Precedence Table of operands

Highest	()	Parenthesis	Left to Right
	[]	sq bracket	
	.	dot	
	->	arrow	
	++, --	postfix inc/dec	
	++, --	prefix inc/dec	Right to Left
	+, -	unary plus/minus	
	!, ~	not, bitwise complement	
	(type)	type operator	
	*	Dereference operator	
	&	address of	
	size of	size in bytes	
	*, /, %	multiply, divide, modulus	L to R
	+, -	add, sub	
	<<, >>	bitwise left shift/right	
	<, <=	less than, less than equal	
	>, >=	greater than, greater than equal	
	==, !=	relational equal, not equals	Left to Right
	&&	bitwise and (AND)	
	^	bitwise or (OR)	
	|	bitwise exclusive or	
	&&	logical and	
	||	logical or	
	?:	ternary	
	=	assignment	
	+=, -=	add/sub assignment	
	*=, /=	multiply/divide assignment	Right to Left
	%=, &=	modulus/bitwise assignment	
	^=, |=	bitwise inclusive/exclusive OR	
	<<=, >>=	bitwise left shift/right shift	
Lowest	,	comma operator	Left to Right

15/7/23

① Data Structure: the ways we organise data
Mathematical and logical model of organizing the interrelated data

Mathematical model: predifined structure of organizing data

Structure: storage and operation to acuss that stored data

logical model: the operations we perform on data to acuss dat

Ye hai data aise aise iss type se store hoga yahan yahan ye cheez rahegi and kahan store hogi (structure + mathematical model) and kaise store hogi (logical model)

② Data storage structure:
i) Linear: Elements are arranged in linear (sequencial) fasion
Eg: array
linked list
queue
stack

ii) Non linear: Elements are arranged in non linear fasion
Eg: tree (binary, general, binary search, heap
avial, full
graph
hash table

③ Classification of data structure
the way data structures are used, ki kis purpose ke liye hum kis data structure ko hum use karna hai use basisis pe alag hota hai
i) Container class (array, linked list)
ii) Indexing class (binary search tree, AVL, hash table, B, B+)
iii) Priority class. (queue, stack, heap)

Why data structures:

DS is used taki hum algo mai Input → Algorithm → Output
kam se kam mehnat, jo input
len and usko read karne mai and output
generate karne mai algorithm kam se kam efforts lagaye
(time and space complexity) taki jaldi se jaldi result mile

Toh issliye aisa data structure select karna hota hai jo ki best ho time and space mai.

④ Operations which are performed on data structures.

i) Traversing
ii) Insertion
iii) Deletion
iv) Searching
v) Merging
vi) Sorting

i) Traversing: Visiting each and every element atleast and atmost once

ii) Insertion: Adding an element in data structure.
→ overflow: error can occur during run time when the limit is reached for inserting (storing new data

iii) Deletion: removing an element from data structure
→ underflow error: kuch hai hi nahi khali hai toh delete kya karun

iv) Searching: Finding an element in data structure
needed for searching: data structure → variables → values to
→ results — successfull search = index of found element be searched
— unsuccessfull search = lower bound − 1

v) Merging: Combining of 2 data structures of same type into one

vi) Sorting: arranging of data in either ascending or decending

⑤ Analysis of algorithm:
This is done on the basis of
i) Run time complexity
ii) Space Complexity (excluding input space)
ie time no. of additional after input taken by alg
ie time no. of additional elements are getting stored.

Run time complexity is dependent on many external factors by wall clock time
hardware
background programs
clicks from mouse or keyboard
programming language
multiple ways to write same code (implementation)
OS

Calculation of run time are → wall clock time
→ number of operations or no. of steps to run

Eg: no. of steps to run

```
c = a+b;      1
d = c*2;      1
return d;     1
```

time complexity = 3

for loops it depend on the ending condition

```
for(i=0; i<=n; i++)      n+1
{
    a = 2+i;             n
}
```

* $\log < n < n^2 < n^3 < 2^n < n!$
$1 < n < n\log n < n^2 < n^3 < 2^n < n!$

Eg:

no. of inputs	no. of steps
2	4
15	17
200	202
1000	1002
n	n+2

n+2 : rate of growth; generalization

rate of growth: what rate the no. of steps (run time complicity) increases with increment in input size.

$T(n) = n+2$ if increasing n linear increase of complexity

$T(n) = n^2+3$ if increasing n parabolic growth.

$T(n) = n^2+n+5$

$T(n) = 2^n+4$

$T(n) = n^n+4$

$T(n) = n^2+2n+6$

→ run time growth	growth complexity
i) logarithmic	$\log n$
ii) exponential	2^n, n^n
iii) linear	n
iv) constant (single	1 (just representation not actual cost of it)
v) quadratic	n^2
vi) logarithmic linear	$n \log n$
vii) quadratic logarithmic	$n^2 \log n$
viii) cubic	n^3

Eg: for (i = 1; i <= n; i++)   n+1 | +1 is the last time it ran when it checked condition as false
{
n = m+2   n
}
2n+1

2n+1 : total no. of step run
it is having linear growth

Eg: for (i = 1; i <= n; i++)   n+1
{
for (j = 1; j <= n; j++)   n * (n+1)   → +1 case of last check when condition is false
{
n = m+2   n * n   → n for 1st loop and n for 2nd loop
}
}
$2n^2 + 2n + 1$

$2n^2 + 2n + 1$: total no. of step run
its having quadratic complexity

Eg: for (i = 1; i < n; i = i*2)
{
n = m+2
}

for (i = n; i > 1; i = i/2)
{
n = m+2
}

let n = 16
i = 1
i = 2
i = 4
i = 8

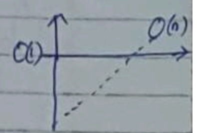
$\log_2 n$: total no. of steps ($2^x \le n$)
logarithmic complexity

if loop runs n times : RT complexity =
$2^x \le n$
$x = \log_2 n$

★ Worst case is the best to tell the time and space for any program

⑥ Asymptotic notation

i) Big O : provides tighest upper bound
(₹ 2000 papa ne diye hain)
O(n), O(n^2), O($\log_2 n$), O(1)
iss kam mai bhi kaam ho sakta hai



ii) Omega Ω : provides tighest lower bound
(min ₹ 500 toh lag he jayenge)
Ω(n), Ω(n^2), Ω($\log_2 n$), Ω(1)
iss zyada bhi lag sakti hain but min ₹ 500

iii) Theta Θ : provides average bound
(₹ 1162 given and exact ₹1162 he spend honge)

if for any problem O(n) & Ω(n)
then complexity = Θ(n)

Cases :
Best Case : for a particular input algo takes min time.

Worst Case : for a particular input algo takes max time

★ agar Θ(1) avg bound constant hai toh lower and upper both bounds are constant

Eg:
```
for(i=1; i<=n; i++)          n+1
{
  for(j=1; j<=n; j++)        n(n+1)
  {
    for(k=1, k<=n; k++)      n²(n+1)
    {
      x = 2 + m;             n³
    }
  }
}
```

RTC : n^3

$[O(n^3)]$ = big O i.e tight upper bound bcoz both mai koi rook nahi rahe toh min and max same toh avg bhi same = $\Theta(n^3)$ final.

Eg:
```
for(i=1; i<=n; i++)          n+1
{
  for(j=1; j<=n; j++)        n(n+1)
  {
    x = m+2;                 n²
  }
  for(k=1; k<=n; k++)        n(n+1)
  {
    y = n+2;                 n²
  }
}
```

RTC : $n^2 + n^2$

$= 2n^2$

$= n^2$ not $2n^2$ as $2n^2$ are the steps but we need highest power variable so n^2 is given eq asymptotically.

$\Theta(n^2)$: koi nahi hai rakne wala to min = max

Eg:
```
for(i=1; i<=n; i++)          n+1
{
  for(j=1; j<=n; j=j*2)      n(log₂n)
  {
    y = z*2;                 n(log₂n)
  }
}
```

RTC : $n \log_2 n$

$\Theta(n \log_2 n)$

Eg:
```
for(i=1; i<=n; i=i*2)        log₂n
{
  for(j=1; j<=n; j=j*2)      (log₂n)²
  {
    y = z*2;                 (log₂n)²
  }
}
```

RTC : $(\log_2 n)^2$

$\Theta(\log_2^2 n)$

Eg:
```
for(i=0; i<n; i++)           n+1
{ a = a + sqrt(i); }         n
for(j=0; j<m; j++)           m+1
{
  b = b + sqrt(j);           m
}
```

RTC : $2m + 2n + 2$

$= m + n$

$\Theta(m+n)$

Eg:
```
for (i=1; i<=n; i++)          n+1
{
  for (j=1; j<=i; j++)        n(1+2+3+...+n) = Σn
  {
    x = x+2;
  }
}
```

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

RTC = n^2

$\Theta = \Theta(n^2)$

⑦ Array: Collection of homogeneous elements (same datatype)

Characteristics of array:
i) All elements stored on consecutive memory locations
ii) All elements can be accessed using a set of indices

generally: starting (LB) = 0
ending (UB) = size − 1 $\quad LB \leq UB$

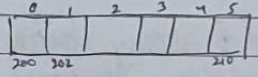
but for DS: name [LB : UB] ✓

Eg: A[2:15] ✓ = 14
B[5:18] ✓ = 14
D[-15:-5] ✓ = 11
E[15:-5] ✗ = -09

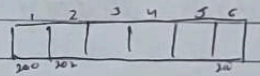
[size: UB − LB + 1]

[Location of element (A[i]) = base + w * (i − LB)]

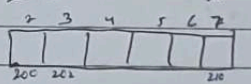
w = size of each element
base = base address of array
LB = lower bound
i = index of element

Eg: A = 

200 + 2*(3−0) = 206

A = 

200 + 2*(3−1) = 204

A = 

200 + 2*(3−2) = 202

index no. (3) ka location:

* To get no. of elements before or from given element = (i − LB)
(i − LB): relative index

⑧ Traversing in Array:

Array ka har element ko ek ek baar visit karna hai

```
for (i = LB ; i <= UB; i++)
  {
    visit A[i];
  }
```

RTC = $\Theta(n)$
SC = $\Theta(1)$ besides space by input space taken by algo and its elements used.

algorithmic: for i = LB to UB, step by 1

Eg: array A[-7:13] each element has 2 locations of memory. Total space needed to store array.

total elements = UB - LB + 1
= 13 + 7 + 1
= 21 × 2 = 42 = total space

Eg: array A[-5:18], is stored in array starting from 1000 each element takes 4 locations, location of A[5] is?

no. of elements = 18 + 5 + 1
= 24

Location A[5] = base + w*(i - LB)
= 1000 + 4*(5+5)
= 1040

⑨ finding maximum / minimum:
Comparison method (bubble method)
Eg: 15, 8, 9, 7, 12, 19, 20, 21, 2, 4, 11, 5, 6, 1, 3

min = 15 1st run (supposed as min then compare further next elements)

no. of comparisons = n - 1 (n = no. of elements)

```
min = A[LB];
for (i = LB + 1; i <= UB; i++)
  {
    if (A[LB] < min)
    {
      min = A[i];
    }
  }
return min;
}
```

RTC = $\Theta(n)$
SC = $\Theta(1)$ as only 2 variables (i, min) LB given, A is array given

If agar humne 1000 variables array mai diye fabhi bhi ye variables (new, algo variables) will be needed to perform action so its a constant.

tournament method

odd inputs

(walkover mil gaya coz iska opponent aya hi nahi)

1) 15, 8, 9, 7, 12, 19, (20)
2) 8 7 12 20
3) 7 12
4) (7)

7 elements 6 comparisons
i.e n-1

even inputs

1) 15, 8, 9, 7, 12, 19, 20 5
2) 8 7 12 5
3) 7 5
4) (5)

8 elements 7 comparisons
n-1

RTC = $\Theta(n)$
SC = $\Theta(n)$

n elements hain to after comparing (stage 1 elements) the next stage elements are half the size of their parent array. i.e $\frac{n}{2}$ size.

n/2 is n i.e linear complexity.

as point of space this algo is bad than the previous algo which was having space constant.

ismai constant iss liye nahi ho paya coz n hai kyunki input jo ki khud hi variable hai

Eg: finding maximum.

```
max = A[LB]
for (i = LB+1 ; i <= UB ; i++)
{
    if (A[i] > max)
        max = A[i]
}
return max;
}
```

no. of comparisons : n-1
RTC : $\Theta(n)$
SC : $\Theta(1)$ no new variable was created

Eg: Find min & max at same time:

General: Finding min = n - 1
Finding max = n - 1
$2n - 2$

Optimised:
agar do numbers hain aur ek chota hai vo consider hoga for smallest number for now but that bigger number will be not at all considered again for smaller and comparing with other biggers.

$\overline{a_1, a_2}, \overline{a_3, a_4}, \overline{a_5, a_6} \ldots\ldots \overline{a_{n-1}, a_n}$

smaller $n/2$ | larger $n/2$

preparing 2 lists	$\frac{n}{2}$
to find min	$\frac{n}{2} - 1$
to find max	$\frac{n}{2} - 1$
	$\frac{3n}{2} - 2$

// agar odd no. of values hue to last wala new element ko dono mai copy kardunge without comparision coz last ke liye koi pair hi nahi jisse compair kar sake

(odd) Total no. of comparisions = $\left\lceil \frac{3n}{2} \right\rceil^{ceil} - 2$

(even) Total no. of comparisions = $\left\lceil \frac{3n}{2} \right\rceil - 2$

RTC = O(n)
SC = $\frac{n}{2} + \frac{n}{2} = n$

ye general wala se $\frac{3}{2} = 01.5$ i.e 0.5 se better hai as
general = 2.0
optimise = 1.5 0.5 better hai

Eg: i) No. of comparisions to find max of 160 elements is?

$n-1 = 160-1 = 159$

ii) find max & min of 300 numbers:

$\frac{3n}{2} - 2 = 3*150 - 2 = 448$

iii) min no of comparisions to find min & max of 123 numbers

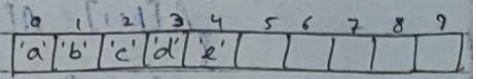
$\left\lceil \frac{3*123}{2} \right\rceil^{c} - 2 = \frac{369}{2} - 2 = \lceil 184.5 \rceil^{c} - 2$

$= 185 - 2 = 183$

(10) Insertion in array: At the end

i) At last
ii) Based on index

i) At the end

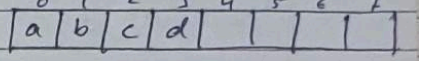


actual UB = 4 (algorithmic UB)
ye vo UB hai jisse pata chalta hai last element kahan present hai

coz agar UB last index of array then array full hai toh usmai insertion ke liye space nahi to for size increase karo then then last mai insert karo agar UB < Last index then UB ke baad mai he insert kar do

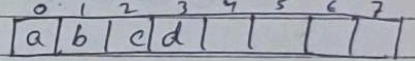
RTC = O(1)

case1: index = UB+1, x

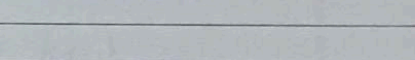


i.e insertion at last index
results in no shifting of any element but addition of that element at the last space.

case2: in between the indexes



index = 2
elmt = x
this is possible only by shifting the elmts c & d one, one index behind i.e



case 3: insertion (item, index)

```
{
for (int k = UB, k >= index; k--)
  {
    A[k+1] = A[k];
  }
A[index] = item;
UB++;
n++;
}
```

RTC = O(n) maybe kam bhi ho sakta hai

(11) Deletion in array: At the end.

Deletion memory mai kabhi nahi hota hai ya toh jo pointers hota hai unki pointing uss value pe band hoti hai ya fir usko overwrite kiya jata for new value kabhi bhi kuch delet nahi hota hai

LB = 0
UB = 4
n = 5 i.e there may exist any garbage value

case 1: deletion based on index

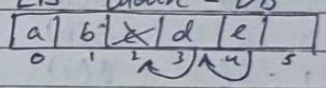
LB ≤ index ≤ UB

index to delet = UB
no elements are shifted

case 2:

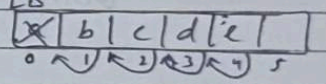
LB < index < UB

index = 2



case 3:

index = LB



n-1 elements are shifted

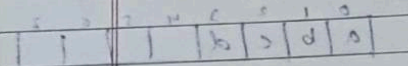
```
for (k = index; k <= UB+1, k++)
  {
    A[k] = A[k+1];
  }
  UB--;
  n--;
```

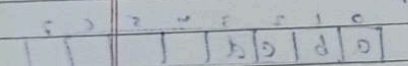
RTC = O(n)

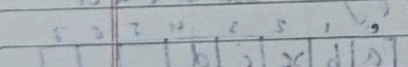
case 3: insertion (item, index)

```
{
for (int k = UB, k >= index; k--)
  {
    A[k+1] = A[k];
  }
  A[index] = item;
  UB++;
  n++;
}
```

RTC = O(n) maybe kam bhi kaam hojaye







(ii) Deletion in array: At the end.

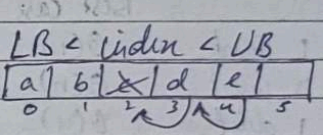
Deletion mumary mai kabhi nahi hota hai ya toh jo pointer hota hai unki pointing uss value pe band hati hai ya fir usko overwrite kiya jata for new value kabhi bhi kuch delet nahi hota hai.

LB = 0
UB = 4
n = 5 i.e there may exist any garbage value

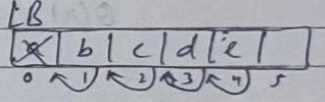
case 1: deletion based on index LB ≤ index ≤ UB

index to delet = UB
no elements are shifted

case 2: index = 2 LB < index < UB



case 3: index = LB



n-1 elements are shifted

```
for (k = index; k <= UB+1, k++)
{
   A[k] = A[k+1];
}
  UB--;
  n--;
```

RTC = O(n)

Eg i) No. of shifting of the elements to insert a new element at starting index in an array of 200 elements is?

Ans = 200

Eg ii) No. of shifting of the elements to delet an element at starting index in an array of 200 elements is?

Ans = n - 1 = 199

Eg iii) Consider an array A of n elements ($n \geq 1$). In this array A n insertions and n deletions are performed in arbitrary order. What should be the best case and worst case complexity of all operations on array?

	Best case	worst case
1 insert	$O(1)$	$O(n)$
n insert	$O(n)$	$O(n^2)$
1 delet	$O(1)$	$O(n)$
n delet	$O(n)$	$O(n^2)$
	$O(n)$	$O(n^2)$

(12) Searching in array

i) Linear Search
ii) Binary Search

i) Linear search :

```
int linearsearch (A[], LB, UB, item)
{
  for (k = LB, k <= UB, k++)
  {
    if (A[k] == item) {
      return k;
      break;
    }
  return LB-1;
}
```

RTC = $O(n)$

→ It can be applied on any kind of array.

ii) Binary search:

0	1	2	3	4	5	6	7	8	9
1	4	9	11	15	21	23	29	34	50

search = 23
low = 0
high = 9
mid $\frac{0+9}{2} = 4$

Algorithm:

```
int binary_search (A[], LB, UB, item)
{
1) Low = LB ; High = UB;
2) mid = (low + high) / 2;
3) while ((A[mid] != item) && (low <= high))
   {
     if (item < A[mid])
        high = mid - 1;
     else
        low = mid + 1;
     mid = (low + high)/2;
   }
4) if (A[mid] == item)
     return mid;
   else
     return (LB - 1);
}
```

RTC: O(log, n)

Eg: Consider a sorted array of size n with duplicate elements. You have been given an element k, what is the time complexity to find that the element k is appeared atleast n/2 times in the array or not.

0 1 2 3 4 5 6 7 8 9 10 11
0, 3, 6, 6, 6, 6, 6, 6, 6, 6, 9, 12 ; k = 6

i = index 1st appeared
O(log, n)

given: inplace sorted array

$A\left[i + \frac{n}{2} - 1\right] = item$;
return true;

Ans = O(log, n)

(13) 2D array : collection of 1D array

uses : 1) matrix storage
2) graph data structures

Eg. 3×4 array

a_{00}	a_{01}	a_{02}	a_{03}
a_{10}	a_{11}	a_{12}	a_{13}
a_{20}	a_{21}	a_{22}	a_{23}

3×4

datatype name [size (row)][size (column)]
int A [3] [4]

r = 0, 1, 2
c = 0, 1, 2, 3 ⇒ A[0:2][0:3]

$LB_i = 0$ $LB_j = 0$ Lower bounds
$UB_i = 2$ $UB_j = 3$ Upper bounds

$[a_{00} + a_{01} + a_{02} + a_{03}] + [a_{10} + a_{11} + a_{12} + a_{13}] + [a_{20} + a_{21} + a_{22} + a_{23}]$

→ no. of rows = $UB_i - LB_i + 1$

→ no. of columns = $UB_j - LB_j + 1$

→ array size = $m \times n$ (m = no. of rows, n = no. of columns)

Eg: find size of array A [-2:4] [5:18]
4 + 2 + 1 = 7
18 - 5 + 1 = 14

total elements = 7 × 14 = 98

→ Storage pattern of 2D array in memory
i) row major order
ii) column major order

→ Location A[i][j] = base + w * (relative index of element)
= base + w * [total no. of elements in row from LB_i to (i-1) + no. of elements from column no. LB_j to (j-1) within ith row]

Row major

Location = base + w * [(i - LB_i) * n + (j - LB_j)]

Eg: Base = 200, m = 3, n = 4; LB_i = 0, LB_j = 0, w = 2

loc (A[2][2]) = 200 + 2 [(2-0) * 4 + (2-0)]
= 200 + 2 (8 + 2)
= 200 + 20
= 220

Column major

Location = base + w * [(i - LB_i) + (j - LB_j) * m]

Eg: A [-5:7] [2:12], starting address of array in memory is 1000
Each element occupies 4 memory locations
1) What is address of A[0][9] in Row major order
2) What is address of A[5][7] in Row major order

1) A[0][9] = 1000 + 4 * [(0 - (-5)) 11 + (9 - 2)]
= 1248

2) A[5][7] = 1000 + 4 * [(5 - (-5)) * 11 + (12 - 7)]
= 1460

Eg: A[2:12][-6:5], Address start from 400, each element occupy 4 memory spaces using column major order
1) address of A[8][2]
2) address of A[3][4]

1) loc(A[8][2]) = 400 + 4 * [(8-2) + (2-(-6)) * 11]
= 776

2) loc(A[3][4]) = 400 + 4 * [(3-2) + (4-(-6)) 11]
= 844

(14) Lower triangular matrix:

a_{00} a_{01} a_{02} a_{03} originally n x n sq matrix
a_{10} a_{11} a_{12} a_{13}
a_{20} a_{21} a_{22} a_{23} A[i][j] [0 i<j ; non zero i>=j
a_{30} a_{31} a_{32} a_{33}

no. of elements = 1+2+3+....+n
$= \frac{n(n+1)}{2}$

Row major

0	1	2	3	4	5	6	7	8	9
a_{00} 200	a_{10} 202	a_{11} 204	a_{20}	a_{21}	a_{22}	a_{30}	a_{31}	a_{32}	a_{33} 218

Row major ord

0 - 9 is relative indexing.

