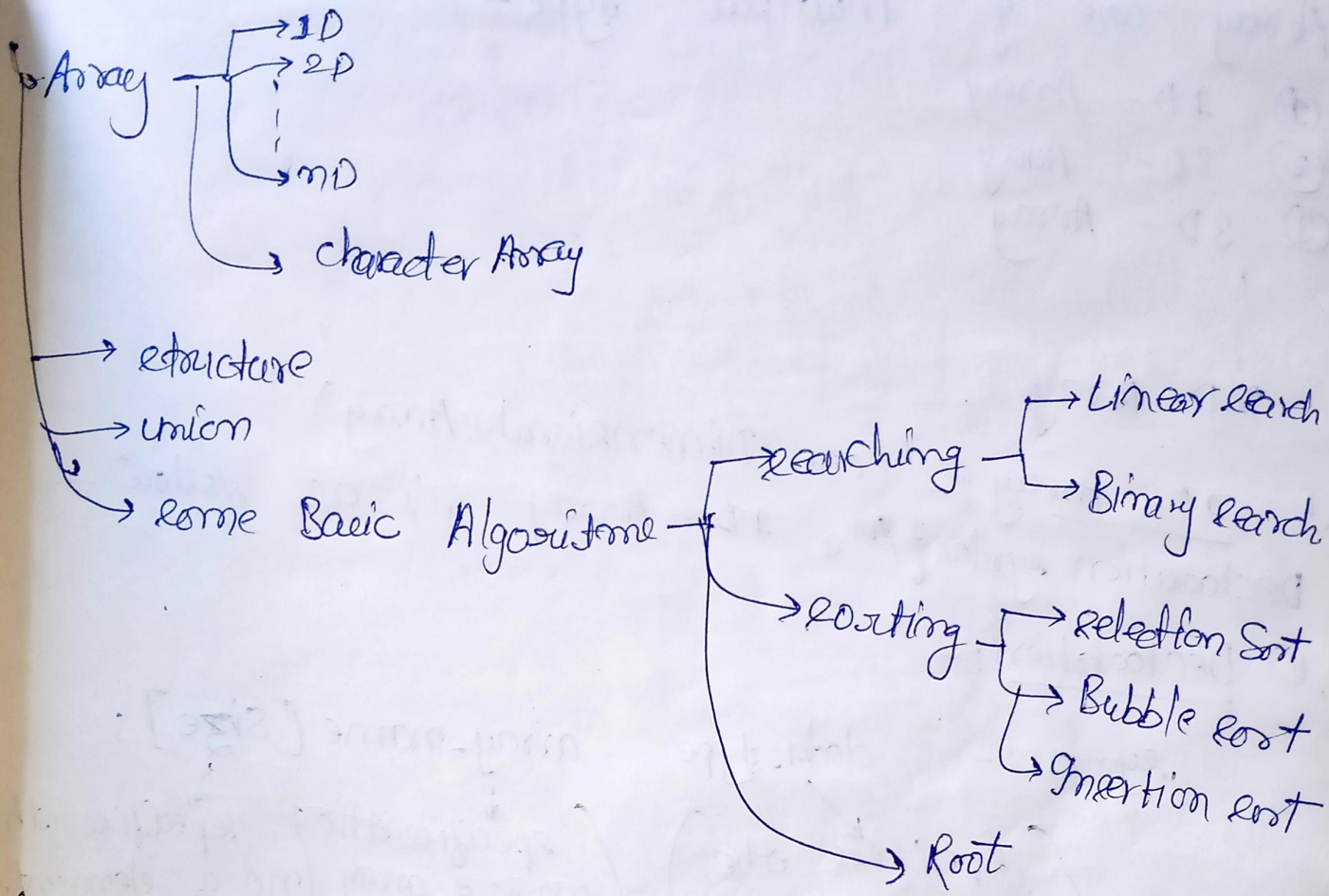
Unit-4



Array:-

Array is a finite collection of Homogeneous elements in Contiguous Memory locations.

Here finite means fixed and finite size (no. of element)
Homogeneous means all elements are of same datatype.
Contiguous Memory locations means Consecutive memory locations

The elements will be store in sequence one after another in memory.

