Revise

:- ARRAY :-

It is the collection of similar data elements stored in contigious memory.

Formula to calculate memory :-

address(a[i]) = b + i*w

w = word size

b = base address

i = index no.

Ex - int a[5];

5000	5002	5004	5006	5008
a[0]	a[1]	a[2]	a[3]	a[4]

address of a[3] = b + i x w

= 5000 + 3 x 2

= 5006

operations on array :-

1) Traversing 2) Insertion 3) Deletion 4) Searching 5) Sorting

(1) Traversing :- This operation is used to visit all the elements in an array.

'c' code

```
void traverse(int a[], int n)
{
  int i;
  for (i=0; i<n; i++)
    process(a);
}
```

Algorithm

1) [Initialize counter]
   set i=0
2) Repeat step 3 & 4 till i<n
3) Apply process to a
4) set i = i+1
   end of for loop
5) Exit.

(2) Insertion :- The operation is used to insert an element into an array provide it is not full.

'c' code

```
void insert( int a[], int n, int pos, int item)
{
  int i;
  for(i=n-1; i>=pos; i--)
    a[i+1] = a[i];
  a[pos] = item;
  n=n+1;
}
```

Algorithm:

void insert(int a[], int n, int pos, int item)

1) [initialize counter] set i = n-1
2) Repeat step 3 & 4 till i >= pos
3) set a[i+1] = a[i]
4) set i = i-1
   end of for loop
5) set a[pos] = item
6) set n = n+1,

3) Deletion : The operation is used to delete an element from an array.

c' code:

```
void delet(int a[], int n, int pos)
{
int i;
for(i = pos; i<(n-1; i++)
    a[i] = a[i+1];
n = n-1;
}
```

Algorithm

void delet(a[], n, pos)

1) [Initialize counter]
   set i = pos
2) Repeat step 3 & 4 till i<(n-1
3) set a[i] = a[i+1]
4) ~~n = n-1~~
   Set i = i+1
5) set n = n+1
6) Exit

4) Searching:

```
void search(int a[], int n, int item)
{
int i, c=0;
for(i=0; i<n; i++)
  {if(a[i] == item)
      c++;
  }
if(c>0)
  Pf("Element found")
else Pf("Element Not found");
}
```

Algorithm:

1) void search(a[], n, item)
1) [Initialize counter]
   i=0, c=0
2) Repeat step 3 to 5 till i<n
3) if a[i] = item
4) set c = c+1
5) set i = i+1
6) if c>0
7) write element found
8) else
   write Not found
9) Exit.

5) Sorting :

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

Algorithm

void Sort(a[], n)

1) Initialize i=0
2) Repeat step 3 to 10 till i<n-1
3) set j = i+1
4) Repeat step 5 to 9 till j<n
5) if a[i] > a[j]
6) set temp = a[i]
7) set a[i] = a[j]
8) set a[j] = temp
9) set j = j+1
   end of for loop
10) set i = i+1
    end of for loop
11) Exit.

Memory Representation of a matrix :-

There are 2 convention of storing any matrix in memory.

(1) Row major order :-

In row major order elements of a matrix are stored on a row by row basis, that is all the elements of first row are filled, then in second row and so on.

a[3][4] →

a[0][0] → a[0][1] → a[0][2] → a[0][3]

a[1][0] ← a[1][1] → a[1][2] → a[1][3]

a[2][0] ← a[2][1] → a[2][2] → a[2][3].

a[0][0]	a[0][1]	a[0][2]	a[0][3]	a[1][0]	a[1][1]	a[1][2]	a[1][3]	a[2][0]	a[2][1]	a[2][2]	a[2][3]

if array of a[m][n] is there &

Calculate the address of a[i][j] formula is

$$add(a[i][j]) = b + ((i - LBR) \times n + (j - LBC)) \times w$$

b = base address
i = row no
j = col no
w = word size
n = total no of columns

LBR :- Lower bound row (starting index of row)

LBC :- Lower bound col (starting index of column)

Q) Calculate the address of a[7][8] element of matrix a[20][25]?