→ loc(A[i][j]) = base + w * $\left[\frac{i(i-1)}{2} + (j-1)\right]$

no. of elements from 1st = 1
no. of elements from 2nd = 2
no. of elements from 3rd = 3
⋮
no. of elements from (i-1) = +i-1
$\frac{i(i-1)}{2}$

Eg: base = 200, w = 2, A[3][2], A[0:3][0:3]

loc A[3][2] = 200 + 2 * $\left[\frac{(3-1)3}{2} + (2-1)\right]$
= 200 + 2 (3 + 1)
= 208

relative index = $\frac{3(3-1)}{2} + (2-1) = 8$

Column major

no. of elmts in 1st column = n
no. of elmts in 2nd column = n-1
⋮
no. of elmts in (j-1) column = n-(j-2)
no. of elmts in (j)th column = n-(j-1)

$= n + (n-1) + (n-2) + \ldots + (n-(j-2))$
$= n(j-1) - [1 + 2 + 3 + \ldots + (j-2)]$
$= n(j-1) - \left[\frac{(j-1)(j-2)}{2}\right]$

$\left[n(j-1) - \left[\frac{(j-1)(j-2)}{2}\right] = \text{no. of elmts}\right]$

$\text{Loc}[A[i][j]] = \text{base} + w * \left[n(j-1) - \frac{(j-1)(j-2)}{2} + (i-j)\right]$

Eg: Consider a 2-D array of size 8×8. The starting address of array in memory is 1000. Each elmnt occupies 4 memory locations what is the address of elmnt A[4][2] in row major order of lower triangular matrix:

B = 1000, w = 4, i = 4, j = 2, n = 8

RMO: $1000 + 4 * \left[\left(\frac{4*3}{2}\right) + (2-1)\right] = 1028$

CMO: $1000 + 4 * \left[8*(2-1) - \frac{(2-1)(2-2)}{2} + (4-2)\right]$
$= 1040$

Eg: Advantages of Array:
i) Elements can be accessed randomly by using the index number.
ii) Using array other data structures like linked list, stacks queues, tries, graphs etc can be implemented
iii) 2D array can be used for matrices.

Eg: Disadvantages of Array:
i) The number of elements to be stored in an array should be known in advance.
ii) Allocating more memory than the requirement can lead to wastage of memory space and less allocation of memory can lead to overflow problems.
iii) Array is static structure. Once declared size cannot be altered
iv) Insertion and deletion is quite difficult.
v) Elements must be stored in consecutive memory locations

* array contigous address while LL non contigous in heap memory and can be stored anywhere but connected

(13) Linked list

→ Linear data structure
→ Linear order maintained using pointers (link)
→ list contains nodes
→ Node contains 2 fields
i) Data (key or value) | Data | link |
ii) link (next or forward)

link gives the addrs where next data is stored
it keeps the address of that node data just like pointers

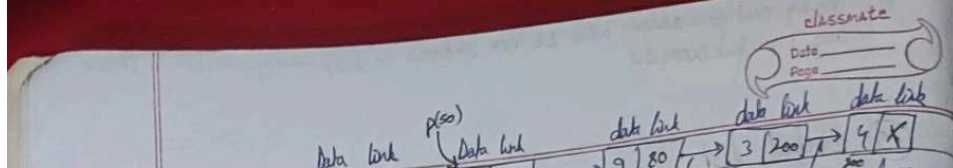
→ There is a list pointer which points to the 1st node of list
→ Null pointer (Hard stop pointer) to end the further linking making the end of list as no more nodes exist

→ Head / start / first pointer to start the 1st node of the list.

```
struct node
{
  int n;
  struct node * link;

};
```

* link : pointer which stores the address of 1st node
koi bhi node ko access karna ho toh pointer ka use karte hain.

start → data = 5; i.e old data node '6' is overwritten by '5' as new value
start → link → data = 15; access of data part



i) start → data = 5; ∴ data value of 1st node is changed to '5'

or

ii) start → link → data = 15; ∴ data value of 2nd node is changed to '15'

or

iii) start → link = null; address of node 1 is updated to "null"

or

iv) start → link → link = null; address of node 2 is changed to "null"

Empty list: when the list is empty i.e. no data value then the start pointer will point to null by default

```
if (start == null)
{
    printf("Empty list");
}
```

```
if (!start)
{
    printf("Empty list");
}
```

Single node:

start → [ ][X]

```
if (start → link == null)
```

```
if (!start → link)
```

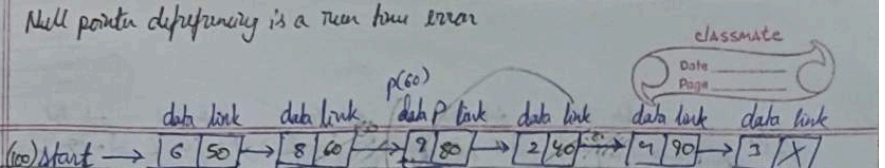
v) p → link → link → link = p; (node 5) address of p = 50

vi) p = start → link → link (node 2) address = 60

vii) p → link → link = p (node 3) address = 50

viii) printf("%d", start → link → link → link → link → data); = 4

* Null pointer dereferencing is a run time error

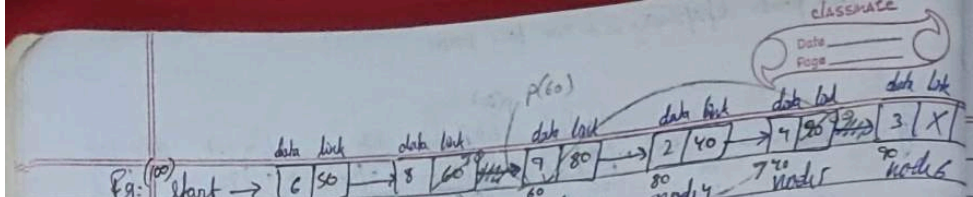



```
struct node *p;
p = start → link → link;        p = 60
start → link → link = p → link;
p → link → link = p;
printf("%d", start → link → link → link → link → data); = 2
```

(start → link: 1st node's address; start → link → link: 2nd node's address; start → link → link → link: 3rd node's address; start → link → link → link → link: 4th node's address)

Pointers	Address
start having address of node 1 as	(100)
link (1st) is having the address of node 2 as	(50)
link (2nd) is having the address of node 3 as	(60)
link (3rd) is having the address of node 4 as	(80)
link (4th) is having the address of node 5 as	(40)
link (5th) is having the address of node 6 as	(90)
link (6th) is having · null (end of list)	

[link (i) is having the address of node (i+1)]

Eg: 

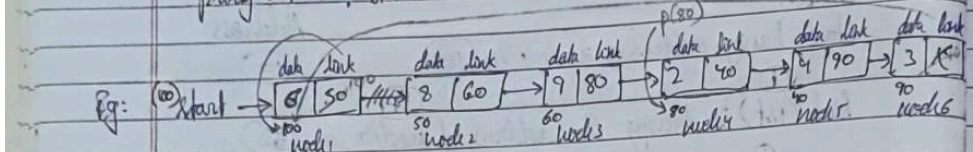
struct node *p;

p = start → link → link; p = 60

start → link → link = p → link → link;

p → link → link → link = p;

printf("%d", start → link → link → link → data);

Eg: 

struct node *p;

p = start → link → link → link; p = 80

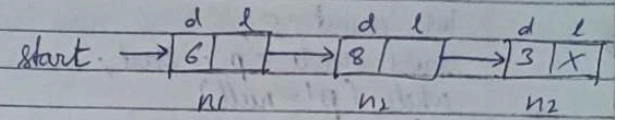
p → link → link → link = start;

start → link = p → link → link → link;

printf("%d", start → link → link → link → link → data);

Eg: Advantage of linked list over array:
It has dynamic memory allocation

→ Traversing in linked list




```
struct node *p;
p = start;
while (p != null)        or  while(p)
{
  print (p → data)
  p = p → link;
}
```

RTC = O(n)
Hamesha ye loop n time

→ Finding address of last node:
The disadvantage of linked list is that we can have only sequencial access no random access and will have to traverse complete list

Last node link will always having null.

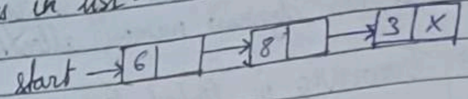
```
struct node *p = start;
while (p → link != null)
{
  p = p → link;
}
return p;
```

// agar empty list hai or phle he null aa jaye to dikkat krne wagi

```
if (p == null)
  return null;
```

RTC = O(n)

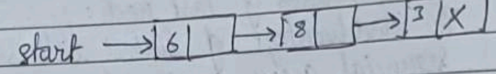
→ Counting the nodes in list:




```
int count = 0;
struct node *p = start;
while (p != null)
{
    count++;
    p = p -> link;
}
return count;
}
```

RTC = O(n)

→ Searching in linked list:



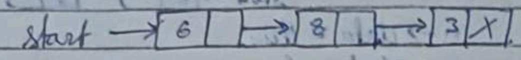
// as we cannot reach any mid index (m) and last (n) in constant time as we cannot get that node in linked list without knowing the index before hand or without traversing entire list due to which we cannot use binary search.
so we have to apply linear search

agar element nahi mila then return: null

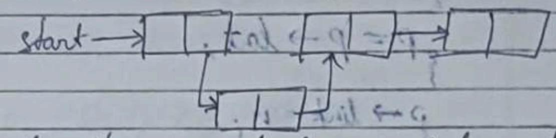
```
searching (start, item) {
    struct node *p = start;
    while (p != null)
    {
        if (p -> data == item)
            return p;
        p = p -> link;
    }
    return p;        // or return null   cuz last null hai and null reached ie not found
```

RTC = O(n)

→ Insertion in linked list:



// unlike array ismai naya insert karna hai toh koi bhi shifting nahi hogi bus change the links and good to go.



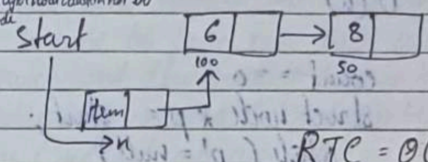
ab aisa karne ke liye ek new node banana padega which will have the new data to be added along with the link having address of next node which makes linked list dynamic and memory efficient over array

struct node *n = (struct node *) malloc (sizeof (struct node));

explicit typecast karna hota of memory of node and its a node type ka pointer hai jiske address banega usko hum ek pointer mai copy karte hai — use sizeof node as to make being size of struct and malloc ke create

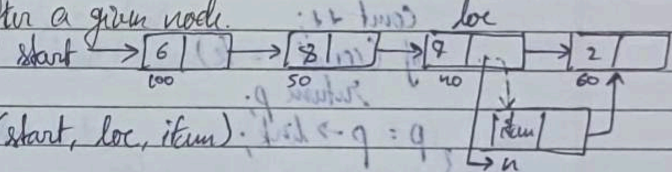
i) Insert at beginning

```
n -> data = item      // ye inputted item hai after node creation not the item of no add node
n -> link = start
start = n
```



RTC = O(1)

ii) Insertion after a given node.

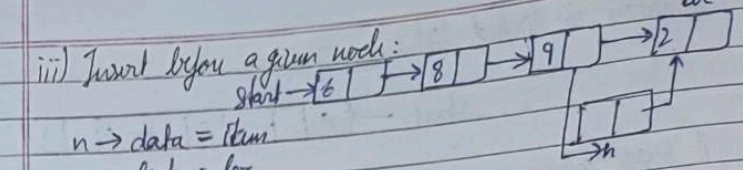



```
insertion (start, loc, item)
{
    n -> data = item
    n -> link = loc -> link
    loc -> link = n
}
```

RTC = O(1)

// loc last node ko point karega tab bhi ye theme points kaam karenge

iii) Insert before a given node:




```
n → data = item
n → link = loc
struct node *p = start
while (p → link != loc)
{
  p = p → link;
}
p → link = n;
```

RTC = O(n)

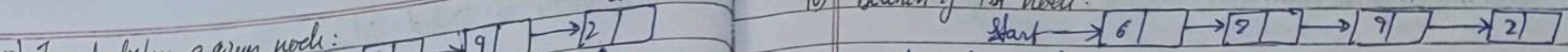
Eg: What is complexity to insert a new element node in a sorted linked list
i) O(n) coz traverse puri karni hogi coz nahi pata kahan karni hai

Eg: What is the complexity to find kth node from starting in a singly linked list using constant spaces
i) O(n)

```
count = 0
struct node *p = start;
while (p != null)
{
  count ++;
  if (count == k)
    return p;
  p = p → link;
}
```

Eg: What is the complexity to find kth node from starting in a singly linked list using constant space.
O(n)

iv) Deletion of 1st node:




```
struct node *p = start;
start = start → link;
free (p)
```

// free: memory of the pointer i.e deletion kar bhi free space nahi hogi (nodes)

v) Deletion of last node:

```
struct node *p = start, *q = start;
while (p → link → link != null)
{
  p = p → link;
}
q = p → link;
p → link = null;
free (q);
```

RTC = O(n)

vi) Deletion of last node: last is Given
last ek pointer pointing to last node address but uska address pata hai but uske pichle ka nahi toh firse ek bar puri list traverse karni hogi to reach 2nd last node of list so as to assign 2nd last node address as null and last pointer will also point at that address (2nd last node originally)

```
struct node *p = start;
while (p → link != last)
{
  p = p → link;
}
struct node *q = last;
free (q)
p → link = null
last = p;
```

RTC = O(n)

vii) Deleting a given node:

```
struct node *p = start;
while(p→link != null)
{
    p = p→link; }
p→link = loc→link;                RTC = O(n)
free(loc)
```

viii) Deletion of a given node without using any other pointer

M1 ye kai bhi valid way nahi hai ek bomb lagaya usse ek toh maar hi gaya saza-e-mot 10 ai maar gaye saza-e-mot i.e loss of data se koi farak nahi

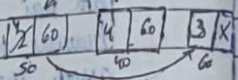
```
start = loc → link;    // uske phele ke sab mar gaye
```

M2

If start is also not given only loc is given

police ko rajnaki nila to raju ko maar maar ke usse gulal karwaya

```
loc → data = loc → link → data
loc → link = loc → link → link
```



this works loc pointer doesn't point to last node.

ix) Deletion before a given node (loc not on 1st)

```
struct node *p = start;
while(p → link → link != loc)
{
    p = p → link;
}
p → link = loc;                RTC = O(n)
```

ix) Deletion after a given node (loc not at last)

```
lock → link = loc → link → link               RTC = O(1)
```

- Application of linked list:

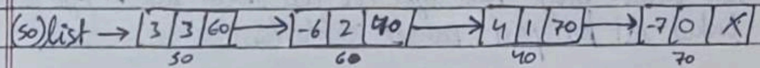
i) Univariate (single variable)

ii) Bivariate (double variable)

i) Univariate Polynomial:

node : | coff | varid | address |

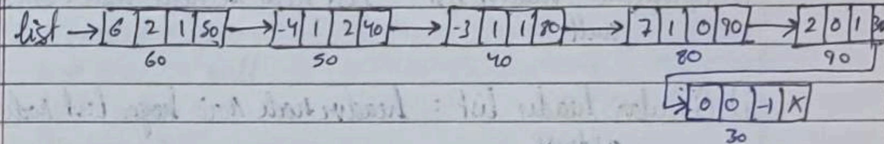
$3x^3 - 6x^2 + 4x - 7$



ii) Bivariate Polynomial:

node : | coff | var1 | var2 | add |

$6x^2y - 4xy^2 - 3xy + 7x + 2y - 1$



Eg: What is the complexity to delete a given key value node from a sorted linked list?

O(n)

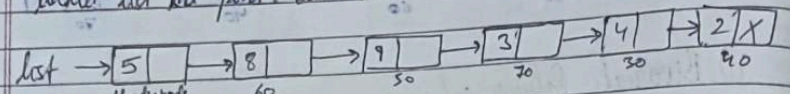
Eg: What is the complexity to remove a node from last and to insert it at the starting of a singly linked list.

Θ(n)

- Disadvantage of singly list:
→ The list part of last node is not utilized
→ The address of predecessor is not known
→ Skipping backward is not possible

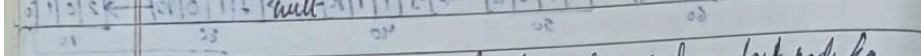
- Header list:
Contains special first node called header node
Header node contains some summary information.

kabhi kabhi jab linked list ke nodes bar bar count karne padte
kuch insertion deletion kiya first count isse bar bar traversing hogi
i.e har bar O(n) complexity.
so header node is created having the count of all the nodes
new variable isliye nahi liya coz uss linked list ki info uss
linked list ka part banake rakhi hai



ab humko address pata hoga usse fir constant lagega

i) Grounded header list: last node ke node mai bola hai



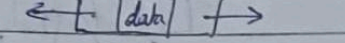
ii) Circular header list: header node mai hoga last node ka address

element storage and start of traves start from node next to header but still counting mai phli node header ki hai but data storage start from next to header node)

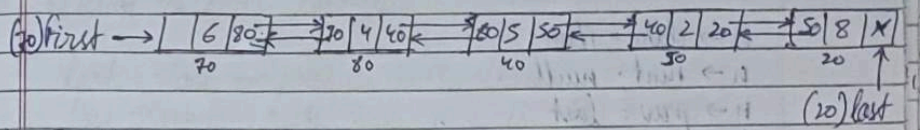
Grounded	Circular
struct node *p = list → link	struct node *p = list → link
while (p != null)	while (p != list)
{	{
p → data	p → data;
p = p → link;	p = p → link;
}	}

O(n)

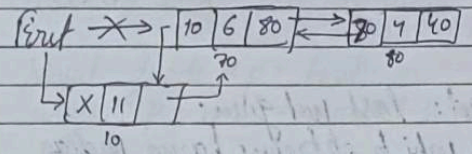
(15) Doubly linked list: dono taraf jane mai help hogi




```
struct dnode
{
    char data;
    struct dnode * next;
    struct dnode * previous;
};
```



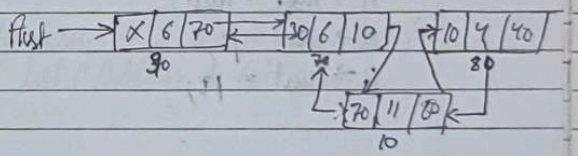
i) Insertion at starting:



n → data = item
n → next = first
n → prev = null
first → prev = n
first = n

RTC: O(1)

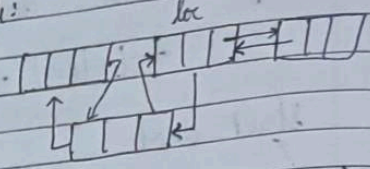
ii) Insertion after a node



n → data = item
n → next = loc → next
n → prev = loc
n → next → prev = n
loc → next = n

RTC = O(n)

iii) Insertion before a given node:




```
n → data = item
n → next = loc
n → prev = loc → prev
n → prev → next = n
loc → prev = n
```

RTC = O(1)

v) Insertion at the end: last given:

```
n → data = item
n → next = null
n → prev = last
last → next = n
last = n
```

RTC : O(1)

vi) Insertion at the end: last not given:
last to address pata nahi to ab traverse karna padega

```
struct dnode *p = first;
while ( p → next != null )
{
    p = p → next;
}
n → data = item;
n → next = null;
n → prev = p;
p → next = n;
```

vii) Deletion from starting:

```
first = first → next
first → prev = null
```

RTC : O(1)

viii) Deletion of a given node:

```
loc → prev → next = loc → next
loc → next → prev = loc → prev
```

RTC = O(1)

ix) Deletion at end: last given

```
last = last → prev
last → next = null
```

RTC : O(1)

x) Deletion at end: last not given

```
struct dnode *p = first
while ( p → next → next != null)
{
    p = p → link. }
p → next = null;
```

RTC : O(1)

Eg: What is the complexity to insert at starting of circular singly linked list?

O(n)

Eg: What is the complexity to remove a node from last from a circular singly linked list. (Address of last given)

O(n)

(17) • Brute force approach : (Bubble, Selection, Insertion) (linear)
• Divide and conquere : Sorting (merge, quick), binary

Bubble Sort
Selection Sort
Insertion Sort
Merge Sort
Quick sort
Heap sort
Binary Search
Linear Search

(18) Bubble Sort : Sabse bada sabse last position pe settle and same is done with n-1 elements

I_1 : largest settled at last position.
I_2 : 2nd largest settled at 2nd last.
I_3 :
⋮
I_n : n-1 th largest settled at 2nd positions.

Eg: 3, 5, 2, 4, 1 → 5 swaps

I_1 : $\overline{3,5}, 2, 4, 1$
$3, \overline{5, 2}, 4, 1$
$3, 2, \overline{5, 4}, 1$
$3, 2, 4, \overline{5, 1}$
$3, 2, 4, 1, 5$ ⑤

I_2 : $\overline{3, 2}, 4, 1, 5$
$2, \overline{3, 4}, 1, 5$
$2, 3, \overline{4, 1}, 5$
$2, 3, 1, \overline{4, 5}$ ④

last pe phle he biggest reached so will have n-1 comparisons

I_3 : $\overline{2, 3, 1}, 4, 5$
$2, \overline{3, 1}, 4, 5$
$2, 1, 3, 4, 5$ ③ n-2

I_4 : $\overline{2, 1}, 3, 4, 5$
$1, 2, 3, 4, 5$ ② n-3

If there are n no. of elements in a given array, n-1 iterations are required to sort the array

for 1st iteration : j = 0 to n-2 i=1 : $\frac{n-i-1}{n-1-1=n-2}$

for 2nd iteration : j = 0 to n-3 i=2 : $\frac{n-i-1}{n-2-1=n-3}$

for 3rd iteration : j = 0 to n-4 i=3 : $\frac{n-i-1}{n-3-1=n-4}$

```
for(i=0; i<n-1; i++)
{
  for(j=0; j<n-i-1; j++)
  {
    if(a[j] > a[j+1])
    {
      temp = a[j];
      a[j] = a[j+1];
      a[j+1] = ;
    }
```

```
for(i=1; i<=n-1; i++)
{
  for(j=0; j<=n-i-1; j++)
  {
    if(a[j] > a[j+1])
    {
      temp = a[j];
      a[j] = a[j+1];
      a[j+1] = temp;
    }
```

→ SC : i, j, temp : O(3) = O(1)
→ RTC : 1 + 2 + 3 + ... + (n-1)
$= \frac{n(n-1)}{2} = \frac{n^2-n}{2}$
$O(n^2)$

→ In place sorting : no new array or elements are created
→ Stable : maintains the relative order of equal elements

Optimized bubble sort

2, 1, 5, 4, 3

I₁: 2, 1, 5, 4, 3
1, 2, 5, 4, 3
1, 2, 5, 4, 3
1, 2, 4, 5, 3
1, 2, 4, 3, 5

I₂: 1, 2, 4, 3, 5
1, 2, 4, 3, 5
1, 2, 4, 3, 5
1, 2, 3, 4, 5

```
int f;
for (i=0; i<n-1; i++)   // yahi bubble baar baar starting se check karta coz bubble jaise sabse bada last mai jata (largest)
{ f=0
for (j=0; j<n-i-1; j++)  waise he last ka check mat karo
{
if (a[j] > a[j+1])   // baram para f se hoga but i,j ko restart kar rha.
{
temp = a[j];
a[j] = a[j+1];
a[j+1] = temp;
f = 1;     // swapping hue hai coz unsorted thi array.
} // if
} // end for (inner)
if (f == 0)     // ek baar bhi agar swap nahi hua i.e array sorted thi toh next iteration (i) pe badho
{
break;
}
f = 0.
} // outer for
```

★ Selection sort only kind of sort which will have max swaps n-1 which is min for all other sorts
Selection is best amongst all comparison base algorithms.