1D Array :- (one Dimen

Array are of Multipal type -

1. 1D - Array
2. 2D - Array
3. 3D - Array

⋮

nD - Array

① 1D - Array :- (one Dimensional Array)

Declaration syntax of a 1D - Array is as follow :-

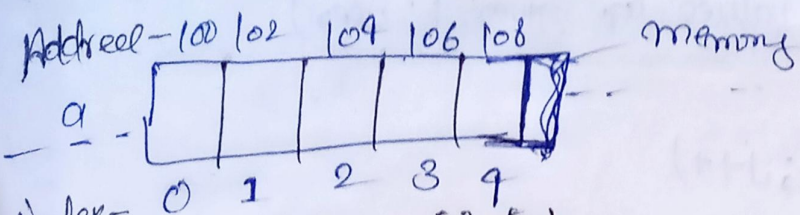
① Declaration :-

syntax :- data_type array_name [size] ;

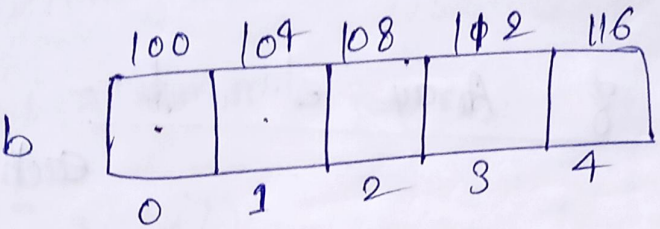
- data_type : specifies the type of elements that will be stored in the array.
- array_name : specifies the name of array which user defined
- [size] : specifies the no of elements in the array.

for ex.

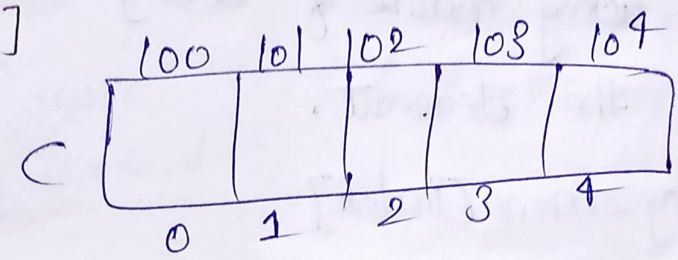
```
int a[5];
Char c[5];
float b[5];
```



for float b[5]



for char c[5]



② Initialization :-

(a) Individually :-

```
a[0] = 5 ;
a[1] = 10 ;
a[2] = 3 ;
a[3] = 8 ;
a[4] = 5 ;
```

(b) Altogether Declaration :-

```
a[5] = { 5, 10, 3, 8, 5 }
```

© Using loops :- (let values are given by user)

```
for (i=0; i<=4; i++)
{
    scanf("%d", &a[i]);
}
```

③ Accessing of Array elements :-

each element of array is accessed using name of array along with the index of the element.

array_name[index]

Input Array

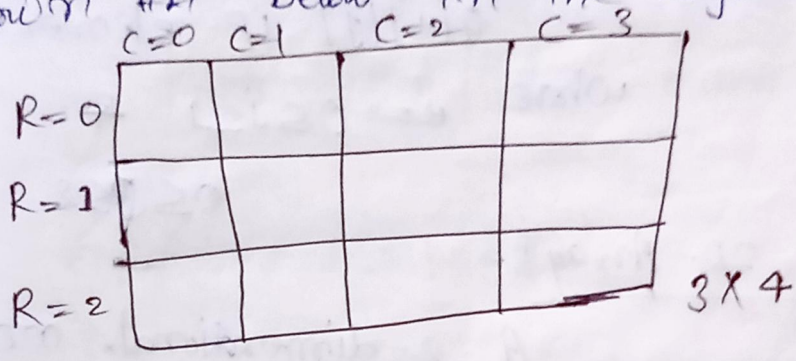
```
Void main()
{ int a[5], i;
// input array
for(i=0; i<=4; i++)
{ scanf("%d", &a[i]);
}
for (i=0; i<=9; i++) // Printing array
{ printf("%d", a[i]);
}
getch();
}
```

```
/* Program to sum the values in an array */
void main()
{  int a[5], i, sum=0;
   // Input Array
   printf("Enter the elements of array");
   for(i=0, i<=4, i++)
   {  scanf("%d", &a[i]);
   }
   // sum logic
   for(i=0, i<=4, i++)
   {  sum= sum+a[i];
   }
   printf("sum of array elements = %d", sum");
   getch();
}
```

Two dimensional Array [2D-Array]

A Two dimensional Array is an Array of Arrays where it can be considered that a 2D Array is nothing but a 1D-Array where each element is itself an Array.

