Linear data structure using sequential organization

Unit II

By

Pujashree Vidap

INTRODUCTION

- An Array is a **Data Structure** with which we can perform operations on a collection of similar data type such as simple list or tables of information.
- These structures are **composite or structured** data types.
- All elements in the array are of same type i.e. int, char, float etc.

- The individual elements within the array can be accessed by the integer called ***Index.***
- eg-: int array[9]

9	8	1	-1	4	2	6	15	3
0	1	2	3	4	5	6	7	8

LB
(Lower Bound)

UB
(Upper Bound)

- Length of an Array=UB-LB+1

- The zeroth element (9) in the list can be accessed by array[0].

- An ***Index*** is also called a ***Subscript.***

- Therefore, individual elements of an Array are called ***subscripted*** variables eg. array[0], array[1] etc.

- Thus, an Array can be defined as a finite ordered collection of items of same type.

Types of Array

ARRAYS

- One Dimensional Array
  (An Array whose elements are specified by a single subscript
  Eg. A[0])
- Multi Dimensional Array
  (An Array whose elements are specified by two or more than two subscripts
  Eg. A[10][10] is a two dimensional array)

ARRAY AS AN ADT

abstract typedef<eltype,ub>ARRTYPE(ub,eltype);

Condition type(ub)==int;

abstract eltype extract(a,i)

ARRTYPE(ub,eltype) a;

int i;

precondition 0<=i<ub;

postcondition extract $==a_{i;}$

abstract store(a,i,elt)

ARRTYPE(ub,eltype) a;

int i;

precondition 0<=i<ub;

postcondition a[i]=elt;

ARRAY AS AN ADT
Array is set pairs<index,value>

Class Generalarray

{

Generalarray(int j, Rangelist list, float initvalue=defaultvalue);

// constructor produce new array of appropriate type and size (produce j dimension array of float with initialvalues)

float retreive(index i);

void store(index i ,float x)

};

One Dimensional Array-:

- 1-D Arrays are suitable for processing lists of items of identical types.

- They are very useful for problems that require the same operation to be performed on a group of data items.

Algorithm Traversal (A,N)

{

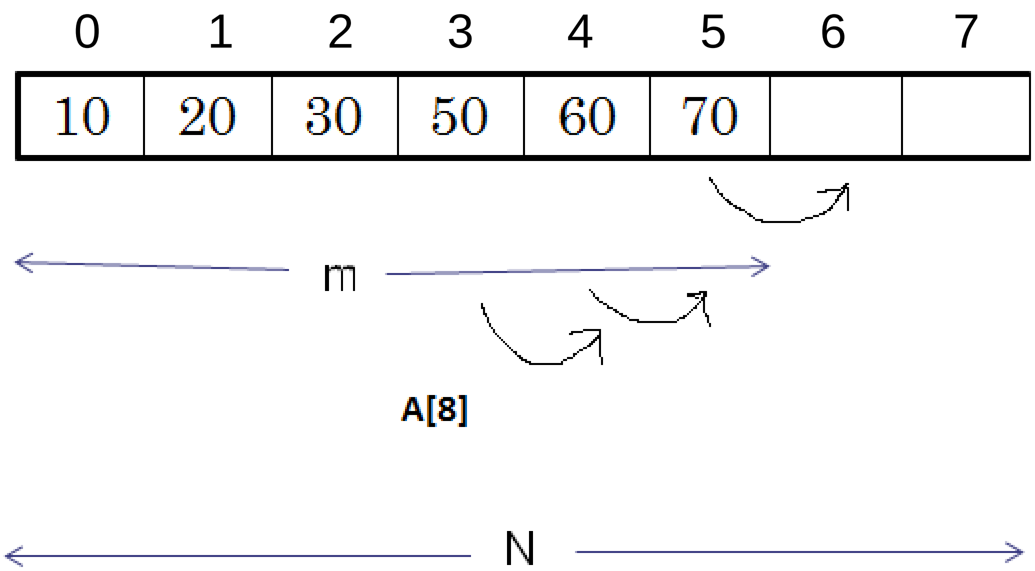
// Description:- A is an array of size N.

1. Read N
2. for(i=0 to N-1)
   - 2.1 Read A[i]
   - 2.2 write A[i]

}

Time Complexity-: ???

2. INSERTION of an element-:



Insert 40 at location 4th

After Insertion-:

0	1	2	3	4	5	6	7
10	20	30	40	50	60	70	

Inserted

m+1

N

```
5. for i=m-1 to loc-1 do
        5.1 A[i+1] = A[i]

6. A[loc-1] = val

7. For i=0 to m do
        7.1 Print A[i]

}
```

Time Complexity-:???

3. Deletion of an element from an array-:

0	1	2	3	4
10	20	30	40	50

N

A[N]

Delete an element at location 3rd

0	1	2	3	4
10	20	40	50	0

N

A[N-1]

Algorithm Deletion(A,N, loc)
{
Description-: A is an Array of size N
loc is the location of deletion.
1.Enter N
2. for (i=0 to N-1)
    2.1 Enter A[i]
3. Enter loc
4. for (i=loc to N-1)
    4.1 A[i-1]=A[i]
5. for (i=0 to N-2)
    5.1 Print A[i]
}

Time Complexity-: ????

ADDRESS CALCULATION-:

One Dimensional Array-:

0	1	2	3	4
A	B	C	D	E

List[5]

Logical View-:

Physical Address		Logical Address
100	A	0
100+1	B	1
100+2	C	2
100+3	D	3
100+4	E	4

- The array elements are stored in contiguous memory locations by sequential allocation techniques.
- The Address of ***ith*** element of the array can be obtained if we know-:
  1. The starting address i.e. the address of the first element called ***Base Address*** denoted by **B**
  2. The size of the element in the array denoted by **W**

Address of List [i] = B + (i – LB) * W

where LB is Lower Bound of the array

List[5] = List[0---4] ; LB=0

ARRAYS AS PARAMETERS

Pass by value
Pass by reference
Returning an array from function.

```
sum(a,ub);//calling
void sum(int a[ ],int size);
```

SEQUENTIAL MEMORY ORGANIZATION

- Advantages and disadvantages ??

PROBLEM

- Write a program that accepts an array of n numbers and returns the median of numbers in the array.
- Consider the linear array AAA [5:50], BBB [-5:10] and CCC [18]

a) Find the number of elements in each array.

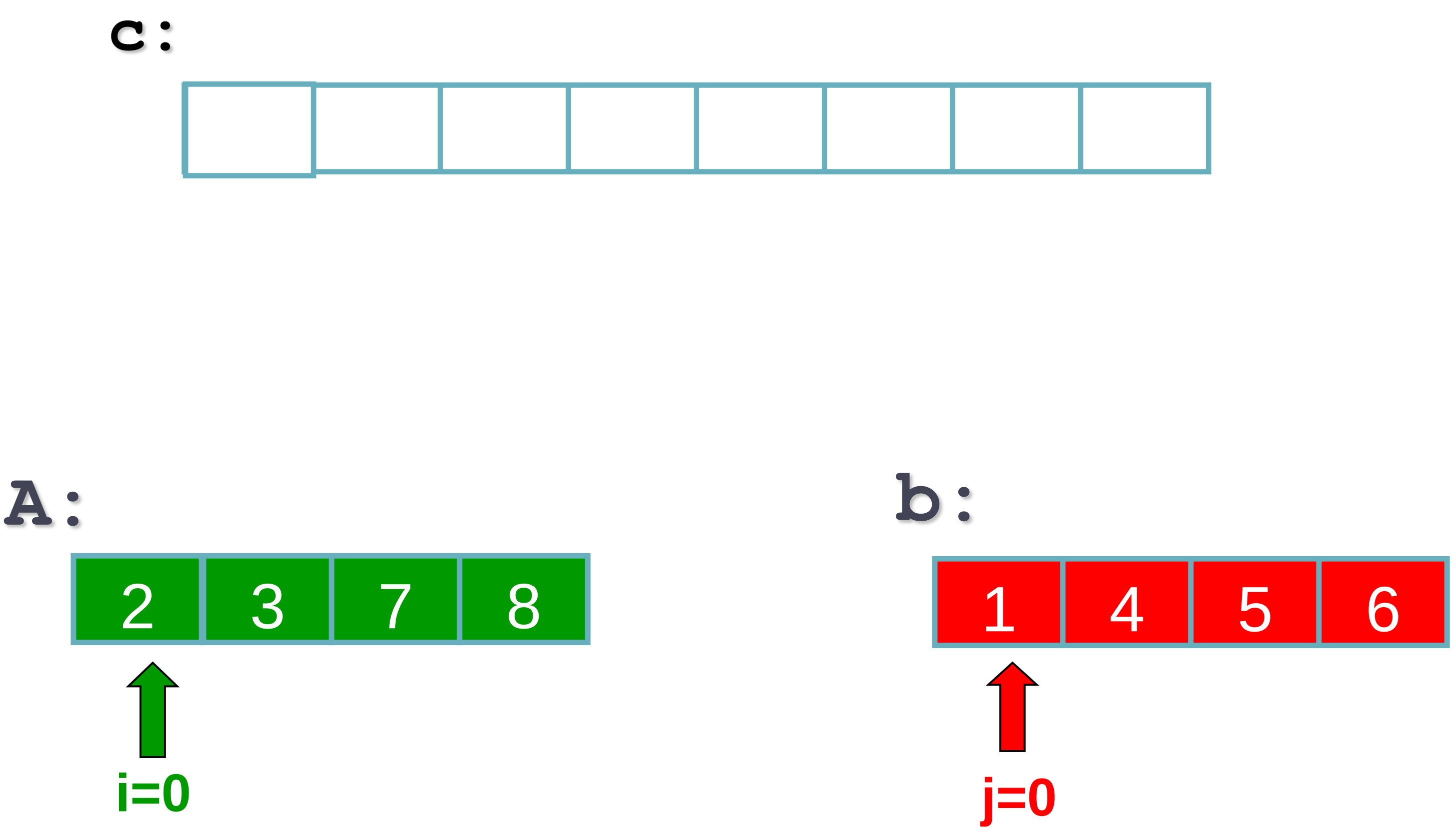
b) Suppose Base (AAA) =300 and w=4 word per memory cell for AAA. Find the address of AAA [15], AAA [35] and AAA [55].

