Unit-II

“Linear Data Structure using Squential Organization”

SYLLABUS

Concept of Sequential Organization, Overview of Array, Array as an Abstract Data Type, Operations on Array, Merging of two arrays, Storage Representation and their Address Calculation: Row major and Column Major, Multidimensional Arrays: Two-dimensional arrays, n-dimensional arrays. Concept of Ordered List,

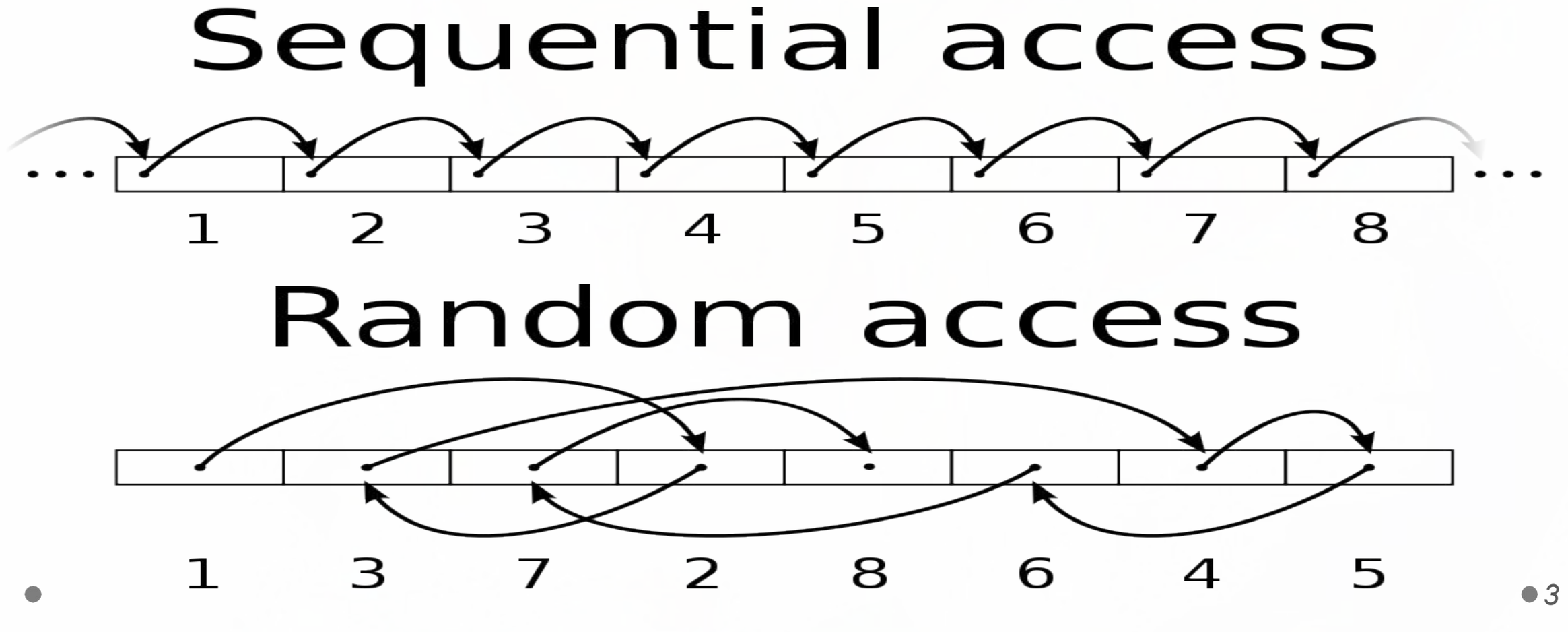
Single Variable Polynomial: Representation using arrays, Polynomial as array of structure, Polynomial addition, Polynomial multiplication.

Sparse Matrix: Sparse matrix representation using array, Sparse matrix addition, Transpose of sparse matrix- Simple and Fast Transpose, Time and Space tradeoff.

SEQUENTIAL ORGANIZATION

- **Definition :**

"In computer science, **sequential** access means that a group of elements (such as **data** in a memory array or a disk file or on magnetic tape **data** storage) is accessed in a predetermined, ordered sequence.**Sequential** access is sometimes the only way of accessing the **data**, for example if it is on a tape."



Linear DataStructure Using Sequential Organization

- **Definition :**

"The data structure where data items are organized sequentially or linearly one after another is called as **Linear Data Structure**"

A **linear data structure** traverses the **data** elements sequentially, in which only one **data** element can directly be reached. Ex: Arrays, Linked Lists.

Array

- **Definition :**

❖ "An array is a finite ordered **collection of homogeneous data elements** which provides **direct access** (or random access) to any of its elements.

❖ An array as a **data structure** is defined as a set of pairs (index,value) such that with each index a value is associated.

- **index** — indicates the location of an element in an array.
- **value** - indicates the actual value of that data element.

❖ Declaration of an array in 'C++':

- **int Array_A[20];**

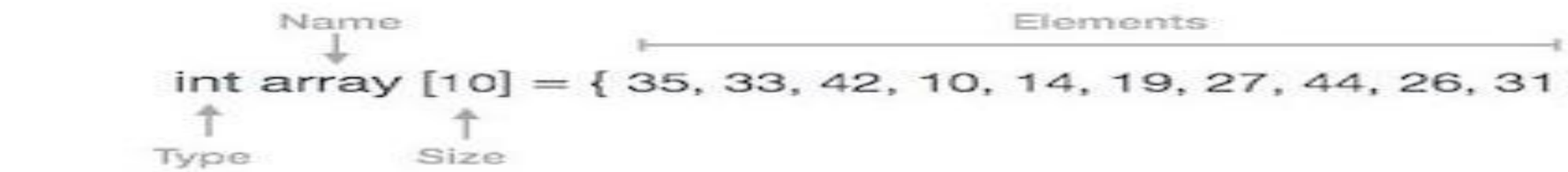
"

Array

- **Array Representation**
- Arrays can be declared in various ways in different languages. For illustration, let's take C array declaration.
  - **Index** – Each location of an element in an array has a numerical index, which is used to identify the element.

Array Representation

Arrays can be declared in various ways in different languages. For illustration, let's take C array declaration.



Arrays can be declared in various ways in different languages. For illustration, let's take C array declaration.

- Arrays can be declared in various ways in different languages. For illustration, let's take C array declaration.
- As per the above illustration, following are the important points to be considered.

Array

- **Array Representation**
- Index starts with 0.
- Array length is 10 which means it can store 10 elements.
- Each element can be accessed via its index. For example, we can fetch an element at index 6 as 9.
- Basic Operations
- Following are the basic operations supported by an array.
- **Traverse** − print all the array elements one by one.
- **Insertion** − Adds an element at the given index.
- **Deletion** − Deletes an element at the given index.
- **Search** − Searches an element using the given index or by the value.
- **Update** − Updates an element at the given index.

Characteristics of array

- An array is a finite ordered collection of homogeneous data elements.
- In array, successive elements of list are stored at a fixed distance apart.
- Array is defined as set of pairs-(index and value).
- Array allows random access to any element
- In array, insertion and deletion of element in between positions requires data movement.
- Array provides static allocation, which means space allocation done once during compile time, can not be changed run time.

Advantage of Array Data Structure

- Arrays permit efficient random access in constant time o(1).
- Arrays are most appropriate for storing a fixed amount of data and also for high frequency of data retrievals as data can be accessed directly.
- Wherever there is a direct mapping between the elements and there positions, arrays are the most suitable data structures.
- Ordered lists such as polynomials are most efficiently handled using arrays.
- Arrays are useful to form the basis for several more complex data structures, such as heaps, and hash tables and can be used to represent strings, stacks and queues.

Disadvantage of Array Data Structure

- *Arrays provide static memory management. Hence during execution the size can neither be grown nor shrunk.*
- *Array is inefficient when often data is to inserted or deleted as inserting and deleting an element in array needs a lot of data movement.*
- *Hence array is inefficient for the applications, which very often need insert and delete operations in between.*

Applications of Arrays

- *Although useful in their own right, arrays also form the basis for several more complex data structures, such as heaps, hash tables and can be used to represent strings, stacks and queues.*
- *All these applications benefit from the compactness and direct access benefits of arrays.*
- *Two-dimensional data when represented as Matrix and matrix operations.*

Memory Representation and Calculation

A. Memory Representation

— Memory arrangement after declaring one dimensional array shown in Fig. 2.3.1.

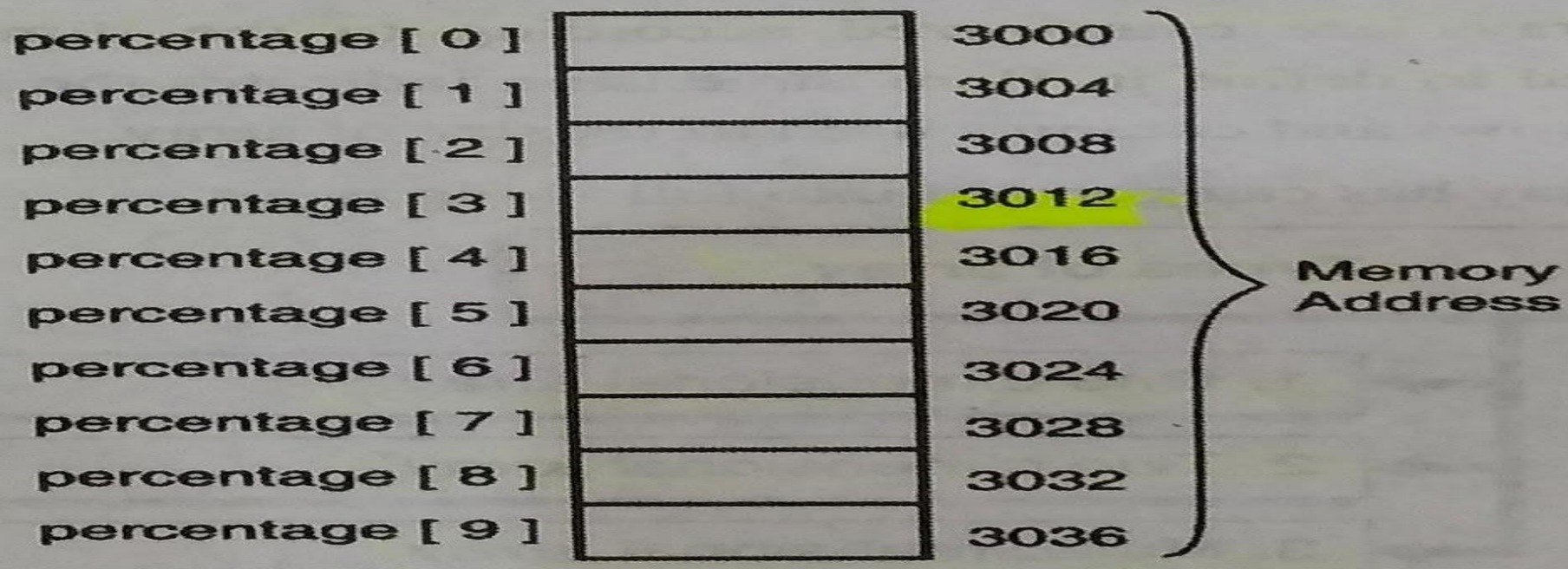


Fig. 2.3.1

— Initialize an array to store percentage of 10 students

```
percentage[0] = 99.07;
percentage[1] = 79.47;
percentage[2] = 60.60;
percentage[3] = 54.04;
percentage[4] = 80.60;
percentage[5] = 91.70;
percentage[6] = 90.30;
percentage[7] = 85.27;
percentage[8] = 99.70;
percentage[9] = 94.50;
```

ADDRESS CALCULATION

- The address of the i^{th} element is calculated bythe following formula

(Base address) + (offset of the i^{th} element from base address)

Here, base address is the address of the first element

☞ **Address calculation of 1D array**

Address of arr[i] = base address + i * element_size

Address of percentages[3] = 3000 + 3 * sizeif(float)

= 3000 + 3 * 4 = 3012

ADT for an array

- Arrays are stored in consecutive set of memory locations.

- Array can be thought of as set of index and values.
- For each index which is defined there is a value associated with that index.
- There are two operations permitted on array data structure .retrieve and store

ADT for an array

- CREATE()-produces empty array.

- RETRIVE(array,index)->value

  Takes as input array and index and either returns appropriate value or an error.

- STORE(array,index,value)-array used to enter new index value pairs.

Introduction to arrays

Representation and analysis
Type variable_name[size]

Operations with arrays:

Copy
Insert
Delete
Search
Sort
Merging of sorting arrays.

ARRAY AS AN ADT

- ADT is written with the help of instances and operations
- Formally ADT is a collection of domain, operations, and

  axioms (or rules)
- For defining an array as an ADT, we have to define its very

  basic operations or functions that can be performed onit
- The basic operations of arrays are creation of an array, storing an element, accessing an element, and traversing the array

List of operation on Array :-

- **1.** Inserting an element into an array
- 2. deleting element from array
- 3. searching an element from array
- 4. sorting the array element

Copy operations on array:

```
#include<stdio.h>
#include<conio.h>
void main()
{
   int a1[100],a2[100],i,n;
   printf("Enter size of array:");
   scanf("%d",&n);
   printf("Enter array elements: ");
   for(i=0;i<n;i++)
   {
   scanf ("\n%d",&a1[i]);
   }
   printf("First array elements: \n ");
   for(i=0; i<n; i++)
   {
   printf("\n%d",a1[i]);
   }
   printf("\n Coping array elements: \n");
   for(i=0;i<n;i++)
   {
     a2[i]=a1[i];
   }
   printf("Second array elements: ");
   for(i=0; i<n; i++)
   {
   printf("\n%d",a2[i]);
   }
   getch();
}
```

Output:

```
Enter size of array:5
Enter array elements: 1
2
6
8
9
First array elements:

1
2
6
8
9
 Coping array elements:
Second array elements:
1
2
6
8
9
```

Insertion operation on array:

ARRAY IN C PROGRAMMING

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

ARRAY IN C PROGRAMMING

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

Size(10) 9 elements

Val = 50

Index = 4

λ = 8

ARRAY IN C PROGRAMMING

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

a

Size of (a) 9 elements

val = 50

index = 4

i = 8

Ques: Program to insert a given value at a given index in an array.

0	1	2	3	4	5	6	7	8	9

```
#include<stdio.h>
int main()
{
int a[10],i,j,pos,val;
for(i=0;i<9;i++)
{
printf("\nEnter Number:");
scanf("%d",&a[i]);
}
printf("\nEnter Index to Insert:");
scanf("%d",&pos);
printf("\nEnter Value to Insert:");
scanf("%d",&val);
for(i=8;i>=pos;i--)
a[i+1]=a[i];
a[pos]=val;
printf("\nArray after Insertion":);
for(i=0;i<10;i++)
printf("\n%d",a[i]);
return 0;
}
```

Ques: Program to insert a given value at a given index in an array.