A Two-D array can be visualized as an arrangement of elements in Row-Column formate. As shown in below in the digram.



A Two dimensional array can be declared initialize and accessed in following ways.

(1) Declaration of 2D Array :-

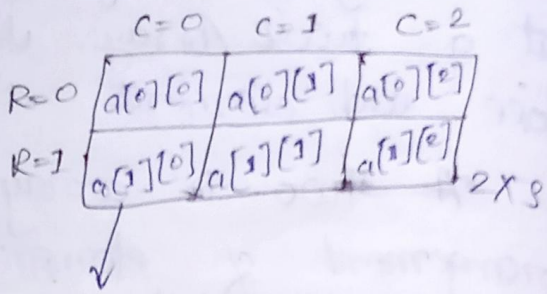
The syntax of declaration of a 2D-Array is given below -

```
data_type  array_name [Row_style][Column_style]
```

- data_type → specifies data type of element to be stored in the 2D Array
- array_name → [User defined name given to the 2D-Array]
- Row_style → specifies No of Row
- Column_style → No of Column

For e.g- Consider a 2D Array having 2 Row and 3 Columns and stores integer values. then it can be declared as-

```
int [2][
    int a[2][3];
```



a[i][j] is element name where i= $0 \le j \le 1$ & $0 \le j \le 2$

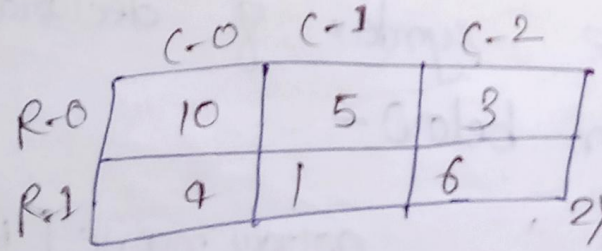
(2) Initialization of 2D-Array:-

A 2-dimensional array can be initialized in any of the following three ways-

(2.1) Individually each element:-

```
R-0 { a[0][0] = 10;
      a[0][1] = 5;
      a[0][2] = 3;

R=1 { a[1][0] = 4
      a[1][1] = 1
      a[1][2] = 6
```



= 6 elements

(22) Initialization with declaration :-

Row, Column

int a[2][3] = {10,5,3,4,1,6}

R-0 R-1

or

int a[2][3] = { {10,5,3}, {4,1,6} }

R-0 R-1

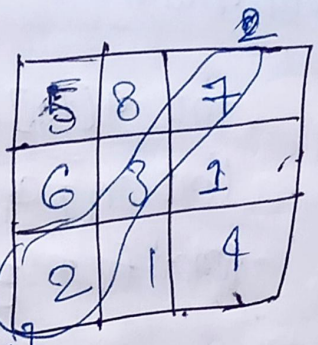
(23) Using loops (Nested)

```
for (i=0; i<=1; i++)
{
    for(j=0; j<=2; j++)
    {
        scanf("%d", &a[i][j]);
    }
}
```

i=0, j=0 , a[0][0]
j=1 , a[0][1]
j=2 , a[0][2]

i=1, j=0 , a[1][0]
j=1 , a[1][1]
j=2 , a[1][2]

```
void main()
{ int a[3][3], sum, i, j;
  for(i=0; i<=2; i++)
  {   for(j=0; j<=2; j++)
      scanf("%d", &a[i][j]);
  }
j=2;
for(i=0; i<=2; i++)
{   for(j=2; j>=0; j++)
    sum = sum + a[j][j];
}   j--;
// End of main.
```



3×3

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

/* M.I Matrix multiplication */

c[0][0] = a[0][x] * b[x][0]
c[0][1] = a[0][x] * b[x][1]
c[1][0] = a[1][x] * b[x][0]
c[1][1] = a[1][x] * b[x][1]