Write a program that reads name of the students, age, roll no, Phone of 4 students and perform the following operation

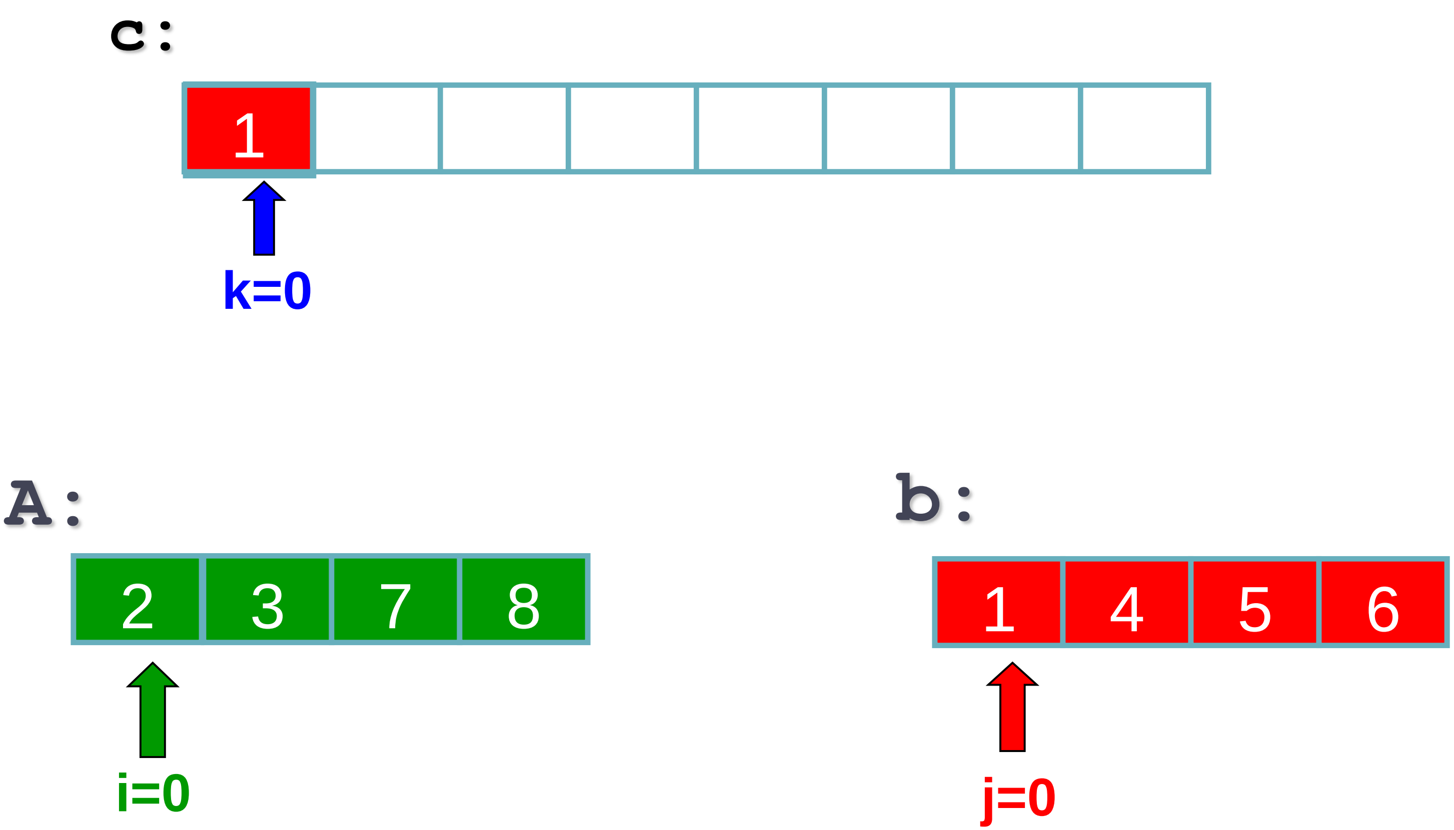
1. Search a record of particular student.
2. Insert a particular record at specific position.
3. Delete a particular record from specific position.

Write a program to merge two sorted array in ascending order.

