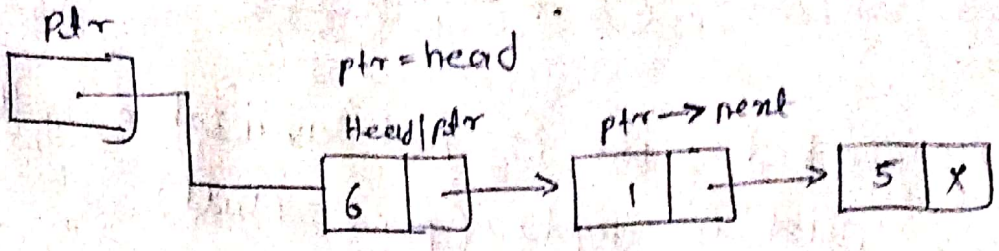
ex:-



Scanned with CamScanner

Traverse a linked list :-

ex:-



(1) while (ptr != Null)

(2) set ptr = ptr → next.

set ptr → info

How to represent structure of a node in a linked list:

```
typedef struct node {
        data_type info;
        struct node* next;
};
```

ex:-

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

Scanned with CamScanner

Insertion of an element in a linked list or singly linked list.

① When linked list is not available, then show list is empty. (ptr == Null)

② Before insertion first create a node.

new = (node *) malloc (sizeof (node))

③ new → info = item

new → next = Null

set head = new

Algorithm :- to insert at beginning of list.

① if (ptr == NULL), then
write list is empty.

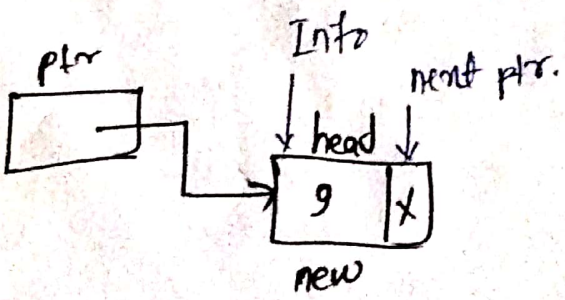
② set new = (node*) malloc (sizeof (node))

③ then insert item at new node

set new → info = item

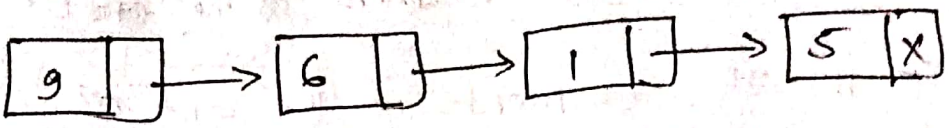
④ here inserted node is a single node then
set new → next = Null
set head = new

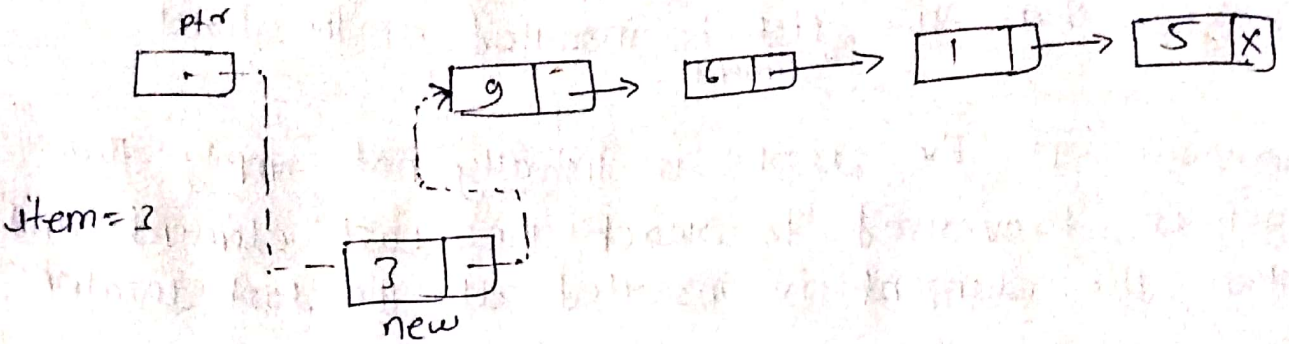
ex:- item = 9



Scanned with CamScanner

ex:





new → next = head

set head = new.

Begin

① set new = (node*) malloc (size of (node))

set new → info = item.

② if (ptr == NULL), ~~item~~

write list is empty

then set new → next = Null

③ else

set new → next = head

set head = new.

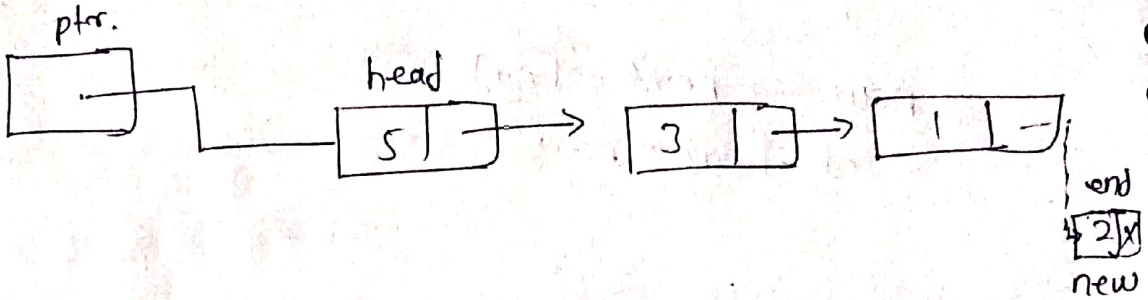
Scanned with CamScanner

Insertion of an element at the end of the list:-

To insert element at the end of list, first we check (test) whether the given list is empty or not. if yes, then the list is inserted as the first element.

however if the list is initially not empty then list is traversed to reach the last element, and then the element is inserted at the last location

ex:-



Algorithm

Begin

(1) set new = (node *) malloc(sizeof(node))
set new → info = item

(2) set new → next = Null || become last node.

(3) if (ptr == Null)
set head = new.

(4) set ~~end = new~~ loc = head

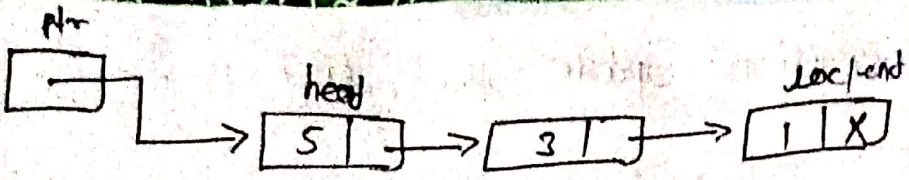
(5) while (end → next != Null)
loc = loc → next.

(6) set loc → next = new
set end = new.

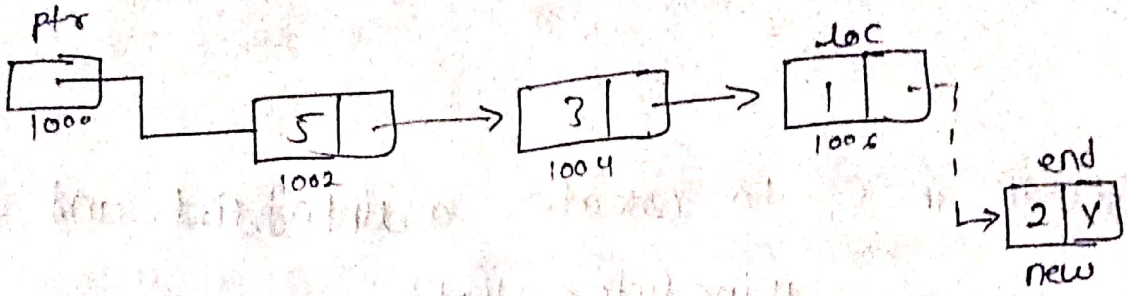
end

Scanned with CamScanner

ex:



Sol



Insertion of an item/node at specified location in a given list. :-

assume → a node. we have to create.

Algorithm :-

To insert at location ( )

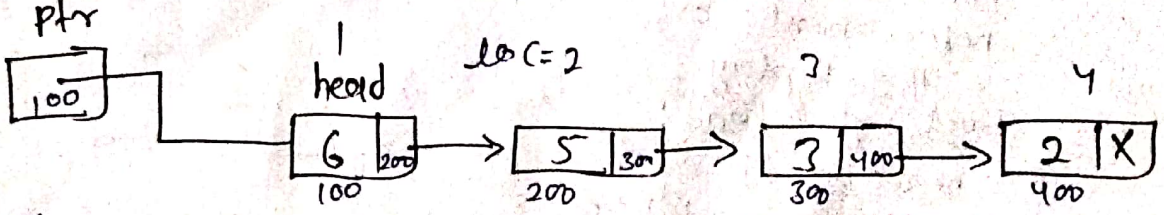
$i = 1$, $loc = 2$

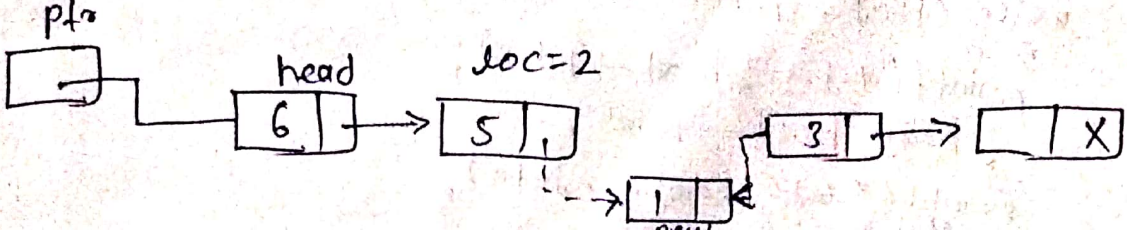
① while ($i < loc$)
    i++

② set loc = head
    loc = loc → next || for traversing.

③ set new → next = loc → next.
    set loc → next = new.

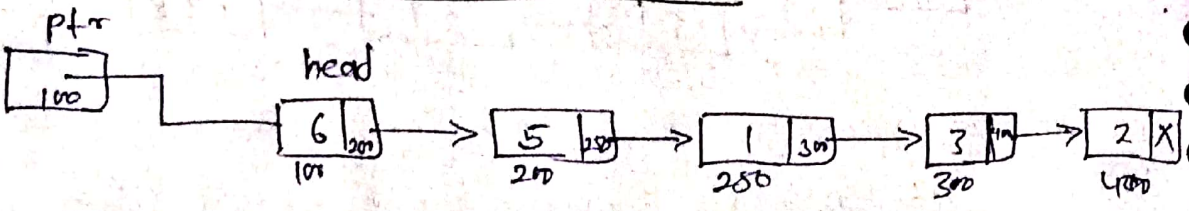
ex:





Scanned with CamScanner

After insertion update the list as —



① WAP in C to create a linked list and display it.

```
#include<stdio.h>
#include<stdlib.h>
struct node
{
 int data;
struct node* next;
};
int main()
{
struct node *node1, *node2, *node3;
//create space for node
node1 = (struct node*) malloc(sizeof(struct node));
node2 = (struct node*) malloc(sizeof(struct node));
node3 = (struct node*) malloc(sizeof(struct node));
 printf("enter the data of linked list = ");
scanf("%d %d %d", &node1->data, &node2->data, &node3->data);
   //Link the node
node1->next = node2;
node2->next = node3;
node3->next = 0;
   // To display
struct node * head;
   head = node1;
while (head->next != 0)
{
printf("%d->", head->data);
   head = head->next;
}printf("%d", head->data);
return 0;
```

Scanned with CamScanner

* To delete an item/node from a given list :-

(1) Delete an item/node from beginning of list.
(2) Delete an item/node from end of list.
(3) Delete an item/node from given location of list.