$$add(a[7][8]) = b + ((i - 0) \times n + (j - 0)) \times w$$

Let starting address is 1000, & word size is 2 byte

$$= 1000 + (7 \times 25 + 8) \times 2$$

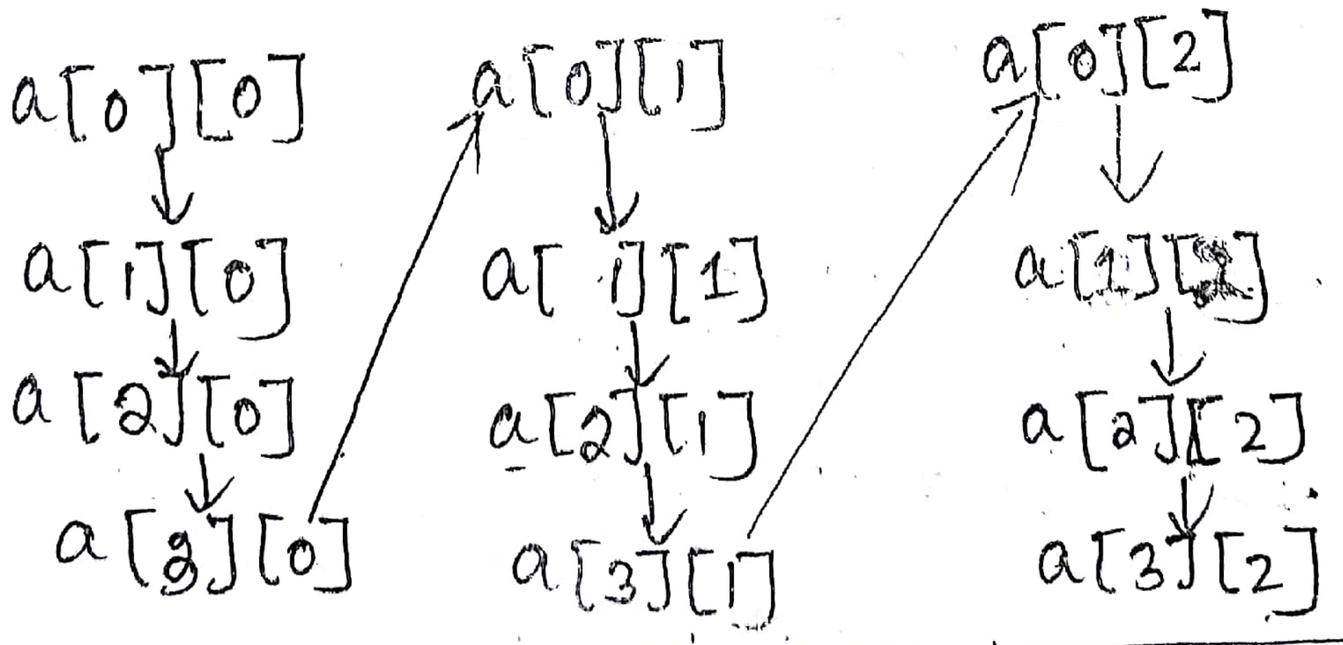
$$= 1000 + (175 + 8) \times 2$$

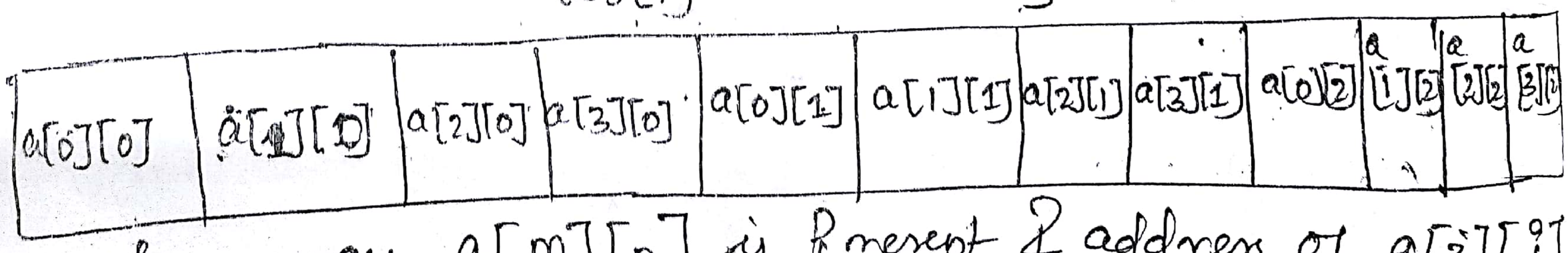
$$= 1366.$$

(ii) Column-major order :-

In this elements are stored column by column that is all the elements in first column are stored in their order of rows, then the 2nd column & so on

Ex :- a[4][3]





of array a[m][n] is present & address of a[i][j]

element of array a[m][n] is

$$add(a[i][j]) = b + ((j - LBC) \times m + (i - LBR)) \times w$$

where m is total no. of row.