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

```
#include<stdio.h>
int main()
{
int a[10],i,j,pos,val;
for(i=0;i<9;i++)
{
printf("\nEnter Number:");
scanf("%d",&a[i]);
}
printf("\nEnter Index to Insert:");
scanf("%d",&pos);
printf("\nEnter Value to Insert:");
scanf("%d",&val);
for(i=8;i>=pos;i--)
a[i+1]=a[i];
a[pos]=val;
printf("\nArray after Insertion":);
for(i=0;i<10;i++)
printf("\n%d",a[i]);
return 0;
}
```

input

index = 5

Pos = 5

Val = 30

```
#include<stdio.h>
#include<conio.h>
int main()
{
int a[10],i,j,pos,val;
for(i=0;i<9;i++)
{
printf("\nEnter Number:");
scanf("%d",&a[i]);
}
printf("\nEnter Index to Insert:");
scanf("%d",&pos);
printf("\nEnter Value to Insert:");
scanf("%d",&val);
for(i=8;i>=pos;i--)
a[i+1]=a[i];
a[pos] =val;
printf("\nArray after Insertion:");
for(i=0;i<10;i++)
printf("\n%d",a[i]);
return 0;
}
```

Output:

```
C:\Users\Dell\OneDrive\Desktop\array_insertion.exe

Enter Number:1

Enter Number:2

Enter Number:3

Enter Number:4

Enter Number:5

Enter Number:6

Enter Number:8

Enter Number:9

Enter Number:10

Enter Number:10

Enter Index to Insert:6

Enter Value to Insert:7

Array after Insertion:
1
2
3
4
5
6
7
8
9
10
Process returned 0 (0x0)   execution time : 37.890 s
Press any key to continue.
```

To Remove A Given Number From The Array.

ARRAY IN C PROGRAMMING

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

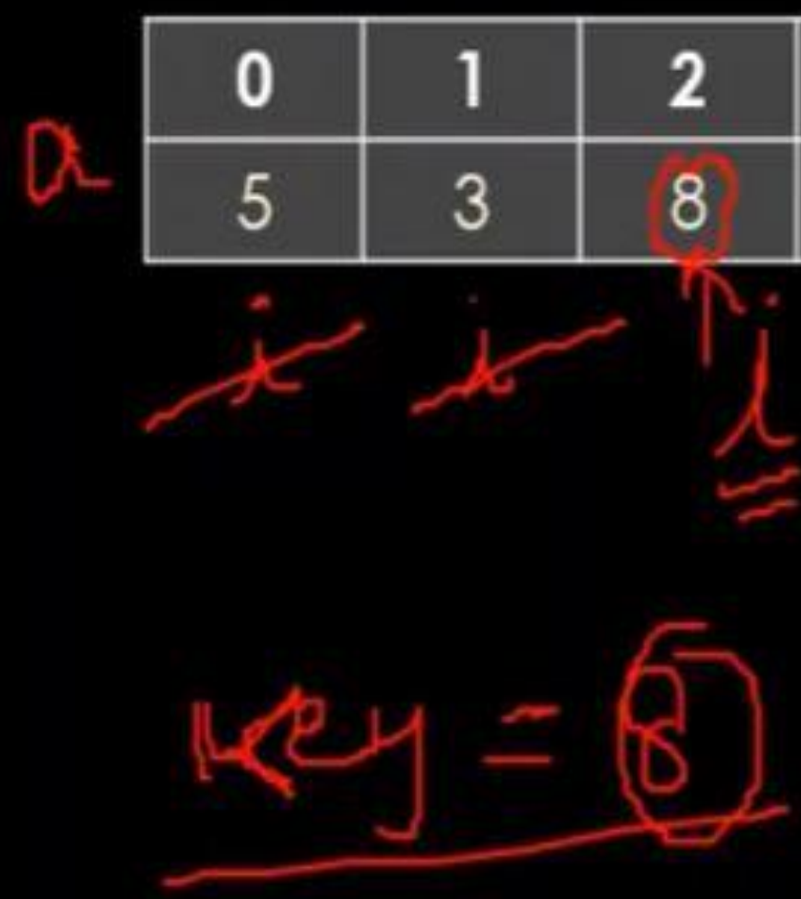
ARRAY IN C PROGRAMMING

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

key = 8

ARRAY IN C PROGRAMMING

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



ARRAY IN C PROGRAMMING

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

Ques: Program to remove a given number from the array.

and so on.

0	1	2	3	4	5	6	7	8	9

```
#include<stdio.h>
int main()
{
int a[10],i,j,t,index,key;
for(i=0;i<10;i++)
{
printf("\nEnter Number:");
scanf("%d",&a[i]);
}
printf("\nEnter Element to Remove from the list:");
scanf("%d",&key);
index=9;
for(i=0;i<=index;i++)
{
if(a[i]==key)
{
for(j=i+1;j<=index;j++)
a[j-1]=a[j];
i--;
index--;
}
}
printf("\nArray after removing %d is:",key);
for(i=0;i<index;i++)
printf("\n%d",a[i]);
return 0;
}
```

ARRAY IN C PROGRAMMING

Ques: Program to remove a given number from the array.

and so on.

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

a

```
#include<stdio.h>
int main()
{
int a[10],i,j,t,index,key;
for(i=0;i<10;i++)
{
printf("\nEnter Number:");
scanf("%d",&a[i]);
}
printf("\nEnter Element to Remove from the list:");
scanf("%d",&key);
index=9;
for(i=0;i<=index;i++)
{
if(a[i]==key)
{
for(j=i+1;j<=index;j++)
a[j-1]=a[j];
i--;
index--;
}
}
printf("\nArray after removing %d is:",key);
for(i=0;i<index;i++)
printf("\n%d",a[i]);
return 0;
}
```

Ques: Program to remove a given number from the array.

and so on.

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

```
#include<stdio.h>
int main()
{
int a[10],i,j,t,index,key;
for(i=0;i<10;i++)
{
printf("\nEnter Number:");
scanf("%d",&a[i]);
}
printf("\nEnter Element to Remove from the list:");
scanf("%d",&key);
index=9;
for(i=0;i<=index;i++)
{
if(a[i]==key)
{
for(j=i+1;j<=index;j++)
a[j-1]=a[j];
i--;
index--;
}
}
printf("\nArray after removing %d is:",key);
for(i=0;i<index;i++)
printf("\n%d",a[i]);
return 0;
}
```

Ques: Program to remove a given number from the array.

and so on.

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

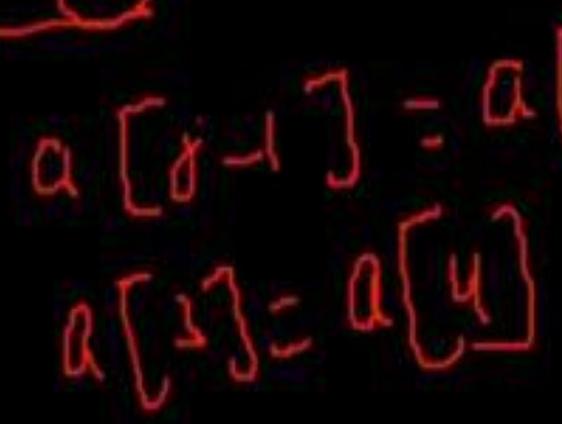
```
#include<stdio.h>
int main()
{
int a[10],i,j,t,index,key;
for(i=0;i<10;i++)
{
printf("\nEnter Number:");
scanf("%d",&a[i]);
}
printf("\nEnter Element to Remove from the list:");
scanf("%d",&key);
index=9;
for(i=0;i<=index;i++)
{
if(a[i]==key)
{
for(j=i+1;j<=index;j++)
a[j-1]=a[j];
i--;
index--;
}
}
printf("\nArray after removing %d is:",key);
for(i=0;i<index;i++)
printf("\n%d",a[i]);
return 0;
}
```

ARRAY IN C PROGRAMMING

Ques: Program to remove a given number from the array.

and so on.

```
#include<stdio.h>
int main()
{
int a[10],i,j,t,index,key;
for(i=0;i<10;i++)
{
printf("\nEnter Number:");
scanf("%d",&a[i]);
}
printf("\nEnter Element to Remove from the list:");
scanf("%d",&key);
index=9;
for(i=0;i<=index;i++)
{
if(a[i]==key)
{
for(j=i+1;j<=index;j++)
a[j-1]=a[j];
i--;
index--;
}
}
printf("\nArray after removing %d is:",key);
for(i=0;i<index;i++)
printf("\n%d",a[i]);
return 0;
}
```



Ques: Program to remove a given number from the array.

```
#include<stdio.h>
int main()
{
int a[10],i,j,index,key;
for(i=0;i<10;i++)
{
printf("\nEnter Number:");
scanf("%d",&a[i]);
}
printf("\nEnter Element to Remove from the list:");
scanf("%d",&key);
index=9;
for(i=0;i<=index;i++)
{
if(a[i]==key)
{
for(j=i+1;j<=index;j++)
a[j-1]=a[j];
i--;
index--;
}
}
printf("\nArray after removing %d is:",key);
for(i=0;i<=index;i++)
printf("\n%d",a[i]);
return 0;
}
```

Output:

C:\Users\Dell\OneDrive\Desktop\remove.exe

```
Enter Number:1

Enter Number:2

Enter Number:3

Enter Number:4

Enter Number:5

Enter Number:6

Enter Number:7

Enter Number:8

Enter Number:9

Enter Number:10

Enter Element to Remove from the list:5

Array after removing 5 is:
1
2
3
4
6
7
8
9
10
Process returned 0 (0x0)   execution time : 25.020 s
Press any key to continue.
```

Reverse operation on array

```
#include<stdio.h>
int main()
{
int a[10],i;
for(i=0;i<10;i++)
{
printf("\nEnter Number:");
scanf("%d",&a[i]);
}
for(i=9;i>=0;i--)
printf("\n%d",a[i]);
return 0;
}
```

output

```
Enter Number:2

Enter Number:3

Enter Number:5

Enter Number:6

Enter Number:4

Enter Number:7

Enter Number:8

Enter Number:9

9
8
7
4
6
5
3
2
2
1
Process returned 0 (0x0)   execution time : 12.209 s
Press any key to continue.
```

CODE::BLOCKS

The steps for installing Code::Blocks:

1. Download the setup from the following link: https://sourceforge.net/projects/codeblocks/
2. After downloading when you will open CodeBlocks, it will show some error reporting “no compiler was found”.
3. Now, you have to download one more thing from the following link: https://sourceforge.net/projects/mingw/
4. After downloading a list of packages will be shown, here you have to click on every package and choose “Mark for Installation”.
5. After selecting all the packages, go to the installation menu and then click on “Apply Changes”.
6. A dialog box will appear now, here click on “Apply”. Your installation will start and then you have to wait till it completes. Once it is complete, Code::Blocks is ready to be used

Happy Coding

Two-dimensional Arrays in C

- A two dimensional array is multidimensional array and it is the arrangement of elements in rows and columns
-
- It has two indices, one for row and other for column. First index represents the number of rows, second index represents the number of columns

- **arrayName [x][y];**

	Column 0	Column 1	Column 2	Column 3
Row 0	a[0][0]	a[0][1]	a[0][2]	a[0][3]
Row 1	a[1][0]	a[1][1]	a[1][2]	a[1][3]
Row 2	a[2][0]	a[2][1]	a[2][2]	a[2][3]

2D ARRAY IN C PROGRAMMING

Ques: Accept & Display a 3*3 Matrix.

```
#include<stdio.h>
int main()
{
int a[3][3],i,j;
for(i=0;i<3;i++)
for(j=0;j<3;j++)
{
printf("\nEnter Value for 2D Array:");
scanf("%d",&a[i][j]);
}
printf("\n2D Array Elements are:");
for(i=0;i<3;i++)
{
printf("\n");
for(j=0;j<3;j++)
printf("%d\t",a[i][j]);
}
return(0);
}
```

	0	1	2
0	0,0	0,1	0,2
1	1,0	1,1	1,2
2	2,0	2,1	2,2

2D ARRAY IN C PROGRAMMING

Ques: Accept & Display a 3*3 Matrix.

```
#include<stdio.h>
int main()
{
int a[3][3],i,j;
for(i=0;i<3;i++)
for(j=0;j<3;j++)
{
printf("\nEnter Value for 2D Array:");
scanf("%d",&a[i][j]);
}
printf("\n2D Array Elements are:");
for(i=0;i<3;i++)
{ printf("\n");
for(j=0;j<3;j++)
printf("%d\t",a[i][j]);
}
return(0);
}
```

"C:\Users\Dell\OneDrive\Desktop\2D Matrix.exe"

```
Enter Value for 2D Array:1

Enter Value for 2D Array:2

Enter Value for 2D Array:3

Enter Value for 2D Array:4

Enter Value for 2D Array:5

Enter Value for 2D Array:6

Enter Value for 2D Array:7

Enter Value for 2D Array:8

Enter Value for 2D Array:9

2D Array Elements are:
1       2       3
4       5       6
7       8       9
Process returned 0 (0x0)   execution time : 4.983 s
Press any key to continue.
```

Definition and syntax

Multidimensional arrays can be defined as an array of arrays

General form of declaring N-dimensional array is as follows:

```
data_type name_of_array[size1][size2]...[sizeN];
```

For example:

```
int a[3][4];    //Two Dimensional Array
int a[3][4][6];    //Three Dimensional Array
```

Row-major Representation

Column-major Representation

Columns →

Col1 col2 .... coln

Rows Ro

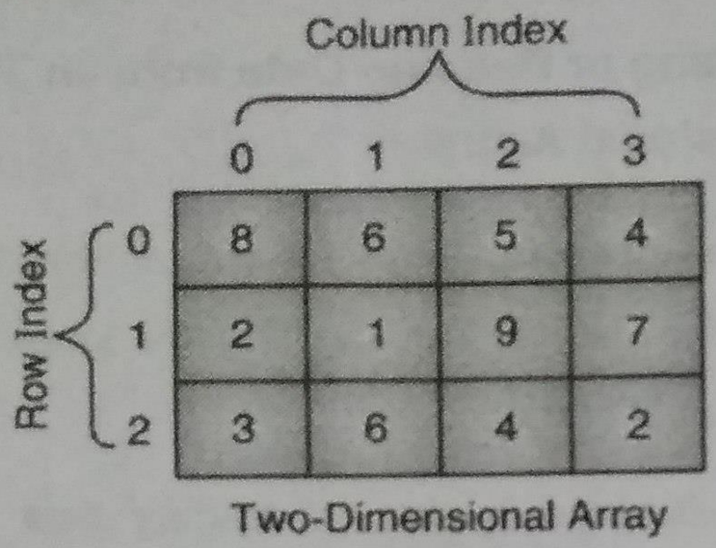
R1 ↓

$$\begin{bmatrix} A_{00} & A_{01} & \dots & A_{0n} \\ A_{10} & A_{11} & \dots & A_{1n} \\ \vdots & \vdots & & \vdots \\ A_{m0} & A_{m1} & \dots & A_{mn} \end{bmatrix}_{m*n}$$