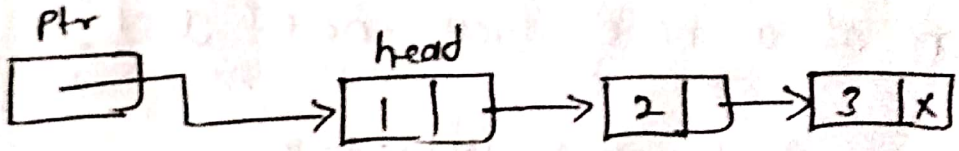
To delete an item from the list, first set the pointer value properly and then the memory occupied by the node to be deleted is deallocate (free)

(1) Deletion from Beginning of list:-

* An element from the beginning of list can be deleted by following the steps:-

(1) Assign the value of head (address of the first element of list). to temporally variable.

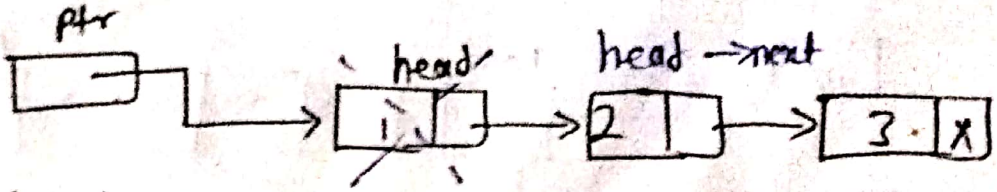
ex:-



set temp = head

(2) Assign the value of next pointer field of the first node to head

ex:-

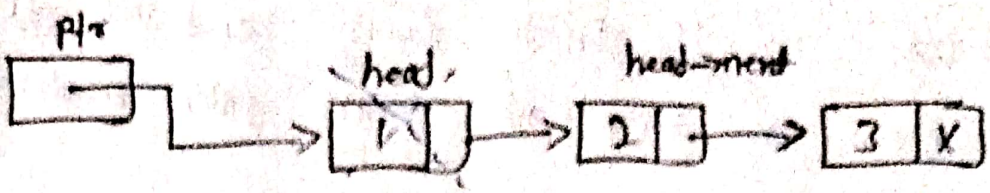


set head = head → next.

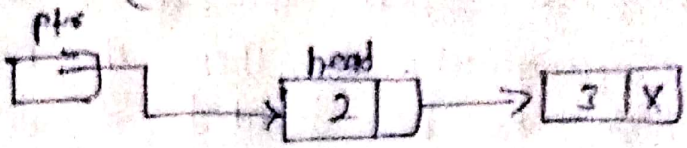
Scanned with CamScanner

(3) deallocate the memory occupied by the deleted node

ex:-



free (temp)



Algorithm

Begin

(1) while (ptr != NULL)

(2) set temp = head

(3) set head = head → next

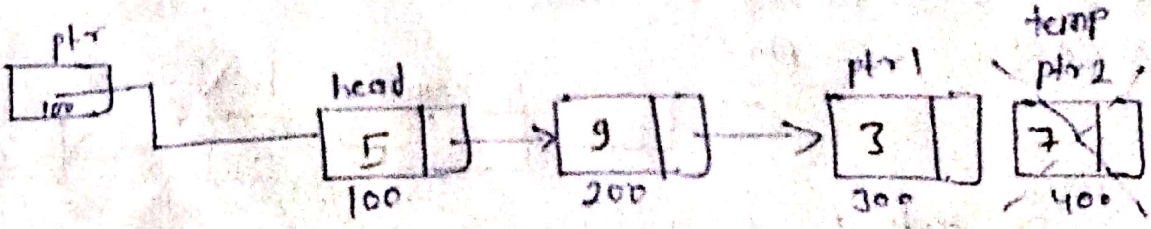
(4) free (temp)

end

Deletion of a node from end of list:-

```
struct node *ptr1, *ptr2
temp = ptr2
while (temp != NULL)
```

ex:-



temp = head

Scanned with CamScanner

Algorithm

Dele_end ( )

struct node *head, *ptr1, *ptr2, *temp;

Begin

① if (head == NULL), then write list is empty and exit.

② else if (head → next == NULL), then

set free (head)

set head = NULL.

③ else

temp = head // ptr2 = temp

While (temp → Next != NULL)

then set ptr1 = ptr2 or ptr1 = temp

set temp = temp → next.

end while.

④ set ptr1 → next = Null

set free (temp)

end.

Scanned with CamScanner

Deletion of an item/node after a given item/node (given location) :-

Algorithm :-

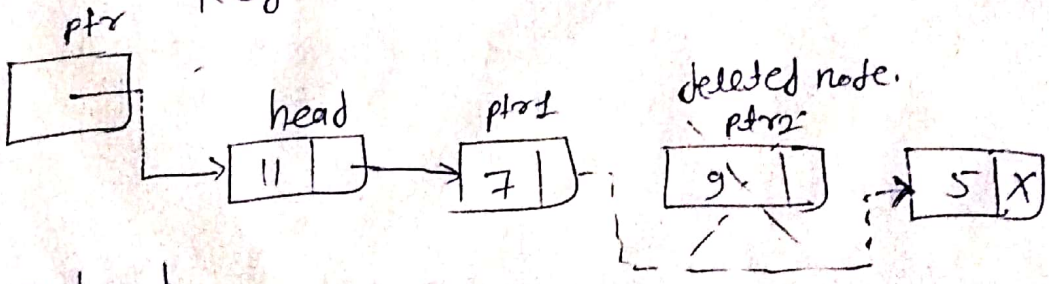
```
Deletion_ After ( )
Struct *head, *ptr1, *ptr2, *temp;

Begin
(1)   ptr1 = head
(2)   while (ptr1 -> next != Null)
      {
(3)       if ( ptr1 -> data == key)
          {
              set ptr2 = ptr1 -> next // after deletion
          set ptr1 -> next = ptr2 -> next

              free ( ptr2)
          }
      end if.
(4)   set   ptr1 = ptr1 -> next.
      }end while

end.
```

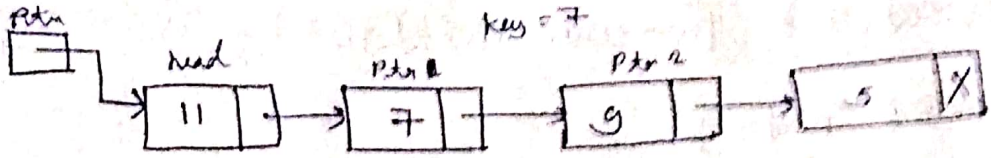
ex:-




```
temp = head
  while ( temp -> next != Null)
        temp = temp -> next.
          if (temp -> data == key)
```

Scanned with CamScanner

Ex

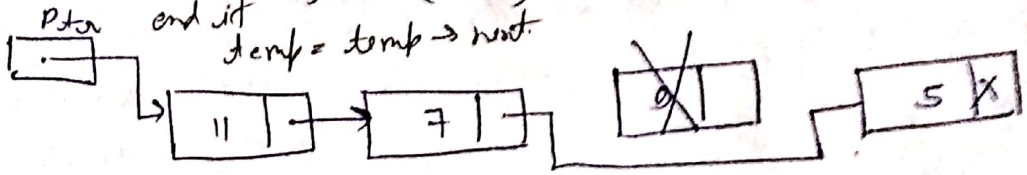



```
temp = head
while(temp -> next != NULL)
if (temp -> data == key)
   temp = temp -> next
```

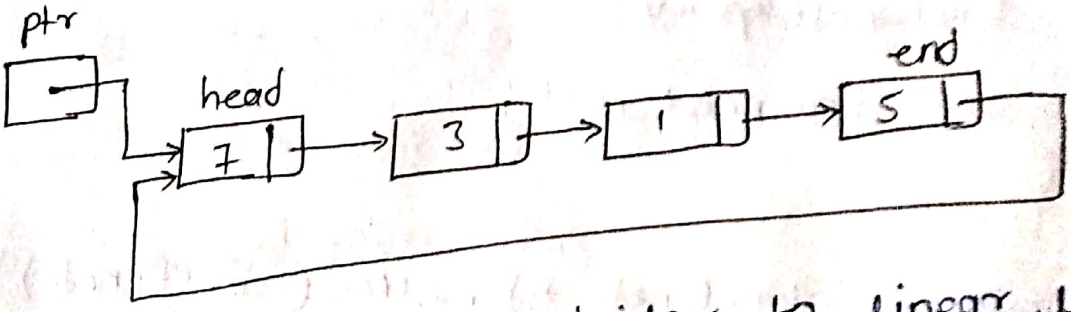
Ptr 2 = temp

Ptr 1 -> next = Ptr 2 -> next

free (Ptr 2)



Circular linked list :-



A circular linked list similar to linear linked list except that the last element or node next pointer fie is not a null pointer, last nod next pointer field store the address of head node (1st node).

* For non-empty circular linked list there is no null point

Scanned with CamScanner

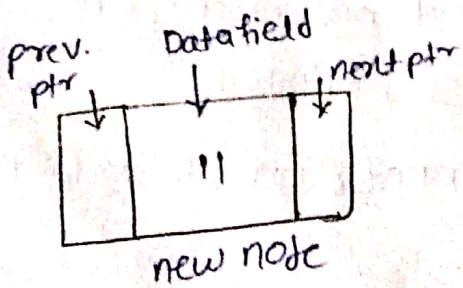
Doubly linked list (Two way linked list) :-

```
struct node {
        data-type data;
        struct node * prev;
        struct node * next;
};
```

```
new = (struct node *) malloc(size of(node));
```

ex:-

```
struct newnode {
        int data;
        struct node *prev;
        struct new node * next;
};
newnode = (struct node *) malloc (sizeof(node));
```

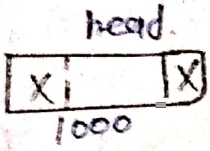


new node —> data = item

new node —> data = 11

Scanned with CamScanner

If there is single node:-



* prev ptr and next ptr will be Null

In a doubly linked list a node consist three field. 1st one is previous pointer and 2nd one data field and 3rd next pointer field.

* When prev. ptr does not store null value then it contained (store) the address of preceding node of current node.

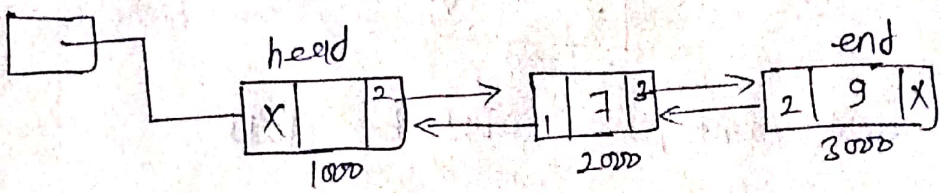
---

* header node :- It is special type of node which contain vital information of all the node.

```
if (head == Null), then
    list is empty
exit.
else if (head != NULL)
        head -> prev. = Null and
        head -> next = Null.
else    while (head -> next ! Null)
forward -> head = head -> next.
backword  head = head -> prev.
```

Scanned with CamScanner

ex:-



* Two way linked list beneficial over singly linked linked because doubly linked can be traversed in in both direction.

  i.e backward direction and forward direction.

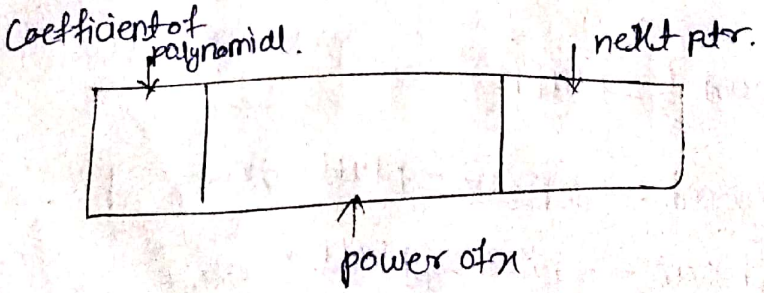
Applications of linked list :-

① To implement other data structure such as stack, Queue, tree and Graph by using linked list.