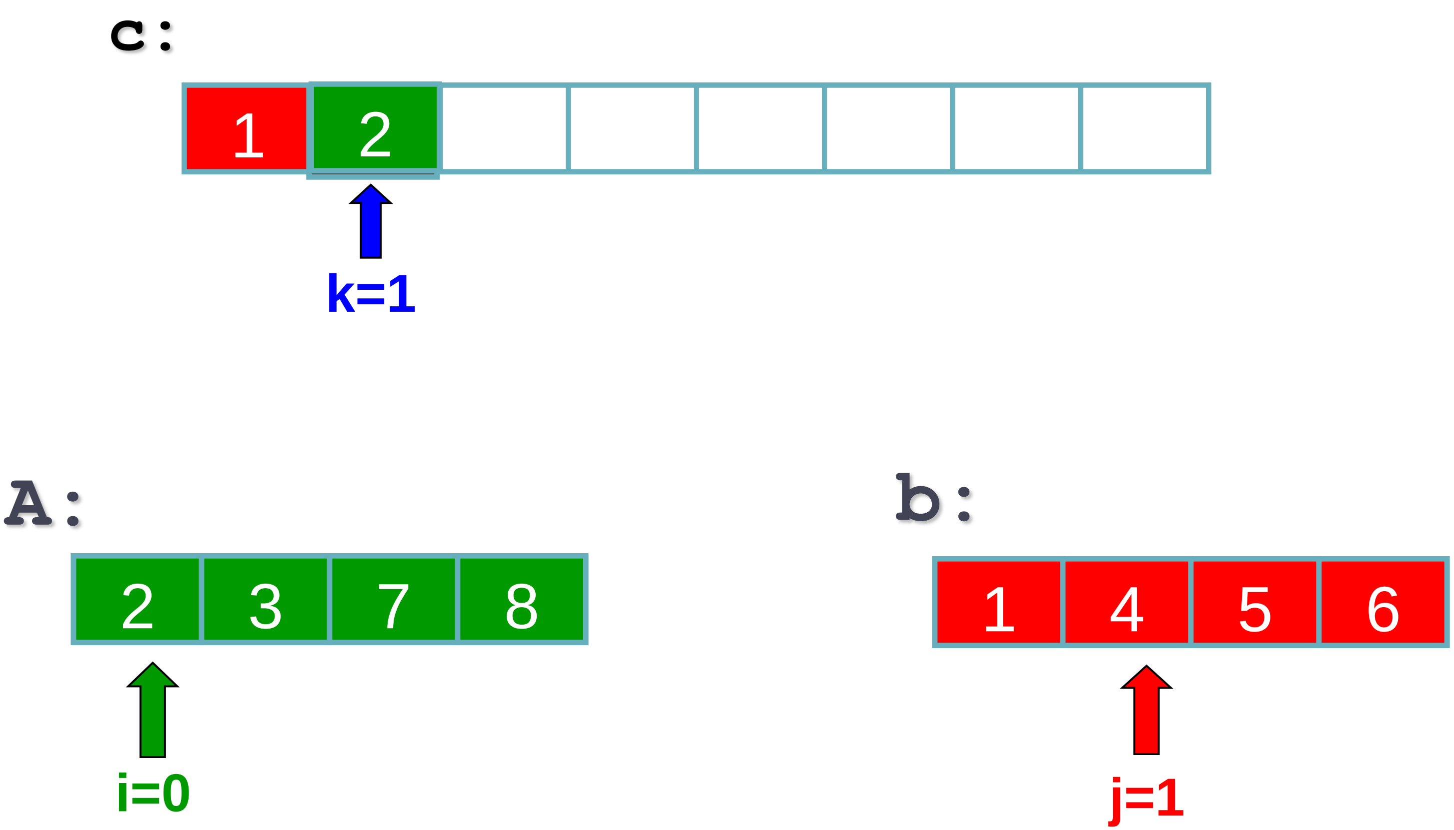
MERGE-SORT: MERGE EXAMPLE



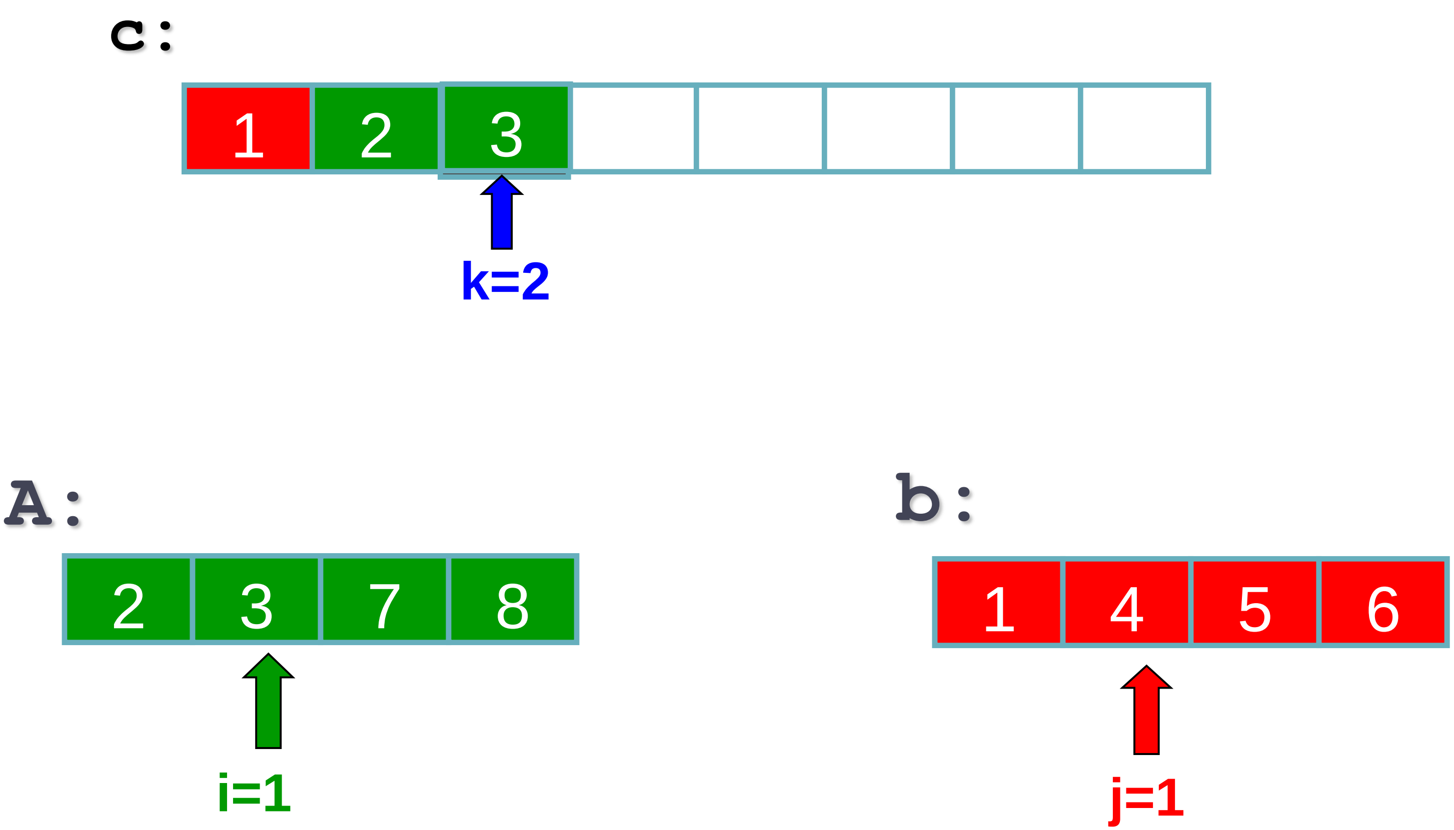
MERGE-SORT: MERGE EXAMPLE



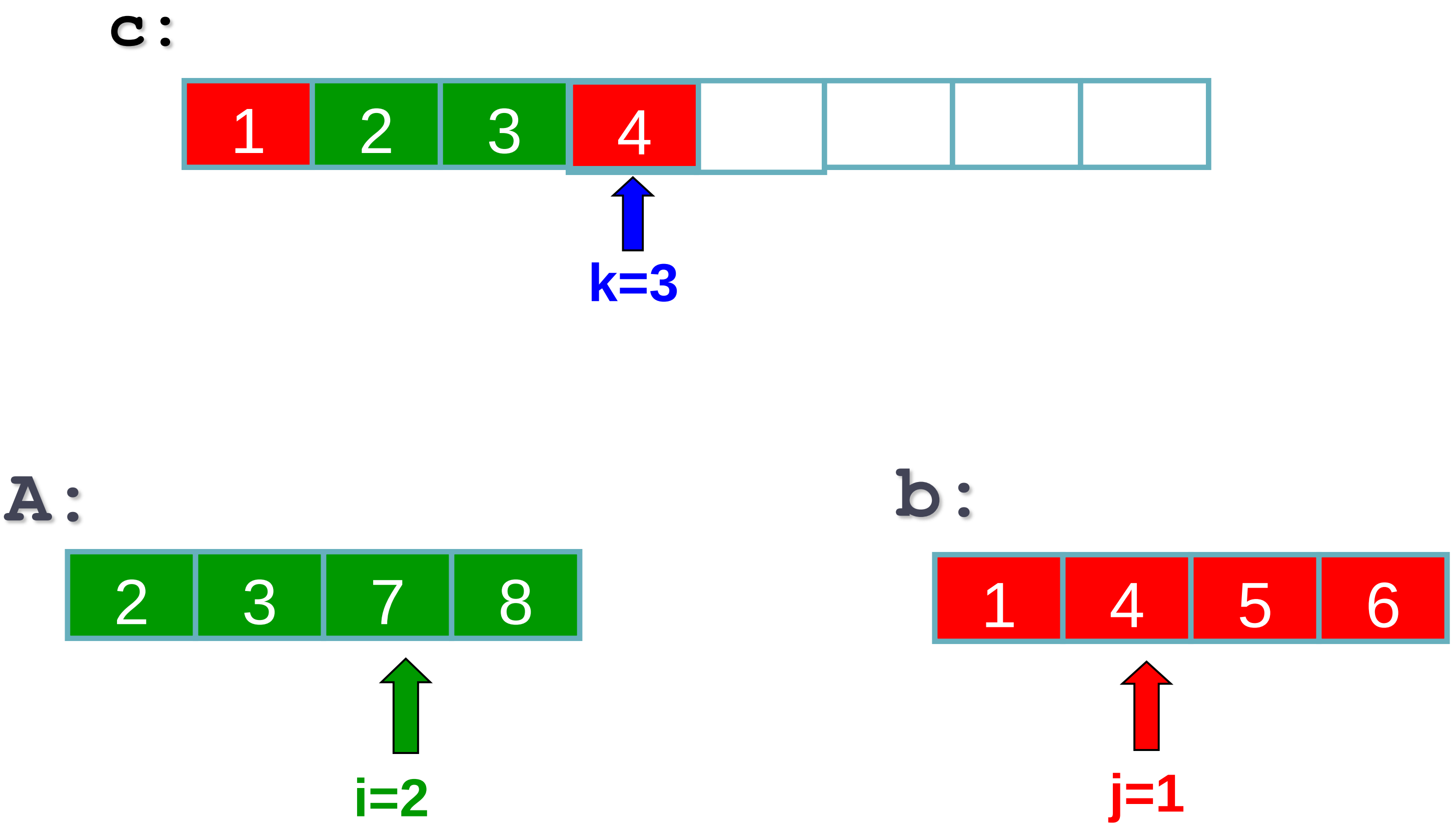
MERGE-SORT: MERGE EXAMPLE



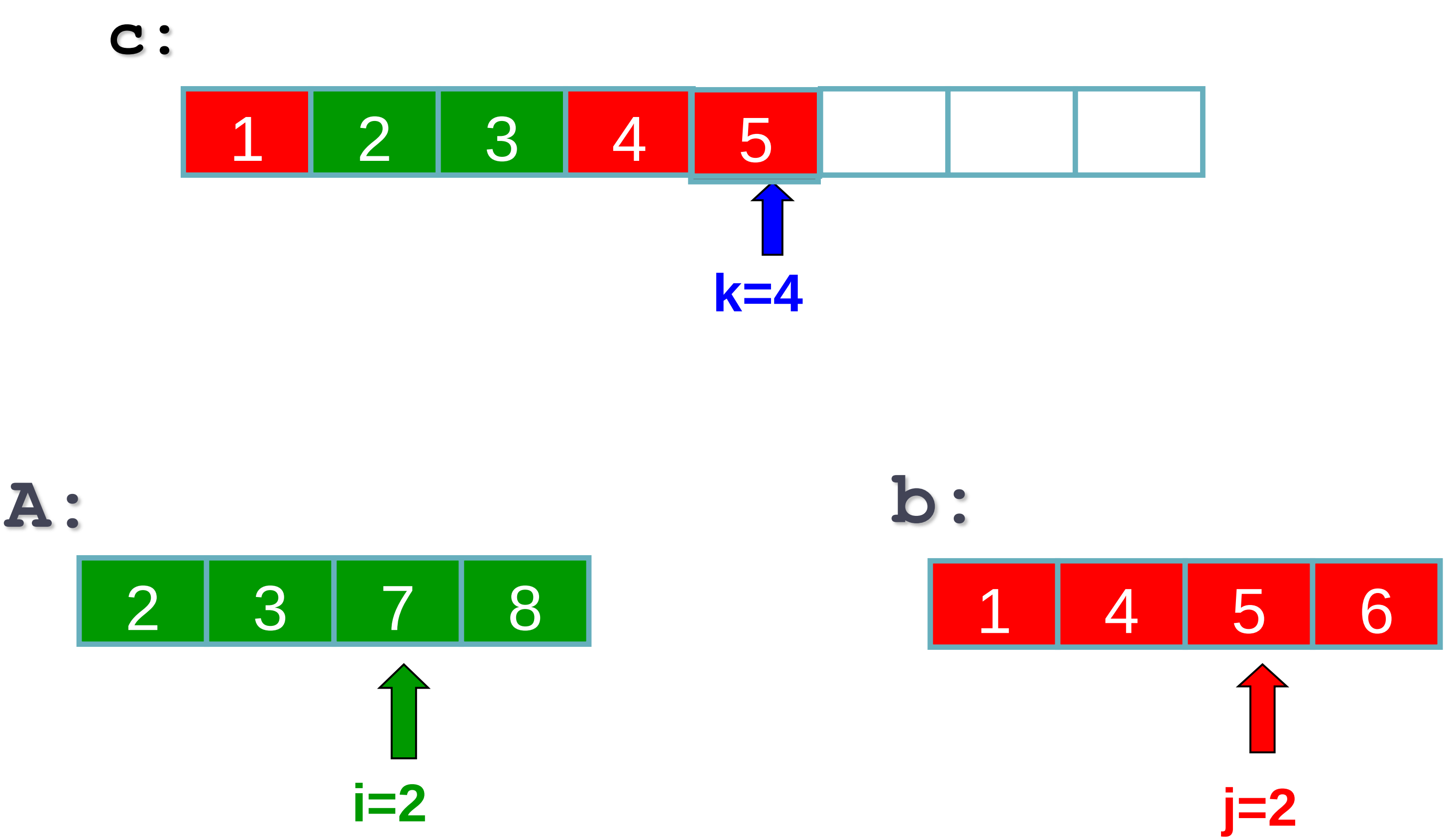
MERGE-SORT: MERGE EXAMPLE



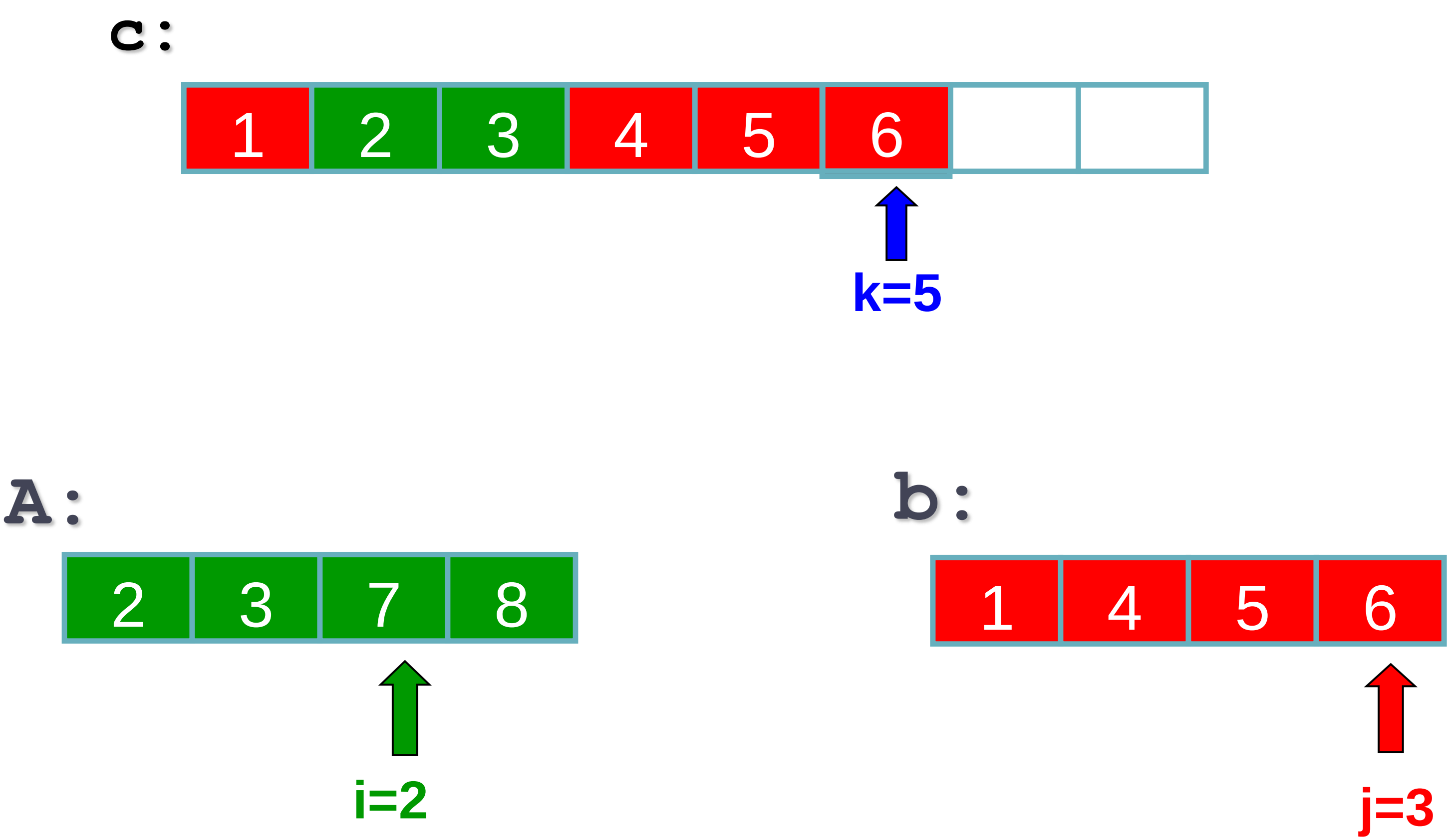
MERGE-SORT: MERGE EXAMPLE



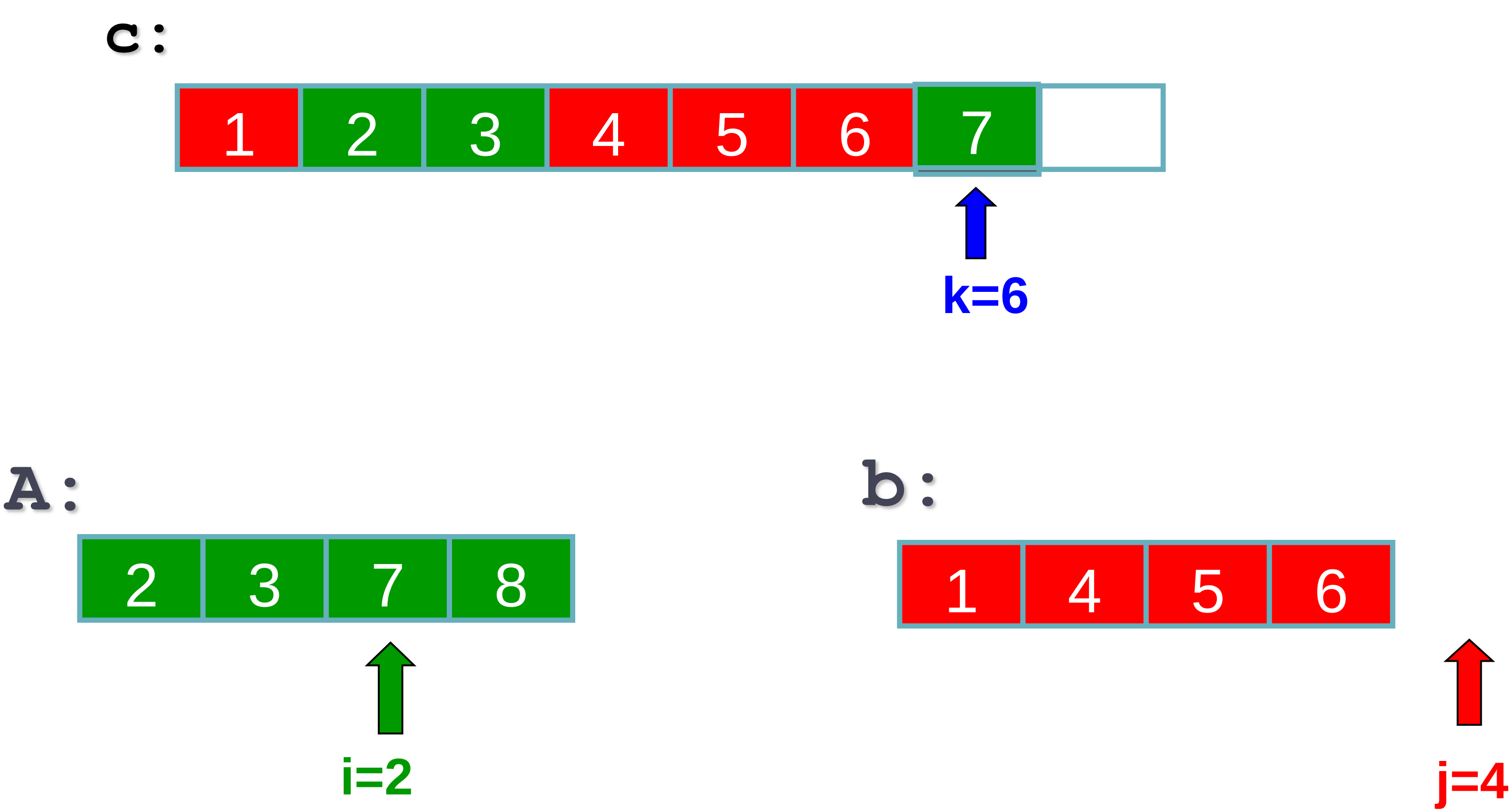
MERGE-SORT: MERGE EXAMPLE



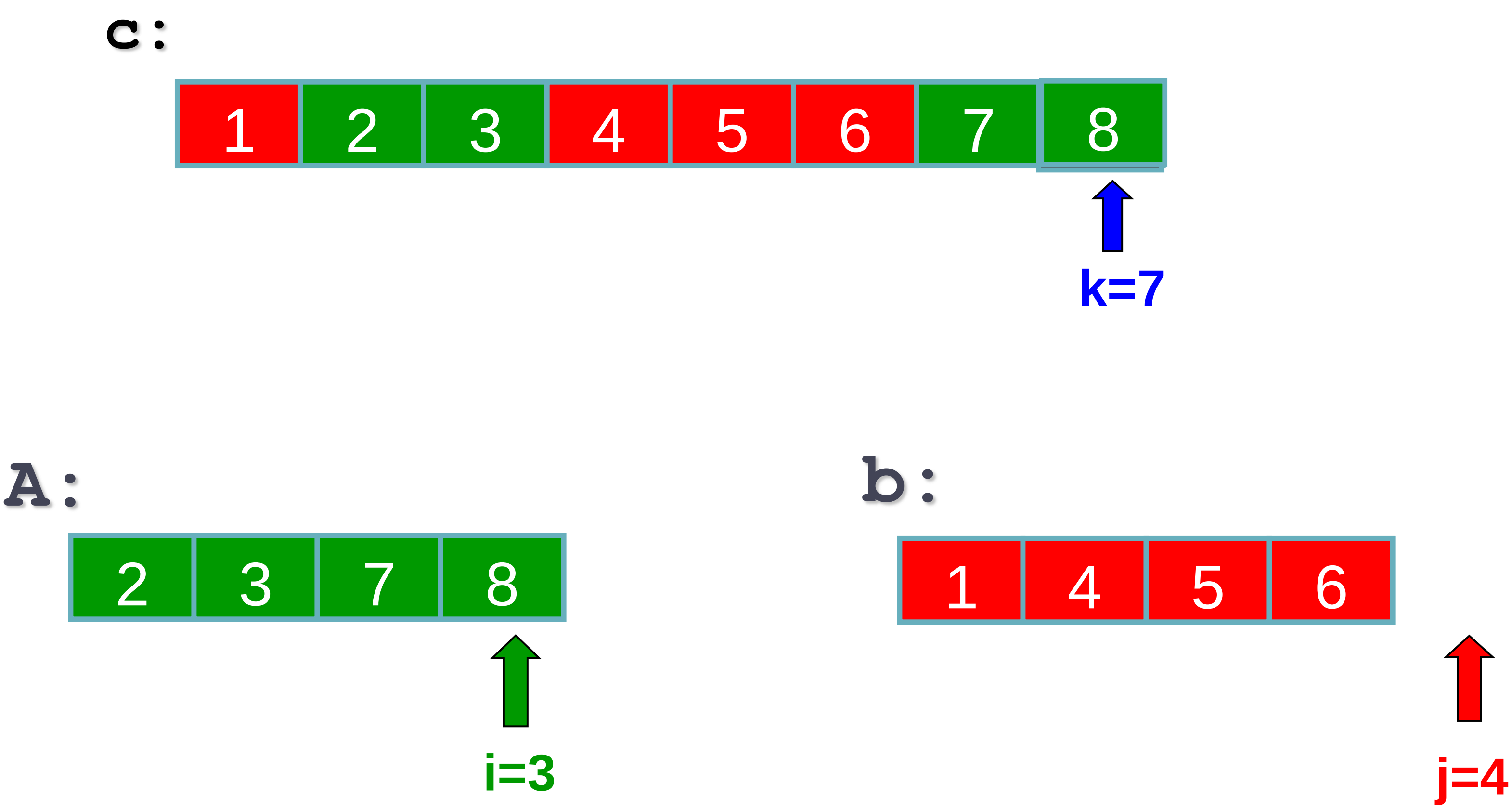
MERGE-SORT: MERGE EXAMPLE



MERGE-SORT: MERGE EXAMPLE



MERGE-SORT: MERGE EXAMPLE



MERGE-SORT: MERGE EXAMPLE

c:

1	2	3	4	5	6	7	8

k=8

A:

2	3	7	8

i=4

b:

1	4	5	6

j=4

Algorithm Merge (A ,b, c, n, m)

{

//A is an array of n elements, b, is an array of m,c is the final merged array

1. i := 0;
2. j:= 0;
3. k := 0;
4. Do
   - 4.1 If(A [i] <= b [j])
     - 4.1.1 c [k] := A [i]