$0 \le x \le 2$

c[0][0] = a[0][0] * b[0][0] + a[0][1] * b[1][0] + a[0][2] * b[2][0]

c[0][1] = a[0][0] * b[0][1] + a[0][1] * b[1][1] + a[0][2] * b[2][1]

```
Void main()
{
    int a[10][0], b[10][10], c[0][10], i, j, x;
    int r1, c1, r2, c2;
    printf("Enter the number of rows & Column of matrix a[][]");
    scanf("%d%d", &r1, &c1);
    printf("Enter the number of rows & Column of matrix b[][]");
    scanf("%d%d", &r1, &c1);
    if (c1 != r2)
        printf("Matrices are not compatible for mulu");
    else
    { // Input Matrix a[][]
        for(i=0; i<=r1-1; i++)
        {
            for (j=0; j<=c1; j++)
                scanf("%d", &a[i][j]);
        }
        // Input Matrix b[][]
        for(i=0; i<=r2-1; i++)
        {
            for (j=0; j<=c2-1; j++)
                scanf("%d", &b[i][j]);
        }
```

// logic for multiplication.

```
for (i=0; i<=r1-1; i++)
{  for (j=0; j<=c2-1; j++)
   {  c[i][j]=0;
      for (x=0; x<=c1-1; x++)
      {
         c[i][j] = c[i][j] + a[j][x] * b[x][j];
      }
   }
// Display matrix c[][]
for (i=0; i<=r1-1; i++)
{  for (j=0; j<=c2-1; j++)
   {  printf ("%d", c[i][j])
   }
   printf ("\n");
}
getch();
}
// End of main().
```

$[\ ]_{2\times3} * [\ ]_{3\times}$

$= [\ ]_{3\times3}$

$x = 2 = c_1 - 1$

-: strings :-

In C programming a string is define as a sequence of characters terminated by a null character written as '\0'.

for ex. "Rahul", "Roy 1", "Rajnish Tripathi".

In C language strings are represented and stored as character array. Where each element of the character array stores a single character, of the string (in sequence)

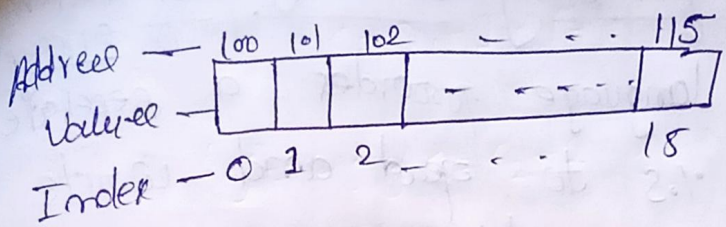
Thus a character array can be declared to store some string the charcter array can be declared follow —

```
char arr_name[size];
```

- `char` — data type of the elements in the array (which is character here)
- `arr_name` — user defined name given to the array
- `[size]` — No of characters of the string to be stored +1 atleast

for example. the declaration of a character array 'name' which can hold, which can store the name of some person maximum upto 15 characters along, as follow.

char name[16];



Initialization:-

The elements of a character array can be initialized in following two most common ways

eg. char name[6] = {'R', 'a', 'h', 'u', 'l', '\0'};

or n[0] = 'R';

or

char name[] = "Rahul"

↓

note - here the string Rahul contain 5 character so the Compiler automatic alocates 6 memory locations (1 extra for string null character)

Other

or for(i=0; i<=4; i++)

{ printf("%c", name[i]);

}

Better way ↓

for(i=0, n[i] != '\0'; i++)

{ printf("%c", name[i]);

}

Reading and writing of strings using character array:

In c programming language provides a seperate Access specifier i.e- %s to read and write sequence of characters of a string

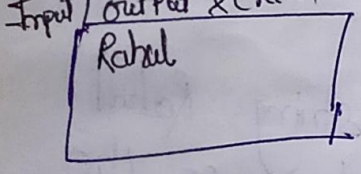
Following is the example of reading sequence of characters using '%s' Access specifier.

```
char name[12];
```

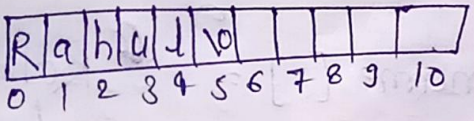
Read/Input string

```
scanf("%s", name);
```

Input/output screen



Write/output string



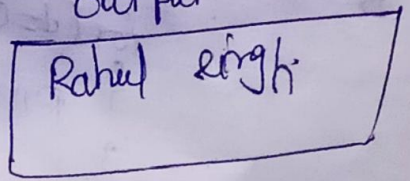
The drawback of %s Access specifier that it stops reading the input as soon as it Encounts a whitespace (Normal space, Tab space etc)

There is another built in function provided by c which can read the input character sequence that include white space too.