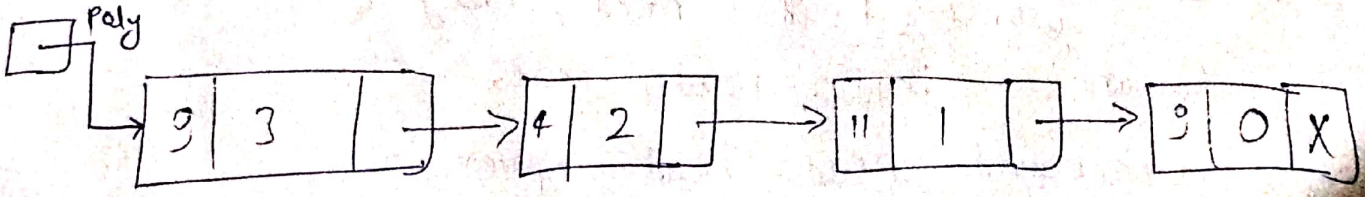
② To maintain directory ↓ by using linked list.
dynamically

③ To manipulate and to represent polynomials equation by using linked list.

ex: $9x^3 + 4x^2 + 11x + 9$



* Structure of node to represent a polynomial term.



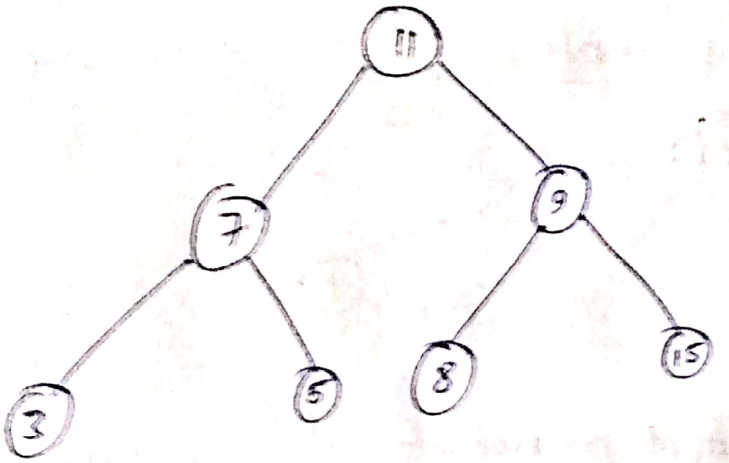
Scanned with CamScanner

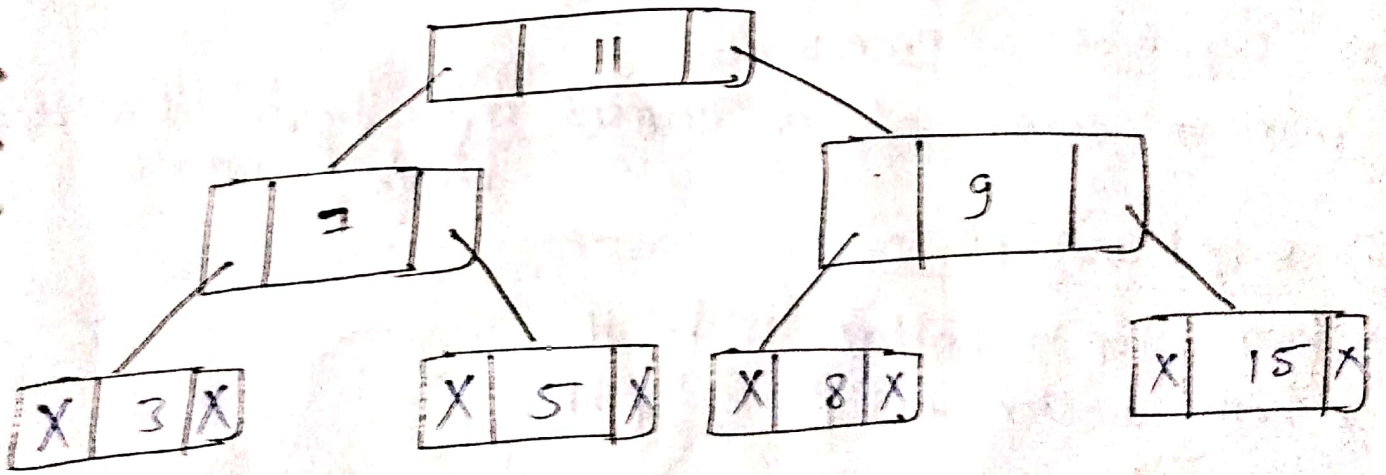
Structure

```
struct poly {
    int coeff;
    int power;
    struct node * next;
};
```

(4) To represent a binary tree by using linked list.

eg:-





Scanned with CamScanner

Trees

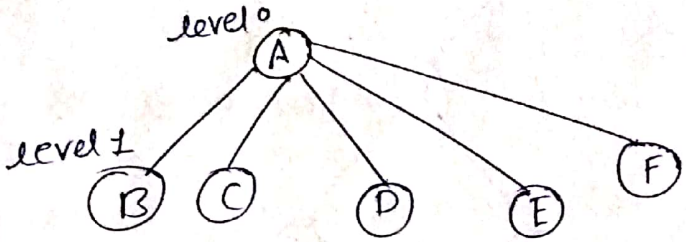
It is non primitive data structure and hierarchical representation.

* It is a collection of nodes and edges.

A tree (T) is a finite non-empty set of elements.

Trees Terminologies :-

(1) level of an element or node :-



(2) Degree of an element or node. —
Number of children it has.

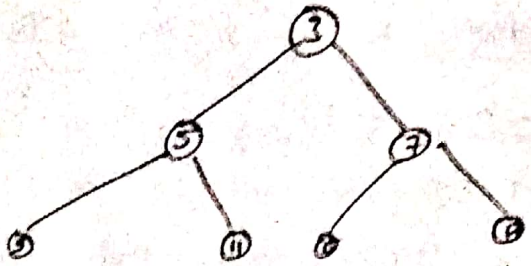
(3) Degree of a Tree :-
maximum degree of an element is the degree of a Tree.
become.

(4) Height and depth :-
* from top to bottom is height.
* from bottom to top is depth.

Binary Tree :- A binary tree (T) is either empty or has a finite collection of elements.

ex:- 3, 5, 7, 9, 11, 15, 17

Scanned with CamScanner

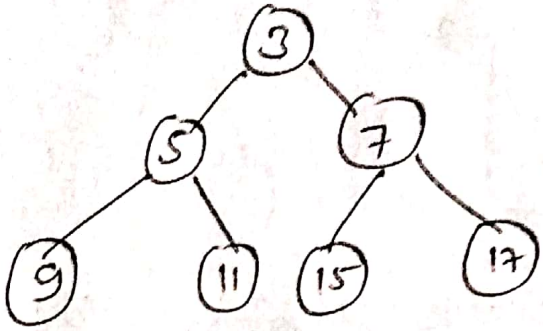


Full Binary Tree

properties of Binary Tree :-

(1) A binary tree with (n) elements where $n>0$, has exactly (n-1) edges

ex:-



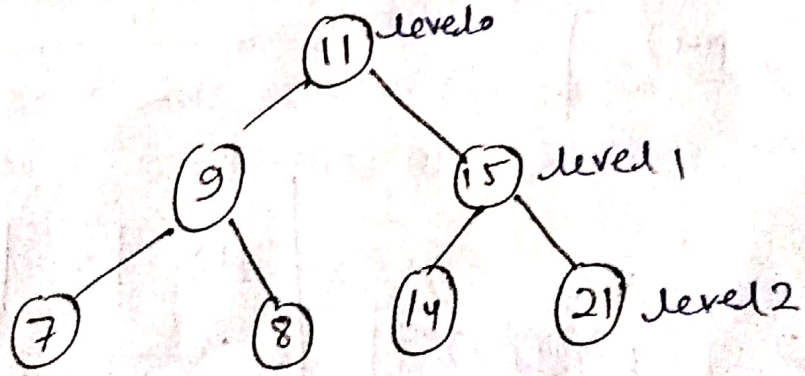
where $n>0$

node (n) = 7

edges = (n-1) = 6

(2) A binary of height H, where $H \geq 0$, has at least H node and atmost $2^{H+1}-1$ elements in it.

ex:-



$= 2^{H+1} - 1$

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

No. of node. $= 2^3 - 1 = 7$

Scanned with CamScanner

for a single particular node:-

$2^4 = 2^0 = 1$

$2^4 = 2^1 = 2$

$2^4 = 2^2 = 4$

Sum of all ~~node~~-level node $= 2^0 + 2^1 + 2^2 = 7$

mathematically :-

$$\text{Total No. of node} = \sum_{i=0}^{n-1} 2^{i+1} - 1$$

$$\text{Total level of nodes} = \sum_{i=1}^{n} 2^i - 1$$

(3) The height of a binary that contains (n) elements where $n \geq 0$, is at atmost an element and atleast

$$H \geq 0 \left\lceil \log_2^{(n+1)} \right\rceil$$

Ex:- $n = 7$

$$H \geq \left\lceil \log_2 (n+1) \right\rceil$$

$$H \geq \left\lceil \log_2 8 \right\rceil$$

$$H \geq \left\lceil \log_2 2^3 \right\rceil$$

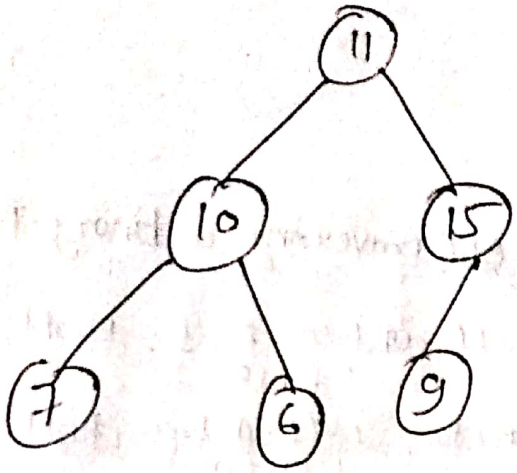
$$H \geq \lceil 3 \rceil$$

$$H \geq 3$$ Ans

Scanned with CamScanner

* Complete binary Tree — A complete binary tree has maximum no. of possible nodes at all levels except the last level, and all the nodes of last level appear as far as left possible.

ex:-

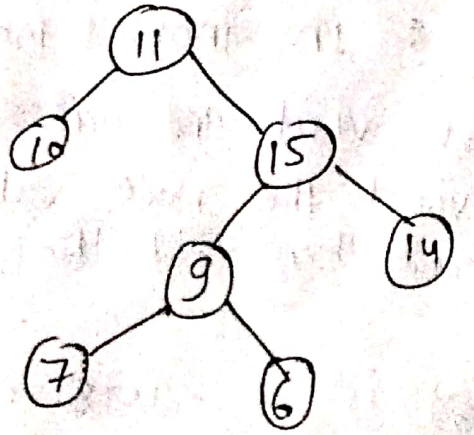


it is complete binary tree

* Those node in which all levell are same and found at last is called leaf node or Terminal node..

Strictly Binary Tree :- If every non leaf node in a binary tree has non empty left and right subtree, the tree is termed as a strictly binary tree.

ex:-



* leafs are not at same level.