⑲ Selection Sort ★ select the min & swap

find min of all the present elements make its index
swap min and 1st element

i) $a_1, a_2, a_3, a_4 \ldots \boxed{a_j} \ldots a_n$ (min)

$a_j, a_2, a_3, a_4 \ldots a_1 \ldots a_n$

ii) $a_j, a_2, a_3, a_4 \ldots \boxed{a_n} \ldots a_n$ (2nd min)

$a_j, a_n, a_3, a_4 \ldots a_2 \ldots a_n$

Eg: 3, 4, 2, 5, 1

	0 1 2 3 4		
I₁:	3, 4, 2, 5, 1	i=4	swap(a[0], a[4])
	1, 4, 2, 5, 3		
	0 1 2 3 4		
I₂:	1, 4, 2, 5, 3	i=2	swap(a[1], a[2])
	1, 2, 4, 5, 3		
I₃:	1, 2, 4, 5, 3	i=4	swap(a[2], a[4])
	1, 2, 3, 5, 4		
I₄:	1, 2, 3, 5, 4	i=4	swap(a[3], a[4])
	1, 2, 3, 4, 5		

```
for (i=0; i<n-1; i++)
{
  min = i;
  for (j=i+1; j<=n-1; j++)
  {
    if (a[j] < a[min])
      min = j;
  }
  temp = a[i];
  a[i] = a[min];
  a[min] = temp;
}
```

max no. of swaps: n-1
RTC: $O(n^2)$
SC: $O(n)$

Optimized code:

```
for (i=0; i<n-1; i++)
{
  min = i;
  for (j=i+1; j<=n-1; j++)   // ith ka next wali se check karo
  {
    if (a[j] < a[min])       // idhar compair karke raho puri array mai
      min = j;               // jisse chota koi hai to index note karlo
  }
  if (i != min)              // maan lo raho 1st pos pe hi min hai
  {                          // min i ka sath hi aage badhte rahega
                             // coz apne aap se 1st pe puri array ka min aajyega
    temp = a[i];             // last mai i iteration overko swap karado
    a[i] = a[min];
    a[min] = temp;
  }
}
```

→ In place : no new datatype, array not used.
→ Unstable : not able to maintain relative position of equal elements

(28) Insertion sort

10, 7, 3, 6, 2, 5, 4

I_1: 10, 7, 3, 6, 2, 5, 4
7, 10, 3, 6, 2, 5, 4

I_2: ab 7, 10 ki sorted list mai 3 ko insert karna hai
3, 7, 10, 6, 2, 5, 4

I_3: ab 3, 7, 10 ki sorted list mai 6 ko insert karna hai
3, 7, 10, 6, 2, 5, 4
3, 7, 6, 10, 2, 5, 4
3, 6, 7, 10, 2, 5, 4

I_4: ab 3, 6, 7, 10 ki sorted list mai 2 ko insert karna hai
3, 6, 7, 10, 2, 5, 4
3, 6, 7, 2, 10, 5, 4
3, 6, 2, 7, 10, 5, 4
3, 2, 6, 7, 10, 5, 4
2, 3, 6, 7, 10, 5, 4

I_5: ab 2, 3, 6, 7, 10 ki sorted mai 5 ko insert karna hai
2, 3, 6, 7, 10, 5, 4
2, 3, 6, 7, 5, 10, 4
2, 3, 6, 5, 7, 10, 4
2, 3, 5, 6, 7, 10, 4

I_6: ab 2, 3, 5, 6, 7, 10 ki sorted list mai 4 insert karna
2, 3, 5, 6, 7, 10, 4
2, 3, 5, 6, 7, 4, 10
2, 3, 5, 6, 4, 7, 10
2, 3, 5, 4, 6, 7, 10
2, 3, 4, 5, 6, 7, 10

* Best for any sorted list to be chcked insertion sort algo is the best but optimized bubble also has O(n) but its optimized.

```
for (i = 0; i <= n-1; i++)
{
    for (j = i ; (j > 0 && a[j-1] > a[j]); j--)
    {
        int temp = a[j-1];
        a[j-1] = a[j];
        a[j] = temp;
    }
}
```

RTC : O(n)
SC : O(1)

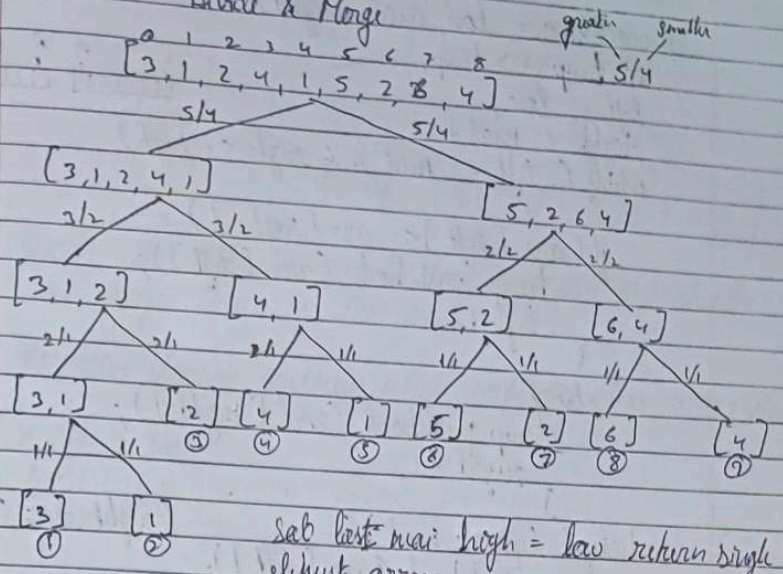
	max inner loop	
i = 1	1	Total : 1+2+3+ ... +n-1
i = 2	2	= (n+1)n / 2
i = 3	3	
⋮	⋮	
i = n-1	n-1	= n²+n / 2

→ worst : O(n²)
→ avg : O(n²)
→ best : O(n) already sorted list. (Total : 1+1+ .... +1 = n-1)

→ In place : No new spaces are created
→ Stable : relative order is maintained
if it had (j >= 0 && A[j] >= temp) = unstable

(22) Merge Sort (Striver)

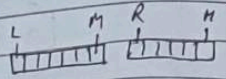
Divide & Merge



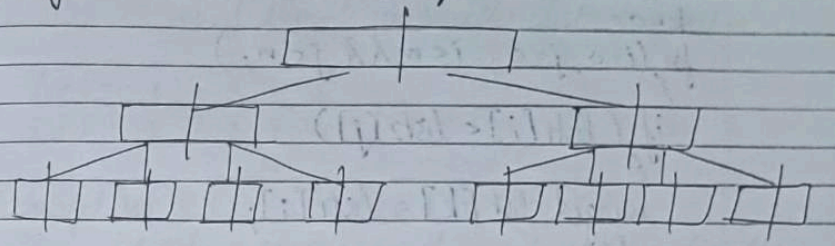
sab list me mai high = low return single element array

```
mergesort (arr, low, high) {
    if (high <= low)
        return arr[low].
    mid = (high + low)/2.
    mergesort (arr, low, mid);   // left para
    mergesort (arr, mid+1, high);  // right half
    merge (arr, low, mid, high);  // merge L&R after R
}
```

```
merge(&arr, low, mid, high){
  vector<int> temp;
  left = low;
  right = mid+1;
  while (left <= mid && right <= high)
    {
    if (arr[left] <= arr[right]) {
       temp.pushback(arr[left]);
       left++;
    }
    else {
       temp.pushback(arr[right]);
       right++;
    }
  while (left <= mid){          // right exhaust ho gyi
    temp.pushback(arr[left]);
     left++;
  }
  while (right <= high){        // left exhaust ho gyi
    temp.pushback(arr[right]);
     right++;
  }
}

mergesort(arr, low, high){
   if (high <= low)
      return arr[low];
   mid = (high+low)/2;
   mergesort(arr, low, mid);
   mergesort(arr, mid+1, high);
   merge(arr, low, mid, high);
}
```



(12) Merge sort: (Divide & conquare)



Divide the list into 2 parts sort each part individually and merge them in the end.

Recursive division continues when each list has 0 or 1 elmt

merge 2 lists:

list 1: 1, 6, 9, 10 → sorted

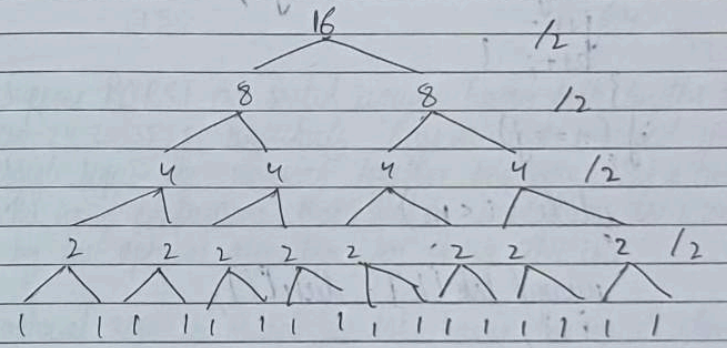
list 2: 2, 4, 8, 13 → sorted

if (i > j); else (j > i)

compair of i & j smallest to be put 1st in the list (bigger one)

1st 2nd 3rd 4th 5th 6th 7th 8th

merged list: 1, 2, 4, 6, 8, 9, 10, 13



$16 = 2^4$

$\log_2 16 = 4$

TC: O(n log n)

```
k=0
for(i=0, j=0; i<m && j<n;)
{
 if (list1[i] < list2[j])
 {
  merged_list[k] = list1[i];
  i++;
  k++;
 }
 else
 {
  merged_list[k] = list2[j];
  j++;
  k++;
 }
 if (m == i)
 {
  while (j<n)
  {
   merged_list[k] = list2[j];
   j++;
   k++; }
  }
 if (n==j)
 {
  while (i<m)
  {
   merged_list[k] = list1[i];
   i++;
   k++;
  }
 }
```

→ best case : no. of comparison to merge 2 sorted list = no. of elements in smaller list ⇒ min(m,n)

list 1 : 1, 2, 3, 4, 5
list 2 : 6, 7, 8, 9, 10
(L_1 = list 1, L_2 = list 2)
merged list : 1, 2, 3, 4, 5, 6, 7, 8, 9, 10

i.e only 4 comparisons of L_1 only and L_2 directly added without any comparisons.

→ worst case : no. of comparisons ⇒ (m+n-1)

list 1 : 1, 3, 5, 7 (i i i i)
list 2 : 2, 4, 6 (j j j)
no. of comp = 4+3-1 = 6
merged list 1, 2, 3, 4, 5, 6, (7) — added without comparisons (i++ j++ i++ j++ i++ j++ i++)

Eg: Consider sorted lists of size 64 & 87 elements. max and min no. of comparisons required to ~~merge~~ merge the lists into 1 sorted list.

max = 64+87-1 = 150
min = min(64, 87) = 64

Eg: Suppose P Q R S T are sorted sequences having ~~both~~ lengths ~~20, 24, 30, 35~~ 20, 24, 30, 35, 50 respectively. They are to be merged into a single sequence by merging together two sequences at a time. The no. of comparisons that will be needed in the worst case by the optimal algorithm for doing this is.

optimal algo ⇒ search for lists having min no. of elements so that min worst is obtained.

(no. of elements after merge)

P 20] ML1(44)
Q 24]
R 30] ML2(65)
S 35]
T 50 L1(50)

ML3(94)] ML4(159)
ML2(65)]

no. of comp = 43+64+93+158 = 358

Eg: Merge sort example:

19, 16, 10, 12 | 9, 1, 4, 7

19, 16 | 10, 12 | 9, 1 | 4 7

19 | 16 | 10 | 12 | 9 | 1 | 4 | 7 — they all are individually sorted

16, 19 | 10, 12 | 1, 9 | 4, 7

10, 12, 16, 19 | 1, 4, 7, 9

1, 4, 7, 9, 10, 12, 16, 19 — 4 comparisons L2 as this

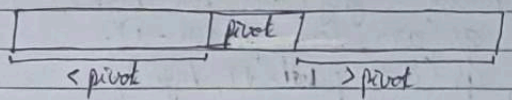
SC = O(n)
RTC = O(n log n) in all cases.
In place: no as requires new array for storage.
Stable: maintains relative position of elements

log n level of divide and merge * n no. of comparisons

★ Merge sort on linked list:
RTC = O(n log n)
SC = O(1)

(2) Quick sort: (better in SC than merge)

(LHS: smaller elements) Pivot (RHS: greater elements)
w.r.t pivot — w.r.t pivot



Eg: | 4 | 6 | 2 | 5 | 7 | 9 | 1 | 3 |

pivot can be any i) random in entire array
ii) 1st or last

but we generally go for 1st element as pivot:
→ i = low
j = high

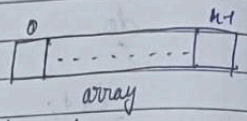
→ jab j & i crosses each other that position is where assumed pivot is to be swapped with j th element

(24) Queue or fifo list (FIRST IN FIRST OUT)

A linear data structure in which insertion can be done from one end (rear end) and deletion is done from other end (front end)

Insertion in the queue: Enqueue
Deletion in the queue: Dequeue

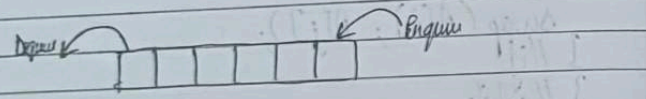
→ Implimentation using array:



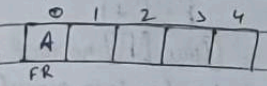
2 indexes
front = stores index of 1st inserted element
rear = stores index of last inserted element

Initially when queue is empty [front = rear = -1]

when empty queue mai enqueue karte hai toh vo 1st element hi 1st and last hai agar mai aur element daalta hun ek line ke liye toh mai hi phela and last hunga



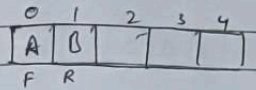
1) Enqueue ('A') assuming n = 5



F = R = 0

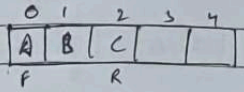
R = R+1;
Queue[R] = item;

2) Enqueue ('B')



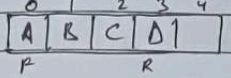
F = 0
R = 1

3) Enqueue ('C')



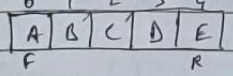
F = 0
R = 2

4) Enqueue ('D')



F = 0
R = 3

5) Enqueue ('E')

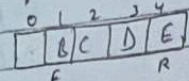


F = 0
R = 4

6) Enqueue ('G') : overflow

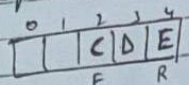
★ (Front == 0 && Rear == n-1) ★

7) Dequeue (): F = 1, R = 4, f = f + 1



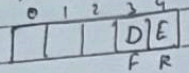
Queue[F] no mad index is moved so no accessibility to A

8) Dequeue ()



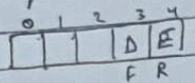
F = 2, R = 4

9) Dequeue ()



F = 3, R = 4

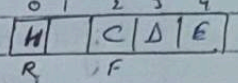
10) Enqueue (H) — For this linear queue its not possible



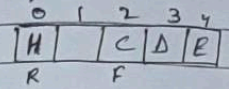
F = 3, R = 4

(13) Circular queue.

Proper utilization of

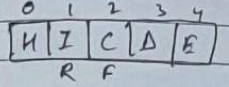


10) Enqueue (H)



F = 2, R = 0

11) Enqueue (I)

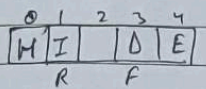


F = 2, R = 1

12) Enqueue (J) overflow tab jab rear ka next front ho jaye circularly ya rear ka next front ho jaye linearly done mai overflow hoga

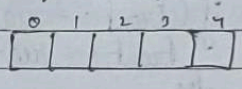
if (front == (Rear + 1) mod n)
common condition for both linear and circular

13) Dequeue ()



F = 3, R = 1

17) Dequeue ()

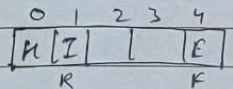


F = -1, R = -1

* enqueue ke case mai jab pehla insertion hota hai tab F & R both are updated otherwise (R++) hota jayega

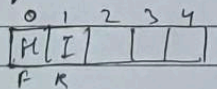
* dequeue ke case mai (F++) hota jayega but when last element is deleted then both F & R are updated

14) Dequeue ()



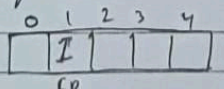
F = 4, R = 1

15) Dequeue ()



F = 0, R = 1

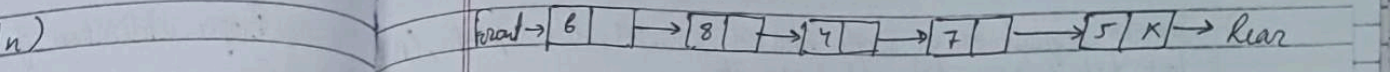
16) Dequeue ()



F = 1, R = 1

```
→ Enqueue (Queue [], Front, Rear, item)
{
  if (Front == (Rear + 1) mod n)
    printf ("Overflow");

  if (Front == Rear == -1) {
    Front = Rear = 0;

  else
    Rear = (Rear + 1) mod n;

  Queue [Rear] = item;
}

→ Dequeue (Queue [], Front, Rear)
{
  if (Front == rear + 1) {
    printf ("underflow");
    return;
  }
  if (Front == rear)
    Front == rear = -1;
  else
    Front = (Front + 1) mod n
}
```

Implementation using linked list



Enqueue ⇒ Insertion from rear ⇒ O(1)
Dequeue ⇒ Deletion from front ⇒ O(1).

Insertion from front ⇒ O(1)
Deletion from rear ⇒ O(n)

→ Other functions

QueueFront(): Return front element of queue without any change in queue

QueueRear(): Return rear element of queue without any change in queue

isEmptyQueue():
- True : Queue Empty
- False : Otherwise

Eg: What is the content of queue after following operations on an empty queue?

Enqueue (X)
Enqueue (R)
Dequeue ()
Enqueue (D)
Enqueue (E)
Dequeue ()

^{3}D ^{6}D
^{1}E ^{2}E ^{4}E ^{5}E
X R D E

Eg: What is the complexity of enqueue and dequeue if the queue is implemented using a singly null terminated linked list when only address of the 1st node is given for linked list?

a) O(1), O(1)
b) O(1), O(n)
c) O(n), O(1)
d) O(n), O(n)

Insertion O(n)
Deletion O(1)

Last ka address nahi hai to pura traverse karna
list traverse karne
logic but deletion from front is constant

- Applications of queue

1) Serving request on a single shared resource, like a printer, CPU task scheduling, disk scheduling etc.
2) When data is transferred asynchronously btw two processes Eg IO buffers, pipes, file IO, etc.
3) Call center phone system uses queue to hold people.
4) Handling of interrupts in real time system.
5) BFS (Breadth First Search)

(26) Doubly ended queue:
The queue in which deletion and insertion can take place from both the ends

→ Types of double ended queue
i) Input restricted DEQ : input restricted to standard end (rear)
ii) Output restricted DEQ : output restricted to standard end (front)

(28) Priority queue : implementation

Handle priority at insertion time
Handle priority at deletion time

Insertion:
Insert the element based on the priority in such a way that the elements are always arranged in decreasing order of priority from front to rear. O(1)

Deletion:
Delete the front element (which is the highest priority element)
O(n)

→ Priority queue : implementation

Insertion : insertion element : O(1)

Deletion : Find the element with least priority and delete it. O(n)

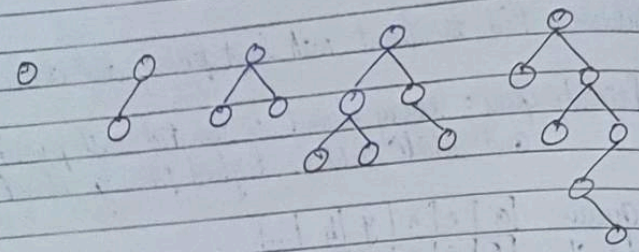
→ Priority queue : every element is inserted with priority attached and while deletion, highest priority element is deleted

queue	a	e	x	y	b	...
priority	5	6	2	9	3	...

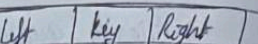
→ Priority queue : can be used to implement

queue
stack

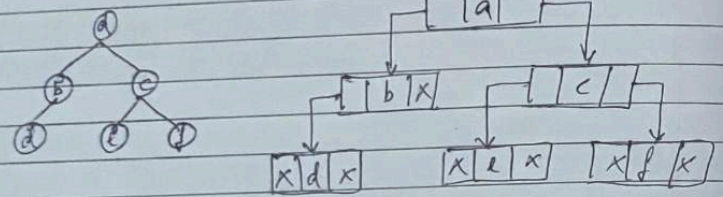
(29) Binary Tree: A tree in which each node has maximum of 2 children only, minimum can be zero.



→ Linked representation:

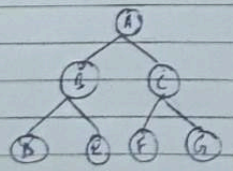



```
Struct BTnode
{
  char key;
  struct BTnode *Lchild;
  struct BTnode *Rchild;
};
```




```
if ((t->Lchild == null) && (t->Rchild == null))
  printf("t points to a leaf node");
```

Eg: A binary tree T has 4 leaves. The number of nodes in T having 2 children are?