In a character array Can be display using %s specifier as below-

```
printf("%s", name);
```

output



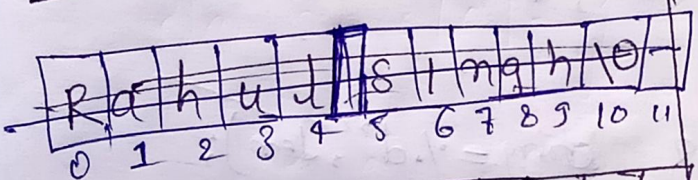
gets(name);

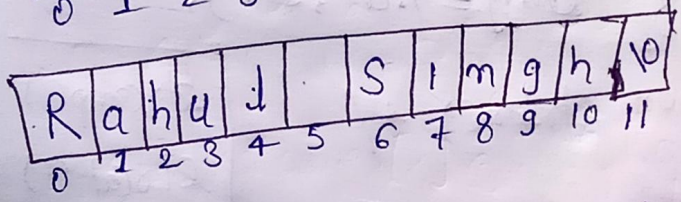
A syntax of using built in function 'gets' is as follows.

gets(name);

Input / output

Rahul singh





another way to print entire character array is to use built in function 'puts' as following.

puts(name);

output

Rahul singh

/* Program to find the length of an input string */

```
Void main()
{
    char S[] = "Rajnish Tripathi";
    int len=0, i;
    for(i=0;  S[i] != '\0' ; i++)
    {
        len++;
    }
    printf("length of string is = %d", len);
    getch();
}
```

Library function in <string.h>	use	Description
① strlen( )	strlen("Hello") or strlen(s)	Find the length of string (or array of char)
② strcpy(,)	strcpy(d, "Hello") strcpy(d,s)	It copies the string in second argument (direct string or char Array) into first argument (Character Array)
③ strcmp(,)	strcmp(S_1, S_2) strcmp(S_1, "Hello") strcmp("Hello", S_2) strcmp("Hello", "Hello")	Compare both the string arguments S_1, S_2 and if they matched return 0 otherwise return Non zero value.
④ strcat(,)	strcat(d, "Hello") strcat(d,s);	Concatenates second argument (string or char array) to first argument

Eg. char t[20] = "Rahul";
char s[7] = "_Singh";
strcat(t,s)
Puts(t);
getch();
}

O/P
Rahul Singh

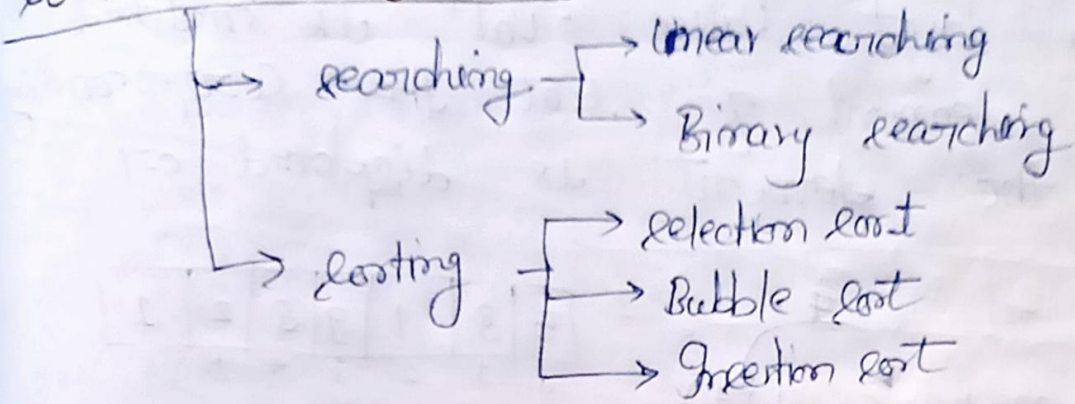
t

R	a	h	u	l	'\0'														
0	1	2	3	4	5	6													19

s

	S	i	n	g	h	'\0'
0	1	2	3	4	5	6

Some basic algorithm:-



searching:-

In searching in programming searching is a process to find whether a target element is present in the given ~~tree~~ list of element or not. If the given element is present in the list then what is the location of the target element in the list.

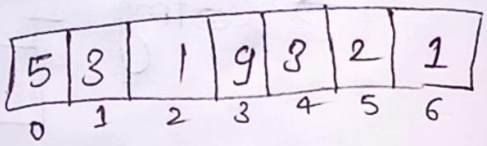
Following two common search Algorithm ↓ or technique Computer science —

1. Linear search
2. Binary search

linear search:-

In linear search technique the target element (val) is sequentially compaired with each element in the list 'l' one by one from beginning to end. If at any

Position the target value 'val' is matched with the element of list the corresponding position of the element is displaced or returns.