     - 4.1.2 i := i + 1
     - 4.1.3 k: = k +1
   - 4.2 else
     - 4.2.1 c [k] := b[j]
     - 4.2.2 j := j +1
     - 4.2.3 k := k +1

   while (i <= n && j <= m);

```
6. While ( j < = m)
        6.1 c[k] := b [j]
        6.2 j:= j + 1
        6.3 k := k +1

7. While( i <=  n )
        7.1  c[k] := A [i]
        7.2   i := i + 1
        7.3    k := k +1


}
```

- <u>Two Dimensional Array-:</u>

A [m] [n] represents an array A where m is no.of rows and n is no.of columns.

Eg-:

A[2][3]

	0	1	2
0	4	5	6
1	7	8	9

m*n
1*2

Row Major Order-:

100	4	0,0
100+1	5	0,1
100+2	6	0,2
100+3	7	1.0
100+4	8	1,1
100+5	9	1,2

In row-major storage, a multidimensional array in linear memory is accessed such that rows are stored one after the other.

Address of A [i] [j]= B + [(i-LB_R)*n + (j-LB_C)]*W

where :

- LB_R is the Lower Bound of Rows

- LB_C is the Lower Bound of Columns

- W is the size of element

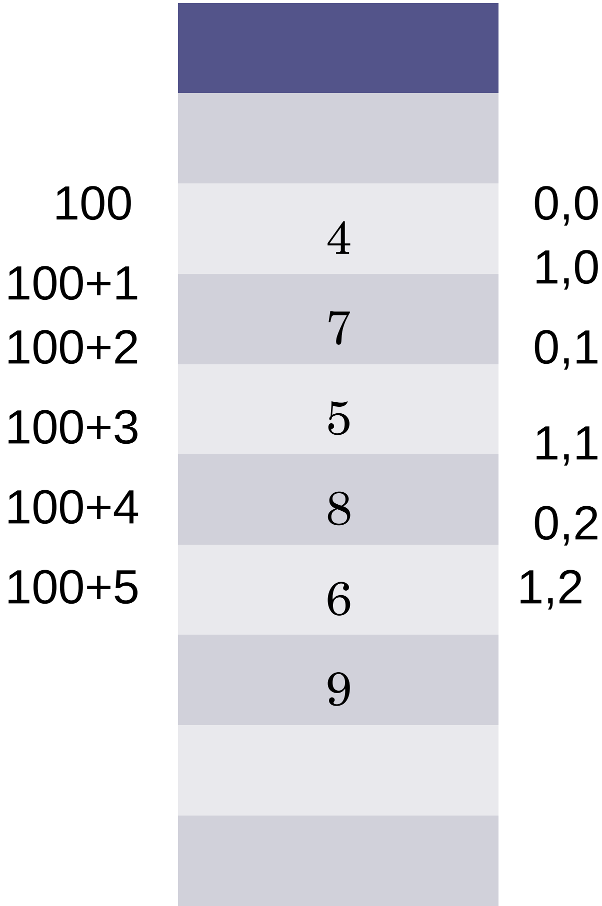
- n is the no.of columns

- Eg-: A[1][2]; m=2;n=3;LB_R=0;LB_C=0

A [1] [2] = 100 + ((1-0) * 3 + (2-0)) *1

= 105

Column Major Order-:



In column-major storage, a multidimensional array in linear memory is accessed such that columns are stored one after the other.

$$\text{Address of } A[i][j] = B + [(i-LB_R) + (j-LB_C)*m]*W$$

where

- LB_R is the Lower Bound of Rows
- LB_C is the Lower Bound of Columns
- W is the size of element
- m is the no.of rows

SOLVE

- Consider the 25*4 matrix array SCORE. Suppose base(SCORE)=200 and there are w=4 words per memory cell. Suppose the programming language stores two dimensional arrays using row major order .Find the address of SCORE[12,3].
- Also solve considering the elements are stored in column major order.