Matrix M =

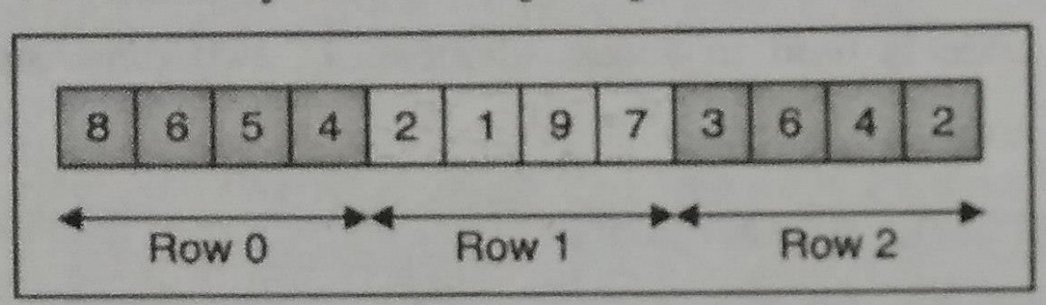
$$\begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \end{bmatrix}$$

Row-major representation

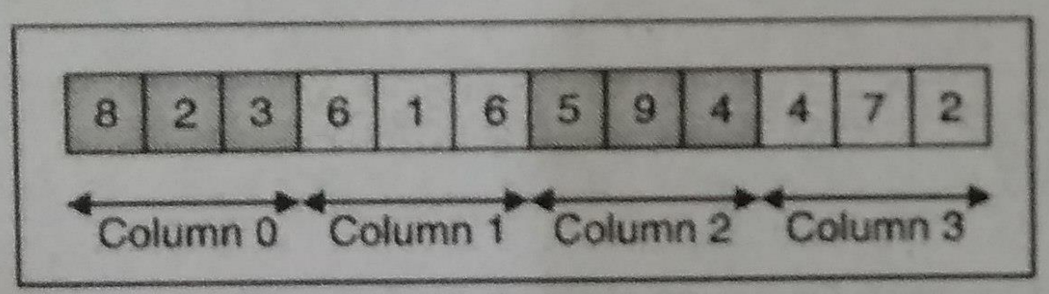


Two-Dimensional Array

- For this array the row-major representation will be :



- For column-major order, all elements of the first column come before all elements of the second column, etc.
- For the given array the column-major representation will be :



Address Calculation of 2D array

mxn matrix
4x3

A. Row major representation

Address of [i][j] = base address + i *c * element_size
+ j * element_size
= base address + (i*c*+j) * element_size

Example

- Adresss of arr[1][2] (Consider base address as 1000)

Arr[1][2 = 1000 + (1*3+2) * sizeof(int)
= 1000 + 10 = 1010

B. Column major representation

Address of [i][j] = base address + j *r * element_size
+ i * element_size
= base address + (j*r*+i) * element_size

Example

- Adresss of arr[1][2] (Consider base address as 1000)

Arr[1][2] = 1000 + (2*4+1) * sizeof(int)
= 1000 + 18 = 1018

1. Row-Major Representation:

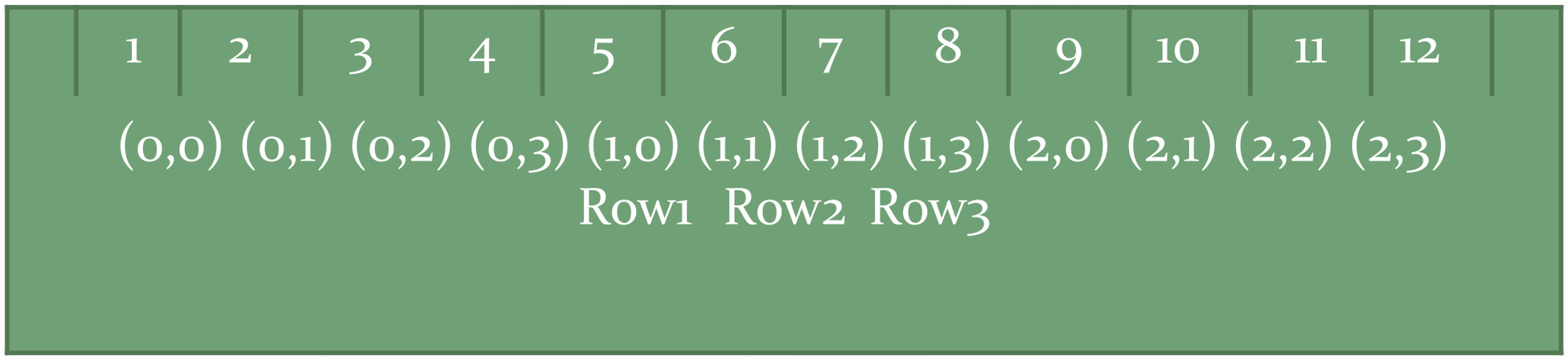
$$Address\ of\ A[i][j] = Base\ Address + [(i * n) + j] * element\ size$$

2. Column-Major Representation:

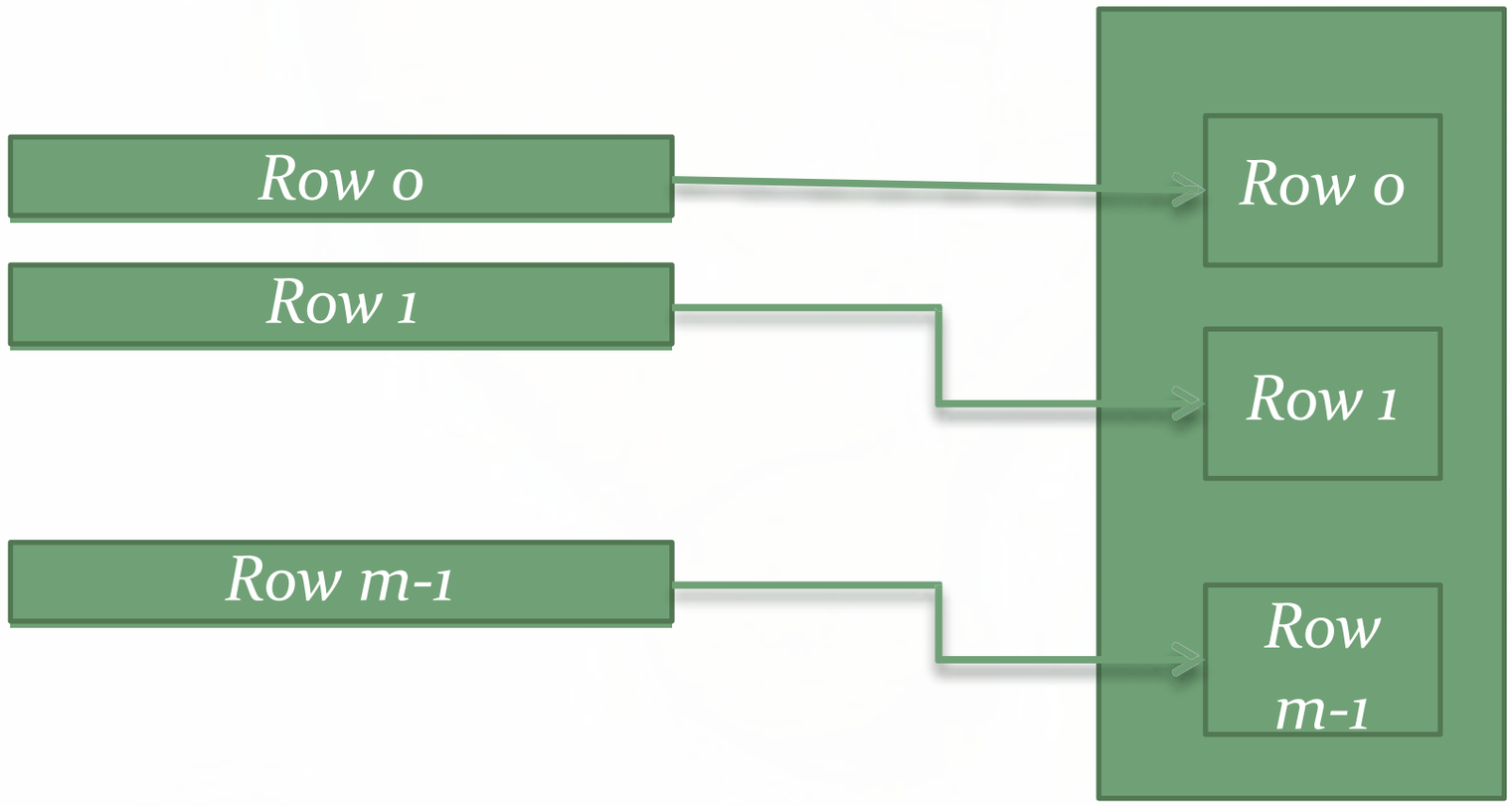
$$Address\ of\ A[i][j] = Base\ Address + [(j * m) + i] * element\ size$$

Row-major representation

- *In row-major representation, the elements of Matrix are stored row-wise, i.e., elements of 1st row, 2nd row, 3rd row, and so on till m^{th} row*



Row major arrangement

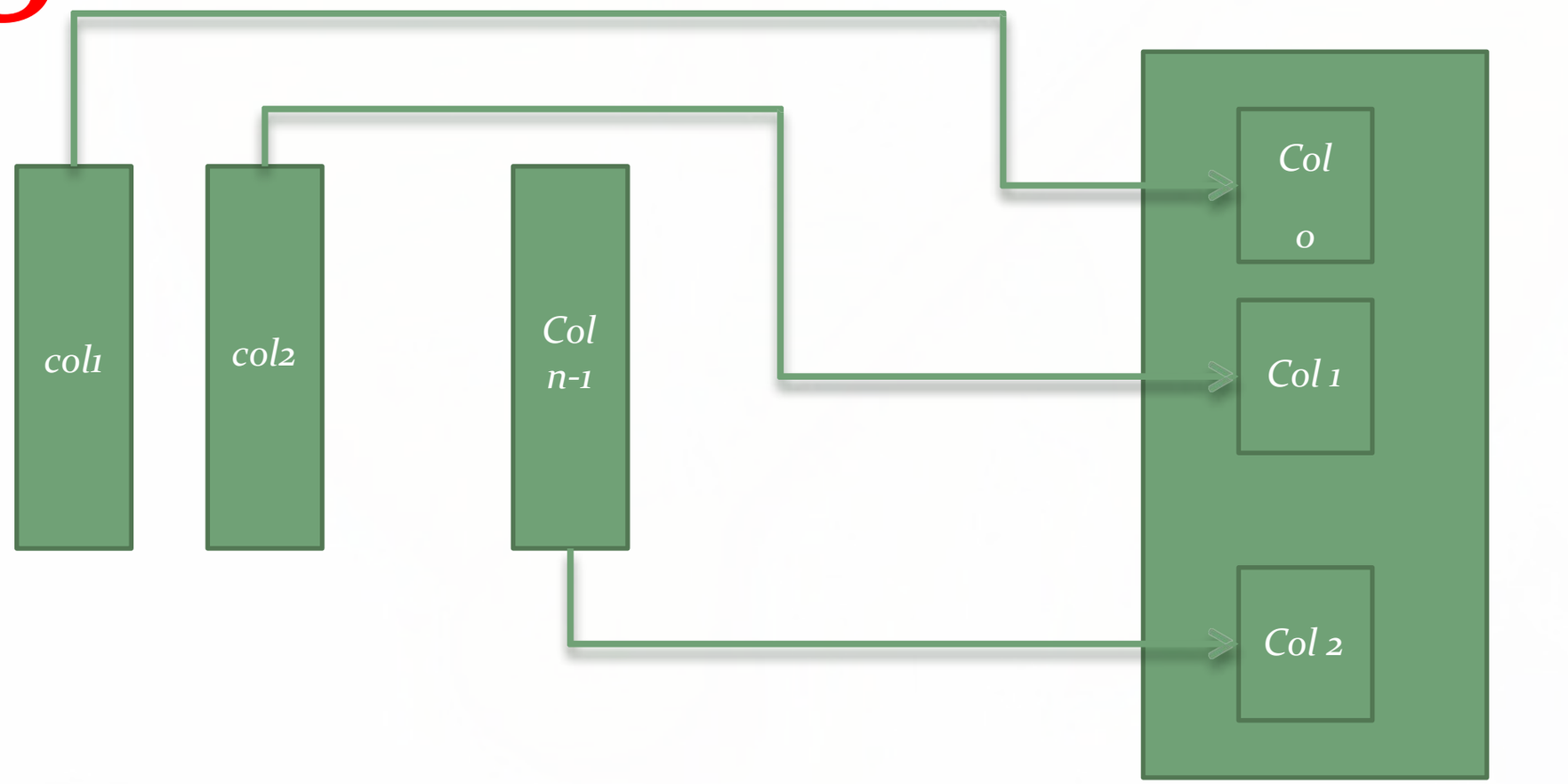


- ***Row-major arrangement in memory , in row major representation***

Column-major representation

- *In **column-major representation** m × n elements of two- dimensional array A are stored as one single row of columns.*
- *The elements are stored in **the memory as a sequence** as first the elements of column 1, then elements of column 2 and so on till elements of column n*

Column-major arrangement



Column-major arrangement in memory , in column major representation

Example 2.1: *Consider an integer array, int A[3][4] inC++. If the base address is 1050, find the address of the element A[2][3] with row-major and column-major representation of the array.*

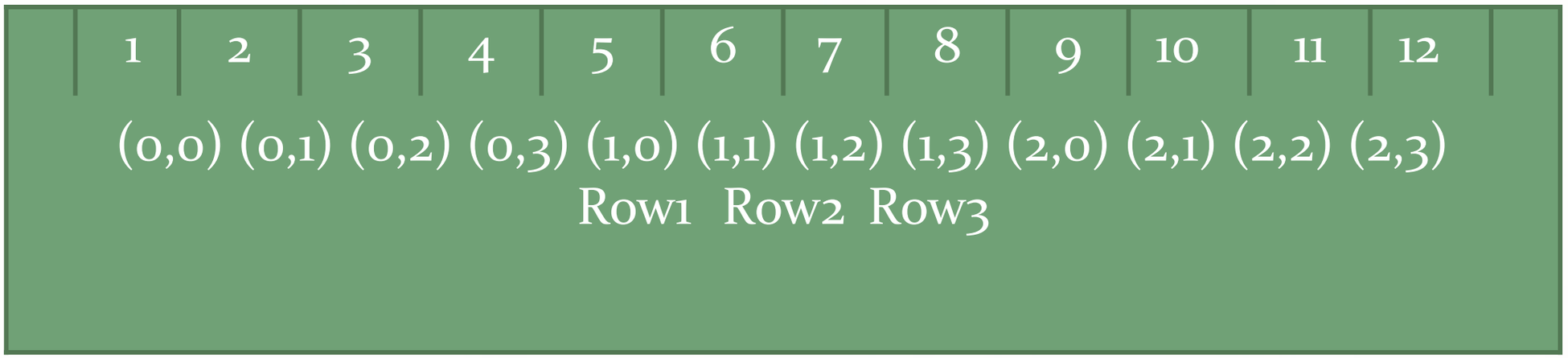
For C++, lower bound of index is 0 and we have m=3, n=4, and Base= 1050. Let us compute address of element A [2][3] using the address computation formula