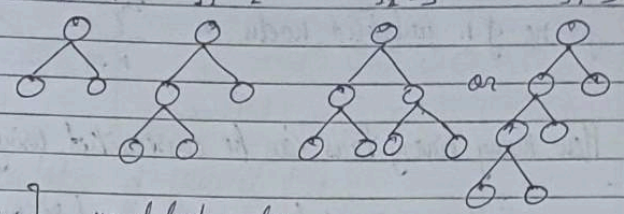
A → BC
B → DE, C → FG
(3)

(30) Handshaking lemma:

I_2	0	1	2	3
L	1	2	3	4

I_2 = internal nodes with 2 children

$I_2 = 0$ $I_2 = 1$ $I_2 = 2$ $I_2 = 3$ $I_2 = 3$



$[L = I_2 + 1]$ no. of leaf nodes

$[I = I_1 + I_2]$ total no. of nodes (internal + external)

$N = L + I$

$= I_2 + 1 + I_1 + I_2$

$[N = I_1 + 2I_2 + 1]$

Eg: In a binary tree, the no. of nodes of degree 1 is 11, and the no. of nodes of degree 2 is 23. Total no. of nodes in the binary tree is?

degree = no. of children

$I_1 = 11$

$I_2 = 23$

$N = 11 + 2(23) + 1$

$N = 58$

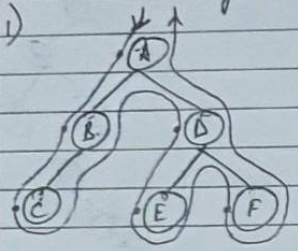
(31) Tree Traversal: recursive node

Preorder: n L R pahile node ko
Inorder: L n R bichi mai node ko
Postorder: L R n last mai node ko

```
void preorder (struct BTnode *t)
{
  if(t)
  {
    printf("%d", t->data);
    preorder(t->leftchild);
    postorder(t->rightchild);
  }
}
```

→ Dot method for tree traversal:

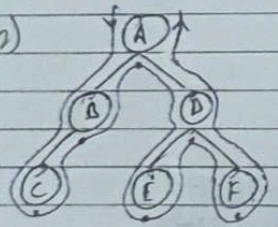
1)



Preorder: left side dot

A B C D E F

2)

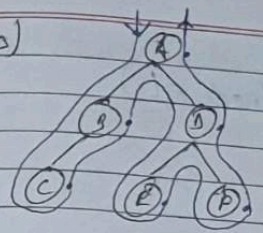


Inorder: center sided dot

C B A E D F

★ General traversal: Left is prioritized.
★ Converse traversal: Right is prioritized

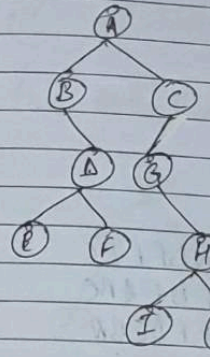
3)



Postorder: right side dot

C B E F D A

Eg:



Preorder: A B D E F C G H I J
Inorder: B E D F A G I H J C
Postorder: E F D B I J H G C A

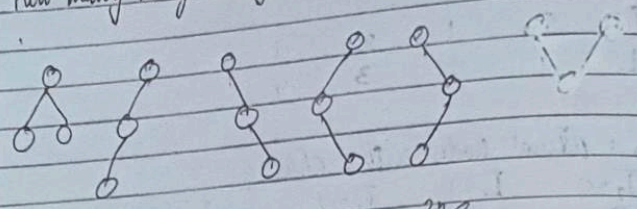
→ Observation from traversal
First symbol of preorder traversal is always root.
Last symbol of postorder traversal is always root.

Eg: Identify the inorder, preorder, postorder traversal

1) G H F E A P K D C B X M postorder
2) E G F H A P M C K D X B inorder
3) M P A E F G H X C D K B preorder

(32) Converse order traversal:
1) Converse Preorder n R L
2) Converse Inorder R n L
3) Converse Postorder R L n

Eg: How many no. of binary trees can be constructed using 3 unlabled nodes?



for no. of n unlabled nodes = $\frac{^{2n}C_n}{n+1}$

Eg: How many binary trees can be constructed using 3 distinct keys?

$\frac{^{2n}C_n}{n+1} \propto n!$

n! se un tree ko arrangment

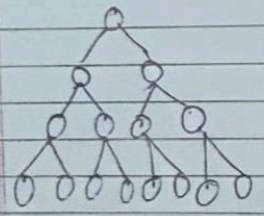
A B C	B A C	C A B
A C B	B C A	C B A

$\frac{^{3\times3}C_3}{3+1} \propto 3!$

$\frac{^{6}C_3}{4} \propto 3! \Rightarrow \frac{6\times5\times4}{3\times2\times1} \propto \frac{1}{4} \times 3\times2\times1$

$= 30$

Eg: minimum and maximum no. of nodes in a binary tree on level number L?

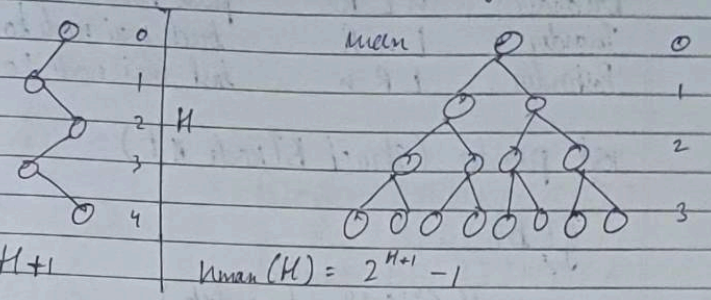


L_0	1
L_1	2
L_2	4
L_3	8

2^L = max no. of nodes

1 = min no. of nodes

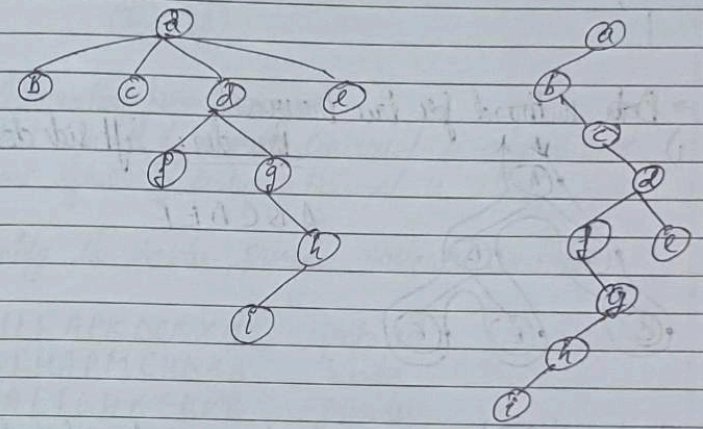
Eg: Minimum and maximum no. of nodes in a binary tree of height H? Note: Height of tree with single node is zero.



$n_{min} = H+1$

$n_{max}(H) = 2^{H+1} - 1$

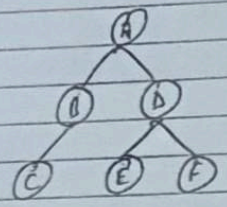
→ BT representation of general tree:
Leftmost child and Right sibling representation.




```
struct LMCRS node
{
   char key;
   struct LMCRS * Leftmostchild;
   struct LMCRS * Rightmostsibling;
};
```

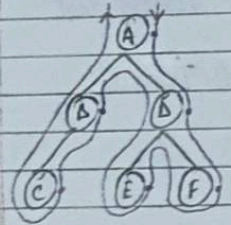
```
void converspreorder (struct BTnode *t)
{
  if (t)
  {
    printf ("%d", t->data);
    converspreorder (t->Rchild);
    converspreorder (t->Lchild);
  }
}
```

Eg:



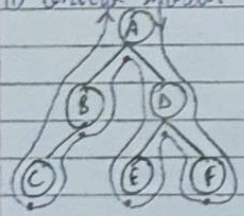
ConPre = ADFEBC
ConIn = FDEABC
ConPost = FEDCBA

i) Converse Preorder =



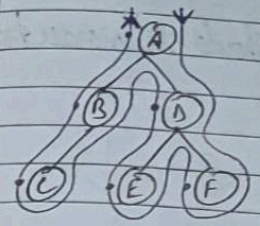
ADEEBC

ii) Converse Inorder



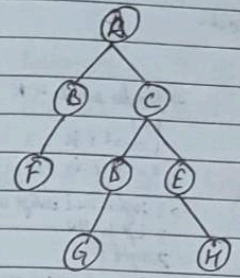
FDEABC

iii) Converse Postorder:



FEDCBA

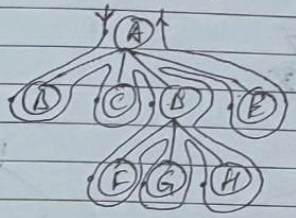
(32) Level order traversal



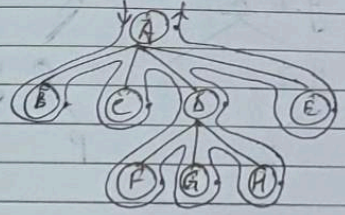
Travel will happen level by level but the order can be any BC or CB

a, bc, fde, gh ✓
a, cb, edf, hg ✓
a, cb, fed, hg ✗

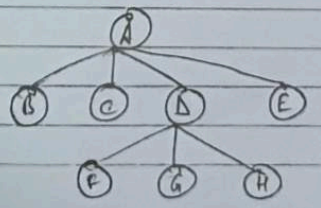
(33) Tree traversal for general tree



i) Preorder
ABCDFGHE



ii) Postorder
BCFGHDEA



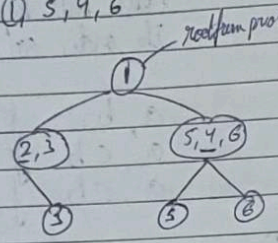
iii) Inorder
dot kab lagana hai that will be provided.

(32) Constructing tree using traversals

Minimum 2 traversal required one should be inorder traversal.

[Preorder, Inorder]
[Postorder, Inorder]
root — left and right subtree

Eg Pre: 1, 2, 3, 4, 5, 6
In: 2, 3, 1, 5, 4, 6



→ Inorder mai L n R
n=1
Left subtree 1 is 2,3
Right subtree 1 is 5,4,6

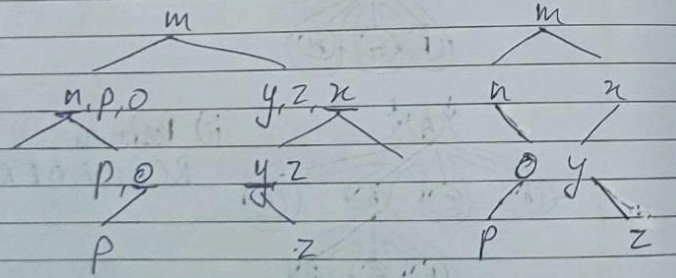
→ 3 right mai isliye coz as per inorder
2 left is NA
2 right is 3

→ 4 is root as per preorder

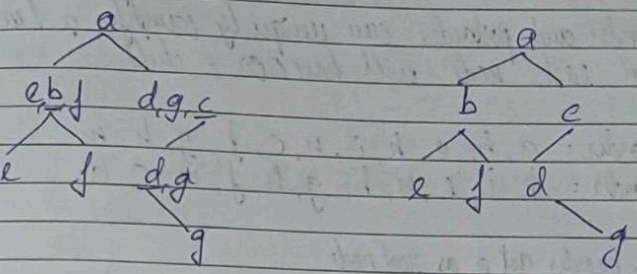
→ 6 is right of 4 as per inorder

→ 5 is left of 4 as per inorder

Eg Pre: (m), (n), o, p, (x), y, z
In: n, p, (o), (m), (y), z, (x)



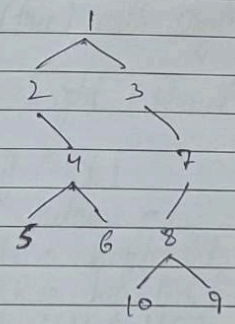
Eg: Post: e, f, b, g, d, c, a
In: e, b, f, a, d, g, c



Eg: Con Postorder: 9, 10, 8, 7, 3, 6, 5, 4, 2, 1
Con Inorder: 7, 9, 8, 10, 3, 1, 6, 4, 5, 2

Converse a convert karne ke liye phle usko reverse kardo ie preorder

	Converse	General
C Post =	9, 10, 8, 7, 3, 6, 5, 4, 2, 1	1, 2, 4, 5, 6, 3, 7, 8, 10, 9 = Pre
C In =	7, 9, 8, 10, 3, 1, 6, 4, 5, 2	2, 5, 4, 6, 1, 3, 10, 8, 9, 7 = G In

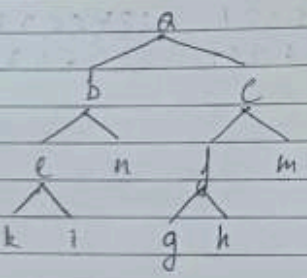


No unique tree without inorder because inorder traversal can identify uniquely that a subtree is left one or right one

Preorder and postorder can uniquely identify a tree only when each node will have 0 or 2 child

Eg: Preorder: a, b, e, k, i, n, c, f, g, h, m
Postorder: k, i, e, n, b, g, h, f, m, c, a

from Preorder got a as root node
b as root node for
e as node for k, i;
i, k as node for leaves.
c as root node equal to b
f pre mai pehle aya to vo node unke under g, h
then comes g, h equal value



The unique tree when have knowledge of 2 or 0 child of node

Eg: What does the following function return if called for root of a binary tree?

```
int getvalue (struct BTnode *t)
{
  int value = 0;
  if (t);
  {
    if (((t->LC) == Null) && (t->RC == Null))
      value = 1;
    else
      value = value + getvalue (t->LC) + getvalue (t->RC);
  }
  return value;
}
```

(34) Height of tree

i) H (tree) with single node = 0 (here empty tree height = -1)

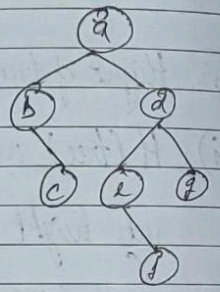
```
int height (struct BTnode t)
{
  if (!t)
    return -1;
  HL = height (t->LC);
  HR = height (t->RC);
  if (HL > HR)
    return HL + 1;
  else
    return HR + 1;
```

ii) H(tree) with single node = 1 or H(empty tree) = 0

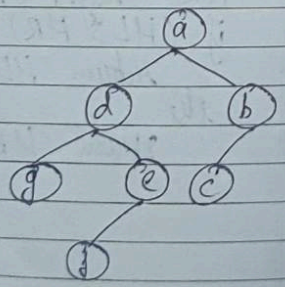
```
int height (struct BTnode *t)
{
  if (!t)
    return 0;
  HL = height (t->LC);
  HR = height (t->RC);
  if (HL > HR)
      return HL+1;
  else
      return HR+1;
}
```

Eg: Consider following routine on binary tree:

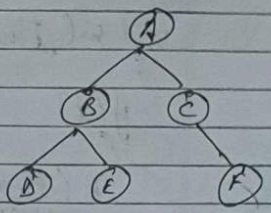
```
void do (struct BTnode *t)
{
  if (t)
  {
    do (t->LC);
    do (t->RC);
    swap (t->LC, t->RC);
  }
}
```



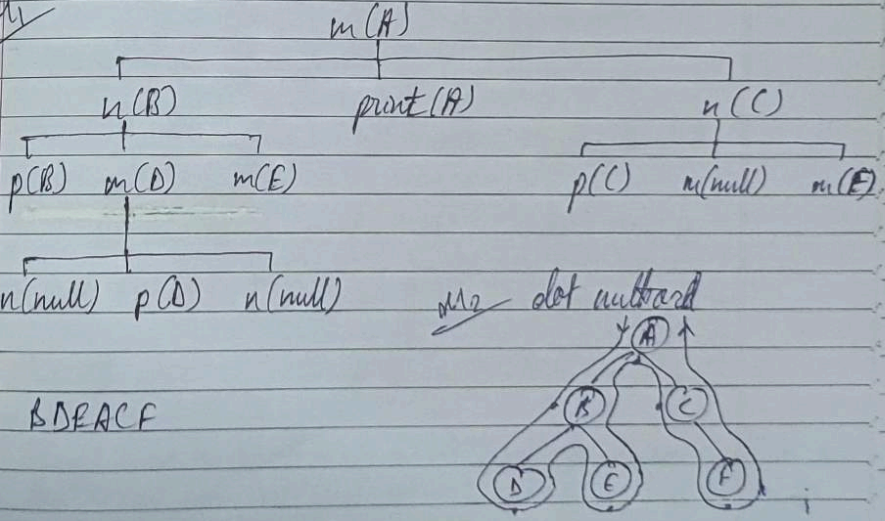
what will be the final tree after the above function is executed for given tree?



Eg: Indirect recursion:
Get the value of m(t) for the given tree?
Consider the following functions:



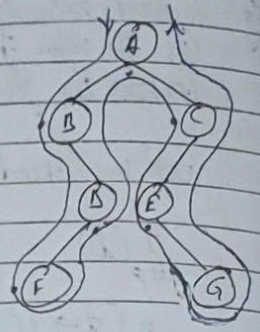

```
void m (struct BTnode *t) // inorder
{
  if (t)
  {
    n (t->LC);
    printf ("%c", t->data);
    n (t->RC);
  }
}

void n (struct BTnode *t) // preorder
{
  if (t)
  {
    printf ("%c", t->data);
    m (t->LC);
    m (t->RC);
  }
}
```



BDEACF

not unique

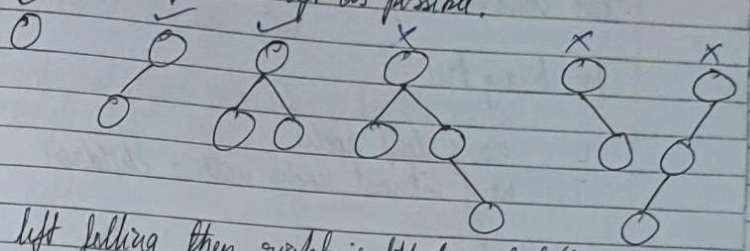
Eg: Get the value of m(t) for given tree?
Consider the following functions




```
void m(struct BTnode *t)
{
 if(t)
 {
  m(t->LC);
  printf("%c", t->data);
  m(t->RC);
 }
}

void m(struct BTnode *t)
{
 if(t)
 {
  printf("%c", t->data);
  m(t->LC);
  m(t->RC);
 }
}
```

(35) Complete binary tree

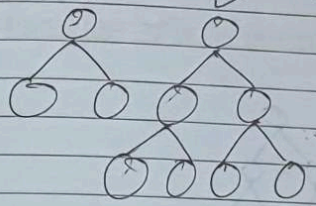
A complete binary tree is binary tree in which every level except possibly the last, is completely filled and all nodes are as far as left as possible.



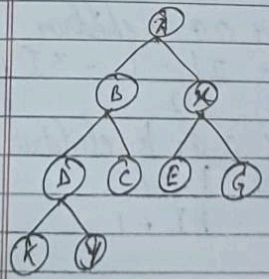
while left falling then right is filled and falling of level then next level left is to be filled

(36) Strictly binary tree:

All levels must be completely filled



(36) Array representation of CBT



root stored at index no. (1)

for a node at index no. (i)
LC of node at index no. (2i)
RC of node at index no. (2i+1)

1	2	3	4	5	6	7	8	9
A	B	X	D	C	E	G	K	Y

If general binary tree ko array mein ke store nahi kiya coz general BT mein kisi level pe max nodes nahi hote but array mein store is useful as memory for general but CBT is useful as array.

(38) Full binary tree (even no. of children) $\leftarrow_2^0$

A full binary tree (sometimes proper binary tree or 2 tree) is a tree in which every node other than the leave has two children

for binary tree:

$$L = I_2 + 1$$

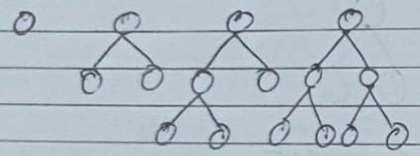
L : no. of leaf nodes
I_2 : no. of internal nodes with 2 children

for full binary tree:

$$L = I + 1$$

L : no. of leaf nodes
I : no. of internal nodes

$N = L + I$ no. of nodes in a full BT is always odd
$N = 2I + 1$
$N = 2L - 1$



- 3 ary tree: each node will have 0 or 3 children.

$$L = 2I + 1, \quad N = \frac{2L - 1}{2} = 3I + 1$$

- k ary tree: each node will have 0 or k children

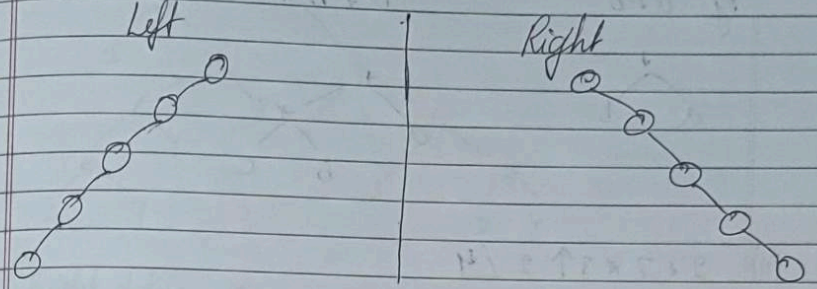
$$L = (k-1)I + 1, \quad N = I + L = kI + 1$$

(39) Left skewed binary tree

Every internal node has only left child.
Only single leaf node.

(40) Right skewed binary tree

Every internal node has only right child.
Only single leaf node.



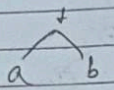
~~Expression tree : To represent a exp~~

(4) Expression tree

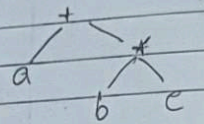
Expression can be represented as tree as per their priority order

operator → root
left operand → left child
right operand → right child

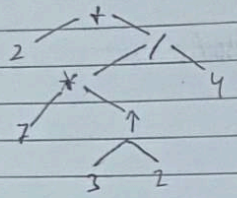
Eg: a+b



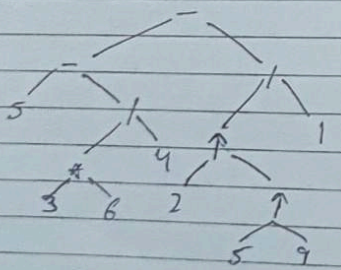
a+b*c



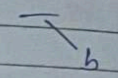
iii) 2+7*3↑2/4



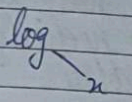
iv) 5-3*6/4-2↑5↑9/1



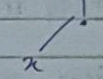
v) -b



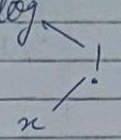
vi) log x



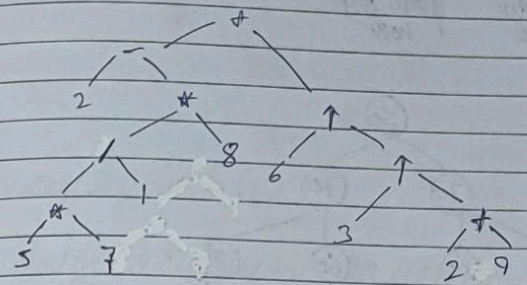
vii) x!



viii) log x!