Scanned with CamScanner

* A strictly binary tree with (n) leaves always contains (2·n−1) ~~(n−1)~~ node.

total leafs = 4 = n

$2 \cdot 4 - 1$

$= 8 - 1$

Node = 7

Traversal a Binary Tree OR (Traversing a binary Tree)

① In-order Traversal → Left_subtree, Root, Right_subtree (L,N,R)

② Pre-order Traversal → Node, left sub_tree, Right_subtree (N, L, R)

③ Post-order Traversal. → Left subtree, Right subtree, Node (L, R, N)

To traverse a binary tree defined by three ways.

① In order Traversal

② Pre order Traversal

③ Post order traversal.

① In order Traversal &- In inorder traversal method of binary tree. first visit and access left_subtree elements, then visit the node value of left subtree, visit the root node, then visit the right subtree and its node value.

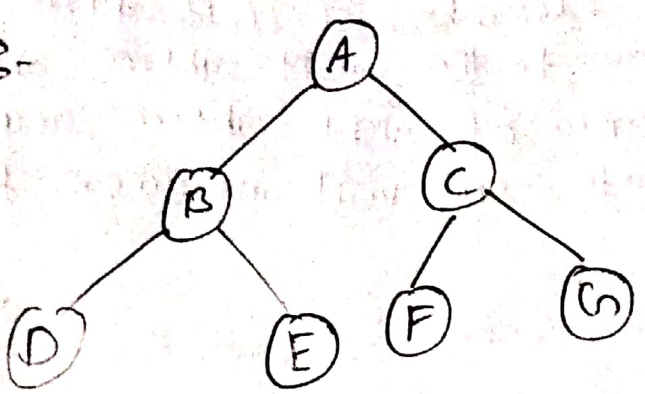
* Traversal method is a recursive process.

x. It is also known as symmetric order.

Scanned with CamScanner

In order Traversing (L, N, R)

ex:-

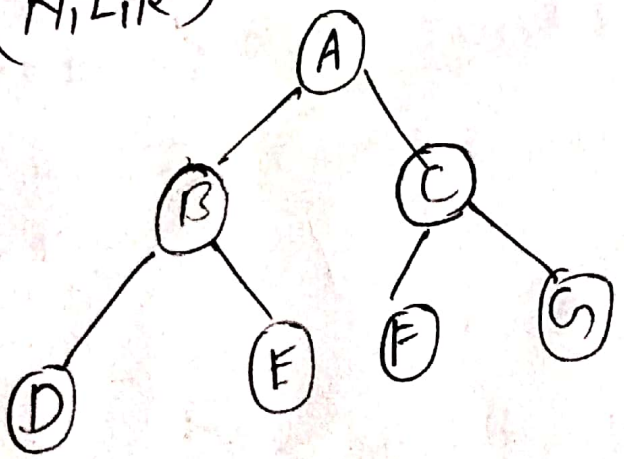


In order := D, B, E, A, F, C, G

(2) Pre order (Depth first order) :- In preorder

traversal first traversed the value of root node then come to left_subtree and traversed left subtree in preorder recursively. after traversing left_subtree then traversed right subtree in pre order recursively.

ex: (N, L, R)

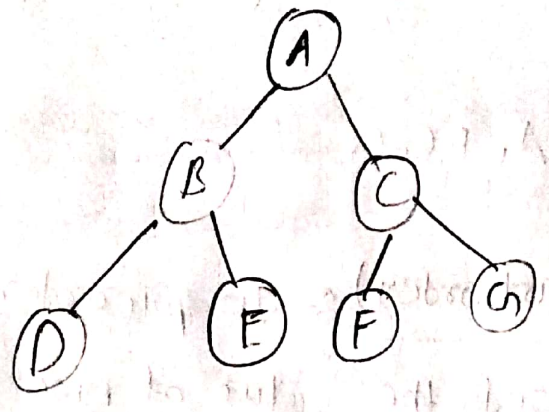


Pre order = A, B, D, E, C, F, G

Scanned with CamScanner

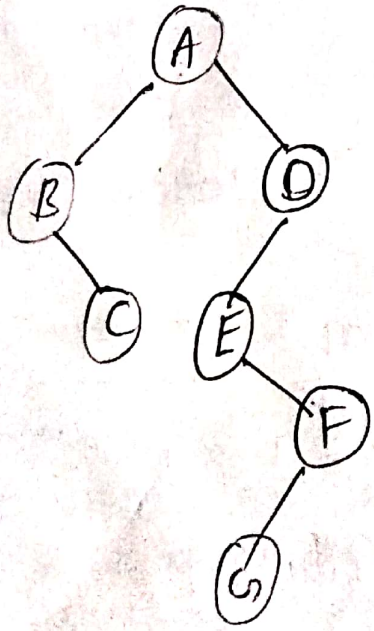
(3) Post order Traversal :- In post order traversal first visit and access the elements of left subtree recursively in post order then come to the right subtree and visit and access the elements of right subtree recursively. then come to root node and visit and access the value of root node.

exo- (L, R, N)



postorder = D, E, B, F, G, C, A

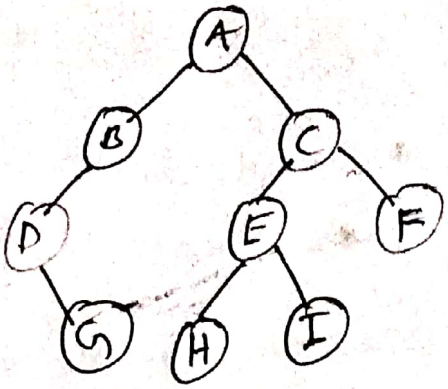
(1)



Inorder = B, C, A, E, G, F, D
preorder = A, B, C, D, E, F, G
postorder = C, B, G, F, E, D, A

Scanned with CamScanner

Q(1)

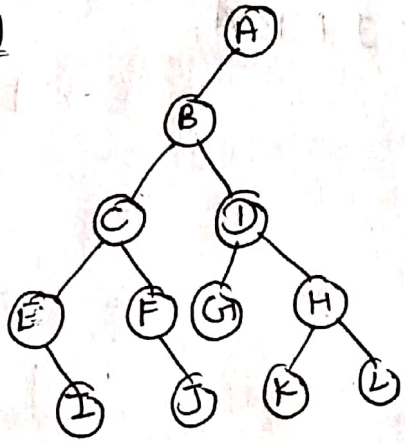


Inorder = D, G, B, A, H, E, I, C, F

Preorder = A, B, D, G, C, E, H, I F

Postorder = D, G, B, B, H, I E, F, C A

Q(2)



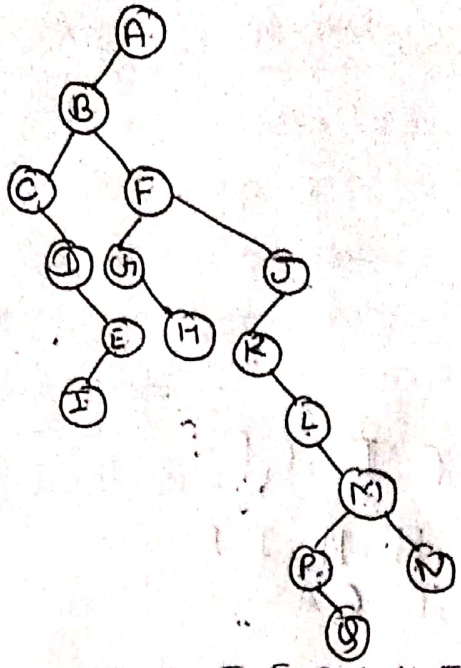
Inorder = E, I C F J B G D K H L A

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

Postorder = I E J F C G K L H D B A

Scanned with CamScanner

Q(5)

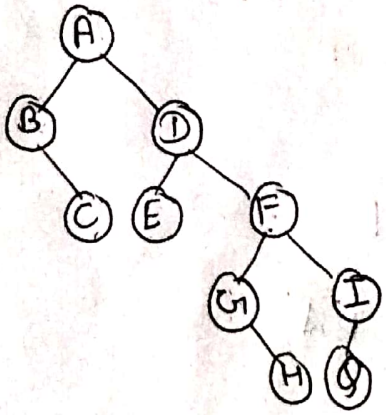


Inorder = I, D, I E B G H F K L P Q M N J A

Preorder = A B C D E I F G H J K K M P Q Q

postorder = I, E, D, C H G Q P N M L K J F B A

Q(6)



Inorder :- B C A E D G H F Q I

Pre order :- A B C D E F G H I Q

Post order :- C B E H G Q I F D A

Scanned with CamScanner

Algorithm of Inorder :-

1. Recursively traverse left subtree.
2. Visit root node.
3. Recursively traverse Right subtree.

Pre order (Algorithm)

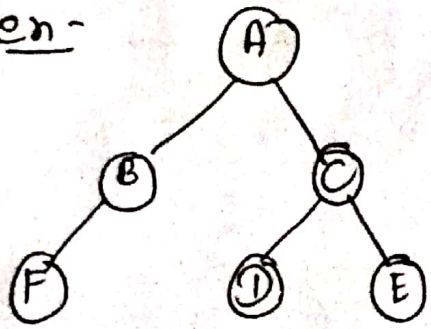
1. Visit root node
2. recursively traverse left subtree.
3. recursively traverse right subtree.

postorder (Algorithm)

1. Recursively traversed left subtree.
2. Recursively traversed right subtree.
3. Visit root node.

Creation of Binary Tree -

ex-



Inorder= F, B, A, D, C, E

Preorder- A, B, F, C, D, E

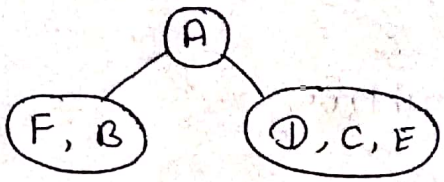
Postorder- F, B, D, E, C, A

Scanned with CamScanner

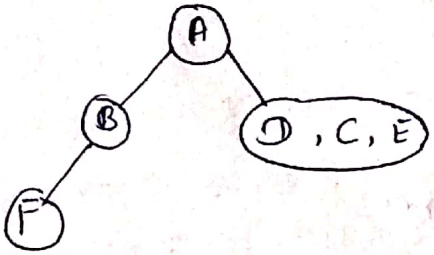
If Inorder & Preorder are given only-

1) Inorder = F, B, (A), D, C, E (Root)
Preorder = (A) B, F, C, D, E

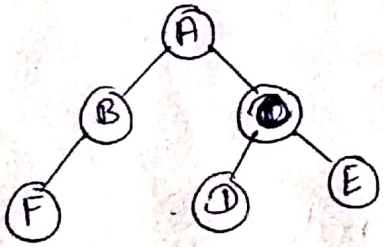
2)



3) Inorder = F, (B)
Preorder = (B), F



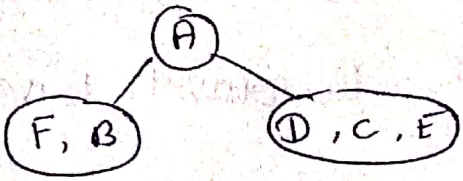
4) Inorder = D (C) E
Preorder = (C) D, E



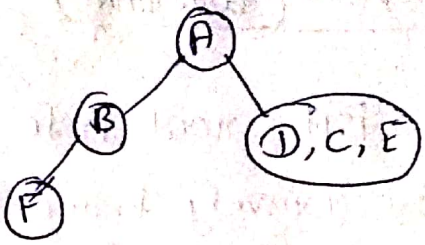
If Inorder & Postorder are given only-

1) Inorder = F, B, (A), D, C, E (Root)
Postorder = F, B, D, E, C (A)

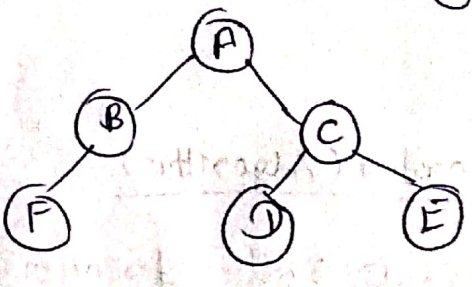
2)