2-D ARRAY

- Declaration:

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

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

  int x[3][4] = {0, 1 ,2 ,3 ,4 , 5 , 6 , 7 , 8 , 9 , 10 , 11}

  int x[3][4] = {{0,1,2,3}, {4,5,6,7}, {8,9,10,11}};

  Second dimension must be specified

- Accessing array elements:

  cout<<a[i][j];

- Inputting array elements:

  cin>>a[i][j];

- Passing 2D arrays to functions

Calling: print(a,row,col);

Prototyping: void print(int[][],int,int);

Defination:void print(int x[][5],int r,int c);

Input in 2-d matrix

```
for(i=0;i<r;i++)
{for(j=0;j<c;j++)
{cin>>a[i][j];
}
```

RETURNING 2D ARRAY

```
int main()
{
  int x[3][3], y[3][3];
  int (*a)[3];
  int i,j;
  printf("Enter the matrix1: \n");
  for(i = 0; i < 3; i++)
    for(j = 0; j < 3; j++)
      scanf("%d",&x[i][j]);
  printf("Enter the matrix2:”);
  for(i = 0; i < 3; i++)
    for(j = 0; j < 3; j++)
      scanf("%d",&y[i][j]);
  a = Matrix_sum(x,y);
  printf("The sum of the matrix is: \n");
  for(i = 0; i < 3; i++){
  for(j = 0; j < 3; j++){
  printf("%d",a[i][j]);
  printf("\t");
  }
  printf("\n");
  }
  return 0;
}
```

```
int (*(Matrix_sum)(int matrix1[][3], int
matrix2[][3]))[3]{
int i, j;
  static int m3[3][3];
  for(i = 0; i < 3; i++){
  for(j = 0; j < 3; j++){
  m3[i][j] = matrix1[i][j] + matrix2[i][j];
  }
  }
  return m3;
  }
```

CONSTRUCTING TWO DIMENSIONAL ARRAY

```
class Test
{
int **p;
int row,col;
public:
Test(int,int);
void getdata();
void display();
};
```

```
void Test:: getdata()
{
for(int i=0;i<row;i++)
{
for(int j=0;j<col;j++)
{
cin>>p[i][j];
}
}
}
```

CONSTRUCTING TWO DIMENSIONAL ARRAY

```
void Test :: display()
{
for(int i=0;i<row;i++)
{for(int j=0;j<col;j++)
{cout<<p[i][j];}
cout<<"\n";
}}
Test ::Test(int x,int y)
{
row=x;
col=y;
p=new int *[row];
for(int i=0;i<row;i++)
{
p[i]=new int[col];
}
}
void main()
{
int m,n;
cout<<" Enter the no. of rows
   and columns ";
cin>>m>>n;
Test t(m,n);
cout<<"enter array";
t.getdata(m,n);
t.display(m,n);
}
```

OUTPUT

Enter the no. of rows and

Columns 2 2

Enter array

2

2

2

2

2   2

2   2

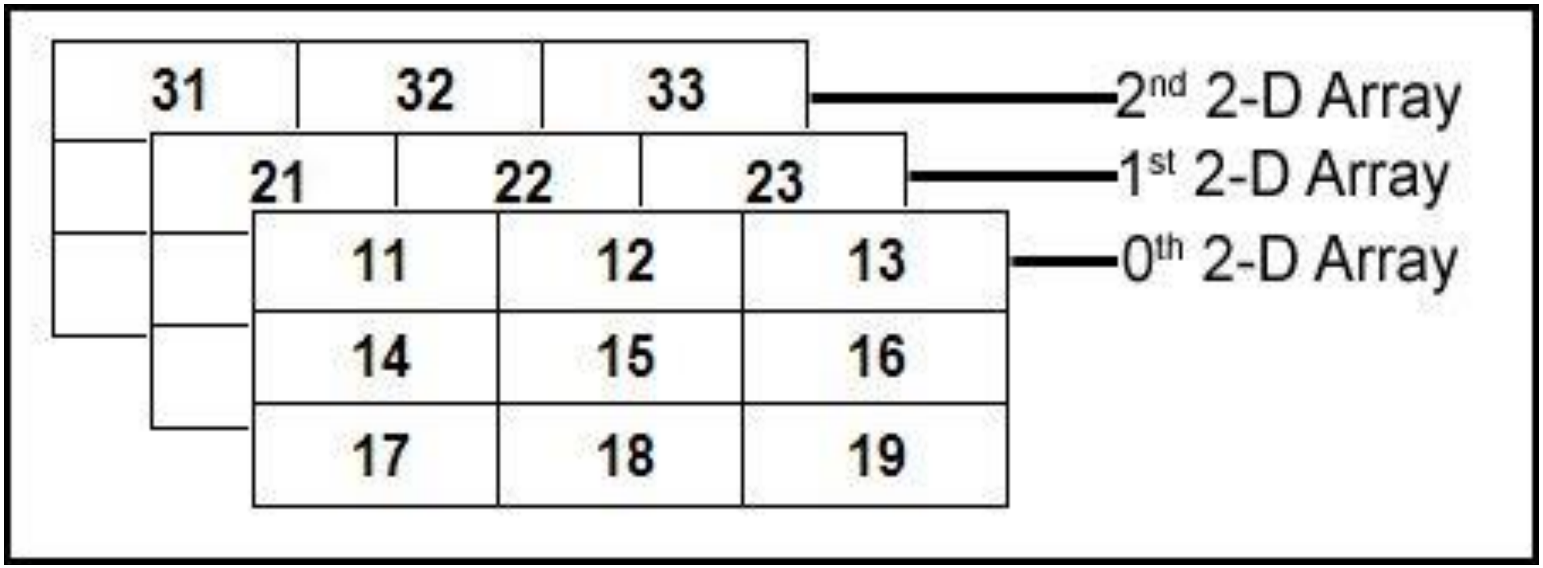
Assignment on 2D array

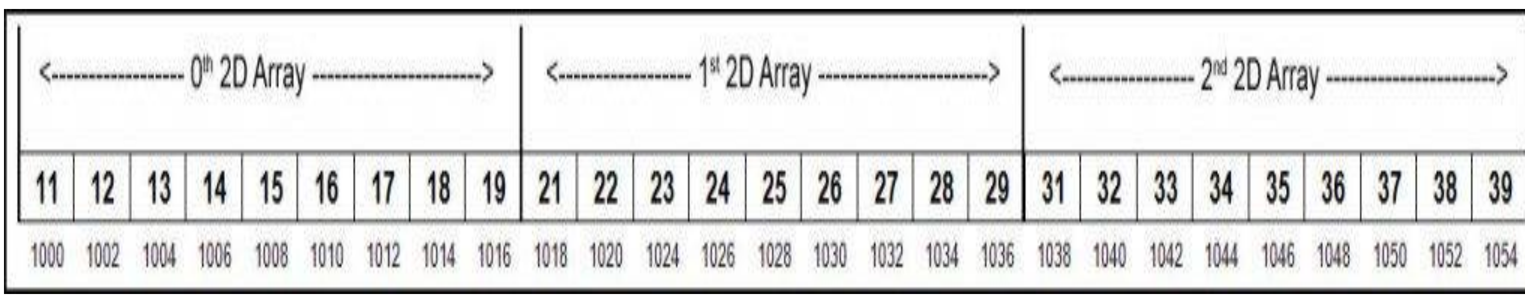
- Matrix operations like addition, subtraction, multiplication, transpose etc.
- Operations on square matrix like symeetric matrix, diagonal matrix, upper triangular, lower triangular and identity matrix etc.
- Print the matrix along with row and column sum
- Generating magic square matrix or checking given square matrix is magic square

MULTI-DIMENTIONAL ARRAY

- We can have three or more dimentions of an array.
- int a[3][3][3]
- No of elements will be:3*3*3=27