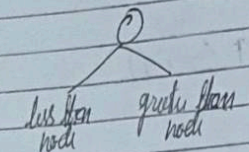
ix) 2-5*7/1*8+6↑3↑(2+9)



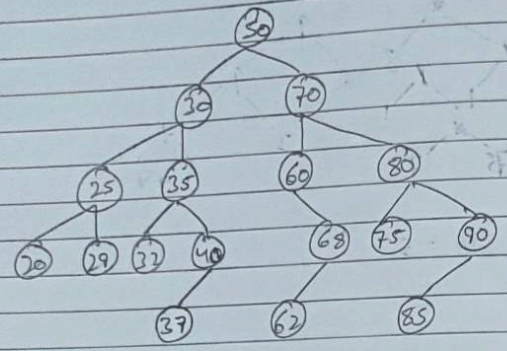
height of tree = 5

(42) Binary search Tree

Binary search tree whose internal node each stores a key greater than all the keys in the node's left subtree and less than those in its right subtree



Eg:



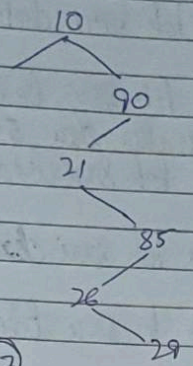
Search 32, 62

Search seq : 50, 30, 35, 32
: 50, 70, 60, 68, 62

```
struct BTnode *search_in_BST(int item, struct BTnode*t)
{
  while(t)
  {
   if (t->data == item)
     return t;
   if (item < t->data)
      t = t->lC
   else
      t = t->RC
  }
  return null;
```

Identifying valid search sequence.

Eg: Search 50 in 10, 90, 21, 85, 26, 29, 72, 27, 50



wrong seq due to (27)

Eg: Identify correct sequence for searching 80 in BST

(X) 9, 120, 12, 58, 88, 81, 86, 50, 84
(X) 198, 47, 111, 58, 71, 90, 65, 75, 54, 89
(✓) 15, 33, 46, 109, 94, 79, 82
(X) 17, 37, 49, 117, 96, 85, 15, 56, 80
(X) 20, 40, 90, 48, 68, 79, 81

(X) 9; 80 se chota hai toh 9 ke baad sari badi values honi chahiye

120: 80 se bada hai toh 120 ke baad sari chhoti values honi chahiye

12: 80 se chota hai toh 12 se sab badi

58: 80 se chota hai toh 58 se sab badi but 50 is smaller than sequence is wrong

198 : 80 se bada toh sari choti value after 198

47 : 80 se choti toh sari badi value after 47

111 : 80 se bada toh sari choti value after 111

58 : 80 se chota toh sari badi value after 58
but 54 smaller than 58

71 : 80 se chota toh sari badi value after 71

90 : 80 se bada toh sari ch[illegible] value after 90

65 : 80 se chota to sari bdi value after 65

15 : 80 se chota toh sari badi value after 15

33 : 80 se chota toh sari badi value after 33

46 : 80 se chota toh sari badi value after 46

109 : 80 se bada toh sari choti value after 109

94 : 80 se bada toh sari choti value after 94

79 : 80 se chota toh sari badi value after 79

82 : 80 se bada toh sari choti value after 82

17 : 80 se chota toh sari badi value after 17
but 15 is

37 : 80 se chota toh sari badi value after 37

49 : 80 se chota to

20 : 80 se chota toh sari badi value after 20

40 : 80 se chota toh sari badi value after 40

90 : 80 se bada toh sari choti value after 90

48 : 80 se chota toh sari badi value after 48

68 : 80 se chota toh sari badi value after 68

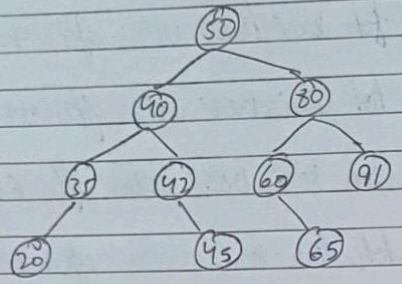
79 : 80 se chota toh sari badi value after 78

81 : 80 se bada toh sari choti value after 81

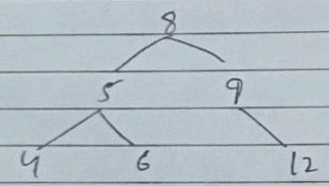
Eg: find 50 in

(A) 80, 73, 65, 45, 57, 49, 71, 82, 50
(B) 40, 45, 90, 80, 70, 60, 63, 56, 50
(C) 24, 34, 46, 98, 47, 80, 75, 61, 48, 50
(D) 28, 95, 29, 94, 37, 85, 89, 71, 43, 50
(E) 30, 39, 84, 81, 73, 67, 45, 59, 35, 50

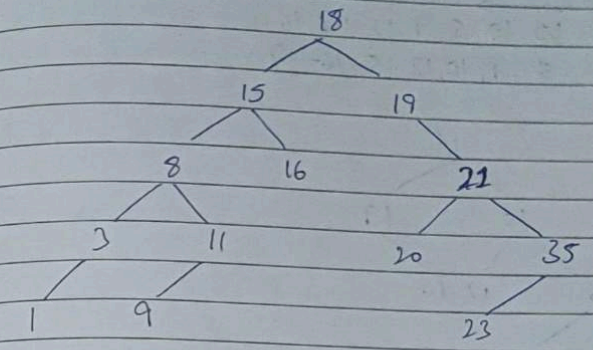
Eg: insert 45, 20, 63 in



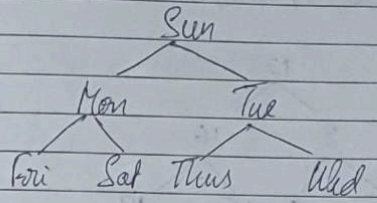
Eg: make BST using 8, 5, 9, 4, 12, 6



Eg: BST using : 18, 15, 8, 19, 21, 35, 23, 11, 9, 16, 3, 1, 20



Eg: Sunday, Monday, Tuesday, Wednesday, Thursday, Friday, Saturday in alphabetical.



(13) Deletion in BST

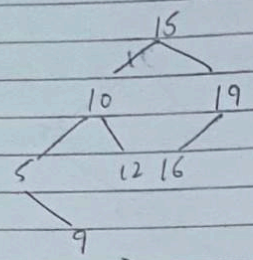
For deletion the key to delet will be given

Search the key in given BST and reach to the node where the key is present

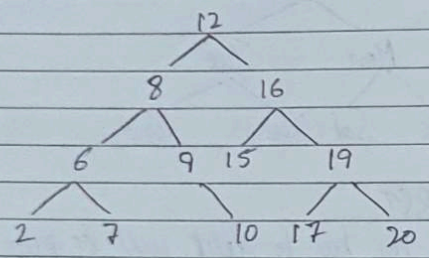
Find out how many children the node has.

(44) Inorder in BST

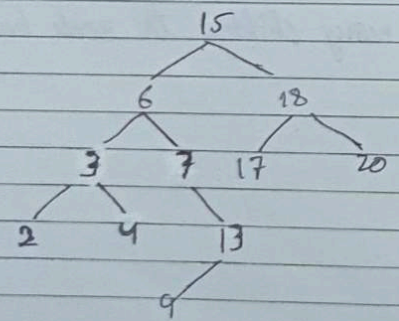
Eg: Preorder: 15, 10, 5, 9, 12, 19, 16
Inorder: 5, 9, 10, 12, 15, 16, 19



Eg: Preorder: 12, 8, 6, 2, 7, 9, 10, 16, 15, 19, 17, 20
Postorder: 2, 7, 6, 10, 9, 8, 15, 17, 20, 19, 16, 12



Eg: Postorder: 2, 4, 3, 9, 13, 7, 6, 17, 20, 18, 15
Preorder: 15, 6, 3, 2, 4, 7, 13, 9, 18, 17, 20

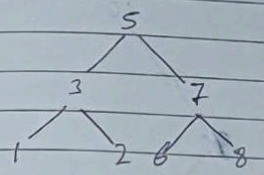


Eg: Which Preorder is correct of BST

A) 5, 3, 1, 2, 4, 7, 8, 6
B) 5, 3, 1, 2, 6, 4, 8, 7
C) 5, 3, 2, 4, 1, 6, 7, 8
D) 5, 3, 1, 2, 4, 7, 6, 8

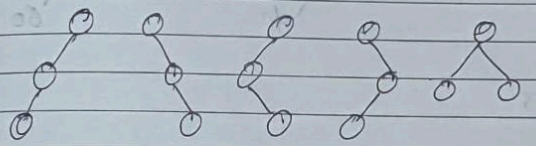
Pre : nLR

D)



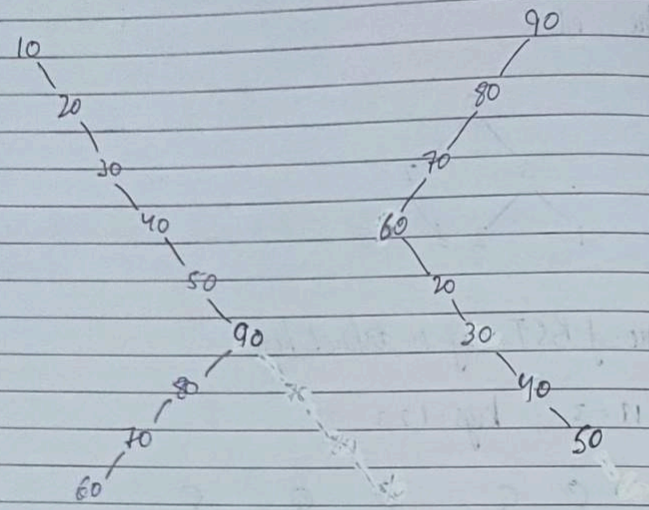
Eg: no. of BST using n distinct keys

n = 3 ; keys: 1, 2, 3



	n unlabled	n distinct
Binary Tree	$\frac{^{2n}C_n}{n+1}$	$\frac{^{2n}C_n}{n+1} \times n!$
Binary Search	—	$\frac{^{2n}C_n}{n+1} \times 1$

Eg: When searching for a key value 60 in a BST, nodes containing the key values 10, 20, 40, 50, 70, 80, 90 are traversed not necessarily in the same order given. How many different orders are possible in which these key values can occur on the search path from the root to the node containing the value 60?



$\frac{7!}{4!\,3!} = 35$

ya to 10 se start sari badi value right mai jao, 90 se start sari choti value left mai jao

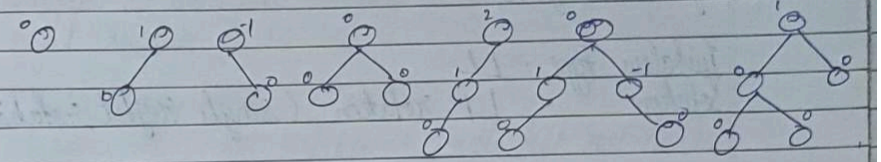
7 keys hai 7 key ki total combinations 7 factorial, unmai se 4 keys ke (10, 20, 40, 50) bare options eliminate karke sirf ek select kiya and same for remaining 3 keys

(18) AVL tree

→ used to get our worst case complexity of BST of O(n)
→ Self balancing BST

height balanced tree = for each node balanced factor should be between (1, 0, -1)

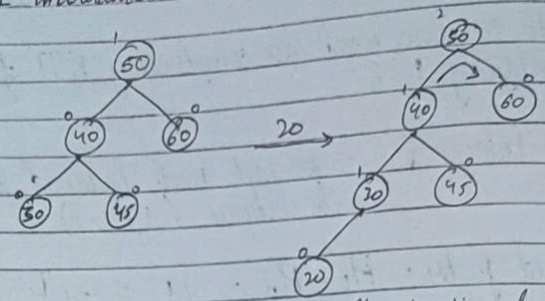
Balanced factor = $H_L - H_R = \{1, 0, -1\}$



Types of unbalances
1) RR right right
2) LL left left
3) RL right left
4) LR left right

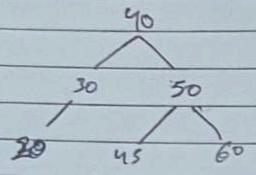
Tree has unbalances due to insertion and deletion of nodes

LL imbalance

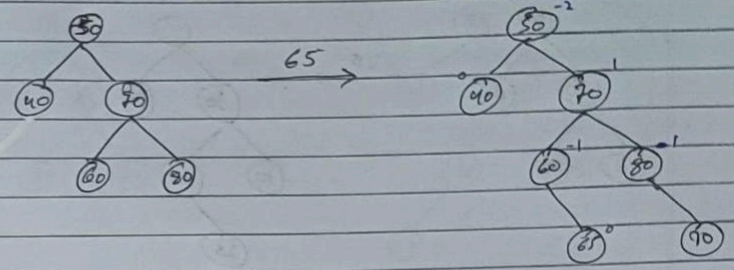


imbalance at 50 (BF = 2) due to the insertion of 20 left to left of subtree of 50

Imbalance type : LL
Solution : LL rotation (single right rotation)

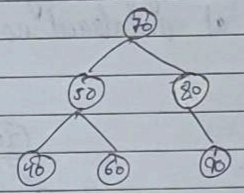


RR imbalance

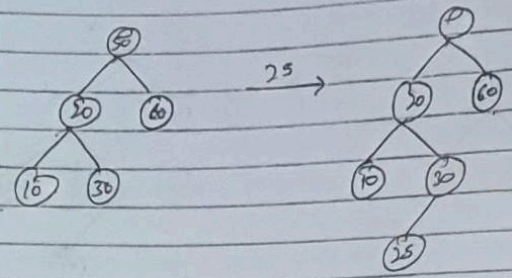


imbalance at 50 (BF = -2) due to the insertion of 90 right to right subtree of 50

Imbalance type : RR
Solution : RR rotation (single left rotation)

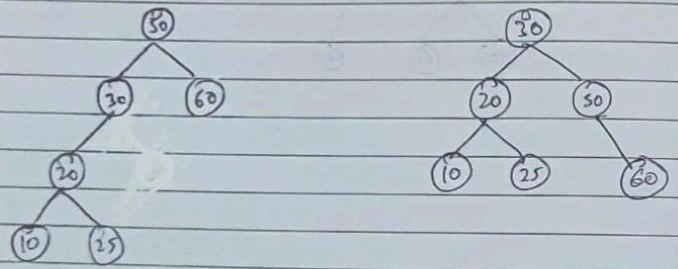


LR imbalance

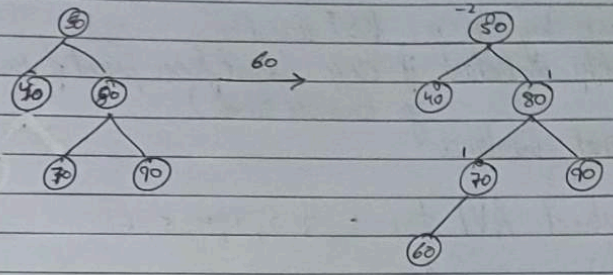


imbalance at 50 (BF = 2) due to insertion of 25 to left and right subtree of 50

Imbalance type : LR
Solution : LR rotation (right right → Left Left)
RR at left child of imbalanced node
LL at imbalanced node

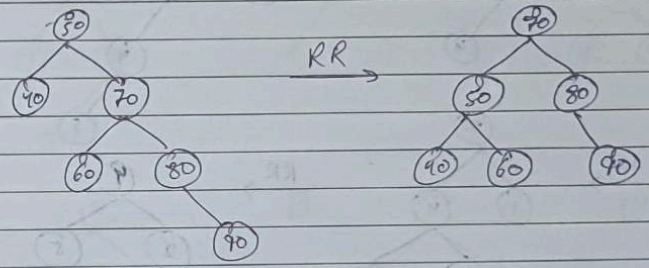


RL imbalance



imbalance at 50 (BF = 2) due to insertion of 60 to right and left subtree of 50

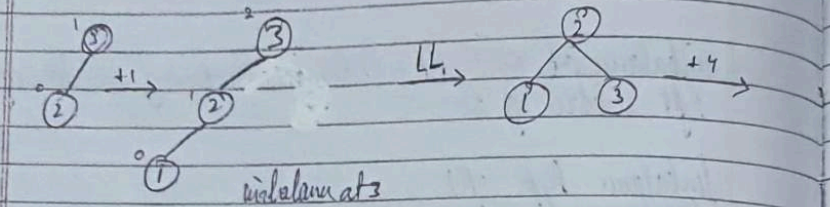
Imbalance type : RL
Solution : RL rotation
LL rotation at right child of imbalance node
RR rotation at imbalanced node

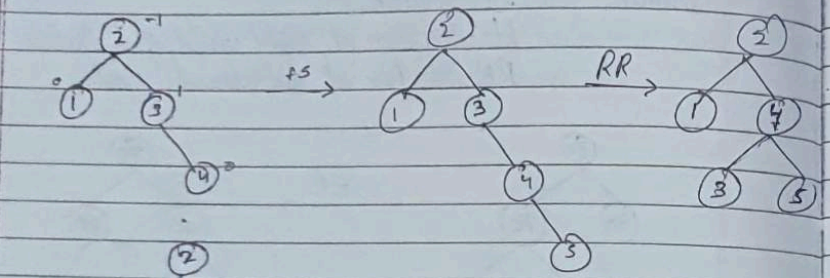


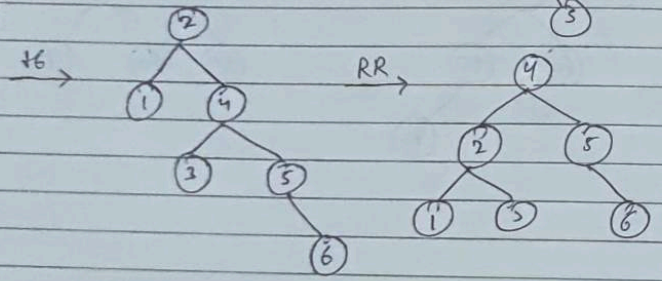
(46) AVL tree insertion:

1) Insert key as per BST insertion
2) Identify imbalance if any (start from parent of new node & go towards root)
3) Eliminate imbalance

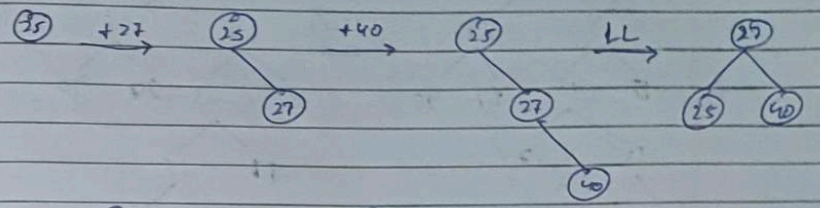
Eg: Construct AVL tree : 3, 2, 1, 4, 5, 6

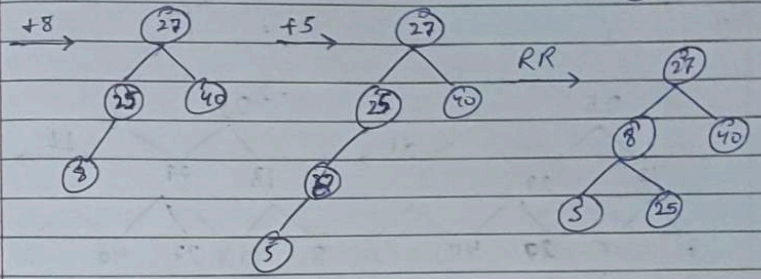


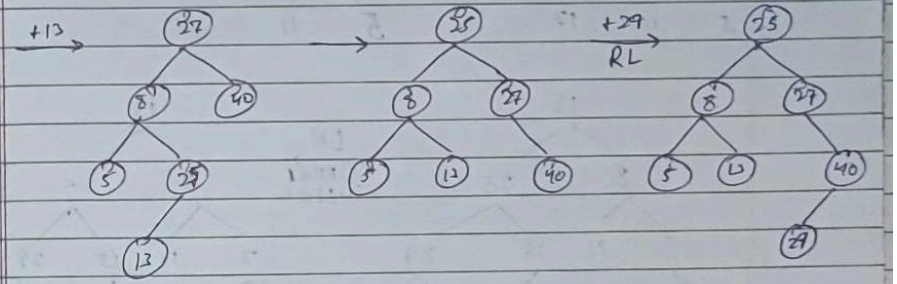


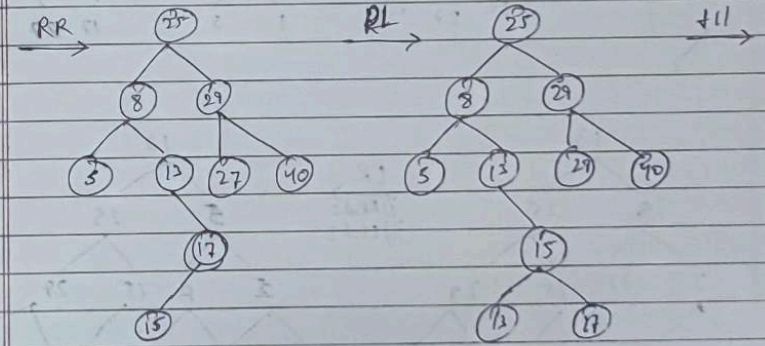


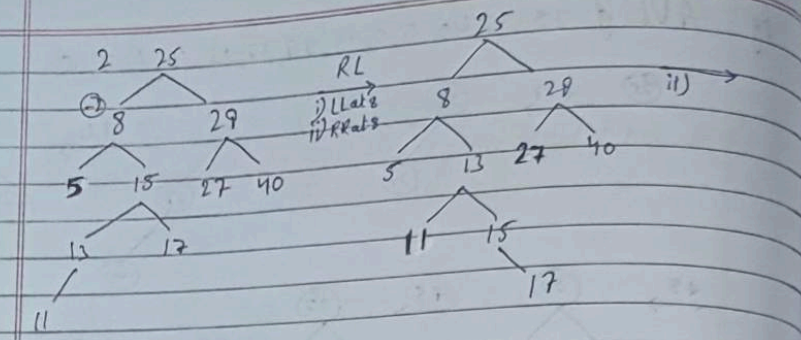
Eg: AVL of 25, 27, 40, 8, 5, 13, 29, 17, 15, 11, 1, 2, 6

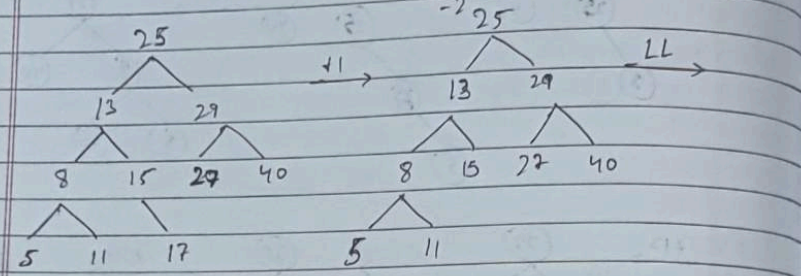


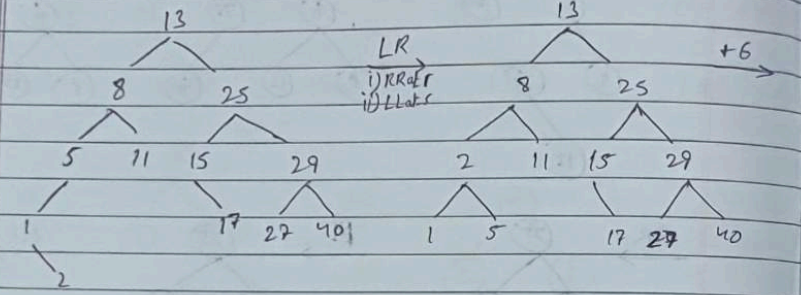


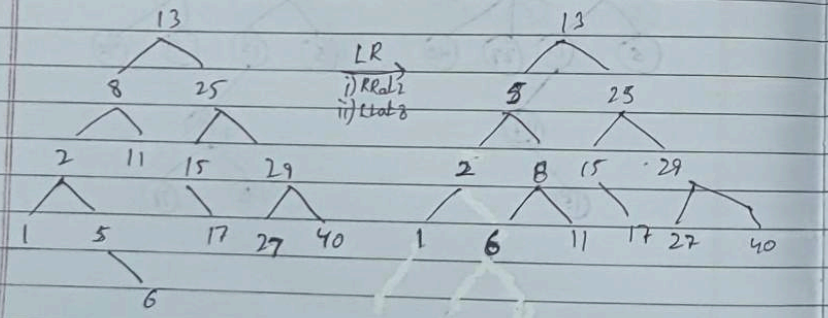




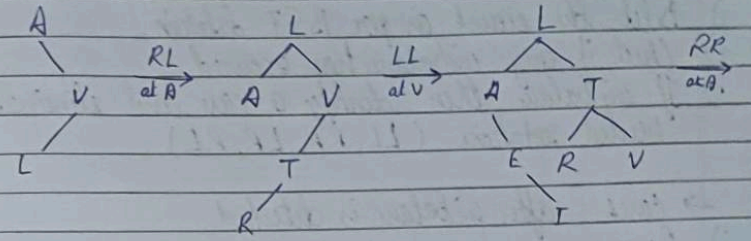


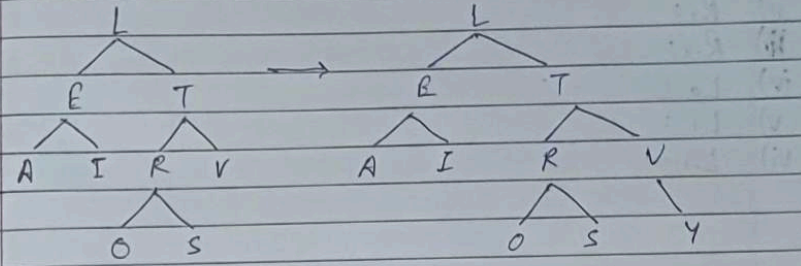






Eg: Create AVL tree : A, V, L, T, R, E, I, S, O, K, Y





(43) Heap

It is a tree based data structure which is essential an almost complete tree that satisfies the heap property.