1. Row-Major Representation:

$$\text{Address of } A[2][3] = \text{Base} + [(i * n) + j] * \text{element size}$$

$$= 1050 + [(2 * 4) + 3] * 2$$

$$= 1072$$



Row-Major Representation of 2-D array

2. ***Column-Major Representation:***

*Address of A [2][3] = Base Address + [(j * m) +i]*2*

*= 1050 + [(3 * 3) + 2]*2*

= 1050 + 22

= 1072

- *Here the address of the element is same because it is the last member of last row and last column.*

	1	2	3	4	5	6	7	8	9	10	11	12	
(0,0) (1,0) (2,0) (0,1) (1,1) (2,1) (0,2) (1,2) (2,2) (0,3) (1,3) (2,3)													
Col 1 Col 2 Col 3 Col 4													

Column-Major Representation of 2-D array

CONCEPT OF ORDERED LIST

LIST

- *Ordered list is the most common and frequently used data object*
- *Linear elements of an ordered list are related with each other in a particular order or sequence*
- *Following are some examples of the ordered list.*

- *1, 3,5,7,9,11,13,15*
- *January, February, March, April, May, June, July, August, September,*
- *October, November, December*
- *Red, Blue, Green, Black, Yellow*

- ***There are many basic operations*** *that can be performed on the ordered list asfollows:*

  - *Finding the length of the list*
  - *Traverse the list from left to right or from right to left*
  - *Access the ith element in the list*
  - *Update(Overwrite) the value of the ith position*
  - *Insert an element at the ith location*
  - *Delete an element at the ith position*

SINGLE VARIABLE POLYNOMIAL

2.8 Single Variable Polynomial

- A polynomial p(x) is the expression in variable x which is in the form $(ax^n + bx^{n-1} + \ldots + jx + k)$, where a, b, c ...., k fall in the category of real numbers and 'n' is non negative integer, which is called the degree of polynomial.
- An important characteristics of polynomial is that each term in the polynomial expression consists of two parts:
  - One is the coefficient
  - Other is the exponent

Example

$10x^2 + 26x$, here 10 and 26 are coefficients and 2, 1 are its exponential value.

A polynomial of a single variable A(x) can be written as

$a_nx^n + a_{n-1}x^{n-1} + a_{n-2}x^{n-2} + \ldots\ldots a_1x + a_0$ where $a_n \neq 0$ and degree of A(x) is n

Single Variable Polynomial

- *Representation Using Arrays*
- *Array of Structures*
- *Polynomial Evaluation*
- *Polynomial Addition*
- *Multiplication of Two Polynomials*

Polynomial Representation

Polynomial Representation

- It is possible to maintain polynomials such as $5x^4 - 10x^3 + 3x^2 + 10x - 1$ using array. On polynomial some basic operations such as addition and multiplication are often implemented. In such case there is need to represent polynomials.
- The easiest method of representing a polynomial having degree **n** is storing the coefficient of (n + 1) terms of the polynomial in the respective array.
- For this purpose each and every array element must have two values; coefficient and exponent.
- In this representation, it is considered that the exponent of each successive term is less than its previous term.

Polynomial Representaiton

- Array representation of polynomial assumes that the exponents of the given expression are arranged from 0 to the highest value (degree), which is represented by the subscript of the array beginning with 0.
- The coefficients of the respective exponent are placed at an appropriate index in the array.

For example

$$Q(x) = 5x^4 - 10x^3 + 3x^2 + 10x - 1.$$

$Q(x) = 5x^4 - 10x^3 + 3x^2 + 10x - 1$

Exponent	Coefficient
0	− 1
1	10
2	3
3	− 10
4	5
5	
6	

Fig. 2.8.1 : Polynomial Representation using Array

- *Polynomial as an ADT, the basic operations are as follows:*
  - *Creation of a polynomial*
  - *Addition of two polynomials*
  - *Subtraction of two polynomials*
  - *Multiplication of two polynomials*
  - *Polynomial evaluation*

Polynomial by using Array

Array

POLYNOMIAL of degree 3 P(x) = $3x^3+x^2-2x+5$						
INDEX i	0	1	2	3	.....	N-1
COEF	3	1	-2	5	....	0

POLYNOMIAL of degree 8 P(x) = $11x^8+5x^6+ x^5+2x^4 -3x^2+ x +10$									
INDEX i	0	1	2	3	4	5	6	7	8
COEF	11	0	5	1	2	0	-3	1	10

Polynomial by using Array

Array

POLYNOMIAL of degree 99 P(x) = x^{99}+ 78

INDEX i	0	1	2	3	.....	99
COEF	1	0	0	0	....	78

Drawbacks of Polynomial by using Array:

1) If exponent is too large then the array size will also be too large. Scanning such big array for reading the polynomial is time consuming.

2) There can be wastage of space if there lies a lot of difference between the exponents of each term of the polynomial

3) The array size need to be predefined while handling the polynomial using arrays

❖ *Structure is better than array for Polynomial:*

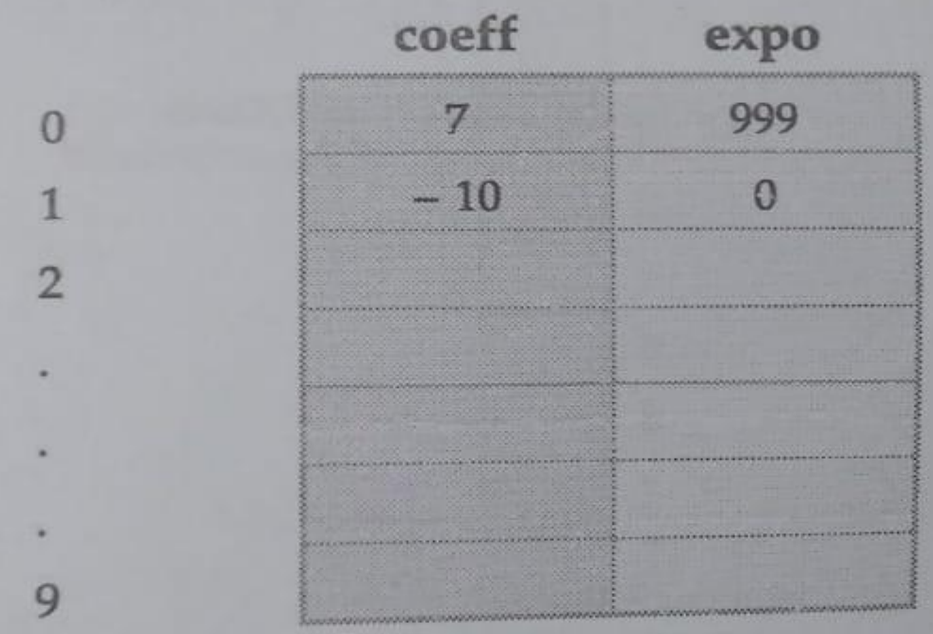
- *Such representation by an array is both time and space efficient when polynomial is not a sparse one such as polynomial P(x) of degree 3 where* $P(x)= 3x^3+x^2-2x+5$
- *But when polynomial is sparse such as in worst case a polynomial as* $A(x)= x^{99} + 75$ *for degree of n =100, then only two locations out of 101 would be used.*
- *In such cases it is better to store polynomial as pairs of coefficient and exponent. We may go for two different arrays for each or a structure having two members as two arrays for each of coeff. and Exp or an array of structure that consists of two data members coefficient and exponent.*

Polynomial by using structure

- *Let us go for structure having two data members coefficient and exponent and itsarray.*

POLYNOMIAL of degree 3 P(x) = $3x^3+x^2-2x+5$						
INDEX i	0	1	2	3	.....	N-1
COEF	3	1	-2	5	....	
EXPO	3	2	1	0	.....	

Here, array P1 is of size 10. But we can store any polynomial such as $7x^{999} - 10$ in it. The representation of the above structure will be



array P1 [10]

Fig. 2.9.2 Representation of Polynomial using Structure

This representation is very much better than previous one because it avoids wastage of space.

Advantages:

Polynomial using Array of Structures

1) There is no limit on the maximum value of the exponent
2) It requires less number of terms although there is vast difference between maximum and minimum value of exponent

Addition of two polynomials:

Polynomial Representation (Simple)

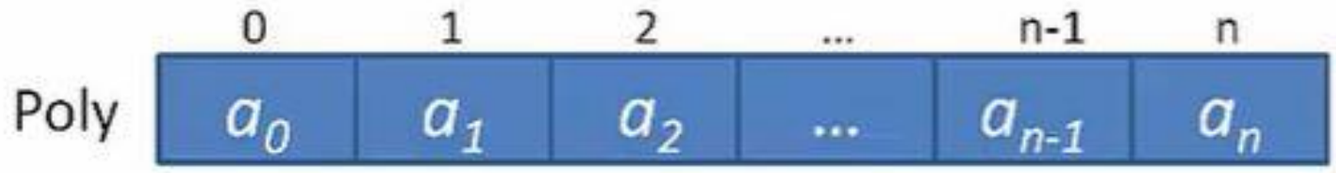
If the Polynomial is $-10 + 3x + 5x^2$ then we can write it as :
$-10x^0 + 3x^1 + 5x^2$

$-10x^0 + 3x^1 + 5x^2$

int Poly[3];

Polynomial Representation (Simple)

A polynomial of a single variable *A(x) can be written as* $a_0 + a_1X + a_2X^2 + \ldots + a_nX^n$ where $a_n \neq 0$ and degree of *A(X) is n.*



For a polynomial of degree n, n+1 terms are required

Polynomial Addition (Simple)

$X1 = 3x^1 + 5x^2 + 7x^3$

$X2 = 10x^0 + 3x^1 + 5x^2$

Polynomial Addition (Simple)

$X1 = 3x^1 + 5x^2 + 7x^3$

Degree of X1 is M=3

$X2 = 10x^0 + 3x^1 + 5x^2$

Degree of X2 is N=2

- *Identify the value of Highest degree polynomial.*
- *Write polynomial X3 with degree Max(degree of X1 and degree of X2).*

Polynomial Addition (Simple)

$X1 = 3x^1 + 5x^2 + 7x^3$

$X2 = 10x^0 + 3x^1 + 5x^2$

$X1 = 0x^0 + 3x^1 + 5x^2 + 7x^3$

$X2 = 10x^0 + 3x^1 + 5x^2 + 0x^3$

$X3 = \_\_x^0 + \_\_x^1 + \_\_x^2 + \_\_x^3$

Polynomial Addition (Simple)

$X1 = 3x^1 + 5x^2 + 7x^3$

$X2 = 10x^0 + 3x^1 + 5x^2$

$X1 = 0x^0 + 3x^1 + 5x^2 + 7x^3$

$X2 = 10x^0 + 3x^1 + 5x^2 + 0x^3$

0	1	2	3
0	3	5	7

0	1	2	3
10	3	5	0

$X3 = \_\_x^0 + \_\_x^1 + \_\_x^2 + \_\_x^3$

k= 0	1	2	3

Polynomial Addition (Simple)

$X1 = 3x^1 + 5x^2 + 7x^3$

$X2 = 10x^0 + 3x^1 + 5x^2$

$X1 = 0x^0 + 3x^1 + 5x^2 + 7x^3$

$X2 = 10x^0 + 3x^1 + 5x^2 + 0x^3$

j= 0	1	2	3
0	3	5	7

j= 0	1	2	3
10	3	5	0

$a_0 = 0 + 10 =$

$X3 = \underline{10}\ x^0 + \underline{\ \ }\ x^1 + \underline{\ \ }\ x^2 + \underline{\ \ }\ x^3$

k= 0	1	2	3

```
i=j=k=0
while (i <= M)
 {
 C[k] = A[i] +B[j]
  i = i ++; j = j ++, k = k++
}
```

Polynomial Addition (Simple)

$X1 = 3x^1 + 5x^2 + 7x^3$

$X1 = 0x^0 + 3x^1 + 5x^2 + 7x^3$

0	i= 1	2	3
0	3	5	7

$X2 = 10x^0 + 3x^1 + 5x^2$

$X2 = 10x^0 + 3x^1 + 5x^2 + 0x^3$

0	j= 1	2	3
10	3	5	0

$a_1 = 3 + 3 =$

$X3 = 10x^0 + 6\,x^1 + \_\_\,x^2 + \_\_\,x^3$

0	k= 1	2	3
10			

```
i=j=k=1
while (i <= M)
 {
 C[k] = A[i] +B[j]
  i = i ++; j = j ++, k = k++
}
```

Polynomial Addition (Simple)

$X1 = 3x^1 + 5x^2 + 7x^3$

$X1 = 0x^0 + 3x^1 + 5x^2 + 7x^3$

0	1	2	3
0	3	5	7

$X2 = -10x^0 + 3x^1 + 5x^2$

$X2 = 10x^0 + 3x^1 + 5x^2 + 0x^3$

0	1	2	3
10	3	5	0

$X3 = 10x^0 + 6x^1 + 10x^2 + 7x^3$

0	1	2	3
10	6	10	7

Polynomial Addition

$X1 = 7x^4 + 5x^2 + 3x^1$

$X2 = 5x^3 + 3x^1 - 8x^0$

X1=

	i=0	1	2
Coefficient	7	5	3
Exponent	4	2	1

X2=

	j=0	1	2
Coefficient	5	3	-8
Exponent	3	1	0

X3=

	k=0
Coefficient	
Exponent	

Polynomial Addition

$X1 = 7x^4 + 5x^2 + 3x^1$

$X2 = 5x^3 + 3x^1 - 8x^0$

X1=

	i=0	1	2
Coefficient	7	5	3
Exponent	4	2	1

X2=

	j=0	1	2
Coefficient	5	3	-8
Exponent	3	1	0

4 > 3

CASE-1

If the exponent of the term pointed by j in X2 is less than the exponent of the current term pointed by i of X1 , then copy the current term of X1 pointed by i in the location pointed by k in polynomial X3. Advance the pointer i and k to the next

X3=

	k=0
Coefficient	
Exponent	

Polynomial Addition

$X1 = 7x^4 + 5x^2 + 3x^1$

$X2 = 5x^3 + 3x^1 - 8x^0$

X1=

	0	i=1	2
Coefficient	7	5	3
Exponent	4	2	1

X2=

	j=0	1	2
Coefficient	5	3	-8
Exponent	3	1	0

X3=

	0	k=1
Coefficient	7	
Exponent	4	

```
if(X1[i].expo > X2[j].expo)
{
 X3[k]. coeff = X1[i]. coeff;
 X3[k].expo = X1[i].expo;
 i = i + 1
 k = k + 1
}
```

Polynomial Addition

$X1 = 7x^4 + 5x^2 + 3x^1$

$X2 = 5x^3 + 3x^1 - 8x^0$

X1=

	0	i=1	2
Coefficient	7	5	3
Exponent	4	2	1

X2=

	j=0	1	2
Coefficient	5	3	-8
Exponent	3	1	0

$2 < 3$

X3=

	0	k=1	2
Coefficient	7		
Exponent	4		

Polynomial Addition

$X1 = 7x^4 + 5x^2 + 3x^1$

$X2 = 5x^3 + 3x^1 - 8x^0$

X1=

	0	i=1	2
Coefficient	7	5	3
Exponent	4	2	1

X2=

	j=0	1	2
Coefficient	5	3	-8
Exponent	3	1	0

X3=

	0	k=1	2
Coefficient	7	5	
Exponent	4	3	

CASE-2

If the exponent of the term pointed by j in X2 is greater than the exponent of the current term pointed by i of X1, then copy the current term of X2 pointed by j in the location pointed by k in polynomial X3. Advance the pointer j and k to the next term.