3) Inorder = F, (B)
Postorder = F, (B)

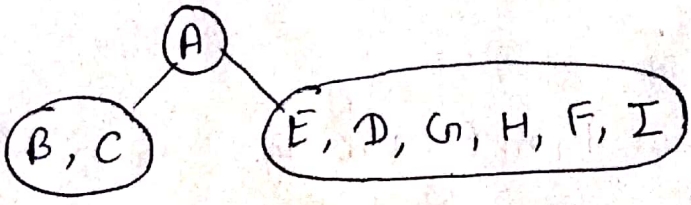


4) Inorder = D (C) E
Postorder = ~~E E~~ D, E, (C)

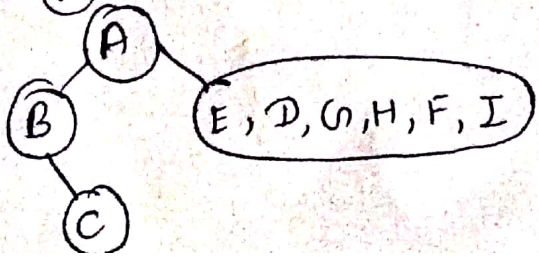


Q Preorder = (A) (root), B, C, D, E, F, G, H, I
Inorder = B, C, (A), E, D, G, H, F, I

2)

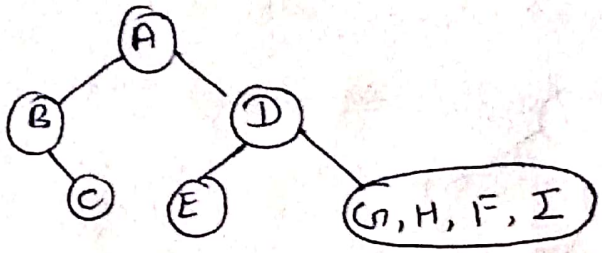


3) Inorder - (B), C
Preorder - (B), C

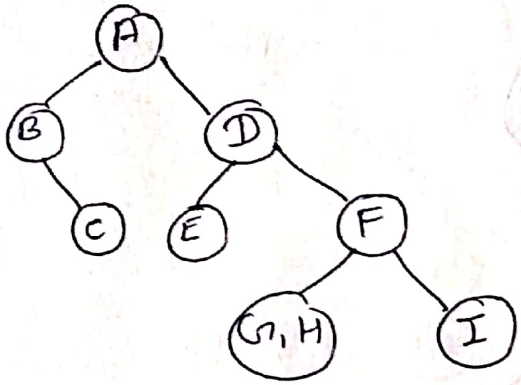


Scanned with CamScanner

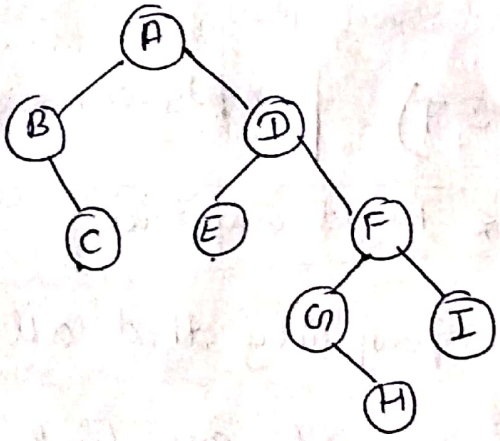
4) Inorder = E, D, G, H, F, I
Preorder = D, E, F, G, H, I



5) Inorder = G, H, F, I
Preorder = F, G, H, I

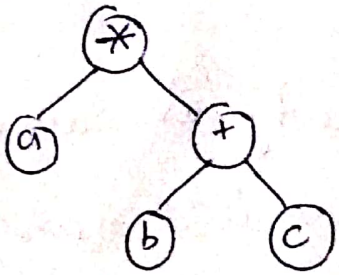


6) Inorder = G, H
Preorder = G, H



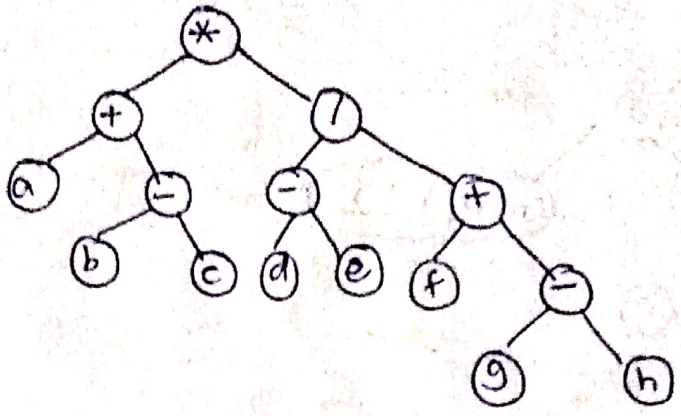
* Binary Tree for Algebri exp. representation

a * b + c

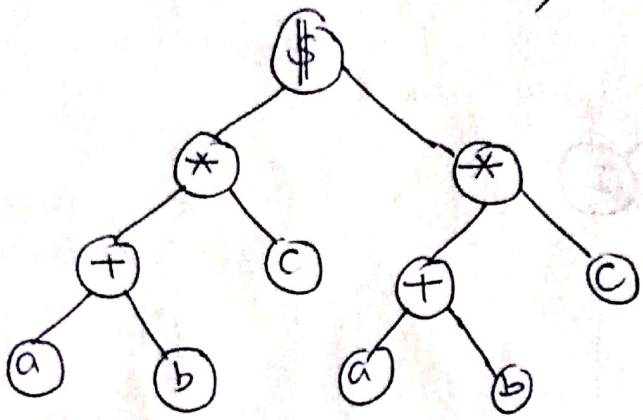


Scanned with CamScanner

Ex- $[a + (b-c)] * [(d-e)/(f+g-h)]$



Q $(a+b*c) \$ ((a+b)*c)$



Binary Search Tree (BST) [empty or non-empty]

Left subtree ≤ Root node ≤ Right subtree

A binary search tree has the property that all the elements in the left subtree of the node are less than or equal to the content of the root node, And all the elements in the right subtree are greater than to the content of root node.

A Binary tree that has these property is called a binary search tree. Binary search tree is traversed by

Scanned with CamScanner

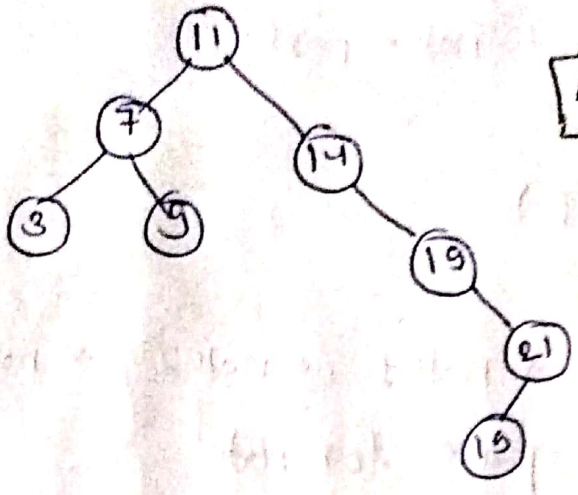
inorder sequence.

Operation on a binary search tree

There are many operations defined in binary search tree

1) Searching 2) Insertion 3) Deletion

Ex⇒ 11, 7, 9, 14, 19, 3, 21, 15 (Do insertion)



LS < Root ≤ RS

In this order

Deletion of an element from binary search tree

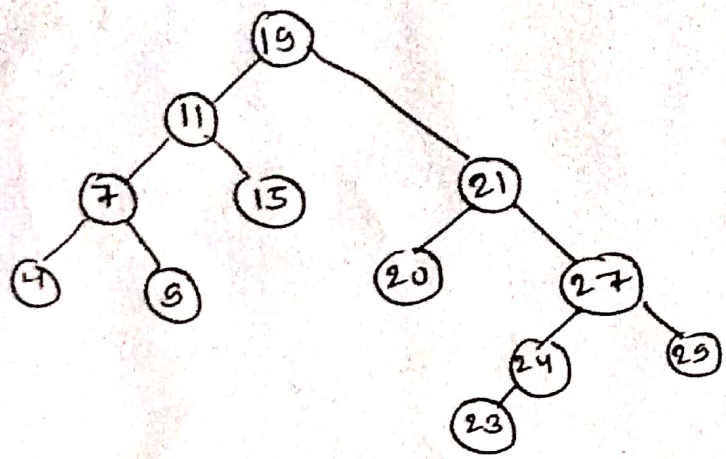
For deletion of any node/element from binary search tree, there are three cases - (1) deletion of leaf node, (2) deletion of a node having one child, (3) deletion of a node having two children.

Deletion of leaf node

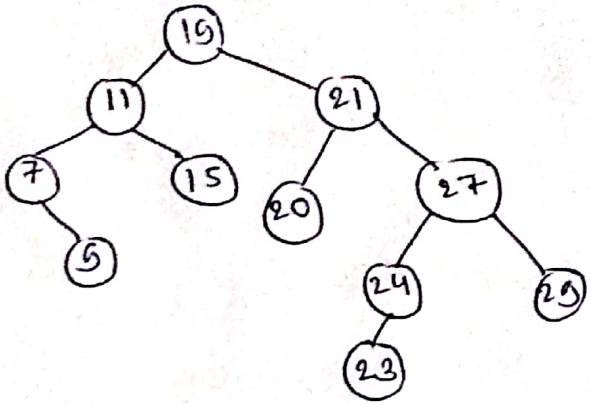
This is the simplest deletion in which, we set the left or right pointer of parent node as NULL.

Scanned with CamScanner

* delete node 4



Before deletion

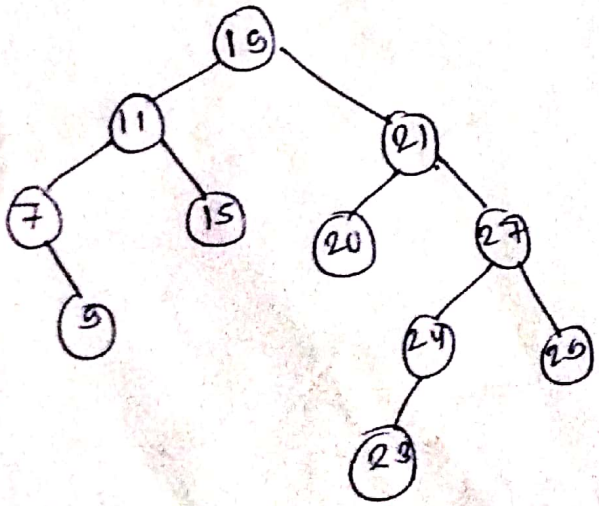


After deletion

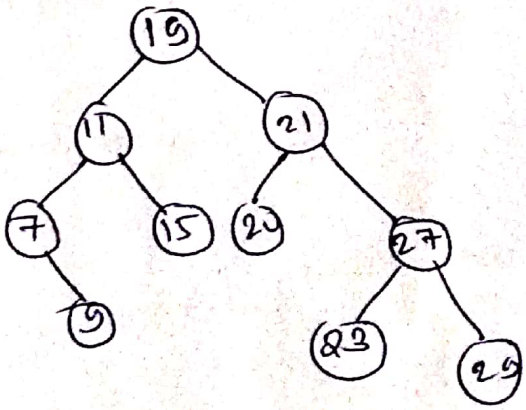
Deletion of a node having one child

Deletion of a node from BST, deleted node have one child node after deletion that node simply put the child node at its parent place.