Eg.

```
for (i=0; i<=6; i++)
{   if (val==a[i])
    {  printf("the target element is present
                                at index = %d", i);
       break;
    }
}
if (i==7)
  printf("Not found");
getch();
```

or

```
int val, i, a[7], flag=0;
for(i=0; i<=6; i++)
{   if (val == a[i]
    { printf("the target element is present at index=%d",i);
      flag=1;
    }
}
if (flag==0)
  printf("Not found");
getch();
```

(2) Binary Search :-

Binary search technique is the faster search mechanism as compared to linear search method. To apply binary search technique the elements given in the list must be in the sorted sequence.

In this technique first of all the element present at the middle position is matched with the target value to be search. If the value is found at the middle position then corresponding index (position) is returned otherwise the process is repeated with the left half or the right half of the list depending on whether the target value is smaller or larger than the value present at the mid position.

Algorithm :-

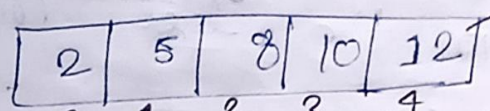
step 1:- Find the middle element at index 'mid'

step 2:- Compare the value at mid with target value 'Val'

step 3:- If value 'Val' is matched with middle value, then return middle value otherwise check whether 'Val' is smaller or larger than mid value.

step 4:- Repeat the process from step 1 to step 3 with left half if val is lesser than mid value otherwise right half.

Ex.



element should be sorted

Val = 10

int l=0, u=4

// l = lower limit
u = upper limit

cycle 1:

mid = (l+u)/2
= (0+4)/2
= 2

a[2] ≠ 10

a[2] < val

l = mid + 1

cycle 2:- l=3, u=4

mid = (3+4)/2 = 3

a[3] ≠ val

l = mid+1 = 4

~~cycle 3~~ - ~~l=4, u=4~~

~~mid = (4+4)/2 = 4~~

~~a[4] ≠ 0~~

Val = 15 l = 0 , u = 4

cycle1 :- mid = (l+u)/2

= 4/2 = 2

a[2] ≠ 15

a[2] < val

l = mid + 1

cycle2 :- l = 3, u = 4

mid = 3+4/2 = 3

a[3] ≠ val

l = mid + 1 = 4

cycle3 :- l = 4 , u = 4

mid = (4+4)/2 = 4

a[4] ≠ val

l = mid + 1 = 5

l > u

Hence proceed stops and element not found.

```
/* WAP to search any value in an array */
void main()
{ int a[10], l=0, u=9, mid, i, val;
  printf("Enter the list of element in sorted order");
  for (i=l; i<=u; i++) // logic to input array
      scanf("%d", &a[i])
  printf("Enter the target value to be searched");
  scanf("%d", &val);
  // logic for Binary search
  while (l<=u)
  { mid = (l+u)/2;
    if(a[mid] == val)
    { printf("element is present at %d", mid);
      break;
    }else
    { if (a[mid]<val)
      { l = mid+1; // change for right half.
      }else
        u=mid-1; // change for the left half
    }
  } // End of while loop
  if(l>u)
  printf("element not found");
} getch();
// End of main function
```

Pseudo Code :-

BinSearch (a, Val, l, u)

```
1.  While l<=u
2.      do mid ← (l+u)/2
3.      If (a[mid] = Val)
4.          then print (Index mid)
5.               break
6.          else if (val < a[mid])
7.               then  u ← mid-1
8.               else  l ← mid+1
9.  If (l>u)
10. then print ("Not found")
11. End
```

in Pseudo code all values start from '1' not from zero.

{ only value is printed so no improper of double code

{ "" is for string.