Polynomial Addition

$X1 = 7x^4 + 5x^2 + 3x^1$

$X2 = 5x^3 + 3x^1 - 8x^0$

X1=

	0	i=1	2
Coefficient	7	5	3
Exponent	4	2	1

X2=

	0	j=1	2
Coefficient	5	3	-8
Exponent	3	1	0

X3=

	0	1	k=2
Coefficient	7	5	
Exponent	4	3	

```
if(X1[i].expo < X2[j].expo)
{
 X3[k]. coeff = X2[j]. coeff;
 X3[k].expo = X2[j].expo;
 j = j + 1
 k = k + 1
}
```

Polynomial Addition

$X1 = 7x^4 + 5x^2 + 3x^1$

$X2 = 5x^3 + 3x^1 - 8x^0$

X1=

	0	1	i=2
Coefficient	7	5	3
Exponent	4	2	1

X2=

	0	j=1	2
Coefficient	5	3	-8
Exponent	3	1	0

1 = 1

X3=

	0	1	2	k=3	4
Coefficient	7	5	5		
Exponent	4	3	2		

CASE-3

If the exponents of the two terms of polynomials X1 and X2 are equal, then the coefficients are added, and the new term is stored in the resultant polynomial X3 and advance I, j and k to track to

Polynomial Addition

$X1 = 7x^4 + 5x^2 + 3x^1$

$X2 = 5x^3 + 3x^1 - 8x^0$

X1=

	0	1	i=2
Coefficient	7	5	3
Exponent	4	2	1

X2=

	0	j=1	2
Coefficient	5	3	-8
Exponent	3	1	0

X3=

	0	1	2	k=3	4
Coefficient	7	5	5	6	
Exponent	4	3	2	1	

Polynomial Addition

$X1 = 7x^4 + 5x^2 + 3x^1$

$X2 = 5x^3 + 3x^1 - 8x^0$

X1=

	0	1	i=2
Coefficient	7	5	3
Exponent	4	2	1

X2=

	0	1	j=2
Coefficient	5	3	-8
Exponent	3	1	0

X3=

	0	1	2	3	k=4
Coefficient	7	5	5	6	
Exponent	4	3	2	1	

```
if(X1[i].expo == X2[j].expo)
{
X3[k].coeff = X1[i].coeff +
                        X2[j].coeff
X3[k].expo = X1[i].expo;
 i = i + 1
 j=j+1
```

Polynomial Addition

$X1 = 7x^4 + 5x^2 + 3x^1$

$X2 = 5x^3 + 3x^1 - 8x^0$

X1=

	0	1	i=2
Coefficient	7	5	3
Exponent	4	2	1

X2=

	0	1	j=2
Coefficient	5	3	-8
Exponent	3	1	0

No more element in i

X3=

	0	1	2	3	k=4
Coefficient	7	5	5	6	-8
Exponent	4	3	2	1	0

CASE-3

If there is no more elements in X1 and there are few elements remaining in X2 then copy rest of the element in X2 to X3 and advance j and k to track to the next term.

Polynomial Addition

$X1 = 7x^4 + 5x^2 + 3x^1$

$X2 = 5x^3 + 3x^1 - 8x^0$

X1=

	0	1	i=2
Coefficient	7	5	3
Exponent	4	2	1

X2=

	0	1	j=2
Coefficient	5	3	-8
Exponent	3	1	0

X3=

	0	1	2	3	k=4
Coefficient	7	5	5	6	
Exponent	4	3	2	1	

```
while (j < n) do
{
 X3[k].coeff = X2[j].coeff;
 X3[k].expo = X2[j].expo;
 j = j + 1
 k = k + 1
}
```

Polynomial Addition

$X1 = 7x^4 + 5x^2 + 3x^1$

$X2 = 5x^3 + 3x^1 - 8x^0$

X1=

	0	1	i=2
Coefficient	7	5	3
Exponent	4	2	1

X2=

	0	1	j=2
Coefficient	5	3	-8
Exponent	3	1	0

X3=

	0	1	2	3	k=4
Coefficient	7	5	5	6	-8
Exponent	4	3	2	1	0

Polynomial Addition

```
Polynomial Polynomial :: Add_Poly(Polynomial X2)
{
 int i = j = k = 0;
 Polynomial X3;
 while (i < X1.Total_Terms && j < X2.Total_Terms)
 {
  if(X1.Poly[i].expo == X2.Poly[j]. expo)
  {
   X3.Poly[k].coeff = X1.Poly[i]. coeff + X2.Poly[j]. coeff
   X3.Poly[k]. expo = X1.Poly[i]. expo;
   i++; j++; k++;
  }
 else if(X1.Poly[i]. expo > X2.Poly[j]. expo)
 {
  X3.Poly[k]. coeff = X1.Poly[i].coeff;
  X3.Poly[k]. expo = X1.Poly[i]. expo;
  i++; k++;
 }
else
 {
   X3.Poly[k]. coeff = X2.Poly[j]. coeff;
   X3.Poly[k]. expo = X2.Poly[j]. expo;
   j++; k++;
  }
  }   // end of while
while(i < X1.Total_Terms)
  {
   X3.Poly[k]. coeff = X1.Poly[i]. coeff;
   X3.Poly[k]. expo = X1.Poly[i]. expo;
   i++; k++;
  }
  while(j < X2.Total_Terms)
  {
   X3.Poly[k]. coeff = X2.Poly[j]. coeff;
   X3.Poly[k]. expo = X2.Poly[j]. expo;
   j++; k++;
  }
  C.Total_Terms = k - 1;
  return X3;
  }
```

3.11.2 C-Function for Addition of Two Polynomials Represented in an Array

Q. Write pseudo C/C++ code to perform addition of two polynomials using arrays. **SPPU - Dec. 18, 6 Marks**

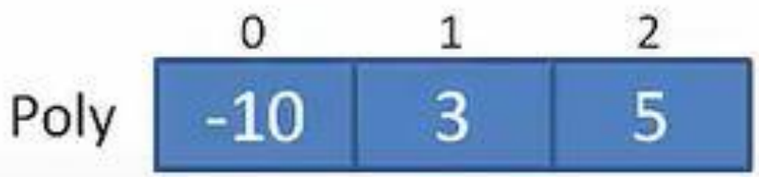
```
void addp(int a[ ], int b[ ], int c[ ])
{
    int m, n, i, j, k;
    m = a[0];
    n = b[0];
    i = j = k = 1;
    while(i< = 2*m && j< = 2*n)
    {
        if(a[i] > b[j])
        {
            c[k] = a[i];
            c[k+1] = a[i+1];
            k = k+2;
            i = i+2;
        }
        else
        if(a[i] < b[j])
        {
            c[k] = b[j];
            c[k+1] = b[j+1];
            k = k+2;
            j = j+2;
        }
```

```
    else
    {
        c[k] = b[j];
        c[k+1] = a[i+1]+b[j+1];
        k = k+2;
        i = i+2;
        j = j+2;
    }
}
while(i < = 2*m)
{
    c[k] = a[i];
    c[k+1] = a[i+j];
    k = k+2; i = i+2i
}
while(j< = 2 = *n)
{
    c[k] = b[j]
    c [k +1] = b[j+1]
    k = k+2; j = j+2;
}
c(0) = k/2;
}
```

Polynomial Representation (Simple)

If the Polynomial is $-10 + 3x + 5x^2$ then we can write it as :
$-10x^0 + 3x^1 + 5x^2$

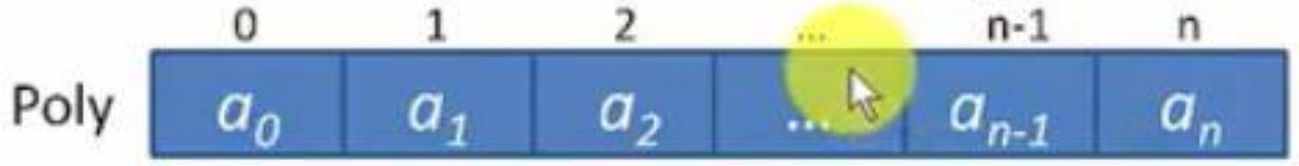
$-10x^0 + 3x^1 + 5x^2$




```
int Poly[3];
```

Polynomial Representation (Simple)

A polynomial of a single variable *A(x) can be written as* $a_0 + a_1X + a_2X^2 + \ldots + a_nX^n$ where $a_n \neq 0$ and degree of *A(X) is n.*



For a polynomial of degree n, n+1 terms are required

Polynomial Multiplication (Simple)

$X1 = 3x^1 + 5x^2 + 7x^3$

$X2 = 10x^0 + 3x^1 + 5x^2$

Polynomial Multiplication (Simple)

$X1 = 3x^1 + 5x^2 + 7x^3$

Degree of X1 is M=3

$X2 = 10x^0 + 3x^1 + 5x^2$

Degree of X2 is N=2

- *Identify the degree polynomials.*
- *Write degree of polynomial X3 as sum of degree of X1 and degree of X2.*

Polynomial Multiplication (Simple)

$X1 = 3x^1 + 5x^2 + 7x^3$

$X2 = 10x^0 + 3x^1 + 5x^2$

$X1 = 0x^0 + 3x^1 + 5x^2 + 7x^3$

$X2 = 10x^0 + 3x^1 + 5x^2$

j= 0	1	2	3
0	3	5	7

j= 0	1	2
10	3	5

$a_0 =$

$i =$

$X3 = \_\_x^0 + \_\_x^1 + \_\_x^2 + \_\_x^3 + \_\_x^4 + \_\_x^5$

k=0	1	2	3	4	5

Polynomial Multiplication (Simple)

$X1 = 3x^1 + 5x^2 + 7x^3$

$X2 = 10x^0 + 3x^1 + 5x^2$

$X1 = 0x^0 + 3x^1 + 5x^2 + 7x^3$

$X2 = 10x^0 + 3x^1 + 5x^2$

j= 0	1	2	3
0	3	5	7

j= 0	1	2
10	3	5

$a_0 = 0 * 10 = 0$

$i =$

$X3 = \_\_x^0 + \_\_x^1 + \_\_x^2 + \_\_x^3 + \_\_x^4 + \_\_x^5$

k=0	1	2	3	4	5

Polynomial Multiplication (Simple)

$X1 = 3x^1 + 5x^2 + 7x^3$

$X2 = 10x^0 + 3x^1 + 5x^2$

$X1 = 0x^0 + 3x^1 + 5x^2 + 7x^3$

$X2 = 10x^0 + 3x^1 + 5x^2$

j= 0	1	2	3
0	3	5	7

j= 0	1	2
10	3	5

$a_0 = \ldots = 0$

$i = \ldots = 0$

$X3 = \_\_x^0 + \_\_x^1 + \_\_x^2 + \_\_x^3 + \_\_x^4 + \_\_x^5$

k=0	1	2	3	4	5

Polynomial Multiplication (Simple)

$X1 = 3x^1 + 5x^2 + 7x^3$

$X2 = 10x^0 + 3x^1 + 5x^2$

$X1 = 0x^0 + 3x^1 + 5x^2 + 7x^3$

$X2 = 10x^0 + 3x^1 + 5x^2$

i= 0	1	2	3
0	3	5	7

0	j= 1	2
10	3	5

$a_0 =$

$i =$

$X3 = \_\_x^0 + \_\_x^1 + \_\_x^2 + \_\_x^3 + \_\_x^4 + \_\_x^5$

k=0	1	2	3	4	5

```
for(i=0;i<=m;i++)
for(j=0;j<=n;j++)
{
 k=i+j
X3[k] = X3[k]+ X1[i] * X2[j]
}
```

Polynomial Multiplication (Simple)

$X1 = 3x^1 + 5x^2 + 7x^3$

$X2 = 10x^0 + 3x^1 + 5x^2$

$X1 = 0x^0 + 3x^1 + 5x^2 + 7x^3$

$X2 = 10x^0 + 3x^1 + 5x^2$

i= 0	1	2	3
0	3	5	7

0	j=1	2
10	3	5

$a_0 = 0 * 3 = 0$

$i =$

$X3 = \_\_x^0 + \_\_x^1 + \_\_x^2 + \_\_x^3 + \_\_x^4 + \_\_x^5$

k=0	1	2	3	4	5
0					

Polynomial Multiplication (Simple)

$X1 = 3x^1 + 5x^2 + 7x^3$

$X2 = 10x^0 + 3x^1 + 5x^2$

$X1 = 0x^0 + 3x^1 + 5x^2 + 7x^3$

$X2 = 10x^0 + 3x^1 + 5x^2$

j= 0	1	2	3
0	3	5	7

0	j=1	2
10	3	5

$a_0 =$ 0

$i =$ 1

$X3 = \_\_x^0 + \_\_x^1 + \_\_x^2 + \_\_x^3 + \_\_x^4 + \_\_x^5$

k=0	1	2	3	4	5
0					

Polynomial Multiplication (Simple)