ex - delete 24



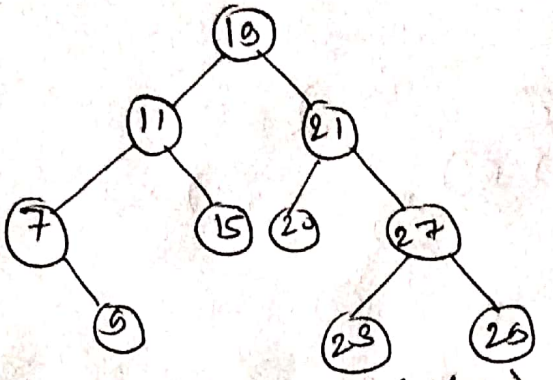
(Before)



(after)

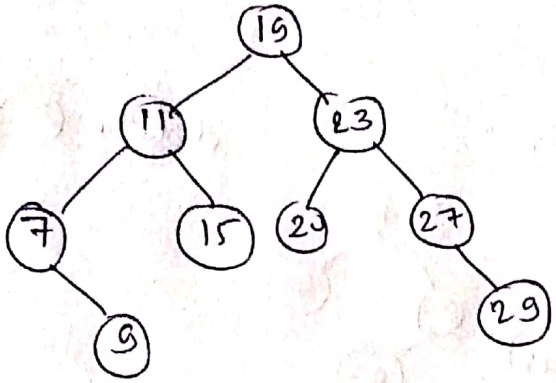
Scanned with CamScanner

Deletion of a node having two child

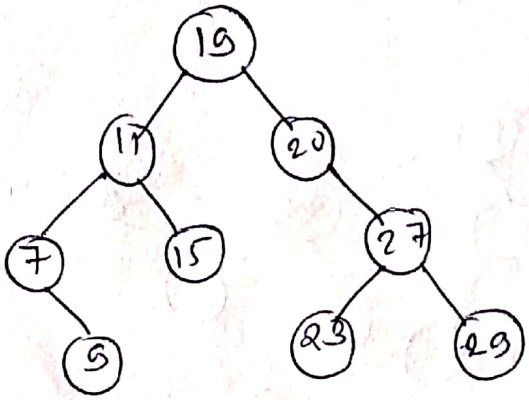


Inorder = 7, 5, 11, 15, 19, 20, 21, 23, 27, 29

(before)



(Case of select successor)



(Case of select predecesor)

(after)

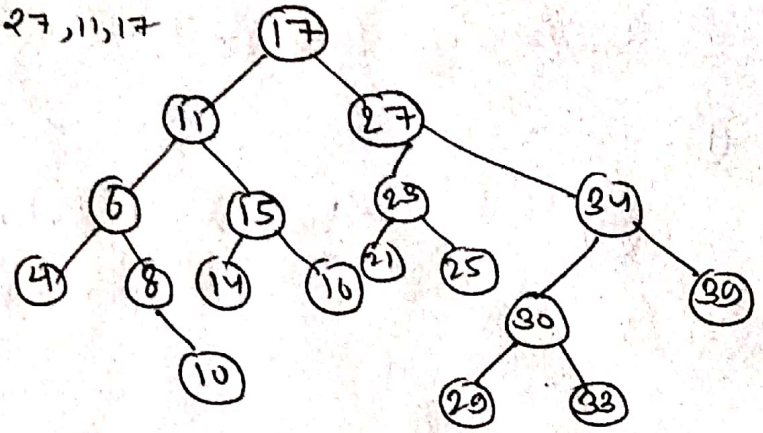
In deletion of a node having 2 child, find-out the inorder successors of delited node and put the successors node at delited place, then readjust the binary search tree

OR

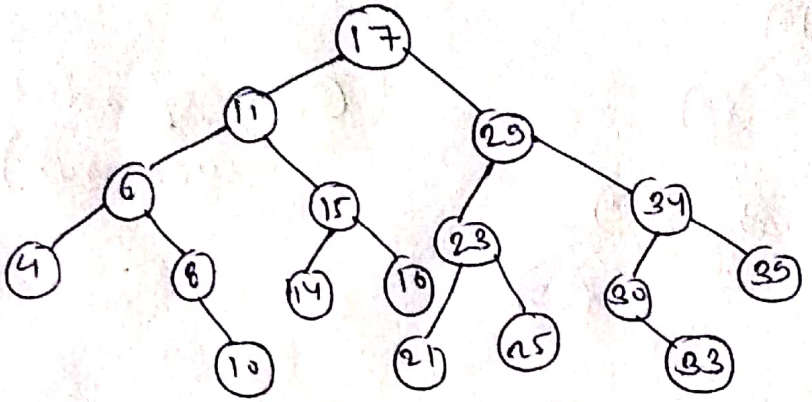
Find out the inorder predecessor of delited node & put it at the delited place, then readjust the BST

Scanned with CamScanner

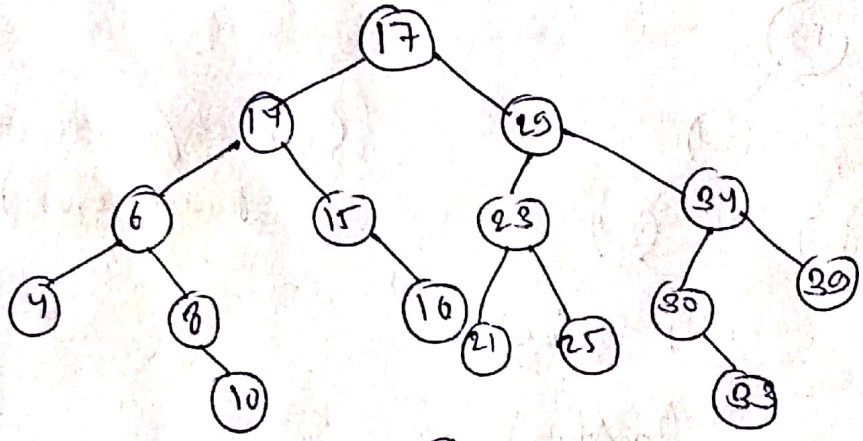
Que Delete 27, 11, 17



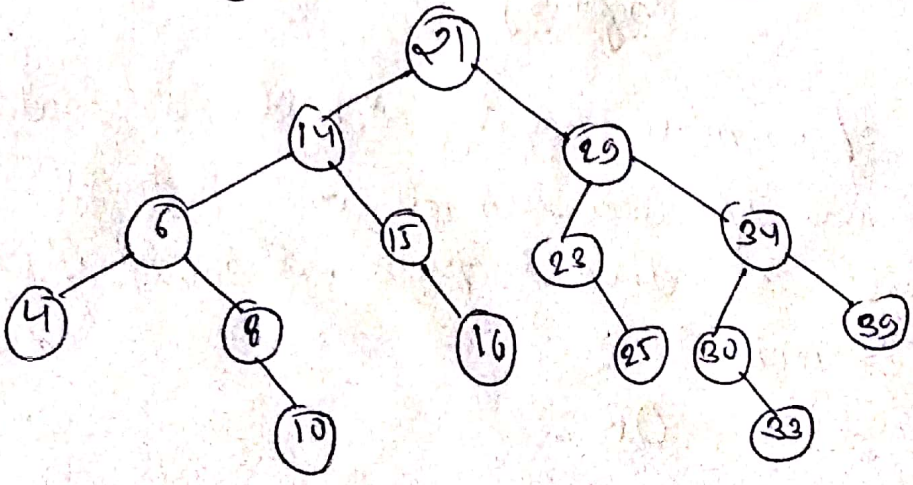
Inorder = 4, 6, 8, 10, ~~11~~, 14, 15, 16, ~~17~~, 21, 23, 25, ~~27~~, 29, 30, 33, 34, 39



delete 27



delete 11



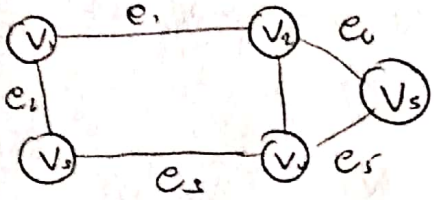
delete 17

Scanned with CamScanner

Graph

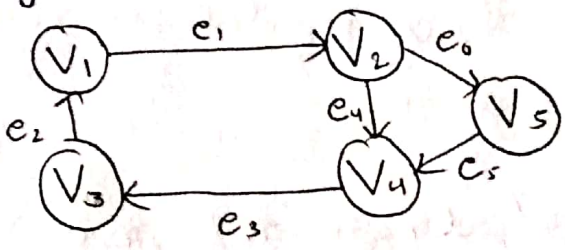
A graph is a ordered set of (V, E) where V is finite non-empty set of vertices and E is a finite non-empty set of edges. Any graph is denoted as $G = \{V, E\}$. A graph can be either directed or undirected. In an undirected graph there is no specific direction associated with edge.

Eg-



(Undirected graph)

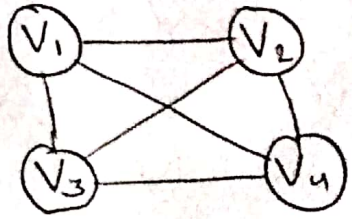
In the directed graph the direction are shown on the edges.



Graph terminologies -

1) Complete graph

follow the property of mesh topology

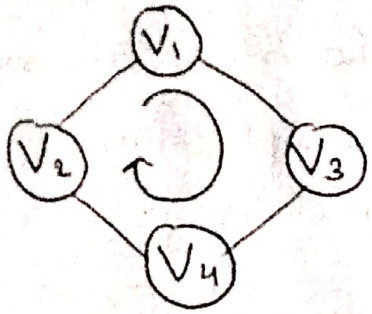


If an undirected graph of 'n' vertices consist of $\frac{n*(n-1)}{2}$ edges then that graph is called a complete graph.

Scanned with CamScanner

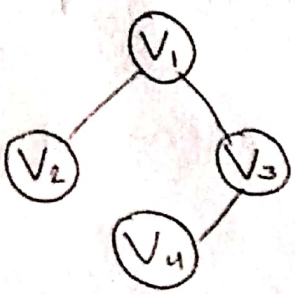
2) Connected graph - A graph is said to be connected if there is a path b/w any two of its vertices (there is no isolated vertex). A connected graph without any cycle is called a tree.

ex -



$V_1 - V_3$
$V_1 - V_3 - V_4$ } Path
$V_1 - V_2 - V_4$

$V_1 - V_3 - V_4 - V_2 - V_1$ (cycle)

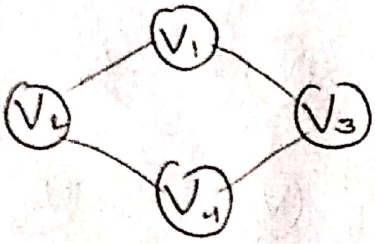


$V_1 - V_3$
$V_1 - V_3 - V_4$ } Path
$V_1 - V_2$

No cycle, this is also known as tree

3) Path - A Path 'P' of length n from a vertex u to vertex v is defined as

ex -



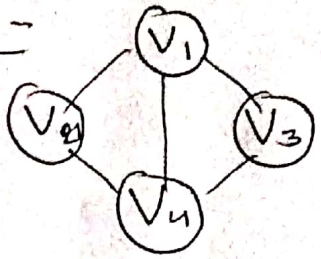
$V_1 - V_2 - V_4$
$V_1 - V_2$
$V_1 - V_3$
$V_1 - V_3 - V_4$
$V_1 - V_3 - V_4 - V_2$ } Path

If there is n vertices in a connected graph then, we can defined (n+1) number of paths

4) Degree of vertex - The degree of vertex u, is degree (u)

Scanned with CamScanner

ex-



degree (V_1) = 3
degree (V_2) = 2
degree (V_3) = 2
degree (V_4) = 3

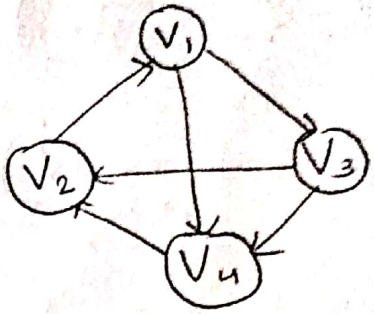
NOTE - Degree of a graph is a max. degree of any vertex in that graph

Degree of vertex is a no. of edges connected to that vertex.