Sorting :-

In Programming sorting is defined as the process of arranging the elements in a given list in increasing or decreasing order. Following are most Commonly used technique of sorting -

1. Selection sort
2. Bubble sort
3. Insertion sort

Ex Input list

8	1	5	2	4	7	6

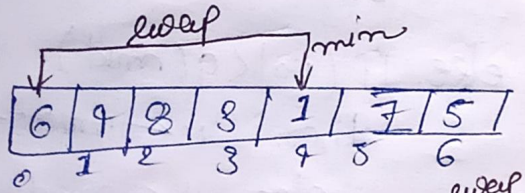
output list

1	2	3	4	5	6	7	increasing order

or

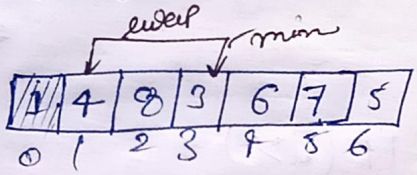
7	6	5	4	3	2	1	decreasing order

1. Selection sort :-



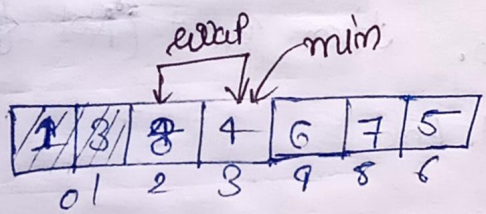
Iteration-1

min = 1
loc = 4



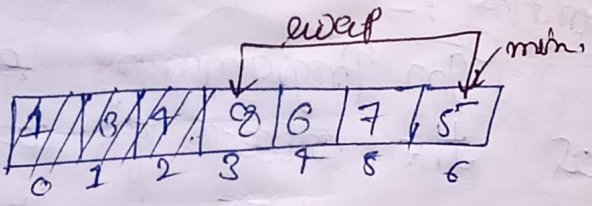
Iteration-2

min = 3
loc = 3



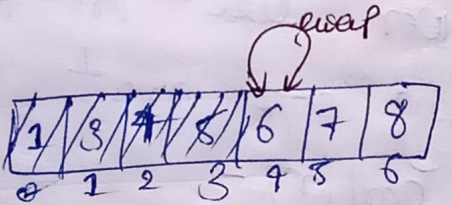
Iteration-3

min = 4
loc = 3



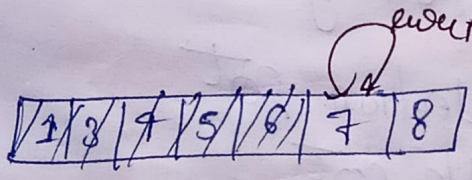
Iteration-4

min = 5
loc = 6



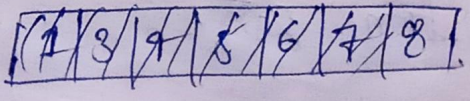
Iteration-5

min = 6
loc = 4



Iteration-6

min = 7
loc = 5



Definition / Algorithm :-

step-1 :- Find the minimum element in unsorted list

step2 :- swap min element with the first element in unsorted part.

step-3 :- Repeat step1 & step-2 till array is sorted

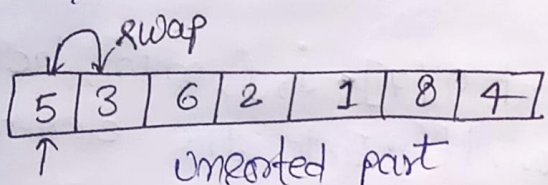
```
Void main ()
{
    int a[7], i, j, min, loc.
    printf("Enter the elements of an array");
    for(i=0; i<=6; i++)
        scanf("%d", &a[i]);
    // logic for selection sort
    for(i=0; i<=5; i++)
    {
        // minimum finding in unsorted part
        min = a[i];
        loc = i;
        for(j=i; j<=6; j++)
        {
            if(a[j] < min)
            {
                min = a[j];
                loc = j;
            }
        }
        // swapping with first element in unsorted part
        a[loc] = a[i];
        a[i] = min;
    }
```

Bubble sort :-

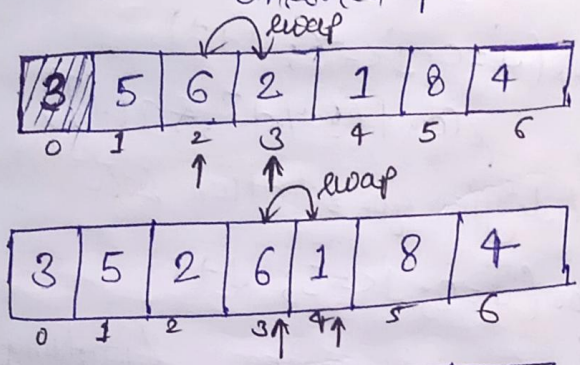
In bubble sort technique in each cycle the (adjecent element (from beginning to end) in the unsorted part of given list are compare and swapped if the first element is larger than second element in this way after one cycle the largest element reches in the last.