$X1 = 3x^1 + 5x^2 + 7x^3$

$X2 = 10x^0 + 3x^1 + 5x^2$

$X1 = 0x^0 + 3x^1 + 5x^2 + 7x^3$

$X2 = 10x^0 + 3x^1 + 5x^2$

i= 0	1	2	3
0	3	5	7

0	1	j= 2
10	3	5

$a_0 =$

$i =$

$X3 = \_\_x^0 + 0\_x^1 + \_\_x^2 + \_\_x^3 + \_\_x^4 + \_\_x^5$

k=0	1	2	3	4	5
0					

Polynomial Multiplication (Simple)

$X1 = 3x^1 + 5x^2 + 7x^3$

$X1 = 0x^0 + 3x^1 + 5x^2 + 7x^3$

i= 0	1	2	3
0	3	5	7

$X2 = 10x^0 + 3x^1 + 5x^2$

$X2 = 10x^0 + 3x^1 + 5x^2$

0	1	j=2
10	3	5

$a_0 = 0 * 5 - 0$

$i = 0 + 2 - 2$

$X3 = \_\_x^0 + \_\_x^1 + \_\_x^2 + \_\_x^3 + \_\_x^4 + \_\_x^5$

k=0	1	2	3	4	5
0	0	0			

Polynomial Multiplication (Simple)

$X1 = 3x^1 + 5x^2 + 7x^3$

$X2 = 10x^0 + 3x^1 + 5x^2$

$X1 = 0x^0 + 3x^1 + 5x^2 + 7x^3$

$X2 = 10x^0 + 3x^1 + 5x^2$

0	i=1	2	3
0	3	5	7

i= 0	1	2
10	3	5

$a_0 =$

$i =$

$X3 = \_\_x^0 + \_\_x^1 + 0\,x^2 + \_\_x^3 + \_\_x^4 + \_\_x^5$

k=0	1	2	3	4	5
0	0	0			

```
for(i=0;i<=m;i++)
for(j=0;j<=n;j++)
{
 k=i+j
X3[k] = X3[k]+ X1[i] * X2[j]
}
```

Polynomial Multiplication (Simple)

$X1 = 3x^1 + 5x^2 + 7x^3$

$X2 = 10x^0 + 3x^1 + 5x^2$

$X1 = 0x^0 + 3x^1 + 5x^2 + 7x^3$

$X2 = 10x^0 + 3x^1 + 5x^2$

0	j= 1	2	3
0	3	5	7

j=0	1	2
10	3	5

$a_0 = 3 * 10 = 30$

$i = 1 \quad 0$

$X3 = \_\_x^0 + \_\_x^1 + \_\_x^2 + \_\_x^3 + \_\_x^4 + \_\_x^5$

k=0	1	2	3	4	5
0	0	0			

Polynomial Multiplication (Simple)

$X1 = 3x^1 + 5x^2 + 7x^3$

$X2 = 10x^0 + 3x^1 + 5x^2$

$X1 = 0x^0 + 3x^1 + 5x^2 + 7x^3$

$X2 = 10x^0 + 3x^1 + 5x^2$

0	j= 1	2	3
0	3	5	7

j=0	1	2
10	3	5

$a_0 =$ 30

$i =$ 1

$X3 = \_\_x^0 + \_\_x^1 + \_\_x^2 + \_\_x^3 + \_\_x^4 + \_\_x^5$

k=0	1	2	3	4	5
0	0	0			

Polynomial Multiplication (Simple)

$X1 = 3x^1 + 5x^2 + 7x^3$

$X2 = 10x^0 + 3x^1 + 5x^2$

$X1 = 0x^0 + 3x^1 + 5x^2 + 7x^3$

$X2 = 10x^0 + 3x^1 + 5x^2$

0	j= 1	2	3
0	3	5	7

0	j=1	2
10	3	5

$a_0 =$

$i =$

$X3 = \_\_x^0 + \_\_x^1 + \_\_x^2 + \_\_x^3 + \_\_x^4 + \_\_x^5$

k=0	1	2	3	4	5
0	30	0	0		

Polynomial Multiplication (Simple)

$X1 = 3x^1 + 5x^2 + 7x^3$

$X2 = 10x^0 + 3x^1 + 5x^2$

$X1 = 0x^0 + 3x^1 + 5x^2 + 7x^3$

$X2 = 10x^0 + 3x^1 + 5x^2$

0	j= 1	2	3
0	3	5	7

0	j=1	2
10	3	5

$a_0 = 3 * 3 = 9$

$i = 1 \quad 1$

$X3 = \_\_x^0 + \_\_x^1 + \_\_x^2 + \_\_x^3 + \_\_x^4 + \_\_x^5$

k=0	1	2	3	4	5
0	30	0	0		

Polynomial Multiplication (Simple)

$X1 = 3x^1 + 5x^2 + 7x^3$

$X1 = 0x^0 + 3x^1 + 5x^2 + 7x^3$

0	j= 1	2	3
0	3	5	7

$X2 = 10x^0 + 3x^1 + 5x^2$

$X2 = 10x^0 + 3x^1 + 5x^2$

0	j=1	2
10	3	5

$a_0 =$

$i =$

$X3 = \_\_x^0 + \_\_x^1 + 9\_\_x^2 + \_\_x^3 + \_\_x^4 + \_\_x^5$

k=0	1	2	3	4	5
0	30	0	0		

Polynomial Multiplication (Simple)

$X1 = 3x^1 + 5x^2 + 7x^3$

$X1 = 0x^0 + 3x^1 + 5x^2 + 7x^3$

0	i= 1	2	3
0	3	5	7

$X2 = 10x^0 + 3x^1 + 5x^2$

$X2 = 10x^0 + 3x^1 + 5x^2$

0	1	j=2
10	3	5

$a_0 = 3 * 5 = 15$

$i = 1 \quad 2$

$X3 = \_\_x^0 + \_\_x^1 + \_\_x^2 + \_\_x^3 + \_\_x^4 + \_\_x^5$

k=0	1	2	3	4	5
0	30	9	0		

Polynomial Multiplication (Simple)

$X1 = 3x^1 + 5x^2 + 7x^3$

$X2 = 10x^0 + 3x^1 + 5x^2$

$X1 = 0x^0 + 3x^1 + 5x^2 + 7x^3$

$X2 = 10x^0 + 3x^1 + 5x^2$

0	1	i=2	3
0	3	5	7

i=0	1	2
10	3	5

$a_0 = 5 * 10 = 50$

$i =$

$X3 = \_\_x^0 + \_\_x^1 + \_\_x^2 + \_\_x^3 + \_\_x^4 + \_\_x^5$

k=0	1	2	3	4	5
0	30	9	15	0	

Polynomial Multiplication (Simple)

$X1 = 3x^1 + 5x^2 + 7x^3$

$X2 = 10x^0 + 3x^1 + 5x^2$

$X1 = 0x^0 + 3x^1 + 5x^2 + 7x^3$

$X2 = 10x^0 + 3x^1 + 5x^2$

0	1	i=2	3
0	3	5	7

j=0	1	2
10	3	5

$a_0 =$ 50

$i =$ 2

$X3 = \_\_x^0 + \_\_x^1 + \_\_x^2 + \_\_x^3 + \_\_x^4 + \_\_x^5$

k=0	1	2	3	4	5
0	30	9	15	0	

Polynomial Multiplication (Simple)

$X1 = 3x^1 + 5x^2 + 7x^3$

$X1 = 0x^0 + 3x^1 + 5x^2 + 7x^3$

0	1	i=2	3
0	3	5	7

$X2 = 10x^0 + 3x^1 + 5x^2$

$X2 = 10x^0 + 3x^1 + 5x^2$

0	j=1	2
10	3	5

$a_0 =$

$i =$

$X3 = \_\_x^0 + \_\_x^1 + 50x^2 + \_\_x^3 + \_\_x^4 + \_\_x^5$

k=0	1	2	3	4	5
0	30	9	15	0	

```
for(i=0;i<=m;i++)
for(j=0;j<=n;j++)
 {
  k=i+j
X3[k] = X3[k]+ X1[i] * X2[j]
 }
```

Polynomial Multiplication (Simple)

$X1 = 3x^1 + 5x^2 + 7x^3$

$X2 = 10x^0 + 3x^1 + 5x^2$

$X1 = 0x^0 + 3x^1 + 5x^2 + 7x^3$

$X2 = 10x^0 + 3x^1 + 5x^2$

0	1	i=2	3
0	3	5	7

0	j=1	2
10	3	5

$a_0 = 5 * 3 = 15$

$i = 2 \quad 1$

$X3 = \_\_x^0 + \_\_x^1 + \_\_x^2 + \_\_x^3 + \_\_x^4 + \_\_x^5$

k=0	1	2	3	4	5
0	30	59	15	0	

Polynomial Multiplication

$X1 = 7x^4 + 5x^2 + 3x^1$

$X2 = 5x^3 + 3x^1$

X1=

	i=0	1	2
Coefficient	7	5	3
Exponent	4	2	1

X2=

	j=0	1
Coefficient	5	3
Exponent	3	1

Coefficient *Exponent*

X3=

	k=0	1	2	3	4	5	6
Coefficient							
Exponent							

- Add Exponent of X1 and X2.
- Multiply the Coefficient of X1 and X2.
- Compare it with existing exponent of X3.
- If a term with equal exponent is found, then add the new Coefficient to that term of polynomial X3, else search for an appropriate position for the Coefficient and insert the same in polynomial X3.

Polynomial Multiplication

$X1 = 7x^4 + 5x^2 + 3x^1$

$X2 = 5x^3 + 3x^1$

X1=

	i=0	1	2
Coefficient	7	5	3
Exponent	4	2	1

X2=

	i=0	1
Coefficient	5	3
Exponent	3	1

Coefficient 7 * 5 = 35 Exponent 4 + 3 = 7

X3=

	k=0	1	2	3	4	5	6
Coefficient							
Exponent							

- Add Exponent of X1 and X2.
- Multiply the Coefficient of X1 and X2.
- Compare it with existing exponent of X3.
- If a term with equal exponent is found, then add the new Coefficient to that term of polynomial X3, else search for an appropriate position for the Coefficient and insert the same in polynomial X3.

Polynomial Multiplication

$X1 = 7x^4 + 5x^2 + 3x^1$

$X2 = 5x^3 + 3x^1$

X1=

	i=0	1	2
Coefficient	7	5	3
Exponent	4	2	1

X2=

	0	j=1
Coefficient	5	3
Exponent	3	1

Coefficient **7 * 3 = 21**

Exponent **4 + 1 =5**

X3=

	0	k=1	2	3	4	5	6
Coefficient	35						
Exponent	7						

NewTerm_exp = X1[i].Exp + X2[j].Exp;

NewTerm_coef = X1[i].Coef * X2[j].Coef;

- Compare it with existing exponent of X3.
- If a term with equal exponent is found, then add the new Coefficient to that

Polynomial Multiplication

$X1 = 7x^4 + 5x^2 + 3x^1$

$X2 = 5x^3 + 3x^1$

X1=

	i=0	1	2
Coefficient	7	5	3
Exponent	4	2	1

X2=

	0	j= 1
Coefficient	5	3
Exponent	3	1

Coefficient **7 * 3 =** Exponent **4 + 1 =**

X3=

	0	k=1	2	3	4	5	6
Coefficient	35	21					
Exponent	7	5					

NewTerm_exp = X1[i].Exp + X2[j].Exp;

NewTerm_coef = X1[i].Coef * X2[j].Coef;

- Compare it with existing exponent of X3.
- If a term with equal exponent is found, then add the new Coefficient to that

Polynomial Multiplication

$X1 = 7x^4 + 5x^2 + 3x^1$

$X2 = 5x^3 + 3x^1$

X1=

	0	i=1	2
Coefficient	7	5	3
Exponent	4	2	1

X2=

	j=0	1
Coefficient	5	3
Exponent	3	1

Coefficient **5** * **5** =

Exponent **2** + **3** =

X3=

	0	k=1	2	3	4	5	6
Coefficient	35	25 21					
Exponent	7	5					

```
flag = 0;
while(TmpIndex < k)      // Insert NewTerm in Polynomial X3
{  if(X3.Poly[TmpIndex].Exp == NewTerm_exp)     // search matching exponent
     {   flag = 1; break;       }
    else if(X3.Poly[TmpIndex].Exp < NewTerm_exp)
    break;      TmpIndex++;      }
```

Polynomial Multiplication

$X1 = 7x^4 + 5x^2 + 3x^1$

$X2 = 5x^3 + 3x^1$

X1=

	0	i=1	2
Coefficient	7	5	3
Exponent	4	2	1

X2=

	0	j=1
Coefficient	5	3
Exponent	3	1

Coefficient **5 * 3 = 15**

Exponent **2 + 1 = 3**

X3=

	0	1	k= 2	3	4	5	6
Coefficient	35	46					
Exponent	7	5					

Polynomial Multiplication

$X1 = 7x^4 + 5x^2 + 3x^1$

$X2 = 5x^3 + 3x^1$

X1=

	0	i=1	2
Coefficient	7	5	3
Exponent	4	2	1

X2=

	0	i=1
Coefficient	5	3
Exponent	3	1

Coefficient 5 * 3 =

Exponent 2 + 1 =

X3=

	0	1	k= 2	3	4	5	6
Coefficient	35	46	15				
Exponent	7	5	3				

Polynomial Multiplication

$X1 = 7x^4 + 5x^2 + 3x^1$

$X2 = 5x^3 + 3x^1$

X1=

	0	1	i= 2
Coefficient	7	5	3
Exponent	4	2	1

X2=

	j=0	1
Coefficient	5	3
Exponent	3	1

Coefficient **3 * 5 = 15** Exponent **1 + 3 = 4**

X3=

	0	1	2	k=3	4	5	6
Coefficient	35	46	15				
Exponent	7	5	3				

Polynomial Multiplication

$X1 = 7x^4 + 5x^2 + 3x^1$

$X2 = 5x^3 + 3x^1$

X1=

	0	1	i= 2
Coefficient	7	5	3
Exponent	4	2	1

X2=

	0	j=1
Coefficient	5	3
Exponent	3	1

Coefficient **3 * 5 =**

Exponent **1 + 3 =**

X3=

	0	1	2	k=3	4	5	6
Coefficient	35	46	15	15			
Exponent	7	5	4	3			

Polynomial Multiplication

$X1 = 7x^4 + 5x^2 + 3x^1$

$X2 = 5x^3 + 3x^1$

X1=

	0	1	i= 2
Coefficient	7	5	3
Exponent	4	2	1

X2=

	0	j= 1
Coefficient	5	3
Exponent	3	1

Coefficient **3 * 3 = 9**

Exponent **1 + 1 = 2**

X3=

	0	1	2	3	k= 4	5	6
Coefficient	35	46	15	15			
Exponent	7	5	4	3			

Polynomial Multiplication

$X1 = 7x^4 + 5x^2 + 3x^1$

$X2 = 5x^3 + 3x^1$

X1=

	0	1	i= 2
Coefficient	7	5	3
Exponent	4	2	1

X2=

	0	j=1
Coefficient	5	3
Exponent	3	1

Coefficient * = 9

Exponent + = 2

X3=

	0	1	2	3	k=4	5	6
Coefficient	35	46	15	15	9		
Exponent	7	5	4	3	2		

3.12.6 Multiplication of Two Polynomials

Q. Write pseudo C/C++ code to perform polynomial multiplication using arrays.

SPPU - Dec. 16, 6 Marks

First polynomial P1 $5 + 9x + 3x^4 + 2x^6$

Second polynomial P2 $6 + 3x + 2x^2$

Step 1 :

Multiplying P1 with the first term of P2

$30 + 54x + 18x^4 + 12x^6$ (say Ptemp)

Ptemp is added to the final polynomial say (P3);

Value of P3 after step 1 = $30 + 54x + 18x^4 + 12x^6$

Step 2 :

Multiplying P1 with the second term of P2

$15x + 27x^2 + 9x^5 + 6x^7$ (say Ptemp)

Ptemp is added to the final polynomial P3

Value of P3 after step2

$= 30 + 69x + 27x^2 + 18x^4 + 9x^5 + 12x^6 + 6x^7$

Step 3 :

Multiplying P1 with the third term of P2.