If degree(u) = 0, it means it is an isolated vertex

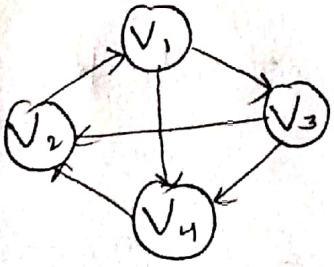
Indegree & Outdegree -

Vertex	indegree	Outdegree
V_1	01	02
V_2	02	01
V_3	01	02
V_4	02	01



5) **Cycle-** A closed walk to the graph with repeated vertices, mostly having the same starting and ending vertex, is called a cycle.

ex-



$V_1 - V_3 - V_4 - V_2 - V_1$
$V_1 - V_3 - V_2 - V_1$
$V_1 - V_4 - V_2 - V_1$
} cycle

Representation of Graph -

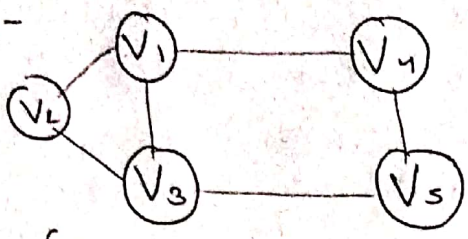
* 1) Adjacency matrix (Two Dimensional Array)
2) Adjacency list (By linked list)

A graph can be represented by these above method.

Scanned with CamScanner

Adjacency matrix representation –

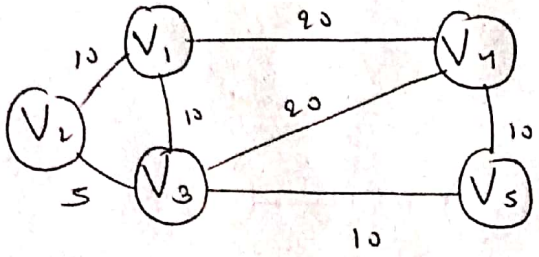
ex-



{Undirected & Unweighted}

	V_1	V_2	V_3	V_4	V_5
V_1	0	1	1	1	0
V_2	1	0	1	0	0
V_3	1	1	0	1	1
V_4	1	0	1	0	1
V_5	0	0	1	1	0

ex- weighted Undirected graph



	V_1	V_2	V_3	V_4	V_5
V_1	0	10	10	20	0
V_2	10	0	5	0	0
V_3	10	5	0	20	10
V_4	20	0	20	0	10
V_5	0	0	10	10	0

Graph traversal

Two Technique

(i) BFS (Breadth - First - Search) – Queue

(ii) DFS (Depth - First - Search) – Stack

BFS (Breadth - First)

It is a technique used for searching a vertex (visition) in a graph. It is also used to decide the order of vertics to be visited in the search process.

There are two traversal technique —

Scanned with CamScanner

BFS (Breadth First Search) Traversal

On BFS, for traversing the graph in this manner, we visit all the vertices on each level.

For example – Let a vertex V_1 in the graph will be visited first, then all the vertices adjacent to V_1 will be traversed & suppose adjacent vertices are $V_2, V_3, \ldots, V_n$ will be printed first. Then again from vertex V_2, find the adjacent vertices of V_2 and so on this process will be continued untill all vertices are covered. To keep track of all the vertices and their adjacent vertices we will make use of queue data structure.

Algo of BFS traversal -

BFS Traversal ( )

1. Define a queue of total size is no. of vertex of graph.
2. Select any vertex as starting point for traversal Visit that vertex and insert it into queue.
3. Visit all the adjacent vertices of the front vertex of the queue, which is not visited and insert it into queue.
4. When there is no new vertex to be visit from the vertex at front of the queue then delete that vertex from the queue.

Scanned with CamScanner

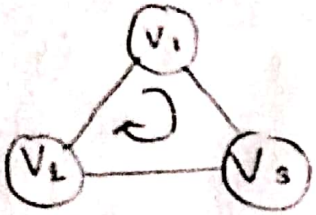
(5) Repeat step 3 & 4 until queue becomes empty.

(6) When queue becomes empty, then it produce a final spaning tree, ~~while~~ by removing unused edges from the graph.

NOTE

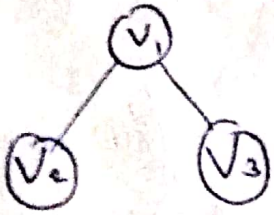
Spanning tree - A spanning tree is a sub-set of graph(G) which has all the vertices with minimum possible no. of edges, hence a spanning tree doesn't have any cycle. If there are "n" vertices in a graph, then there will "n^{n-2}" possible spanning tree of that graph

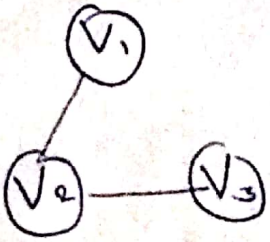
ex :-

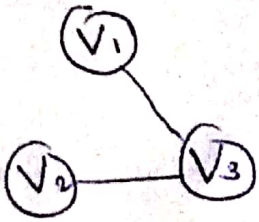


$n = 3$

no. of spanning tree $= 3^{3-2} = 3^1 = 3$

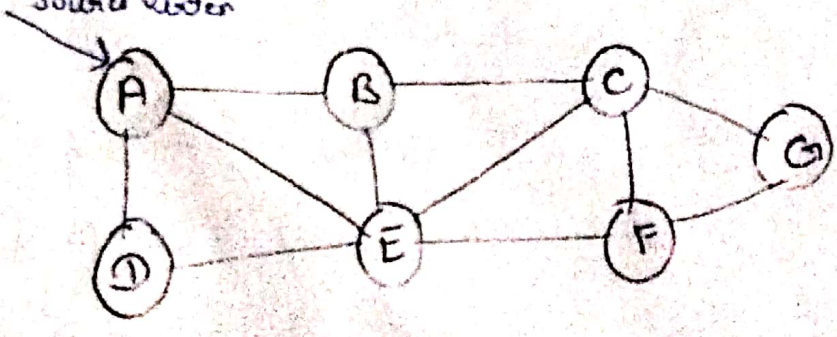






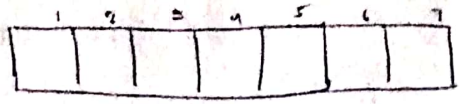
(3 spanning tree)

ex =

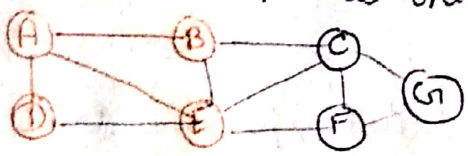


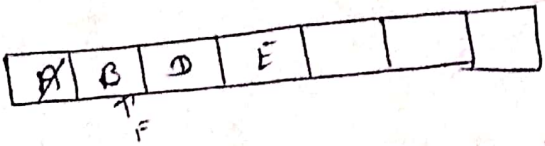
Scanned with CamScanner

Step ① Define a queue of size equal to total no. of vertices in a graph.

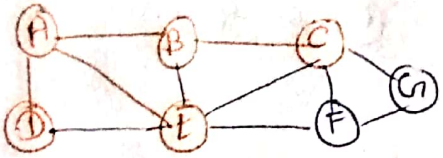


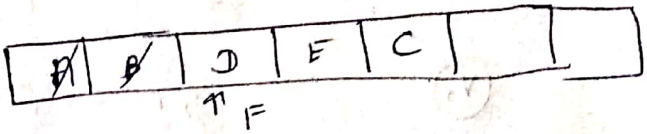
Step ② Select a starting point vertex A, find out all adjacent vertex A and put it onto queue, then remove front vertex A



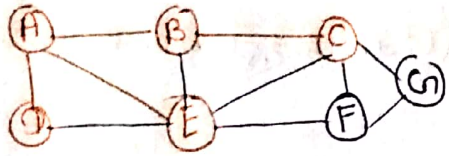


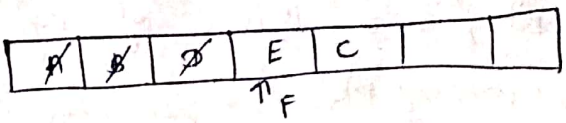
Step ③ Select front vertex B, find out all adjacent vertex of B and put it onto queue, then remove front vertex B



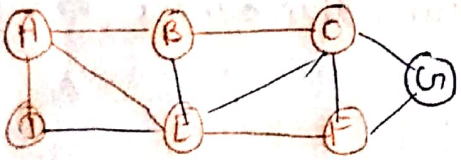


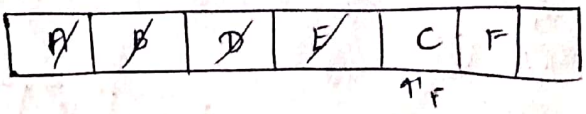
Step ④ Select front vertex D, find out all adjacent vertex of D and put it onto queue, then remove fromt vertex D



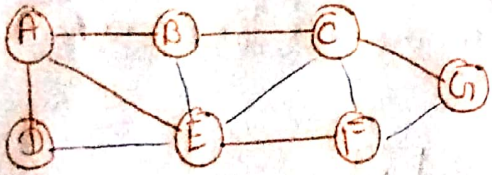


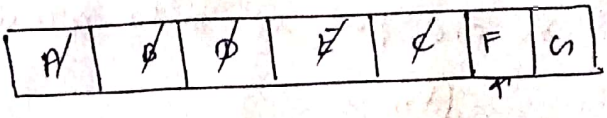
Step ⑤ Select front vertex E, find its adjacent vertices and put it onto queue, then remove front vertex E



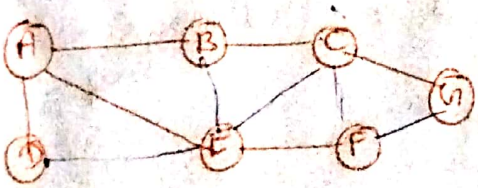


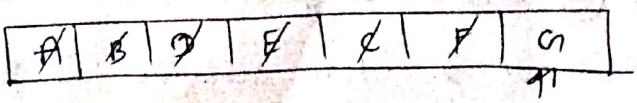
Step ⑤ Select front vertex C, find its all untraversed adjacent vertices and put it onto queue, then remove front vertex C





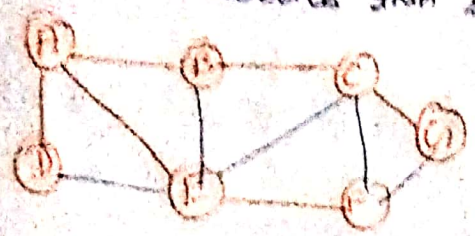
Step ⑥ Select front vertex F, but it has not any untraversed adjacent vertices then simply remove it from the queue

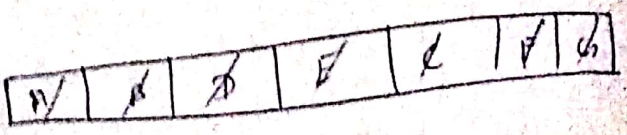




Scanned with CamScanner

Step ④ Select front vertex G, but it has not any untraversed adjacent vertices then simply remove it from the queue