3D REPRESENTATION OF AN WITH ITS MEMORY ALLOCATION





USING MULTI-DIMENTIONAL ARRAYS

```
void main()
{
int a[2][2][2],i,j,k;
for(i=0;i<2;i++)
    for(j=0;j<2;j++)
       for(k=0;k<2;k++)
              cin>>a[i][j][k];
```

0	1
1	2

1	2
2	3

- Row major printing

```
for(i=0;i<2;i++)
    for(j=0;j<2;j++)
       for(k=0;k<2;k++)
           cout<<a[i][j][k]
     cout<<"\n"
```

Output

0(0,0,0)   1(0,0,1)

1(0,1,0) 2(0,1,1)

1(1,0,0)   2(1,0,1)

2 (1,1,0) 3(1,1,1)

- Column major

```
for(i=0;i<2;i++)
    for(j=0;j<2;j++)
       for(k=0;k<2;k++)
           cout<<a[k][j][i];
     cout<<"\n"
```

Output

0(0,0,0)   1(1,0,0)

1(0,1,0) 2(1,1,0)

1(0,0,1)   2(1,0,1)

2 (0,1,1) 3(1,1,1)

ADRESS CALCULATIONS

- Row major
- $A[i][j][k]=b+[(i-lb_i)*d2*d3+(j-lb_j)*d3+(k-lb_k)]*W$

Where b=base address

lb=lower bound

d2=no of rows

d3= no of cols

Where i is plane

j is rows, k is columns

STRING MANIPULATION USING ARRAY

- String?
- Write a program to check whether a string is palindrome or not?
- Write a program to compare two string.
- Write a program to find substring from a given string.

CONCEPT OF ORDERED LIST

- Arrays can be used to implement other data structure like ordered or linear list ex. List of Days of week, values in a deck of cards etc.
- Linear list of elements arranged in particular order.
- List of no. arranged in particular order.
- Can be represented using arrays
- Operations on ordered list : find length, read list(display), retrieve i^{th} element

✓ searching

✓ insertion

✓ deletion

ADT POLYNOMIAL

class polynomial

{ //$A(x)=a_0x^{e0}+\ldots\ldots.a_nx^n$, a set of ordered pair of <e_i,a_i> where a_i is non zero float coefficient and e_i is a non negative integer exponent.

public:

```
polynomial(); //construct the polynomial p(x)=0.
polynomial  add(polynomial poly);
//return sum of polynomials *this and  poly.
polynomial  mul(polynomial poly);
 //return multiplication of polynomials *this and  poly.
float eval(float f);
//evaluate the polynomial *this at f and return result.
}
```

SINGLE VARIABLE POLYNOMIAL

- Polynomial of the form

  $A(x)=C_{m-1}x^{m-1}+c_{m-2}x^{m-2}+\ldots\ldots\ldots c_0x^0$
- Example: $8x^3+3x+6$(array representation)

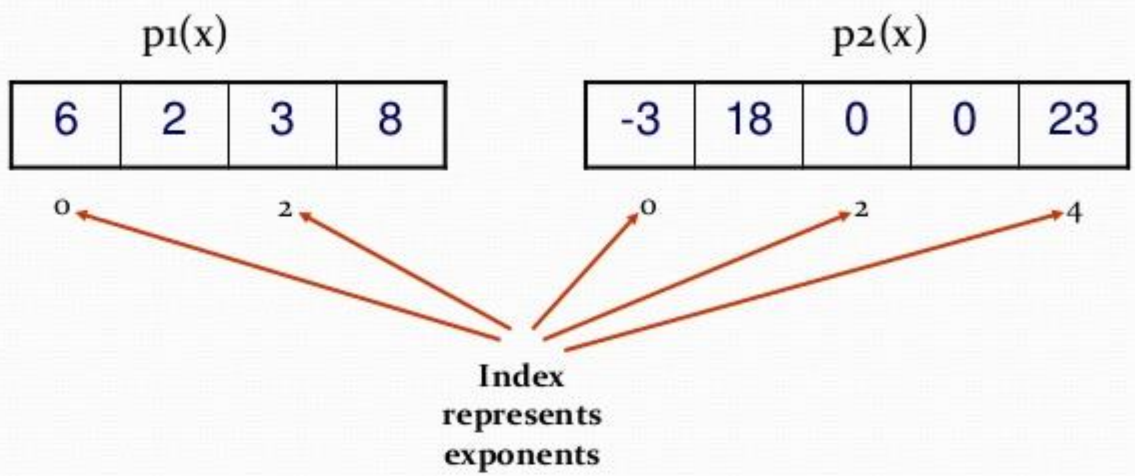
Representation 1:

- Polynomial representation as an array where indices represent the exponent and array element represent corresponding coefficients

6	3	0	8
0	1	2	3

Representation 1:

- Array Implementation:
- $p1(x) = 8x^3 + 3x^2 + 2x + 6$
- $p2(x) = 23x^4 + 18x - 3$



DISADVANTAGE

- If exponent is large ,scanning the large array will be time consuming
- In case of sparse polynomial, lot of wastage will be there.
- Array size needs to be pre-defined.

Representaion 1:

private:

int degree; //degree <=maxdegree

float coef[maxdegree+1];

// if a.degree=n

//a.coef[i]=a_{n-i} 0<=i<=n coefficients are stored in order of decreasing exponents

REPRESENTATION 2 : Using Dynamic array based on degree of polynomial

- private :

```
int degree;
float *coef;
Polynomial::polynomial(int d)
{
degree =d;
coef=new float[degree+1];
}
//here degree much less than Maxdegree so
  disadvantage of representation 1 and 2 are
  overcome
```

Representation 3 :POLYNOMIAL AS ARRAY OF STRUCTURES (Store only nonzero terms to overcome problems with representation 1 and 2

- Example: $8x^3+3x+6$

3	8	1	3	0	6
0	1	2	3	4	5

POLYNOMIAL AS ARRAY OF STRUCTURES

```
class term {
friend polynomial;
private:
float coef;
int exp;  };
class polynomial
{//initializing capacity with appropiate initial value
   and term with 0
private:
term  *termarray;//array of nonzero terms
int capacity;// size of termarray
int terms; //number of nonzero terms
};
```

ADDING TWO POLYNOMIAL: O(m+n)

```
polynomial polynomial ::add(polynomial b)
{ polynomial c;
int apos=0;bpos=0;
while((apos<terms)&&(bpos<b.terms)) {
  if(termarray[apos].exp==b.termarray[b.pos].exp) {
  float t=termarrray[apos].coef + b.termarray[b.pos].coef;
  If(t) c.newterm(t,termarray[apos].exp);
        apos++,bpos++;
   }
  else if (termarray[apos].exp<b.termarray[b.pos].exp){

    c.newterm(b.termarray[bpos].coef,b.termarray[bpos].exp);
        bpos++;
         }
```

```
else
{
 c.newterm(termarray[apos].coef ,termarray[apos].exp);
  apos++;}
}
// add remaining terms of *this
for(;apos<terms;apos++)
c.newterm(termarray[apos].coef,termarray[apos].exp);
for(;bpos<b.terms;bpos++)
c.newterm(b.termarray[bpos].coef,b.termarray[bpos].exp
return c;
}
```

Adding new term and doubling array size when necessary

```
void polynomial::newterm(float coeff,int exp)
{
 if(term==capacity){
   capacity=capacity*2;
   term*temp=new term[capacity];
   Copy(temp, termarray );
   delete[] termarray;
   termarray=temp;
}
termarray[terms].coef=coeff;
termarray[terms].exp=exp;
terms=terms+1;
}
```

Analysis:

$O(n+m)$ where n ,m is the number of terms in A,B.

POLYNOMIAL MULTIPLICATION

- $A(x)=5x^3+2x^2+1$
- $B(x)=2x^3+x$

SPARSE MATRIX

- A sparse matrix is the matrix in which max elements are 0;
- **sparse … many elements are zero**
- **dense … few elements are zero**

Example:

0 0 3 0 4

0 0 5 7 0

0 0 0 0 0

0 2 6 0 0

APPLICATION

- Airline flight matrix.
  - airports are numbered 1 through n
  - flight(i,j) = list of nonstop flights from airport i to airport j
  - n = 1000 (say)
  - n x n array of list references => 4 million bytes
  - total number of flights = 20,000 (say)
  - need at most 20,000 list references => at most 80,000 bytes

  - Diagonal matrix is sparse matrix

PROBLEM

- Not space efficient
- Retrieval time is more

So we can do representation of these matrix as:

1) Rows
2) Column
3) Value

SPARSE MATRIX REPRESENTATION

$$A = \begin{matrix} 0 & 0 & 3 & 0 & 4 \\ 0 & 0 & 5 & 7 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 2 & 6 & 0 & 0 \end{matrix}$$

Can be represented as <row,col,value>

4	5	6
0	2	3
0	4	4
1	2	5
1	3	7
3	1	2
3	2	6

```
class sparsematrix;
class Matrixterm
{
friend class sparsematrix;
private:
int row,col,value;
};
```

In class sparsematrix

```
Private:
int trows,tcols,terms,capacity;
Matrixterm *as;
```

ADT OF SPARSE MATRIX

class sparsematrix

{//a set of triples, <*row, column, value*>, where *row* and *column* are integers and form a unique combination, and *value* comes from the set *item.t*

Public:

Sparsematrix(int r,int c,int t);//constructor used to create sparse matrix

*Sparsematrix transpose();//return the sparse matrix obtained by interchanging row and column value of every triple in *this.*

*Sparsematrix add(Sparsematrix b);//if the dimentions of //*this and b are same then addition is returned.*

*Sparsematrix multiply(Sparsematrix b);//if the no of //coloumn in *this is equal to no. of rows b then //multiplication is returned d[i][j]=∑(a[i][k].b[k][j])*

};

Transpose a Matrix

(1) for each row i

take element <i, j, value> and store it in element <j, i, value> of the transpose.

difficulty: where to put <j, i, value>

(0, 0, 15) ====> (0, 0, 15)
(0, 3, 22) ====> (3, 0, 22)
(0, 5, -15) ====> (5, 0, -15)
(1, 1, 11) ====> (1, 1, 11)

Move elements down very often.