$10x^2 + 18x^3 + 6x^6 + 4x^8$ (say Ptemp)

Ptemp is added to the final polynomial P3 value of P3 after step 3.

$= 30 + 69x + 37x^2 + 18x^3 + 18x^4 + 9x^5 + 18x^6 + 6x^7 + 4x^8$

'C' function for multiplication of two polynomials

```
Polynomial multiply(Polynomial *P1, Polynomial *P2)
{
    Polynomial P3, Ptemp;
    term t;
    int i;
    init(&P3)
    for(i = 0; i < P2 → n; i++)
    {
        init(&Ptemp)
        for(j = 0; j < P1 → n; j++)
        {
            t·power = P1→a[j]·power + P2→a[i]·power;
            t·coeff = P1→a[j]·coeff * P2→a[i]·coeff;
            insert(&Ptemp, t);
        }
        P3 = add(&P3, &Ptemp);
    }
    return(P3);
}
```

```
while (j < N)
{
 NewTerm_exp = X1.Poly[i].Exp + X2.Poly[j].Exp;
 NewTerm_coef = X1.Poly[i].Coef * X2.Poly[j].Coef;
 TmpIndex = 0;
 flag = 0;
  while(TmpIndex < k)
  // Insert NewTerm in Polynomial C
 {
  if(X3.Poly[TmpIndex].Exp == NewTerm_exp)
   // search matching exponent
  {
   flag = 1;
   break;
  }
 else if(X3.Poly[TmpIndex].Exp < NewTerm_exp)
 break;
 TmpIndex++;
 }
```

```
if(flag)    // if found add coefficients
 X3.Poly[k].Coef = X3[k].Coef + NewTerm_coef;
    else  // else add at last location or in between
    {
     if(TmpIndex==k) // add new term at end
     {
      X3.Poly [k].Exp = NewTerm_exp;
      X3.Poly [k].Coef = NewTerm_coef;
      k++;
     }
     else
     {
      // insert new term
      for(p = k; p < TmpIndex; p--)
      {
       X3.Poly [p].Exp = X3.Poly[p].Exp;
       X3.Poly [k].Coef = X3.Poly[p].Coef;
      }
      X3.Poly[TmpIndex].Coef = NewTerm_exp;
      X3.Poly[TmpIndex].Coef = NewTerm_Coef;
      k++;          }
      j++;        }
    i++;     }
 return(C);   }
```

SPARSE MATRIX

In many situations, matrix size is very large but out of it, most of the elements are zeros (not necessarily always zeros).

And only a small fraction of the matrix is actually used. A matrix of such type is called a sparse matrix,

2.9.1 Sparse Matrix Representation

Array Representation of Sparse Matrix

- 2D array is used to represent a sparse matrix in which there are three rows named as
- **Row** : Index of row, where non-zero element is located.
- **Column** : Index of column, where non-zero element is located

SPARSE MATRIX

– **Value :** Value of the non zero element located at index – (row, column)

$$\begin{bmatrix} 0 & 0 & 3 & 0 & 4 \\ 0 & 0 & 5 & 7 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 2 & 6 & 0 & 0 \end{bmatrix} \Rightarrow$$

Row	0	0	1	1	3	3
Column	2	4	2	3	1	2
Value	3	4	5	7	2	6

Fig. 2.9.1 : Sparse Matrix

Matrix A =

0	0	0	0	0
0	1	0	0	0
0	0	9	8	0
0	3	0	0	0
0	0	0	5	4
0	0	2	3	0
0	0	0	0	0

Alternate Representation of Sparse Matrix A=

Rows	Columns	8 NonZero Entries
1	1	1
2	2	9
2	3	8
3	1	3
4	3	5
4	4	4
5	2	2
5	3	3

Sparse matrix and its representation

Transpose Of Sparse Matrix

- *Simple Transpose*
- *Fast Transpose*

Transpose Of Sparse Matrix

- To transpose a matrix, we must interchange the rows and columns. This means that if an element is at position [j][k] in the original matrix, then it is at position [k][j] in the transposed matrix.
- When k = j, the elements on the diagonal will remain unchanged. Since the original matrix is organized by rows, our first idea for a transpose algorithm might be the following:

for (each row k)

take element (k,j, value) and

store it in (j,k, value) of the transpose;

For example

Sp1

Index	Row	Column	value
0	3	4	4
1	1	0	5
2	1	2	3
3	2	1	1
4	2	3	2

Fig. 2.9.2(a) : Spares Matrix

Sp2

Index	Row	Column	value
0	4	3	4
1	0	1	5
2	1	2	1
3	2	1	3
4	3	2	5

Fig. 2.9.2(b) : Transpose Matrix

Transpose Of Sparse Matrix

Step 1 : copy elements of column 0 of sp1 to sp2.

sp1

Row	Column	Value
3	4	4
1	[0]	5
1	2	3
2	1	1
2	3	2

element of column 0

sp2

Row	Column	Value
4	3	4
0	1	5

Step 2 : copy elements of column 1 of sp1 to sp2.

sp1

Row	Column	Value
3	4	4
1	0	5
1	2	3
2	[1]	1
2	3	2

element of column 0

sp2

Row	Column	Value
4	3	4
0	1	5
1	2	1

Step 3 : copy elements of column 2 of sp1 to sp2.

sp1

Row	Column	Value
3	4	4
1	0	5
1	[2]	3
2	1	1
2	3	2

element of column 0

sp2

Row	Column	Value
4	3	4
0	1	5
1	2	1
2	1	3

Time complexity of manual technique is O(mn).

$$\begin{pmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \end{pmatrix} \qquad \begin{pmatrix} 1 & 5 & 9 \\ 2 & 6 & 10 \\ 3 & 7 & 11 \\ 4 & 8 & 12 \end{pmatrix}$$

Sparse matrix transpose

transpose

Sparse Matrix and its Transpose

Let B=

3 x 4

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

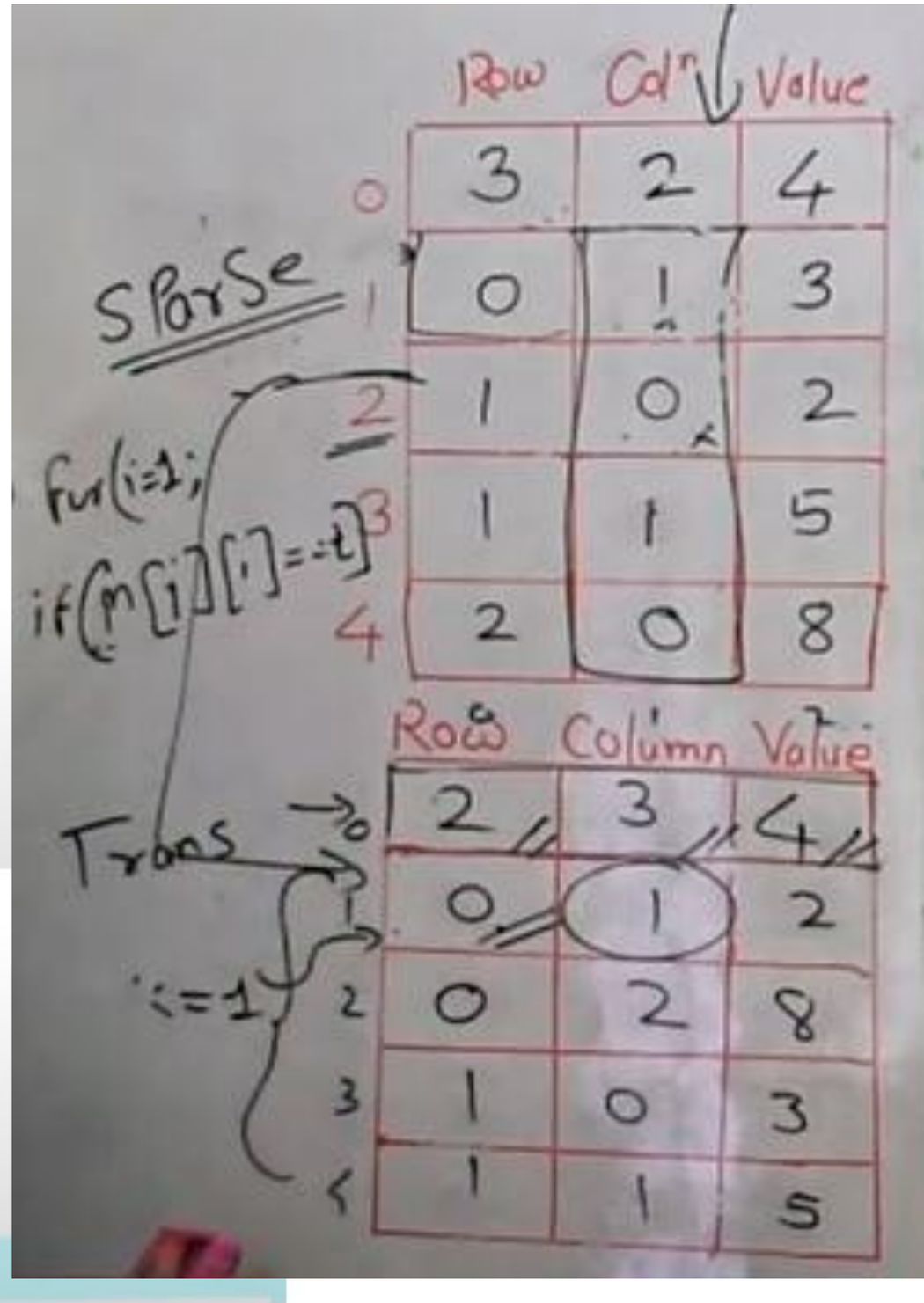
B^T=

4 x 3

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

Pseudocode to perform the simple transpose of sparse matrix

```
void SparseMatrix::transpose()
{
    int transpose[100][3],k=1;
    for(int z=0;z<sparse[0][1];z++)
    {
        for(int i=1; i<=sparse[0][2];i++)
        {
            if(sparse[i][1] == z)
            {
                transpose[k][0] = sparse[i][1];
                transpose[k][1] = sparse[i][0];
                transpose[k][2] = sparse[i][2];
                k++;
            }
                transpose[0][0] = sparse[0][1];
                transpose[0][1] = sparse[0][0];
                transpose[0][2] = sparse[0][2];

        }
}
```



Simple Sparse matrix transpose

- *Time complexity will be O (n. T)*

$$= O(n \cdot mn)$$

$$= O(mn^2)$$

which is worst than *the conventional transpose with time complexity O (mn)*

Fast Transpose

- It is very fast but little complex method to find out transpose of sparse matrix
- Time complexity is O(no. of columns + no. on non-zero values)

Limitation of Simple transpose

- The computing time of simple transpose is O(terms*columns).
- Without triplet representation original matrix transpose requires O(rows*columns).
- It the terms is of the order of rows*columns then the triplet form will require O(rows*column^2) which is worst then the regular representation.
- Hence for better time efficiency of triplet representation fast transpose algorithm is used.
- The time complexity of fast transpose is O(terms + columns).

Fast Transpose of Sparse matrix

Fast transpose uses two additional arrays to store the count of occurrence of each column number and the start position of each row.

Position of a column (converted to row) is calculated in the transposed matrix and then it is placed.

The time complexity of fast transpose is O(terms + column) hence it is efficient than simple transpose.

Fast Transpose

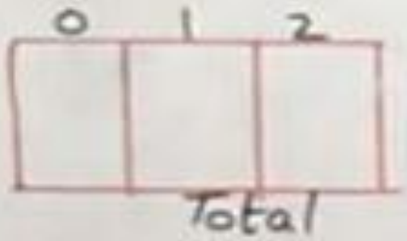
- It is very fast but little complex method to find out transpose of sparse matrix
- Time complexity is O(no. of columns + no. on non-zero values)

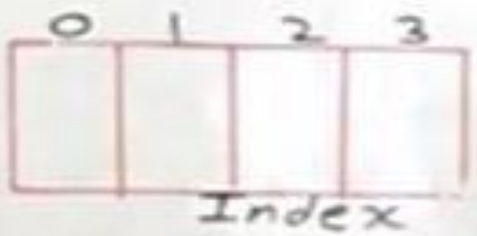
Steps:

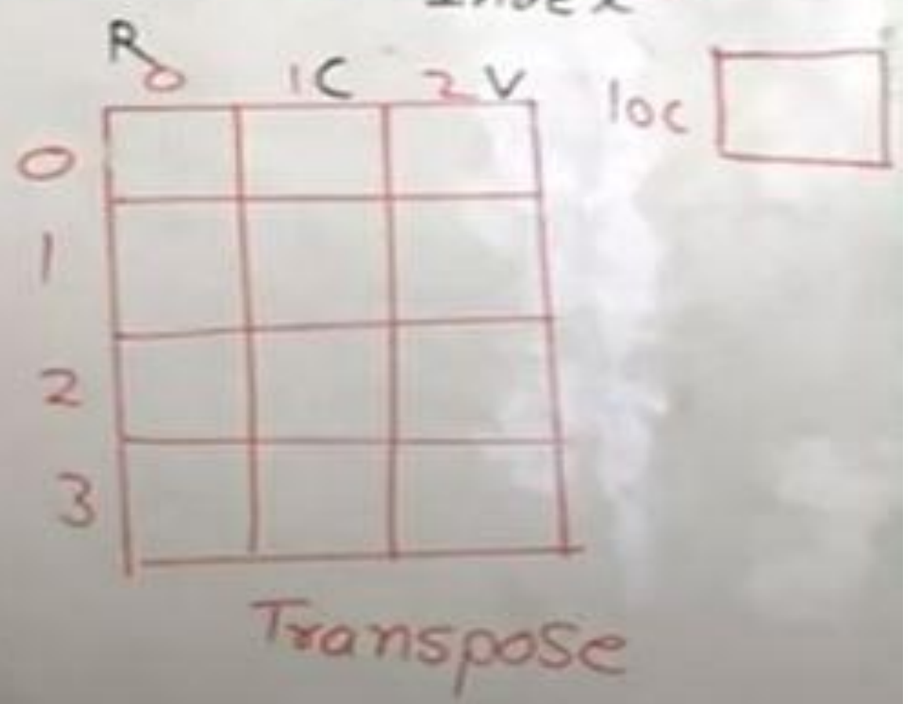
1. Obtained the triplet form of sparse matrix
2. Create 2 one dimensional arrays : total_array and index_array
3. Size of total array is the total number of columns in the original matrix
4. At every index of total, put the number of times respective index appear in second column of sparse matrix
5. Size of index array : size of total array + 1
6. Assign index[0] = 1 and index[i] = index[i-1] + total[i-1]
7. Now traverse the sparse matrix from second row and consider column elements
8. Location = index[Col_No]
9. $location^{th}$ index of transpose matrix = swapped row from original matrix.
10. Increase index[Col_No] by 1
11. Repeat steps from 7 to 10 for the remaining triplets of original matrix