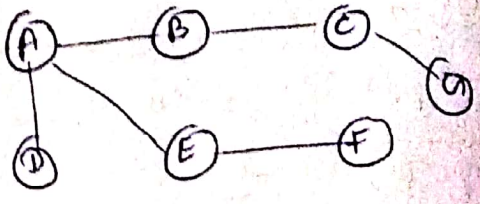
Point-

A, B, D, E, C, F, G

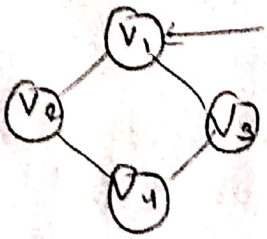
no. of spinning tree = 7^{7-2}

$= 7^5$

Obtained spinning tree

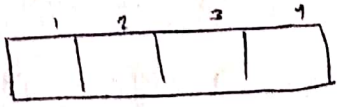


Ques -

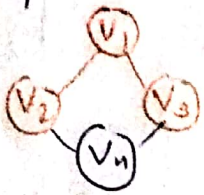


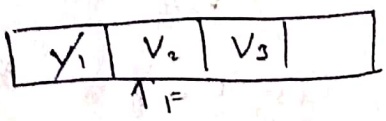
starting point (given)

Step ① Define a queue of size equal to total no. of vertices in a graph

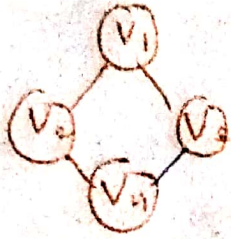


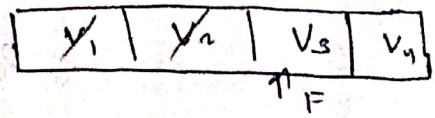
Step ② Select a starting point vertex V_1, find out all adjacent vertex V_1 and put it onto queue, then remove front vertex V_1





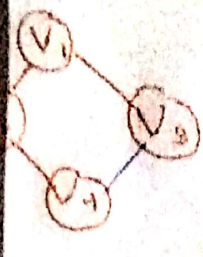
Step ③ Select a front vertex V_2, find out all adjacent vertex V_2 and put it onto queue, then remove front vertex





Step ④ Select a front vertex V_3, find out all adjacent vertex of V_3, then remove it from queue

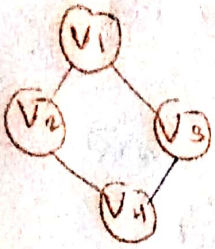
Scanned with CamScanner



V_1	V_2	V_3	V_4

F

Step (5) Select a front vertex V_3, then it no its untraversed, adjacent, then simply remove it from queue

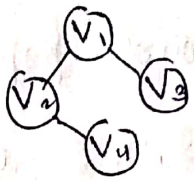


V_1	V_2	V_3	V_4

Point- V_1, V_2, V_3, V_4

no. of spinning tree = 4^2

Obtained spinning tree



Depth-First-Search (DFS) Traversal –

The strategy followed by DFS is as its name, to search "deeper" in the graph whenever possibe. In DFS, edges are explored out of the most recently discovered vertex, unexplored edges living it. When all of vertices have explored, the search "backtracks" to explore edges. This process continued until we visit all vertices of the graph. This entire process is repeated until all the vertices arediscovered. DFS traversal of a graph produces spanning tree as final result. Use stack datastructure with maximum size of total no. of vertices in the graph

To Implment DFS Traversal -

Algorithm

(1) Define a stack of size total no. of vertices in graph.

Scanned with CamScanner

(2) Select any vertex as starting point and visit that vertex, push it onto the stack.

(3) Visit any one of the adjacent vertex of the vertex which is at top of the stack, which is not visited and push it onto the stack.

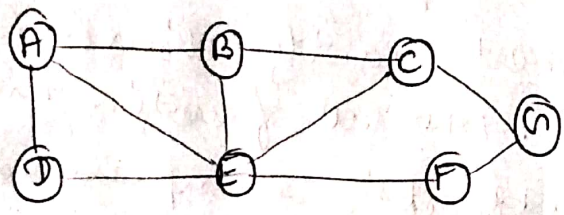
(4) Repeat step 3, until there path are no new vertex to be visit from the vertex on the top of the stack.

(5) When there is no new vertex to be visit, then use backtracking and pop one vertex from the stack

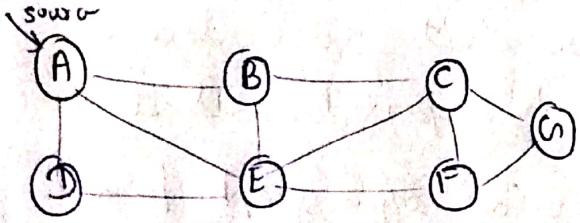
(6) Repeat step 3, 4, 5 until stack becomes empty.

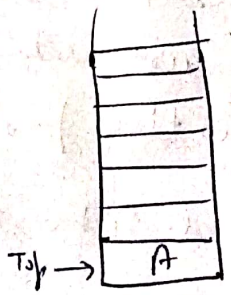
(7) When stack becomes empty, then it produce final spanning tree by removing unused edges from the graph

Ex -



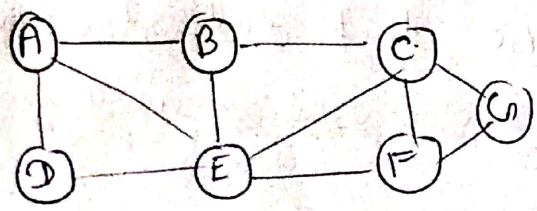
1)

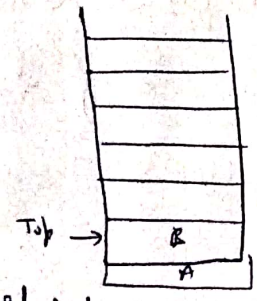




Select source & push it onto stack

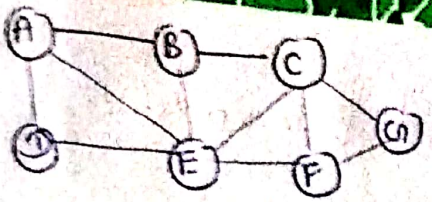
2)

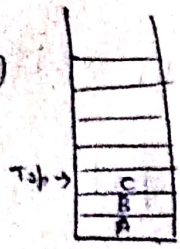




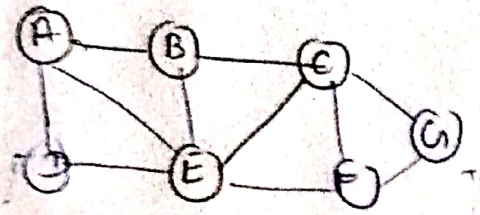
Put any adjacent of A which is not visited and push it onto stack
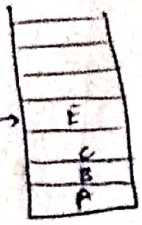
Scanned with CamScanner

4) 

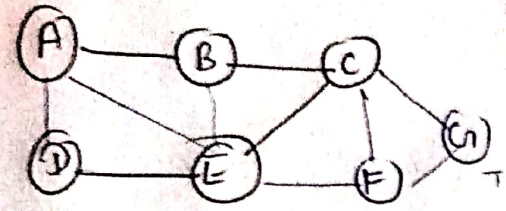


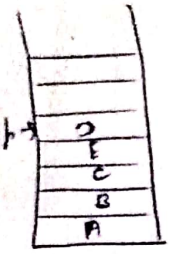
Visit adjacent of B, which is C, push it onto stack

4) 



Visit adjacent of C & select any unvisited adjacent and push it onto stack

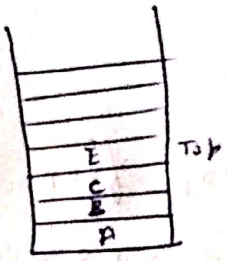
5) 



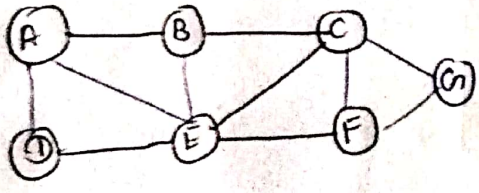
Select any unvisited adjacent of E & push it onto stack

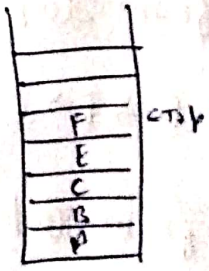
6) There is no new unvisited adjacent of vertex D, so we backtrack and pop from the top of the stack

Point = D →

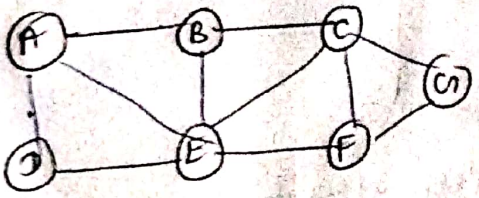


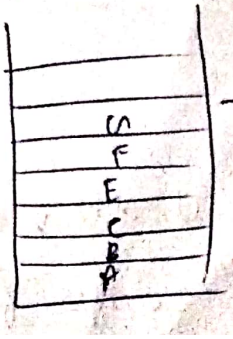
7) Select vertex E & visit any unvisited adjacent vertex which is F & push it onto the stack





8) Select unvisited ~~element~~ adjacent of F & push it onto stack

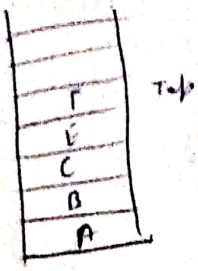




Scanned with CamScanner

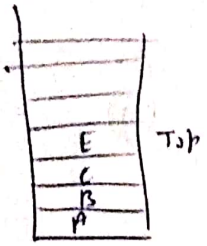
9) There is no new vertex to be visited from vertex G, So, we we backtrack & pop

Point = D → G



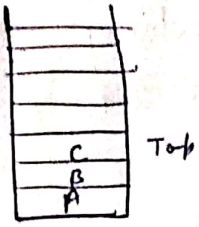
10) Again pop from top

Point = D → G → F



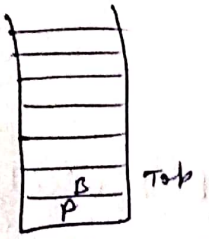
11) Again pop from top

Point = D → G → F → E



(12) Again pop from top

Point = D → G → F → E → C



(13) Again pop from top

Point = D → G → F → E → C → B



(14) Again pop from top

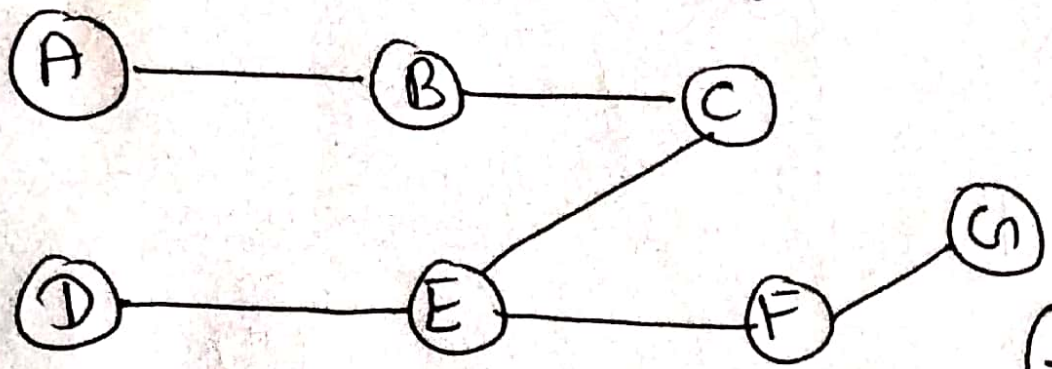
Point = D → G → F → E → C → B → A



Now we traversed all vertices of given graph, remove unused edges from the graph

Scanned with CamScanner

then we obtain final spanning tree



(final spanning tree)

Scanned with CamScanner