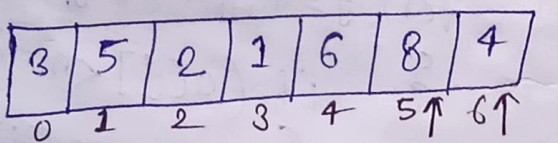
This process is Repeated again and again till the entire array is sorted.

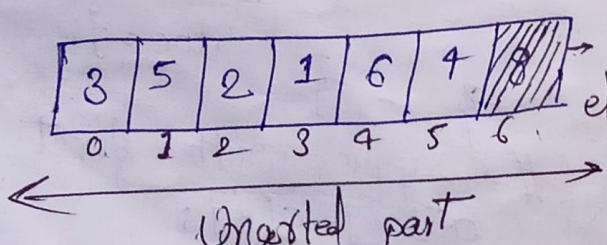
Input array -



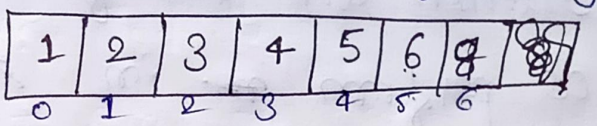
Iteration - 1







Repeating the procedure again and again for remaining unsorted part, we get following sorted array.




```
/* Void main()
{
    int a[7], i, j, t;
    printf("Enter the list of elements");
    for(i=0; i<=6; i++)
    {
        for(j=0; j<=5-i; j++)
        {
            if(a[j] > a[j+1])
            {
                t = a[j];
                a[j] = a[j+1];
                a[j+1] = t;
            }
        }
    }
    // Print array a[]
    for(i=0; i<=6; i++)
        printf("%d", a[i]);
    getch();
}
```

Insertion sort :-

Insertion sort follows the same processes as perform in sequencing method used in playing card game. Insertion sort follows following step.

step-1 - Consider the first element as sorted one.

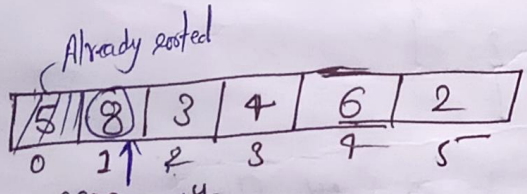
step-2 pick the first element from unsorted list.

step-3 scan the sorted list (in the left)

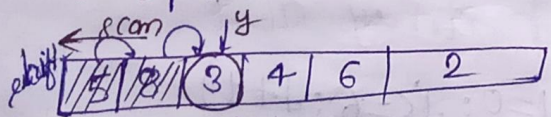
step-4 Insert the pick element after the first element that is smaller than it.

step-5 Repeat step② to step④ until entire list is sorted.

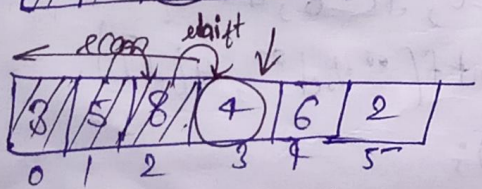
Input array



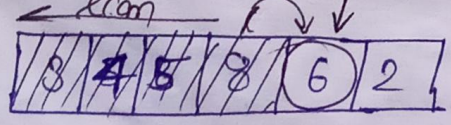
Pass-1/ Iteration-1



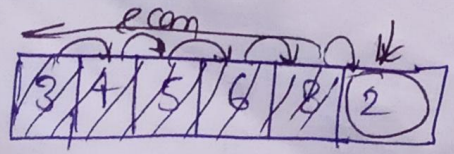
Pass-2/



Pass-3



Pass-4



Pass-5:- | 2 | 3 | 4 | 5 | 6 | 8 | Array is sorted.

```
# Void main()
{   int a[6], i, y, j;
    printf("Enter the element of list");
    for(i=0; i<=5; i++)
        scanf("%d", &a[i])
    for(i=1; i<=5; i++)
    {   y = a[i]; // Picked element to be sorted.
        for(j=i-1; j>=0; j--) // scan left in sorted list
        {   if(a[j]>y)
            {   a[j+1] = a[j];
            }
            else
                break;
        }
        a[j+1] = y;
    }
// Printing the sorted Array
for(i=0; i<=5; i++)
{   printf("%d", a[i])
}
getch();
}
// End of main function
```