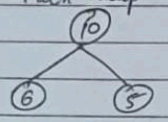
→ Properties of heap:

i) It should be a complete binary tree
ii) Max heap or Min heap property.

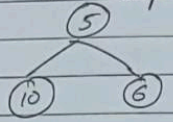
Max heap: The root node value must be greater than the nodes of the child.

Min heap: The child nodes must be greater than the root node

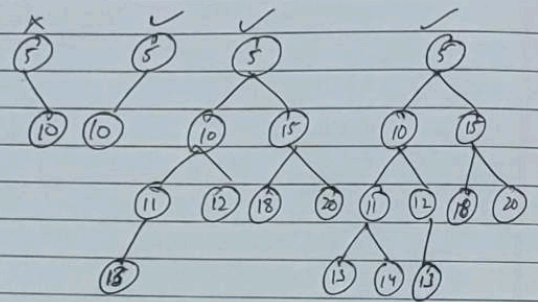
Max Heap



Max Heap

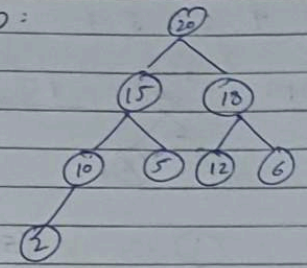


Eg: Min:



☆ indexing in CBT starts from 1 as

Eg: Max heap:



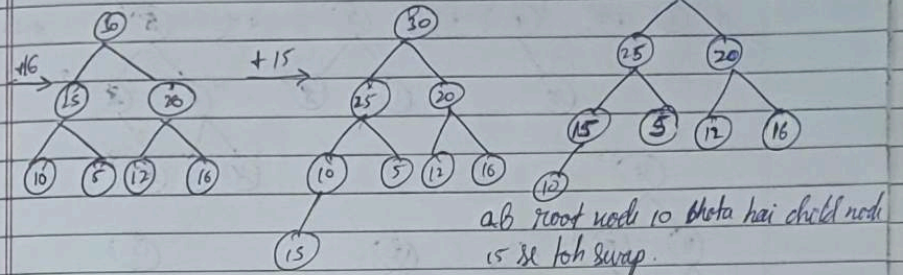
root at index = 1
left child on index = 2i
right child on index = 2i+1

-1	2	3	4	5	6	7	8
20	15	18	10	5	12	6	2

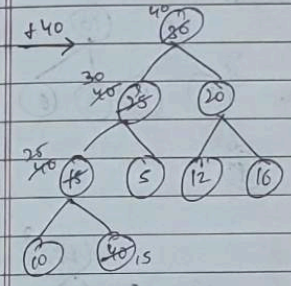
or

20, 15, 18, 10, 5, 12, 6, 2

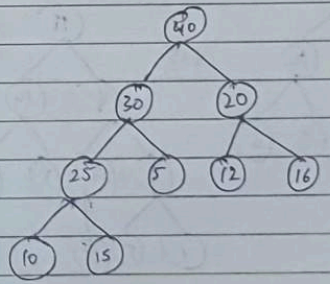
Eg: Insert: 16, 15, 40 in max heap:



ab root node 10 chota hai child node 15 se toh swap.

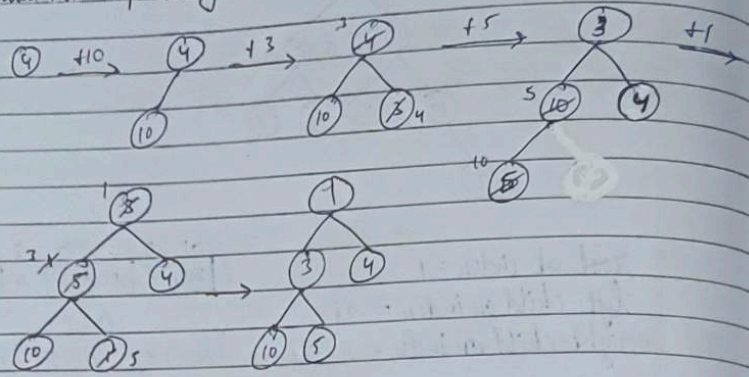


i) ab root node 15 chota hai child node 40 se toh swap
ii) then root node 25 chota hai child node 40 se to swap
iii) then root node 30 chota hai child node 40 se toh swap
40 goes on top

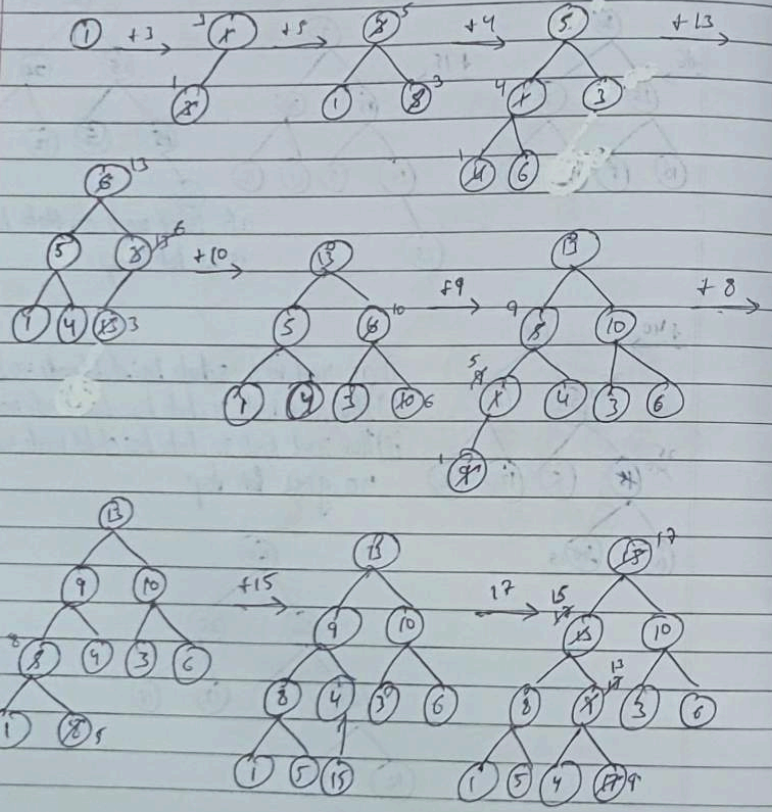


★ Search min element in max heap] O(n)
Search max element in min heap
★ Search min in min heap] O(1)
Search max in max heap

Eg: Build Min heap using : 4, 10, 3, 5, 1



Eg: Build Max heap using : 1, 3, 5, 4, 6, 13, 10, 9, 8, 15, 17

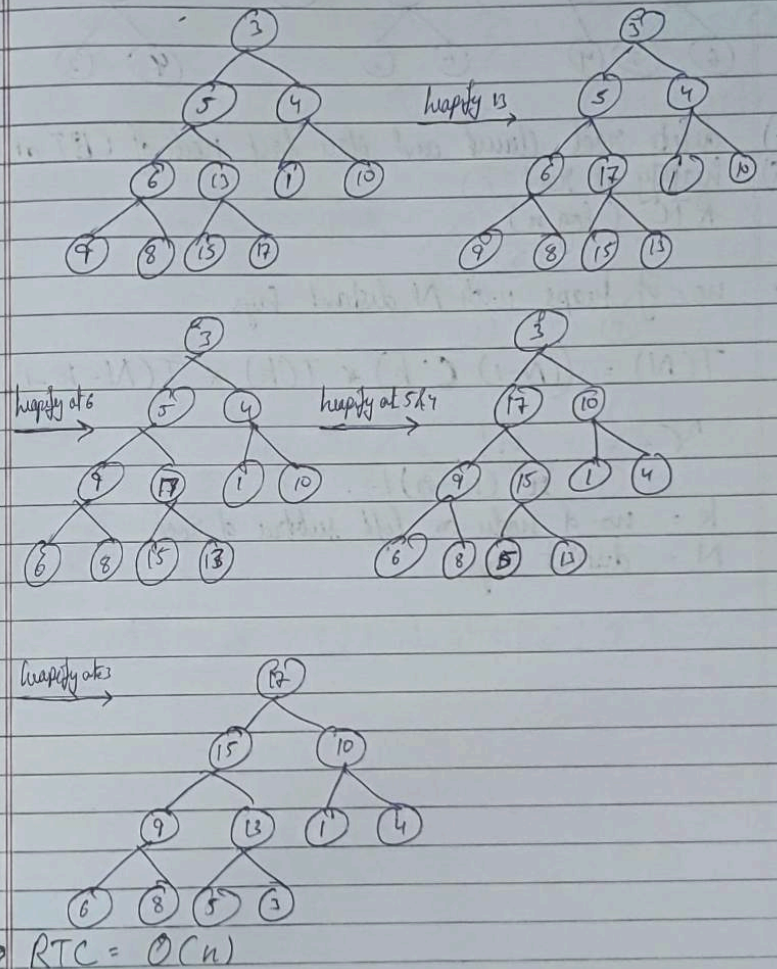


→ One insertion : O (log n)
→ n insertions : O (n log n)

(or) Heapify :

Eg: Built Max heap using : 1, 3, 5, 4, 6, 13, 10, 9, 8, 15, 17

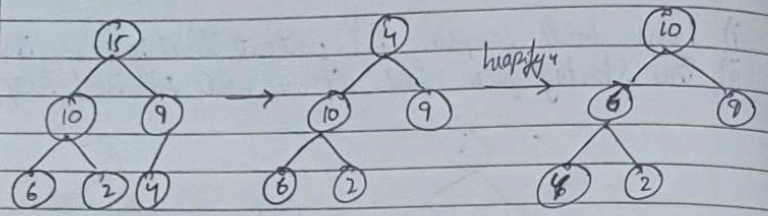
i) First built complete BT without thinking as per BST rules
ii) Then starting from last internal node till root heapify each.



→ RTC = O(n)

(50) Deletion in heap (as its priority class)
So when deletion happens only for a specific elemt only. minheap: min elemt ka deletion
maxheap: max elemt ka deletion
i.e delete root elemt

Eg: delete 15:



i) delete root elemt and place last node of CBT at root
ii) heapify at root
RTC (log n)

- no. of heaps with N distinct keys

$$T(N) = ((N-1) C\ k) * T(k) * T(N-k-1)$$

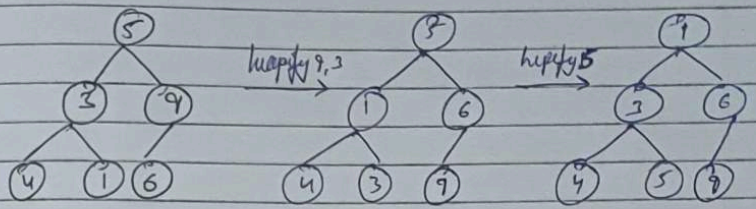
$$^nC_r = \frac{n!}{r!(n-r)!}$$

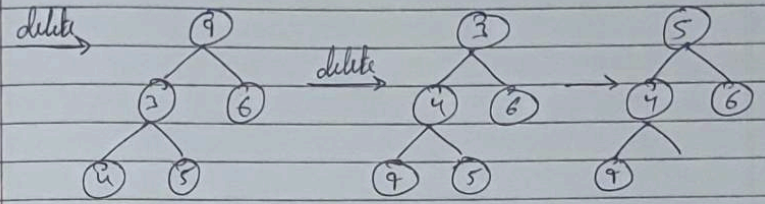
k: no. of nodes on left subtree of root
N: distinct keys

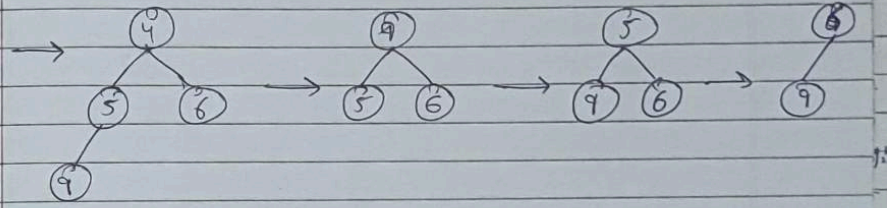
(51) Heap sort:

1) Ascending order:
i) Construct a min heap using given keys
ii) delete elemnts from heap one-by-one and insert those into a list

Eg: keys: 5, 3, 9, 4, 1, 6 (min heap for ascending)







→ 9 ⇒

1	3	4	5	6	9

* instructions means lots several things
* // : comments

25/6/23

C program

→ first.c

```
#include <stdio.h> // preprocessor directive

int main () { // execution of program starts from here
    printf ("Hello World");
    return 0;    // as in main int is given will return control back
} // end
```

cmd:

gcc first.c

- Variable: They are used to store value during prog execution.

```
a = 1;
b = 4.7;
c = 'A';
```

Their values can be changed.

- rules for naming variables
1) First character must be an alphabet or underscore (_).
2) No (commas, blank spaces) allowed
3) No special character other than (_)
4) Variable names are case sensitive.
5) Not start with number.

* its a case sensitive language.

- Constant: The values which cannot be changed.

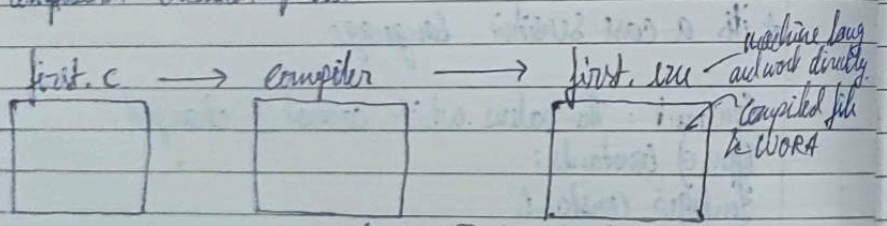
Types of constants:
Integer constants
Real constants
character constants (must be enclosed in single inverted comma)

* control + / : comment out whole selected
* String : "" ; char : ''
** Select karke run the only selected is executed
• keywords : They are the reserved words with predefined meaning to a compiler.

auto	double	int	struct
break	long	else	switch
case	return	for	typedef
char	register	goto	union
continue	short	return	unsigned
const	signed	float	void
default	sizeof	if	volatile
do	static	enum	while

→ rules
1) Every program execution starts from main () function.
2) All statements terminates with a semicolon.
3) Instructions are case sensitive
4) Instruction are executed in same order as they are written i.e top to bottom
5) Comments are overlooked by compiler.
// Single line
/* multi line */

Compilation execution process



gcc first.c -o first.exe
fir + tab => ./first.exe
(c mein dena hai to lag jayega)
(binary files is already exist WORK)

& : address of
% : format specifier

→ library functions : pre written functions used to do certain functions

1) printf ("This is %d", i);

2) %d for integer
3) %f for real value
4) %c for characters

%d, c, f : format specifier

int a = 4 printf (a) = 4
int b = 8.5 printf (b) = 8

* Recieving input from user :
In order to take inputs from user and assign it to a variable, we use "scanf" function

scanf ("%d", &i); # input from user
↳ This & is important

& is the "address of" operator and it means that the supplied value should be copied to the address which is indicated by variable i (to store the recived value)

☆ Size chart

char = 1	-128 to 127
short = 2	-32768 to 32767
int = 4	
long = 4	
float = 4	
double = 8	
long long = 8	

Operations & Instructions

C has a set of instructions to perform actions

→ Types of instructions
1) Type declaration instructions
2) Arithmetic instructions
3) Control instructions

1) Type declaration instruction : for the type of data type of the variable used.

```
int a ;
float b ;
```

Eg: int i=10, int j=i
float b=(i+j)/5

Eg: int a, b, c, d;
a=b=c=d=30; ⇒ each will get value as 30

2) Arithmetic instructions :

int i = (3*2)+1
(operators: *, +; operands: 3, 2, 1)

operands can be int / float etc
+, -, *, / are arithmetic operator

Eg: int b=2, c=3;
int z; z=b*c ✓
int z, b*c=z ✗

* snippets : store of user created code lines
boilerplate : for c language it is set of pre written codes which are basic necessary of a code. $1 : 1st cursor ; $2 = 2nd cursor arrival // tab to move

modular operator : %
→ gives remainder
→ cannot be applied on float
→ sign is same as the operator in the numerator
Eg : (5)%(-2) = +1
(-5)%(2) = -1

Note → No operator is assumed to be present before hand.
int i = ab ✗
int i = a*b ✓

Note → There is no operator to perform exponentiation in C language so we have to use pow(x,y) from math.h

3) Control instructions :
Determine the flow of control in a program :
Four types of control instructions in C are
1) Sequence Control
2) Decision Control
3) Loop control
4) Case Control

→ Type conversion

Arithmetic operations between
int and int : int
int and float : float
float and float : float

5/2 = 2 5.0/2 = 2.5

2/5 = 0 2.0/5 = 0.4

Eg: #include <stdio.h> #include <math.h>
int main ()
{
int a = 2.5
int b = 8.8
int c = a * b;
printf ("%d", c);
scanf ("%d", &a); // override a = 4.0
scanf ("%d", &b); // override b = 5.0
c = a * b; // c = 20
printf ("%d", c);
return 0;
}

c = 16

int main ()
{
int a, b, c = a * b
scanf ("%d", &a);
scanf ("%d", &b);
printf ("power of b on a %f", pow (a, b)); // f as double
printf ("value of c %d", c); // c = 0
}

if used d then output = 0

→ int a = 2.5 ⇒ a = 2
int b = 8.8 ⇒ b = 8
int c = 2 * 8 ⇒ c = 16

→ float a = 2.5
printf ("%d", a) ⇒ 2
printf ("%f", a) ⇒ 2.5

→ float c = 2 * 8 ⇒ c = 16.000
printf ("%d", c) ⇒ 16

* float : 6 decimal places : 0.000000

• operator precedence

* / % > + - > = : BEDMAS

→ for same priority Left to right

Eg: data type return of (3.0 / 8 - 2)?
= double

Eg: step evaluation of 3 * x / (y - z + k
where (z = 3, y = 3, k = 1, x = 2)

3 * 2 / 3 - 3 + 1
6 / 3 - 3 + 1
2 - 3 + 1
0

Eg: 3.0 + 1 will be ?
floating

- Decision making instruction in C

i) If else
ii) switch case

i) If else: The syntax of an if else statement in C looks like

```
if (condition to be checked)
{
  statements;
}
else
{
  statements;
}
```

ii) switch case: It is used when you have to make a choice b/w numbers of alternatives for a given variable. Works with integer only and char

```
switch (integer)
{
  case C1:
    {
    // statements;
    break;
    }
  case C2:
    {
    // statements;
    break;
    }
  default:
    {
    // statements;
    break;
    }
```

- Relational operators in C
They are used to evaluate conditions (true or false) inside the if statements.

==, >=, >, <, <=, !=

= assignment
== equality

The condition can be any valid expression. In C a non zero value is considered to be true.

- Logical operators:
They are used to provide logic to our C language

&&, ||, ! : AND, OR, NOT

Eg: 1 and 0 ⇒ 1 && 0 = false
0 && 0 = false
1 && 1 = true

1 or 0 ⇒ 1 || 0 = true
0 || 1 = true
0 || 0 = false.

not (3==3) ⇒ ! (3==3) = false
! (3>30) = true

As the no. of conditions increase, the level of indentation increases. This reduces readability logical operators come to rescue in such case.