Q) Suppose an array VAL[15][10] is stored in the memory with each element requiring 4 bytes of storage in c programming. If the base address of array VAL is 1500, determine the location of VAL[12][9] when the array VAL is stored (i) row-major order (ii) column-major order

Ans:

$m = 15$ $\quad w = 4$ $\quad LBR = 0$

$n = 10$ $\quad i = 12$ $\quad LBC = 0$

$b = 1500$ $\quad j = 9$

(i) $add(a[i][j]) = b + ((i - LBR) \times n + j) \times w$

$= 1500 + (12 \times 10 + 9) \times 4$

$= 2016$

(ii) $add(a[i][j]) = b + ((j - LBC) \times m + (i - LBR)) \times w$

$= 1500 + (9 \times 15 + 12) \times 4$

$= 2088$

Sparse Matrix -:

> A matrix A is said to be sparse if many of its elements are zero.

> A matrix that is not sparse is called dense

> It is not possible to define an exact boundary between dense and sparse matrix.

Ex -:

$$\begin{bmatrix} 7 & 0 & 0 & 1 & 0 & 2 \\ 0 & 1 & 9 & 0 & 0 & 0 \\ 0 & 0 & 0 & 7 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 8 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 3 & 0 & 0 & 0 \end{bmatrix}$$

Array Representation of sparse Matrix (Triplet form):-

> In array Rep^n an array triplet of type
(row, col, element)

> It is used to store the elements which are nonzero where ~~first~~ field of the triplet is used ~~is used~~ to record row position, second to column position, and third one to record the nonzero element of sparsematrix

	0 Row	1 col	2 Nonzero element
A[0]	6	6	8
[1]	0	0	7
[2]	0	3	1
[3]	0	5	2
[4]	1	1	1
[5]	1	2	9
[6]	2	3	7
[7]	4	0	8
[8]	5	2	3

Array Matrix of Sparse Matrix would be

A[9][3]
- 1st for row no
- 2nd for col no
- rrd for nonzero values.

A[0] → 8 nonzero value
A[1] → row is used for storing total row, col, nonzero values

Program:

WAP for sparse matrix -:

```
#include <stdio.h>
#include<conio.h>
void main()
{
int x[10][10], y[10][3];
int i, j, col, row, k=1, z=0, nz=0;
clrscr();
Pf("Enter row & col size");
sf("%d %d", &row, &col);
for(i=0; i<row; i++)
  {
  for(j=0; j<col; j++)
    {scanf("%d", &x[i][j]);
    }
  }
```

```
It is now the matrix is ...");
for(i=0; i<row; i++)
  {
  for(j=0; j<col; j++)
    {
    printf("%d", a[i][j]);
      if(a[i][j] != 0)
        k++;
    else
      z++;
    }
  printf("\n");
  if(nz < z)
  {
  b[0][0]=row;
  b[0][1]=col;
  b[0][2]=nz;
for(i=0; i<row; i++)
{
for(j=0; j<col; j++)
  {
  if(a[i][j] != 0)
    {b[k][0] = i;
    b[k][1] = j;
    b[k][2] = a[i][j];
    k++;
    }
  }
}
printf("The sparse matrix is \n");
for(i=0; i<k; i++)
  {for(j=0;j<3;j++)
    {printf("%d", b[i][j]);
    }
  printf("\n");
}
```

```
    }
else
    {
    pf("Not a sparse Matrix");
    }
 getch();
}
```

WAP to delete all the occurance of an element in an array.

```
void main()
{ int i, a[10], n, el, j;
clrscr();
pf("Enter the size\n");
sf("%d", &n);
pf("Enter elements");
for(i=0; i<n; i++)
  sf("%d", &a[i]);
pf("Enter the element to dele");
sf("%d", &el);
i=0;
while(i<n)
{ if(a[i]==el)
  { for(j=i; j<n-1; j++)
      a[j]=a[j+1];
    i=0;
    n=n-1;
  }
else
    i++;
}
for(i=0; i<n; i++)
  pf("%d", a[i]);
getch();
}
```

OR

```
void main()
{
int a[10], p[10], i, n, el, c=0, k
clrscr();
pf("Enter size");
sf("%d", &n);
pf("Enter element");
for(i=0; i<n; i++)
  sf("%d", &a[i]);
pf("Enter the element to bedel");
sf("%d", &el);
for(i=0; i<n; i++)
{ if(a[i]==el)
  { p[k]=i;
    c++; k++;
  }
}
for(i=0; i<c; i++)
{ p[i]=p[i]-i;
for(j=p[i]; j<n-1; j++)
  a[j]=a[j+1];
  n=n-1;
}
for(i=0; i<n; i++)
  pf("%d", a[i]);
getch();
}
```