(2) For all elements in column j,
place element <i, j, value> in element <j, i, value>

(1) Represented by a two-dimensional array.

.

(2)Each element is characterized by <row, col, value>.

a[0]	6	6	8
[1]	0	0	15
[2]	0	3	22
[3]	0	5	-15
[4]	1	1	11
[5]	1	2	3
[6]	2	3	-6
[7]	4	0	91
[8]	5	2	28

(a)

transpose $\longrightarrow$

b[0]	6	6	8
[1]	0	0	15
[2]	0	4	91
[3]	1	1	11
[4]	2	1	3
[5]	2	5	28
[6]	3	0	22
[7]	3	2	-6
[8]	5	0	-15

(b)

row, column in ascending order

<u>Sparse matrix and its transpose stored as triples</u>

```
algorithm transpose (matrixterm a[],  matrixterm b[])
   /* b is set to the transpose of a */
{      n := a[0].value;
       b[0].row := a[0].col;
       b[0].col := a[0].row;
       b[0].value := n;
       if (n > 0) {
          currentb := 1;
          for i := 0 to i < a[0].col step 1
             for j := 1 to j <=  n step 1
             if (a[j].col == i) {
                  b[currentb].row = a[j].col;
                  b[currentb].col  = a[j].row;
                  b[currentb].value = a[j].value;
                  currentb++
                }
}
}
```

Scan the array “columns” times. The array has “elements” elements.

O(columns*elements)

Comparison with 2-D array representation

O(columns*elements) vs. O(columns*rows)

elements --> columns * rows when non sparse = O(columns*columns*rows)

Problem: Scan the array "columns" times.

Solution: TO REDUCE COMPLEXITY

Determine the number of elements in each column of the original matrix.

==

Determine the starting positions of each row in the transpose matrix.

a[0]	6	6	8
a[1]	0	0	15
a[2]	0	3	22
a[3]	0	5	-15
a[4]	1	1	11
a[5]	1	2	3
a[6]	2	3	-6
a[7]	4	0	91
a[8]	5	2	28

	[0]	[1]	[2]	[3]	[4]	[5]
ROW_TERMS =	2	1	2	2	0	1
STARTING_POS =	1	3	4	6	8	8

```
algorithm fast_transpose(term a[ ], term b[ ])
 {
      row_terms[MAX_COL],
      starting_pos[MAX_COL];
      num_cols = a[0].col,
      num_terms = a[0].value;
      b[0].row = num_cols; b[0].col = a[0].row;
      b[0].value = num_terms;
      if (num_terms > 0)
      for (i = 0; i < num_cols; i++)
            row_terms[i] = 0;
      for (i = 1; i  <= num_terms; i++)
            row_term [a[i].col]++
      starting_pos[0] = 1;
      for (i =1; i < num_cols; i++)
      starting_pos[i]=starting_pos[i-1] +row_terms [i-1];
```

```
            for (i=1; i <= num_terms, i++) {
                    j = starting_pos[a[i].col]++;
                    b[j].row = a[i].col;
                    b[j].col = a[i].row;
                    b[j].value = a[i].value;
            }
        }
    }
```

ANALYSIS

Compared with previous algorithm.

O(columns+elements) vs. O(columns*element)

Cost: Additional row_terms and starting_pos arrays are required.

SPARSE MATRIX ADDITION

- Algorithm add_sp_mat(sp1[][3],sp2[][3],sp3[][3])

```
{1. if( sp1[0][0] != sp2[0][0] | | sp1[0][1] != sp2[0][1] )
      1.1write("Invalid matrix size ");
      1.2 exit(0);
2. tot1 = sp1[0][2]; tot2 = sp2[0][2]; k1 = k2 = k3 = 1;
3.while ( k1 <= tot1 && k2 <= tot2)
    3.1 if ( sp1[k1][0] < sp2[k2][0] )
          { sp3[k3][0] = sp1[k1][0];
            sp3[k3][1] = sp1[k1][1];
            sp3[k3][2] = sp1[k1][2];
            k3++;k1++;  }
      3.1 else if ( sp1[k1][0] > sp2[k2][0] )
            { sp3[k3][0] = sp2[k2][0];
              sp3[k3][1] = sp2[k2][1];
              sp3[k3][2] = sp2[k2][2];
              k3++;k2++; }
```

```
3.1 else if ( sp1[k1][0] == sp2[k2][0] )
        3.1.1 if ( sp1[k1][1] < sp2[k2][1] )
                { sp3[k3][0] = sp1[k1][0];
                  sp3[k3][1] = sp1[k1][1];
                  sp3[k3][2] = sp1[k1][2];
                   k3++;k1++; }
          3.1.1else if ( sp1[k1][1] > sp2[k2][1] )
                   { sp3[k3][0] = sp2[k2][0];
                     sp3[k3][1] = sp2[k2][1];
                      sp3[k3][2] = sp2[k2][2];
                      k3++;k2++; }
            3.1.1. else
                     { sp3[k3][0] = sp2[k2][0];
                       sp3[k3][1] = sp2[k2][1];
                       sp3[k3][2] = sp1[k1][2] + sp2[k2][2];
                        k3++;k2++;k1++; }
               }
```

```
4.while ( k1 <=tot1 )
    {  sp3[k3][0] = sp1[k1][0];
        sp3[k3][1] = sp1[k1][1];
        sp3[k3][2] = sp1[k1][2];
        k3++;k1++;
      }
5 while ( k2 <=    tot2 )
   { sp3[k3][0] = sp2[k2][0];
     sp3[k3][1] = sp2[k2][1];
     sp3[k3][2] = sp2[k2][2];
      k3++;k2++; }
6.sp3[0][0] = sp1[0][0];
7.sp3[0][1] = sp1[0][1];
8.sp3[0][2] = k3-1;
}
```

COMPLEXITY

- O(m+n) where m and n are the no. of non zero terms in sparse matrix 1 and sparse matrix 2.

CASE STUDY

Use of sparse matrix in social network and maps

- Connection between the people can be shown with the help of sparse matrix
- Direct Route between the cities can be represented as a sparse matrix.
- Represent it with examples!!!

Case studies from Syllabus

- Study use of sparse matrix in Social Networks and Maps.
- Study how Economists use polynomials to model economic growth patterns.
- How medical researchers use them to describe the behavior of Covid-19 virus.

Introduction of Sparse Matrices for Machine Learning

- Large sparse matrices are common in general and especially in applied machine learning
- The interest in sparsity arises because its exploitation can lead to enormous computational savings and because many large matrix problems that occur in practice are sparse.
- sparsity = count zero elements / total elements
- Data:
  - Whether or not a user has watched a movie in a movie catalog.
  - Whether or not a user has purchased a product in a product catalog.
  - Count of the number of listens of a song in a song catalog.

- Some areas of study within machine learning must develop specialized methods to address sparsity directly as the input data is almost always sparse. Examples are:
  - Natural language processing for working with documents of text.
  - Recommender systems for working with product usage within a catalog.
  - Computer vision when working with images that contain lots of black pixels.

```python
from numpy import array
from scipy.sparse import csr_matrix
# create dense matrix
A = array([[1, 0, 0, 1, 0, 0], [0, 0, 2, 0, 0, 1], [0, 0, 0, 2, 0, 0]])
print(A)
# convert to sparse matrix (CSR method)
S = csr_matrix(A)
print(S)
# reconstruct dense matrix
B = S.todense()
print(B)
sparsity = 1.0 - count_nonzero(A) / A.size
```

```
[[1 0 0 1 0 0]
[0 0 2 0 0 1]
[0 0 0 2 0 0]]

  (0, 0) 1
  (0, 3) 1
  (1, 2) 2
  (1, 5) 1
  (2, 3) 2

[[1 0 0 1 0 0]
[0 0 2 0 0 1]
[0 0 0 2 0 0]]

0.7222222222
```

THANK YOU….
ANY QUESTIONS????