* instead of using multiple if we can use (if, else if) ladder

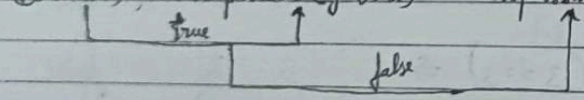
```
if {
  // statements
}
else if {
  // statements
}
else {
  // statements
}
```

// last else is executed when all other if are false

→ priority precedence operator
(!) > (*, /, %) > (+, -) > (<, >, <=, >=) > (==, !=)
&& > || > =

- Ternary/conditional operator:
A short hand if else can be written using the conditional or ternary operator

(condition)? expression (if true) : expression (if false)



Eg:

```
#include <stdio.h>
int main()
{
  int r;
  printf("Enter rating.\n");
  scanf("%d", &r);
  switch(r)
  {
    case 1: {
      printf("Your rating is 1.");
      break;
    }
    case 2: {
      printf("Your rating is 2");
      break;
    }
    case 3: {
      printf("Your rating is 3");
      break;
    }
    default: {
      printf("Invalid\n");
      break;
    }
  }
return 0;
```

```
* if (true)
    printf ("%d", n);
    printf ("%d", n);

if (true) {
    printf ("%d", n);
    printf ("%d", n);
  }

if (true)
    printf ("%d", n);
```

Eg: Calculate income tax paid by employee as given below:

Income slab	Tax
2.5L - 5.0L	5%
5.0L - 10.0L	20%
above 10L	30%

Income < 2.5L : Tax = 0, take input from user

```
# include <stdio.h>
int main () {
printf ("Income \n");
double i; double tax = 0
  scanf ("%d", &i);
  if (i < 2.5)
     tax = 0.0;

  else if (i < 5.0)
     tax = 0.05 * (i);

  else if (i < 10.0)
     tax = 0.20 * (i);

  else
     tax = 0.30 * (i);
  ~~double pay~~ return 0;
}
```

Eg:- Program to find greatest of 4 number entered by user and print all in ascending order

```
# include <stdio.h>
int main ()
{
  printf ("enter 4 numbers \n");
  int a, b, c, d;
  scanf ("%d %d %d %d", &a, &b, &c, &d);
  if (a > b && a > c && a > d)
    printf ("greatest %d", a);

  else if (b > a && b > c && b > d)
    printf ("greatest %d", b);

  else if (c > a && c > b && c > d)
    printf ("greatest %d", c);

  else if (d > a && d > c && d > b)
    printf ("greatest %d", d);

  else
    print ("all equal");

  return 0;
}
```

ii)
```
#include <stdio.h>
int main()
{
printf("Enter 4 numbers \n");
int a, b, c, d, int temp;
scanf("%d %d %d %d", &a, &b, &c, &d);
if (a > b){
temp = a;
a = b;
b = temp;
}
if (b > c){
temp = b;
b = c;
c = temp;
}
if (c > d){
temp = b
c = d
d = temp
}
if (a > b){
temp = a;
a = b;
b = temp;
}
if (b > c){
temp = b;
b = c;
c = temp;
}
if (a > b){
temp = a;
a = b;
b = temp;
}
printf("ascending order %d %d %d %d", a, b, c, d);
}
```

→ while : 1st check then execute
when the end is not known

→ do while : once run the statement within then check
while (condition);

→ for : (initialize; test; increment)
when the end is known

★ (i=10; i = i>0); i++) || (i=0; (i ≠ i<10); i++

as automatically stop at 0 — no output required

• Increment & decrement operator:

++i : 1st increase then use
i++ : 1st use then increase

--i : 1st decrease then use
i-- : 1st use then decrease

Eg: i=5
printf ("%d", i++) i=5
printf ("%d", i) i=6

i=5
printf ("%d", ++i) i=6
printf ("%d", i) i=6

Eg: decrement of for loop
for (i=5; i>0; i--)
printf ("%d", i);

increment of for loop
for (i=0; i<5; i++)
printf ("%d", i);

• continue : The control is used to immediately move to the next iteration of the loop

The control is taken to the next iteration thus skipping everything below "continue" inside the loop for that iteration.

• Break : It is used to exit the current loop irrespective of whether the condition is true or false

When the break is encountered inside the loop, the control is sent outside of the loop

Eg: Print table of 10 in reversed order:

```
# include < stdio.h >
int main ()
{
  printf ("Table of 10 \n");
  for (int i = 10 ; i ; i--)
  {
  printf ("10 x %d = %d \n", i, 10*i);
  }
  return 0;
}
```

Eg: Prime number:

```
# include < stdio.h >
  int main ()
  {
  int n, p = 1;          // p=1 let the number is prime
  scanf ("%d", n);
  for (int i = 2 ; i <= n ; i++)
    {
      if (n % i == 0){
        p = 0;
        break;
      }
    }
  if (p == 0)
    printf ("Not a prime ");
  else
    printf ("Prime ");
  return 0;
}
```

Eg: Number finding game :

WAP to generate a random number and ask players to guess it. If the players guess is higher than the actual number the program displays "Lower the number please". Same when the user guess is too low, the program prints "Higher number please" when the user guesses the correct number, the program displays the number of guesses the player used to arrive at the number

- Functions: also called methods
  They help in reusability
  Saves time and space.

```
#include<stdio.h>
void display();          // function prototype

int main()
{
  int a;
  display();             // function call
  return 0;
}

void display()           // function definition
{
  printf("This is the display function");
}
```

→ Function prototype: This way we tell compiler about function we are going to create / define in program.
  * Void functions don't return any value.

→ Function call: The way we tell the compiler to execute the function body at the time the call is made but program execution starts from main function.

→ Function definition: This part contains the exact set of instructions which are executed during the function call. During this main temporarily stops and wait on later statements when the control return from other function definition.

Eg:
```
int main()
{
  printf("1");
  display();
  printf("2");
  return 0;
}

void display()
{
  printf("3");
}
```
output: 1 3 2

Eg: function calling within function:

```
int main(){
  goodmorning();
  return 0;
}

void goodmorning(){
  goodafternoon();
  printf("Goodmorning \n");
}

void goodafternoon(){
  goodnight();
  printf("Goodafternoon \n");
}

void goodnight(){
  printf("Goodnight \n");
}
```

output:
Goodnight
Goodafternoon
Goodmorning

Important points regarding functions:

i) Execution of a C program starts from main().

ii) A C program can have more than one function

iii) There are two types of functions in C. Lets talk discuss Types of functions

a) Library functions: Commonly required functions grouped together in a library file on disk
Eg: printf, scanf

b) User defined functions: These are the functions declared and defined by the user.

Why to use functions:
→ To avoid repetation the same logic again and again
→ To keep track of what we are doing in a program
→ To test and check logic independently.

Call by value (passing value of argument): The method in which the values of actual parameter are copied to the formal parameter and if any change occur in formal it is (not) reflected in actual parameter.

Call by reference (passing address of argument): The method in which the value of the actual parameter are shared by the formal parameter and if any change occure in formal parameter it is reflected in actual parameter. Here we make use of pointers.

→ Passing values to functions:
We can pass values to a function and can get a value in return from a function (but only a single value)

int sum (int a, int b)

The above prototype means sum is a function which takes values a (int type) and b (int type) and returns in int type

Actual Parameters: The parameters (values) which appear in the function call.

Formal Parameters: The parameters (values) which appear in the function prototype.

Arguments: The actual values passed to the function to make a call. Eg: 2 & 3

Parameters: The values or placeholder for variables in function definition. Eg: a & b

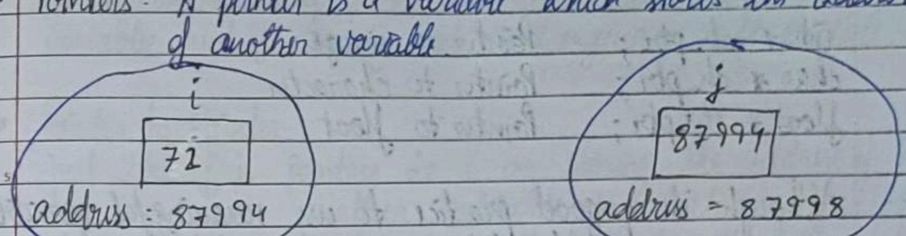
Eg: Print pattern using recursion function

```
*
**
*****
```

```c
# include <stdio.h>
void printpattern (int n);
int main () {
    int n = 3;
    print pattern (n);
    return 0;
}

void printpattern (int n) {
    if (n == 1){
        printf ("*\n");
        return;
    }
    printpattern (n-1);
    for (int i = 0; i<(2*n-1); i++){
        printf ("*");
    }
    printf ("\n");
```

%u = address of operator or "%d", bi pr ie %u & %d works the exact same way
%d = value (numc)
%s = structure
%c = character only

- Pointers: A pointer is a variable which stores the address of another variable



j is a pointer, j points to i
The "address of" (&) operator is used to obtain the address of a given variable (address batao & operator aur jis ka address batao)

&i = 87994
&j = 87998

format specifier for printing pointer address is "%u"

★ The "value at address" operator (*) ★ (address se operator ki value milti)
The "value at address" or * operator is used to obtain the the value present at a given memory address. It is denoted by *.

Value at adrss of
*(&i) = (72)
*(&j) = 87994 (address of i)

*ptr = has the value to which its pointing
ptr = has the address of that pointer

How to declare a pointer?
A pointer is declared using the following syntax (declare a variable as j jo ki integer address ko store karega; j ke address mai address store hoga)

int *j; ⇒ declare a variable j of type int pointer
j = &i ⇒ store address of i in j (a integer pointer)

just like pointer of types integer, we also have pointers to char float etc.

```
int * ch_ptr;      Pointer to integer
char * ch_ptr;     Pointer to character
float * ch_ptr;    Pointer to float
```

Although its a good practice to use meaningful variable names, we should be very careful while reading and working on program from fellow programmers.

Eg: A program to demonstrate pointers.

```
#include <stdio.h>
int main ()
{
    int i = 8
    int *j;
    j = &i;
    printf ("Add i = %u \n", &i);      i = 87994
    printf ("Add j = %u \n", j);       i = 87994
    printf ("Add j = %u \n", &j);      j = 87998
    printf ("Value i = %d \n", i);     i = 8
    printf ("Value i = %d \n", *(&i)); i = 8
    printf ("Value i = %d \n", *j);    i = 8
    return 0;
}
```

```
int i = 34;
int *j = &i;                          j = 87994
printf ("The value of i is %d \n", i);   i = 34
printf ("The value of i is %d \n", *j);  34
printf ("The address of i is %u \n", &i);  87994
printf ("The address of i is %u \n", j);   87994
printf ("The address of j is %u \n", &j);  87998
printf ("The value of j is %u \n", *(&j));
```

* we can go as deep as we want pointer to a pointer to a pointer
  *j **k ***l

* agar variable ka address ho toh you can change the value of variable and if hearing value changing is not possible. *

→ Pointer to Pointer

just like j is pointing to i or storing the address of i we can have another variable k which can further store the address of j. What will be the type of k

```
int **k;
k = &j;
```

	i	j	k	
	72	87994	87998	value stored
	87992	87998	88004	address
	int	int *	int **	type

we can even go further on level and create a variable l of type int *** to store the address of k. we mostly use int * and int ** sometimes in real world.

```
int i = 72;
int *j = &i;   ✓
int **k = &j;  ✓      int ***k = &i  ✗
```

Eg: Call by value:

```
#include<stdio.h>
int main()
{
int x=4, int y=7
printf("The value of x & y %d and %d\n", x,y);
printf("The value of x+y is %d\n", sum(x,y);
printf("
return 0;
}
int sum (int a, int b);
int sum (int a, int b);
{
  int c=a+b;
  b = 324;
  a = 378;
  return c;
}
```

if we define sum as sum (int a, int b), the values 3 and 4 are copied to a and b. Now even if we change a and b, nothing happens to i variable or x & y

Eg: Call by reference:

```
#include<stdio.h>
void wrongswap (int a, int b);
void swap (int *a, int *b);
int main()
{
  int x = 3, int y = 4
  printf("Value of x & y before %d and %d\n", x, y);
  wrongswap (x,y); // passed a copy of values
  swap (&x, &y); // passed address
  printf("The value of x & y after %d and %d\n", x, y);

  return 0;
}

void wrongswap (int a, int b){
  int temp; // treated as local
  temp = a; // treated as local
  a = b; // treated as local
  b = temp;
}

void swap (int *a, int *b){ // address ko (*) variable hi
  int temp;                   store kar sakta hai
  temp = *a; // address ki value ko print karne ke liye * use kiya hai
  *a = *b;
  *b = temp;
}
```

Here we can change the value of a variable in calling function using * and & operators.
* value lene ke liye uss address ki
& address lene ke liye

* scanf uses & coz tells us address pe

- Array:

An array is a collection of similar elements.
int marks[90]: Size of 90
One variable ⇒ capable of storing multiple values

Syntax: int marks[90]; ⇒ 90 integer storage
char name[20]; ⇒ 20 character or string storage
float percent[50]; ⇒ 50 float storage

int marks[5];
marks[0] = 5
marks[1] = 3
marks[2] = 4
marks[3] = 2
marks[4] = 4

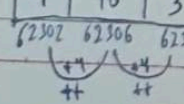
Accessing elements:
Elements of an array can be accessed using :=

scanf("%d", &marks[0]); input 1st value

printf("%d", marks[0]); output 1st value

Initialization of arrays:
There are many other ways in which an array can be initialized.

int cgpa[3] = {9, 10, 5} 1 integer = 4 bytes.
4 x 3 = 12 bytes in memory.



Eg Array to have input of 5

```
#include <stdio.h>
int main()
{
  int marks[5];
  for(int i=0; i<5; i++)
  {
    printf("Enter the value of marks %d", i+1);
    scanf("%d", &marks[i]);
  }

  for(int i=0; i<5; i++)
  {
    printf("The marks of students %d is %d\n", i+1, marks[i]);
  }

  return 0;
}
```

* *ptr kaun se ek variable ka do pata variable ka store karta hai ek pata pointing

→ Pointer Arithematic
A pointer Arithematic can be incremented to point to the next memory location of that type:

Consider the example

```
int i = 32;
int *a = &i;        ⇒ a = 87994
a++;                  a = 87998
char a = 'A';
char *b = &a;         b = 87994
b++;                  b = 87995
float j = 1.7
float *a = &j;
```

i → [32] 87994

pointer type hai ek number hai jisko sub ya add kiya ja sakta hai

as byte size for char is 1

Eg:
```
# include <stdio.h>
int main(){
    int i = 34;
    int *ptr = &i;
    printf("The value of ptr is %u\n", ptr);
    ptr = ptr + 1;        4 x 1 (int)
    ptr = ptr + 2;        4 x 2 (int)
```

ptr ka alag index hai but for now its storing index of i
*ptr = 87994 (i.e index of i is the value of ptr)
&i = *ptr
**ptr = &ptr (i.e **ptr is having index of *ptr as its value)

Following operations can be performed on pointers:
i) Addition of a number to a pointer
ii) Substraction of a number from a pointer
iii) Substraction of one pointer from another
iv) Comparisons of two pointer variables

Eg:
```
# include <stdio.h>
int main()
{
    int marks[4];
    int *ptr;
    ptr = &marks[0];   or ptr = marks acts equally
    for(int i = 0; i < 4; i++)
    {
        printf("Enter value of marks of students %d:\n", i+1);
        scanf("%d", ptr);
        ptr++;   // ptr main hai marks ka index and ptr++
                 incrimant hua marks ke next index mai so
                 indirectly we are moving forward in marks array
                 by using the address of marks assigned in memory.
    }
    for(int i = 0; i < 4; i++)
    {
        printf("The marks of students %d:\n", marks[i]);
    }
    return 0;
}
```

Accessing array using pointers:

0	1	2	3	4
11	13	15	17	19

considering array:
If ptr points to index 0, ptr++ will point to index 1 and so on

* pointers can change the value passed as under is passed

```
int *ptr = &arr[0]  // or ptr = arr;
ptr++;
*ptr will have 13 as its value
```

Passing arrays to functions:
An array can be passed to the function like this

```
printArray (arr, n);          function call
```

```
void printArray (int *, int n);      function prototype
or                                   both way we are passing the address
void printArray (int i[], int n);
```

* Eg: Array to function:

```
#include <stdio.h> or ptr[]   (without converting to array pointer just contacted to array normally)
void printarray (int *ptr, int n)
{
  for (int i = 0; i<n; i++)
  {
    printf ("The value of elemnt %d is %d \n", i+1, *(ptr+i));   (dereferencing or getting)
  }
  ptr[2] = 2; // change the valu as address was passed
}
int main () {                    (ptr+1)(ptr+2) ..... (ptr+6)
  int arr[] = {1, 2, 3, 4, 5, 6}     (4x1)..(4x2) ..... (4x6)
  printarray (arr, 6);   // print all 6 values        address of arr elemnts
  printarray (arr, 7);   // print garbage value after 6   as ptr is now an array
  return 0;                 i.e if the size of array is passed.   ptr[i] is also correct
}
```

* in memory any sized array is stored as a 1D array

→ Multidimensional Array:
An array can be of 2D | 3D | nD
a 2D is defined as:

```
int arr[3][2] = { {1, 4}
                  {7, 9}
                  {11, 23} }
```

(3 row, 2 columns)

```
arr[0][0] = 1
arr[0][1] = 4
```

2D arr storage in memory: arr[0][0], arr[0][1], arr[1][0] ... arr[2][1]

Eg: 2D array stores number of student with his marks:

```
#include <stdio.h>
int main () {
int stud = 5;
int sub = 3;

int marks[5][3];
  for (int i = 0; i<stud; i++) {
  for (int j = 0; j < sub; j++){
    printf ("Enter marks of %d student in %d sub ", i+1, j+1);
    scanf ("%d", &marks[i][j]);
  }
  }

for (int i = 0; i<stud; i++){
  for (int j = 0; j < sub; j++) {
    printf ("Entred marks of student %d in sub %d ");
    printf ("%d %d %d ", i+1, j+1, marks[i][j]);
  }
} return 0;
}
```

Eg: If s[3] is any 1D array then *(s+3) refers to 3rd elemt
False
(s+2) is 3rd elemt.

Eg: WAP to create an array of 10 integers and store multiplication table of 5 in it.

```
#include <stdio.h>
int main(){
    int mul[10];
    for(int i=0;i<10;i++){
        mul[i] = 5*(i+1);
    }
    for(int i=0;i<10;i++){
        printf("5*%d=%d", i+1, mul[i]);
    }
    return 0;
}
```

Eg: To reverse an array:

```
#include <stdio.h>
int main(){
    int arr[] = {1,2,3,4,5,6};
    reverse(arr, 6);
    return 0;
}
void reverse(int *arr, int n){ int temp;
    for(int i=0; i<n/2; i++){
        // arr[i] = arr[n-i-1]
        temp = arr[i];
        arr[i] = arr[n-i-1];
        arr[n-i-1] = temp;
    }
}
```

Eg: Returning 2 values from a function which seems breaking of rules

```
#include <stdio.h>
void SumAndAvg(int i, int j, int *sum, int *avg)
{
    *sum = a+b;        // indirectly value store ki
    *avg = *sum/2;
}

int main(){
    int i, j, sum, avg;
    i=3;
    j=6;
    SumAndAvg(i, j, &sum, &avg);
    printf("The value of sum %d\n", sum);
    printf("The value of avg %d\n", avg);
    return 0;

}
```

Strings:

A string is a 1D character array terminated by a null ('\0')

A null character is used to denote string terminated character are stored in contigous memory locations

Initilization of a string:
Since string is an array of character, it can be initilized as follows:

1) `char s[] = {'H', 'A', 'R', 'R', 'Y', '\0'};`
   - not adding '\0' it is not incorrect but will generate random values
   - denotes the end of string array.

2) `char s[] = "HARRY";` : In this case C adds a null char automatically.

→ Strings in memory:
A string is stored just like an array in the memory as shown below:

H	A	R	R	Y	\0
(210)	(211)	(212)	(213)	(214)	(215)

null character

contigous blocks in memory

3)
```
int main () {
    char str[] = "Harry";
    char *ptr = str;
    while (*ptr != '\0') {
        printf ("%c", *ptr);
        ptr++;
    }
    return 0;
}
```

character by character printing

// \0 - not used then will go out of bound and give some unwanted garbage value.

%c, %d, %f, %s

*** when string initialized by ptr - "Shubham" ✗ not allowed. but "Harry" the stored value cannot be changed. "Harry" can do this

Q2 instead of printing each character of string print the whole wanted string:

```
#include <stdio.h>
int main(){
char *ptr = "Harry";
printf("%s", ptr);
return 0;
}
```

→ ptr has address of the first character of string "Harry" or char ptr[] = "Harry";

→ print the whole string all at once by (%s)

ptr[]	*ptr
pointer allow	pointer not allow

→ compiler will automatically convert Harry to = {'H', 'a', 'r', 'r', 'y', '\0'}

Eg: input from user user inputs from %s

```
#include <stdio.h>
int main(){
char s[34];
printf("Enter your name: ");
scanf("%s", s);
printf("Your name is %s", s);
return 0;
}
```

* scanf automatically adds the null character when the Enter key is pressed.

→ the string should be short enough to fit into the array
→ scanf cannot be used to input multiword strings with space

i) gets(): a function which is used to receive a multiword string with spaces: char s[34];
gets(s);

ii) puts(): a function use to print the string and places the cursor on the next line

(%s enter word
(%c) letter by letter gets entire sentence

→ String functions: they are called using (#include <string.h>)

i) strcpy(): this function is used to copy the content of second string to the first string passed to it.

```
char source[] = "Harry";
char target[30];
strcpy(target, source);   // now target contains Harry
```

target ko andar aa jayega source

Target string should have enough capacity to store the source string.

ii) Strcat(): this function is used to concatinate two strings

```
char s1[11] = "Harry";
char s2[] = "Hellow";
strcat(s2, s1);   // HellowHarry
```

iii) strcmp(): this function is used to compare two strings

It return 0: if strings are equal

It return (-ve): when ASCII character of 1st string mismatch & is not greater than string corresponding mismatch character

bad wala - pehle wala
67-70 = -1

It returns (+ve): when ASCII character of 1st string mismatches & is lesser than the ASCII value of string corresponding to the mismatched character

pehle wala - bad wala
70-67 = +1

iv) Strlen(): measure the length of the given string.

* ptr points to the character of that string not the index, ptr of *ptr is having the address of it

Eg:
```
# include <stdio.h>
# include <string.h>
int main (){
    char str1[45] = "Hello";   → str[] : needs defining of size
    char* str2 = "Harry";      → *str : no need to defining of size
    strcat (str1, str2);
    printf ("Now the str is %s", str2);

    int val = strcmp (str1, str2);
    printf ("The value is %d", val);

    return 0;
}
```

Eg: Get inputs using %c & %s
```
# include <stdio.h>   # include <string.h>
int main (){
    char st1 [34];
    char st2 [34];
    printf ("Enter value of 1st string \n");
    scanf ("%s", st1);
    fflush (stdin);  // function that removes the enter pressed for
    char c;          1st scanf which if not used will be taken
                     by 2nd scanf as a valid input
    while (c!= '\n'); // 2nd enter pressed will get \n (also enter) stored in
    {                   the string as a valid input.
        fflush (stdin); // removes all previous inputs
        scanf ("%c", &c); // even if we give a whole word it will store
        printf ("%c", c); only the 1st letter
        st2[i] = c;
        i++;
    }
    st2 [i-1] = '\0'; // to remove the \n which was stored in st2
    printf ("1st string %s \n", st1);
    printf ("2nd string %s \n", st2);

    return 0;
}
```

// scanf ke baad ek or scanf hai jisme "%c" hi for each character input then if pressed enter that enter is a valid character input for next scanf and as that enter is stored in the buffer which is then picked by the next scanf.

Eg: Create a len function to get the length of the inputted string.
```
// include <stdio.h>
int strlen (char *str)
{
    char *ptr = str;
    int len = 0;
    while (*ptr != '\0'){
        len ++;
        ptr++;
    }
}
```

Eg: Length of a string:
```
# include <string.h>
# include <stdio.h>
int main (){
    char *st = "Harry";
    int a = strlen(st);    // measures length excluding null char
    printf ("Length of st is %d", a);
}
```

Eg: Create a slice() function:

```
#include <stdio.h>
#include <string.h>
void slice (char *st, int m, int n)
{ int i=0; m
while ((m+i)<= n) {
  st[i] = st[i+m];
    i++;
} st[i] = '\0';
}
int main ()
{
char st[] = "Harry";
slice (st, 1, 4);
printf ("%s", st);
return 0;
}
```

Eg: Encrypt a string:

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

void encrypt (char *c)
{
    char *ptr = c;
  while (*ptr != '\0')
  {
      *ptr = *ptr + 1;
       ptr ++;
  }

int main ()
{
  char c[] = "Harry";    // as *c is only read only no error
  encrypt (c);
  printf ("Encrypted is %s \n", c);
  return 0;
}
```

- Structures :

As the arrays and strings can hold similar kind of data and there was a need to have multiple data type values under one data structure.
Eg: data of a employee at a company will have name (string type), date of birth (int), salary (float), address (string) etc
so all this inputs can be stored under one data type.

< Syntax for structures >

```
struct employee {
  int code;
  float salary;
  char name[10];
};
```

This declares a new user defined data type employee

this only ends with a semi colon

Structure in C is a collection of variables of different types under a single name.

Why to use Structures :
We can create the data type in the employee structure, separately but when the number of properties in a structure increase, it becomes difficult for us to create data variables without structure.

(i) data is kept organised using it.
(ii) data managinment becomes easy for very huge data (200 or more)

* Strings initialization in array takes place by strcpy
'.' member operator use to get acess of the members

Eg:
```
#include <stdio.h>
#include <string.h>

struct employee {
    int code;
    float salary;
    char name[10];
};

int main (){
    int a = 34;
    char b = 'g';
    float d = 253.23;

    struct employee e1;
    e1.code = 100;
    e1.salary = 34.54;
    strcpy (e1.name, "Harry");   // e1.name = "Harry"; X
    printf ("%d", e1.code);
    return 0;
}
```

Eg: Write a code to store the data of 2 employees from user defined data. Use the structures declared above

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

struct employee {
    int code;
    float salary;
    char name[10];
};
int main (){
    struct employee e1, e2;
    printf ("Enter for 1st");
    scanf ("%d\n", &e1.code);
    printf ("Enter for salary of 1st");
    scanf ("%f\n", &e1.salary);
    printf ("Enter for name of 1st");
    scanf ("%s\n", e1.name);   // dont need & as its address

    printf ("Enter for code of 2nd");
    scanf ("%d\n", &e2.code);
    printf ("Enter for salary of 2nd");
    scanf ("%f\n", &e2.salary);
    printf ("Enter for name of 2nd");
    scanf ("%s\n", e2.name);   // as its array so its already addressed

    return 0;
}
```

Eg: Array of structures:

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

struct employee {
   int code;
   float salary;
   char name[10];
};

int main () {
   struct employee facebook[100]; // struct employee type ka
                                  ek array jisme e1, e2, e3 .... store ho
   facebook[0].code = 100;
   facebook[0].salary = 100.45;
   strcpy (facebook[0].name, "Harry");

   facebook[1].code = 101;
   facebook[1].salary = 90.2;
   strcpy (facebook[1].name, "Barry");

   facebook[2].code = 102;
   facebook[2].salary = 110.45;
   strcpy (facebook[2].name, "Larry");

   return 0;
}
```

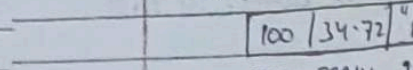
→ Initilization of structure:

```
struct employee harry = {100, 71.22, "Harry"};

struct employee larry = {0}; // all elements set to 0
```

Eg:

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

struct employee {
   int code;
   float salary;
   char name[10];
};

int main (){
   struct employee harry = {100, 34.72, "Harry"};

   printf ("Code is %d: \n", harry.code);
   printf ("Salary is %f: \n", harry.salary);
   printf ("Name is %s: \n", harry.name);

   return 0;
}
```

→ Structure storage in memory:
Structure are stored in contiguous memory locations. For the structure e1 of type struct employee, memory layout looks like this:



In an array of structures, these employee instances are stored adjacent to each other

→ Pointers to structure
A pointer to structure can be created as follows

```
struct employee *ptr;
ptr = &e1;  // store address of the variable of struct employee
*ptr = e1;  // stores value of e1
```

We can now print structure element using:
`printf("%d", (*ptr).code);`

→ Arrow operator
Instead of writing (*ptr.code), we can use arrow operator to access structure properties as follows

(*ptr).code or ptr → code

'→' arrow operator

Eg:
```
#include <stdio.h>
#include <string.h>

struct employee {
  int code;
  float salary;
  char name[10];
};

int main() {
  struct employee e1;
  struct employee *ptr;

  ptr = &e1;
  (*ptr).code = 101;   // or ptr → code = 101 both equal
  printf("%d", e1.code);

  return 0;
}
```

→ Passing structure to a function:
A structure can be passed to a function just like any other data type.

```
void show (struct employee e);
```

Eg: Compile this show function to display the content of employee

```
void show (struct employee emp) {
    printf ("Code of employee %d \n", emp.code);
    printf ("Salary of employee %f \n", emp.salary);
    printf ("Name of employee %s \n", emp.name);
emp.code = 34; // will not effect the value of code as structures
}                  even with pointers are passed as copy not
                   actual value

int main() {
struct employee e1;      // declaring e1
struct employee *ptr;    // declaring ptr

ptr = &e1;    // pointing ptr to e1
ptr → code = 101;
ptr → salary = 11.01;
strcpy (ptr → name, "Harry");

show (e1);
printf ("Code is %d \n", e1.code);

return 0;
}
```

→ Typedef keyword:

We can use the typedef keyword to create an alias name for data types in C.
typedef is more commonly used with structures.

```
typedef struct complex {
    float real;
    float img;
} complex;
```

now complex ~~struct complex~~ C1, C2; for defining complex numbers.

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

typedef struct employee {
    int code;
    float salary;
    char name[10];
} emp;   // now emp is kind of a data type in this code but we can
         also use struct employee but emp is time saving.
void show (struct employee empr) {
    printf ("Code of employee %d \n", empr.code);
    printf ("Name of employee %s \n", empr.name);
    empr.code = 34;
}

int main() {
    emp e1;
    emp *ptr;
    ptr = &e1;
    ptr → code = 100;
    ptr → salary = 11.01;
    strcpy (ptr → name, "Harry"); show (e1);
}
```

Eg: create a program for 2D vector using structure in C? and then sum of these 2 vectors

```
#include<stdio.h>
typedef struct vector{
    int x;
    int y;
}vect;

int main(){
    struct vect v1, v2, sum;
    v1.x = 34;
    v1.y = 54;
    printf("x %d and y %d is \n", v1.x, v1.y);

    v2.x = 20;
    v2.y = 30;
    printf("x %d and y %d for 2nd \n", v2.x, v2.y);
    sumvector(v1, v2);
    return 0;
}

vect sumvector(vect v1, vect v2){
    vect result;
    result.x = v1.x + v2.x;
    result.y = v1.y + v2.y;
    printf("sum of x and y dimensions %d %d \n", result.x, result.y);
}
```

Eg: 20 integers are to be stored in memory what is used array/structure

array: as its single datatype and easy to construct and lesser memory occupation.

Eg: Use of arrow operator in C?

(*ptr).code; or ptr → code;

Eg: Create a code for 5's structure using typedef keyword

Pg: Create a code for an array of 5 complex numbers created in problem 5 and display them with the help of display function. The values must be taken as an input from user.

```
# include <stdio.h>
typedef struct complex {
    int real;
    int complex;
} comp;

int main () {
    comp cnums[5];
    for (int i=0 ; i<5; i++) {
    scanf ("%d", &cnums[i].real);
    printf ("for complex %d \n", i+1);
    scanf ("%d", &cnums[i].complex);
    }

    for (int j=0; j<5 ; j++) {
    display (cnums[i]);
    }
    return 0;
}

void display (comp c) {
    printf ("The value of real %d" c.real);
    printf ("The value of complex %d", c.complex);
}
```

Pg: Create a structure for a bank account of a customer. What field would you use and why?

Eg: WAP for structure capable of storing data, write a function to compare those data (as per date, moth, year)

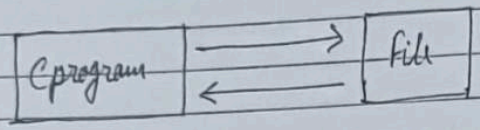
```
#include <stdio.h>

int main(){
  date d1 = {2, 11, 31}
  date d2 = {5, 4, 31}
  display (d1);
  display (d2);
  int a = datecomp (d1, d2)
  printf ("Comparison returns %d", a);

  return 0;
}

int datecomp (date d1, date d2){
  if (d1.year > d2.year){  // year comparison
    return 1;
  }
  if (d1.year < d2.year){
    return -1;
  }
  if (d1.moth > d2.month){  // moth comparison
    return 1;
  }
  if (d1.moth < d2.month){
    return -1;
  }
  if (d1.date > d2.date){  // date comparison
    return 1; }
  if (d1.date < d2.date){
    return -1. }
  return 0; }
```

```
typedef struct date {
  int date;
  int month;
  int year;
} date;

void display (date d){
  printf ("The date is : %d %d %d \n", d.date, d.month
                                     , d.year);
}
```

- File I/O : Input and output

The random access memory is volatile and its content is lost once the program terminates. In order to persist the data forever we make use of files

A file is a data stored in a storage device A 'C' program can talk to the file by reading content from it and write content to it



File pointers:
The file is a structure which needs to be created for opening the file.
A file pointer is a pointer to this structure of file

file pointer is needed for communication between the file and the program

Syntax of file pointer:

```
FILE *ptr;
ptr = fopen ("filename.ext","mode");
```

isko use karke hum file ko khol sakte hain and alag alag modes mai file ko khol sakte hain

* fopen function takes 2 arguments (location + mode in which read or write)

Eg: file basics:

```
#include <stdio.h>

int main(){
  FILE *ptr;
  ptr = fopen("sample.txt","r");   // read file
  ptr = fopen("sample.txt","w");   // write file
  return 0;
}
```

→ File opening modes in C
C offers the coder to select the mode for opening a file. following modes are for I/O files:

"r" : open for reading "rb" : open for reading in binary	If the file doesn't exist, fopen returns null
"w" : open for writing "wb" : open for writing in binary	If the file exists, the content will be overwritten
"a" : open for append	If the file doesn't exist, it will be created automatically

→ Types of files

There are 2 types of files:
1) Text file (.txt, .c)
2) Binary file (.jpeg, .dat)

→ Reading a file: a file can be opened for reading as

```
FILE *ptr;
ptr = fopen ("Harry.txt", "r");
int num;
```

Eg: Let us assume that "Harry.txt" contains an integer, we can read that integer using.

fscanf (ptr, "%d", &num); : Same working as scanf the files counterpart of scanf

This will read an integer from file in num variable
Then modify the code above to check whether the file exists or not before opening the file

```
int main (){
  FILE *ptr;
  int num;
  int num2;
  ptr = fopen ("Harry.txt", "r");
  fscanf (ptr, "%d", &num);
  fscanf (ptr, "%d", &num2); fclose (ptr); //yahan close kardi coz pehle hi
  printf ("The value of num %d \n", num); read karlia hain values.
  printf ("The value of num2 %d \n", num2);
  return 0;
}
```

Harry.txt
34 80
↓ read by num ↓ read by num2

```
int main {
  FILE *ptr;
  ptr = fopen ("Harry.txt", "r");
  int num;
  int num2;
  fscanf (ptr, "%d", &num);
  if (ptr == null){
  printf ("file doesn't exist \n");
  }
  else {
    fscanf ( ptr, "%d", &num);
    fscanf ( ptr, "%d", &num2);
    fclose (ptr);
    printf ("The value of num %d \n", num);
    printf ("The value of num2 %d \n", num2);
  }
  return 0;
}
```

→ Closing the file: It is also very important as the file is taking the resources to run as after read and write value inputted has been assigned to variable so now they are independent of the open file, indicates complete we are done working with file, resources are freed.

```
fclose (ptr);
```

→ Writing to a file:

```
FILE *ptr;
ptr = fopen ("Harry.txt", "w");
```

Eg: Writing to a file:

```
#include <stdio.h>
int main {
  FILE *ptr;
  int num = 45;
  ptr = fopen ("generate.txt", "w");
  fprintf (ptr, "The number is %d", num);
  fclose (ptr);
  return 0;
}
```

// as the generated file does't exist so on running the program it will get created and "The number is 45" will get stored in it

→ fgetc() & fputc() : They are used to read and write a character from to a file (reads character by character)

fgetc(ptr): used to read a character from file (char by char input)

fputc('a',ptr): used to write a character to the file (which is 'a').

Eg: fgetc

```
#include <stdio.h>
int main () {
  FILE *ptr;
  ptr = fopen ("fgetcdemo.txt")
  return 0;
  char c = fgetc (ptr);
  printf ("The character stored %c\n", c);
  printf ("%c\n", fgetc (ptr)); // t
  printf ("%c\n", fgetc (ptr)); // h
  printf ("%c\n", fgetc (ptr)); // i
  printf ("%c\n", fgetc (ptr)); // s

  ptr = fopen ("putcdemo.txt", "w");
  putc ('c', ptr);
  putc ('c', ptr);
  putc ('c', ptr);
  fclose (ptr);
  return 0;
}
```

fgetcdemo.txt

```
this is for fgetc
putcdemo.txt
ccc
```

Eg: read whole file by fgetc.

```
#include <stdio.h>
int main(){
  FILE *ptr;
  char c;
  ptr = fopen("getcdemo.txt", "r");
  c = fgetc(ptr);
  while(c != EOF){
    printf("%c", c);
    c = fgetc(ptr);
  }
  return 0;
}
```

→ EOF = End of file

fgetc returns EOF when all the characters from a file has been read. So we can write a chuck like below to detect end of file

```
while(1){
  ch = fgetc(ptr);
  if(ch == EOF){          // when all the content of a file has been
    break;
  }
}
```

Eg: WAP to read the integers from a file.

pr.txt

24 27 25

```
#include <stdio.h>
int main(){
  FILE *ptr; int a, b, c;
  ptr = fopen("pr.txt", "r");
  fscanf(ptr, "%d%d%d", &a, &b, &c);
  return 0;
  printf("The values of a, b, c %d %d %d", a, b, c);
  return 0;
}
```

Eg: WAP to generate multiplication table of a given number into text format. Make sure that file is readable and well formated

```
#include <stdio.h>
int main(){
  FILE *ptr;
  int num;
  printf("The table of number \n");
  scanf("%d", &num);
  ptr = fopen("table.txt", "w");
  for(int i=0; i<10; i++){
    fprintf(ptr, "%d x %d = %d", num, i+1, num*(i+1));
  }
  fclose(ptr);
  return 0;
}
```

Eg. WAP to read a text file character by character and write this content twice in a separate file

```
#include <stdio.h>
int main (){
    FILE *ptr1;
    FILE *ptr2;
    ptr1 = fopen("a.txt","r");
    ptr2 = fopen("b.txt","w");

    char c = fgetc(ptr1);
    while (c != EOF){
        fputc(c, ptr2);
        fputc(c, ptr2);
        c = fgetc(ptr1);
    }
    fclose(ptr1);
    fclose(ptr2);
    return 0;
}
```

Hellow Harry

→ HHeellloowww HHaarryy

Eg: Take name and salary of two employees as input from the user and write them to a text file in following format:

name1, 3000
name2, 7700

Eg: WAP to modify the file containing an integer to double its

Eg: Project : Game Snake, water, gun
WAP to develop a game playable and print the result after you choose any of the snake, gun, water and on matching which computer choose gunhaki the result.

```
# include < stdio.h >
# include < stdlib.h >
# include   < time.h >

int main ( ) {
  char you, comp;

  srand ( time (0))
  number = rand ( ) % 100 + 1;

  if (number < 33) {
      comp = 's';
  }
  else if ( number > 33 && number < 66 ) {
      comp = 'w';
  }
  else {
      comp = 'g';
  }

  printf (" The 's' for snake, 'w' for water, 'g' for gun);
  scanf ("%c", & you);
  End result = snakewatergun (you, comp);
  printf (" You chose %c and computer choose %c", you, comp);
  if (result == 1 || 0 || -1)
  printf
```

```
printf("
if (result == 0) {
    printf("The game is draw \n");
}
else if (result == 1) {
    printf("You win \n");
}
else {
    printf("You lost \n");
}
return 0;
}

int snakewatergun(char you, char comp) {
// 1 = win, 0 = draw, -1 = lost
if (you == comp) {                    // for draw
    return 0;
}
if (you == 's' && comp == 'w') {      // win
    return 1;
}
else if (you == 'w' && comp == 's') { // lost
    return -1;
}
if (you == 's' && comp == 'g') {      // for lost
    return -1;
}
else if (you == 'g' && comp == 's') { // win
    return 1;
}
if (you == 'w' && comp == 'g') {      // for lost
    return -1;
}
```

```
else if (you == 'w' && comp == 's') { // win
    return 1;
}

if (you == 'g' && comp == 's') {      // win
    return 1;
}
else if (you == 's' && comp == 'g') { // lost
    return -1;
}
}
```

- Dynamic memory allocation

C mai jab program run hota hai toh usko stack mai memory milti hai
Ek function ko run kiya and uske variables ki ek stack memory bnegi un variables ke liye.

Stack Memory	Heap Memory
→ Static memory	→ dynamic memory
→ Destroyed when prg ends	→ Memory is shared with libs, fullcode and on spot memory allocate hoti hai

Heap mai on demand kitni space need hai utni allocate ho jati hai and stack mai fixed amt of memy code ko milti hai means 100 integer array create kiya par use sirf 10 spaces ka kiya 90 waste but for bhi memory 100 ki hi allocated rehti hai

Array banaya tha 7 students ke liye enroll 10 ne kardiya

→ Dynamic Memory Allocation
The way to allocate memory to a data structure during the runtime. We use any DMA function available in C to allocate and free memory during runtime.

→ functions DMA in C
1) malloc()
2) calloc()
3) free()
4) realloc()

1) malloc function(): It stands for memory allocation.
It takes number of bytes to be allocated as an input and returns a pointer of type void.

Syntax: ptr = (int*) malloc(30 * sizeof(int))
(int*): casting void pointer to int; 30: space for 30 ints; sizeof(int): returns size of 1 int

The expression returns a null pointer if the memory cannot be allocated

Eg:
```
#include <stdlib.h>
#include <stdio.h>
int main(){
    int *ptr;
    ptr=(int*)malloc(6*sizeof(int)); //typecast is a must as it returns void pointer
    return 0;
}
```

The expression returns a null pointer if memory cannot be allocated

Eg: For taking inputs:

```
# include < stdlib.h >
# include < stdio.h >
int main (){
  int *ptr;
  ptr = (int *) malloc (6 * sizeof (int));
  for (int i = 0 ; i <6 ; i++){
    printf (" Enter value %d : \n", i);
    scanf ("%d", &ptr[i]);
  }
  for (int j = 0 ; j<6 ; j++){
    printf (" The value %d : %d \n ", i, ptr[i]);
  }
  return 0;
}
```

2) calloc (.) function : It initializes each memory block with a default value of 0.

memory kiya kitna size

Syntax: ptr = (float *) calloc (30, sizeof (float));

Allocate contigous space in memory for 30 blocks (float)

If the space is not sufficient, memory allocation fails and a null pointer is returned

Eg: default value of initialization is 0

```
# include < stdio.h >
# include < stdlib.h >
int main (){
  int *ptr;
  ptr = (int *) calloc (6, sizeof (int));
  for (int i=0 ; i<6 ; i++){
    printf (" Enter value of %d \n", i);
    scanf ("%d", &ptr[i]);
  }

  for (int i = 0 ; i<6 ; i++){
    printf (" The value %d : %d \n ", i, ptr[i]);
  }
  return 0;
}
```

not used then calloc will initialize value 0 on its own as inputs

3) free() function: We can use it to deallocate memory where as when calloc / malloc used deallocation is not automatically.

while calloc / malloc se memory use jitni karni thi thodi and for free jng use karke bachi hua space delete

Syntax: free (ptr); Memory of ptr is released

~~4) realloc () function: Sometimes~~

Eg: use of free function:

```
# include < stdio.h >
# include < stdlib.h >
int main (){
 int * ptr;
 int * ptr2;
 ptr = (int *) malloc (6 * sizeof (int));
 for (int i = 0 ; i < 600 ; i++){
   ptr2 = (int *) malloc (60000 * sizeof (int));
   printf ("Enter value %d : \n", i);
   scanf ("%d", & ptr [i]);
   free (ptr2);
 }
  for (int i = 0 ; i < 6 ; i++){
   printf ("The value are %d : %d \n", i, ptr [i]);
 }
  return 0;
}
```

4) realloc () function: Sometimes the dynamically allocated memory is insufficient or more than required.

realloc is used to allocate memory of new size using the previous pointer and size.

Syntax: ptr = realloc (ptr, nsize);

ptr = realloc (ptr, 3 * sizeof (int));
ptr now points to this new block of memory capable of storing 3 integers

Eg:

```
# include < stdio.h >
# include < stdlib.h >
int main (){
  int * ptr;
  ptr = (int *) malloc (6 * sizeof (int));
   printf ("Enter values %d ", i);
   scanf ("%d", & ptr [i]);
 }
  for (int i = 0 ; i < 6 ; i++){
   printf ("The values are %d \n", ptr [i]);
 }

  ptr = realloc (ptr, 10 * sizeof (int));

  for (int i = 0 ; i < 10 ; i++){
    printf ("Enter values %d \n", i);
    scanf ("%d", & ptr [i]);
 }
   for (i = 0 ; i < 10 ; i++){
    printf ("The values are %d : %d \n", i, ptr [i]);
 }

  return 0;
}
```