Sparse

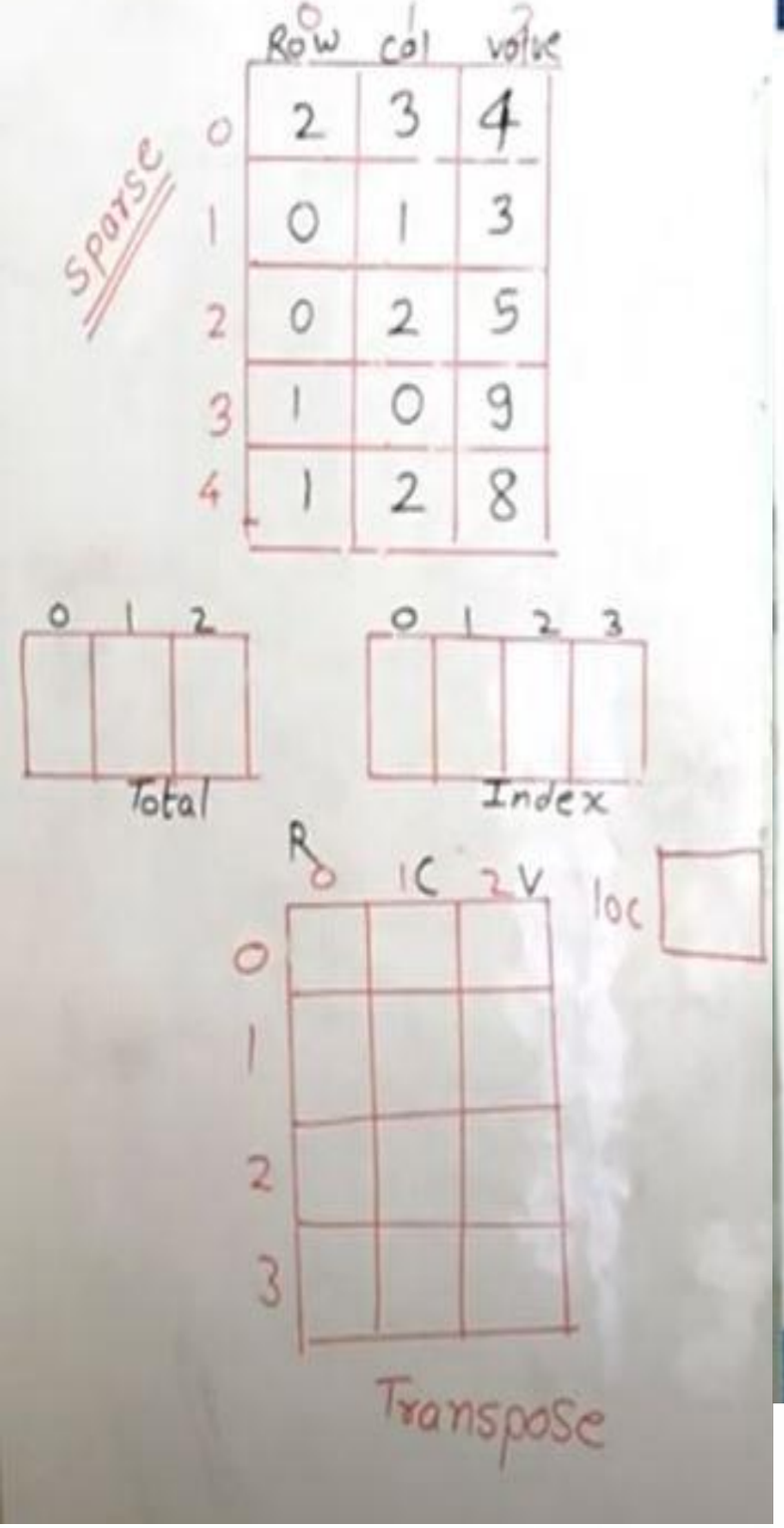
	Row	col	value
0	2	3	4
1	0	1	3
2	0	2	5
3	1	0	9
4	1	2	8







Transpose

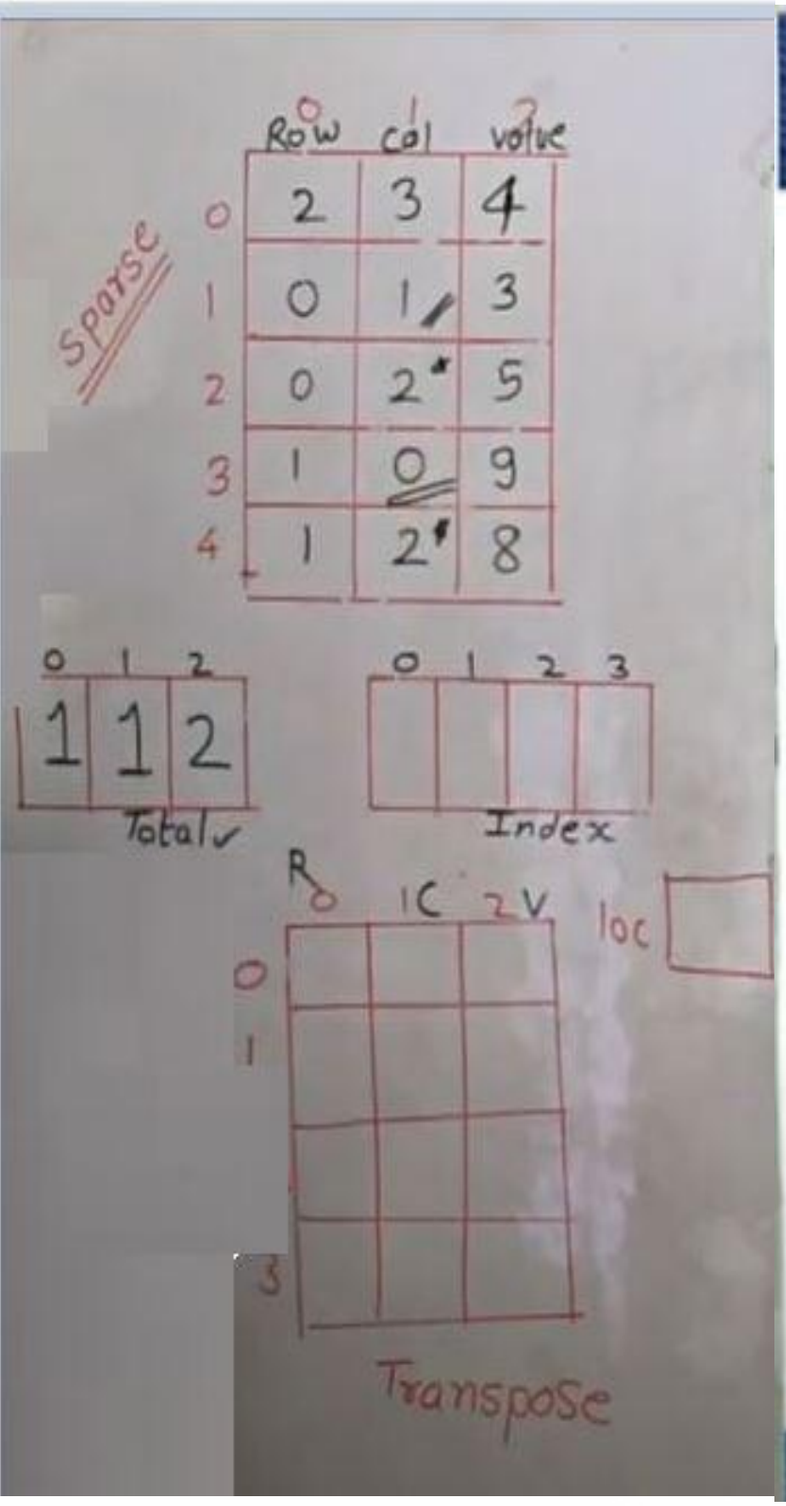


Fast Transpose

- It is very fast but little complex method to find out transpose of sparse matrix
- Time complexity is O(no. of columns + no. on non-zero values)

Steps:

1. Obtained the triplet form of sparse matrix
2. Create 2 one dimensional arrays : total_array and index_array
3. Size of total array is the total number of columns in the original matrix
4. At every index of total, put the number of times respective index appear in secon column of sparse matrix
5. Size of index array : size of total array + 1
6. Assign index[0] = 1 and index[i] = index[i-1] + total[i-1]
7. Now traverse the sparse matrix from second row and consider column elements
8. Location = index[Col_No]
9. $location^{th}$ index of transpose matrix = swapped row from original matrix.
10. Increase index[Col_No] by 1
11. Repeat steps from 7 to 10 for the remaining triplets of original matrix



Fast Transpose

- It is very fast but little complex method to find out transpose of sparse matrix
- Time complexity is O(no. of columns + no. on non-zero values)

Steps:

1. Obtained the triplet form of sparse matrix
2. Create 2 one dimensional arrays : total_array and index_array
3. Size of total array is the total number of columns in the original matrix
4. At every index of total, put the number of times respective index appear in second column of sparse matrix
5. Size of index array : size of total array + 1
6. Assign index[0] = 1 and index[i] = index[i-1] + total[i-1]
7. Now traverse the sparse matrix from second row and consider column elements
8. Location = index[Col_No]
9. $location^{th}$ index of transpose matrix = swapped row from original matrix.
10. Increase index[Col_No] by 1
11. Repeat steps from 7 to 10 for the remaining triplets of original matrix

	Row	col	value
0	2	3	4
1	0	1	3
2	0	2	5
3	1	0	9
4	1	2	8

0	1	2
1	2	

Total

0	1	2	3
1	2	3	5

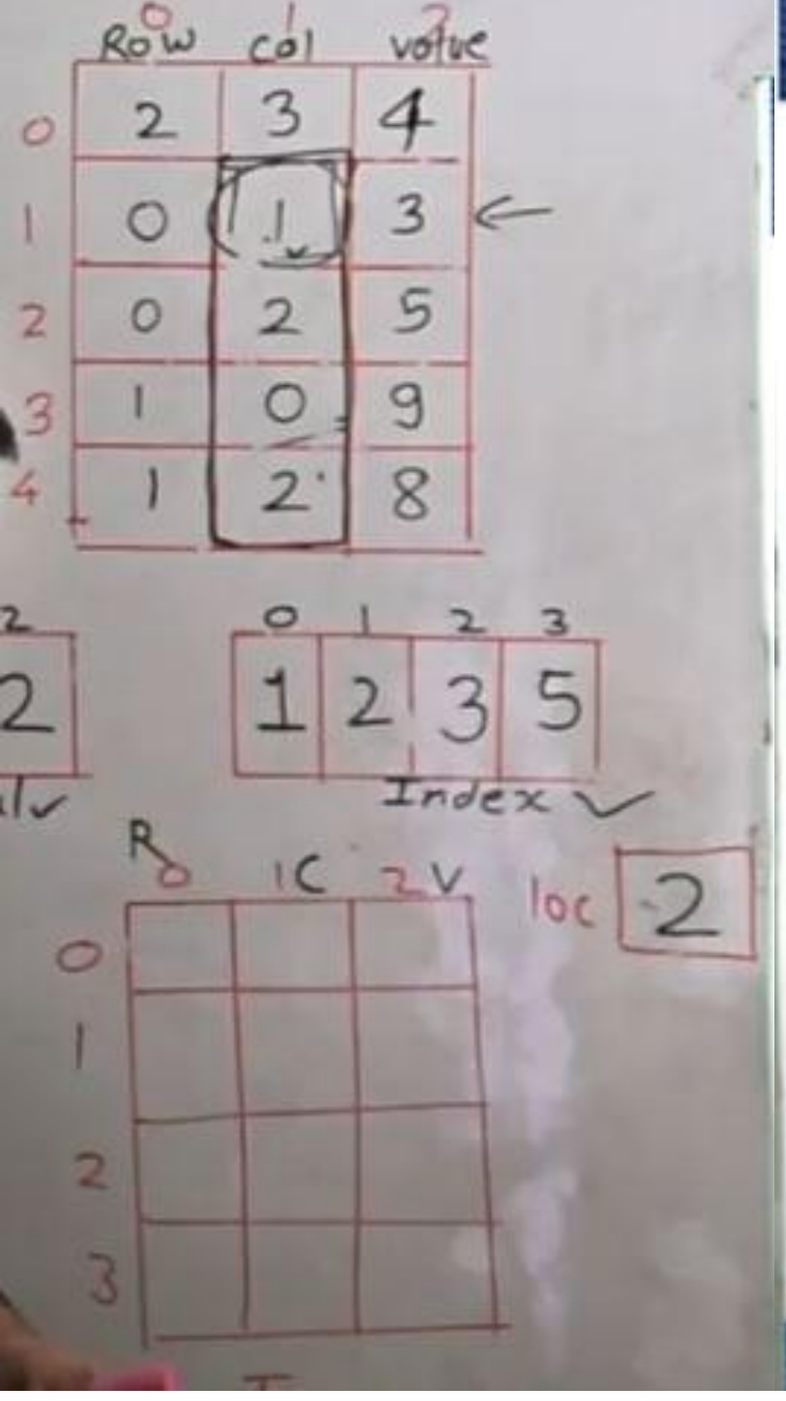
Index

Fast Transpose

- It is very fast but little complex method to find out transpose of sparse matrix
- Time complexity is O(no. of columns + no. on non-zero values)

Steps:

1. Obtained the triplet form of sparse matrix
2. Create 2 one dimensional arrays : total_array and index_array
3. Size of total array is the total number of columns in the original matrix
4. At every index of total, put the number of times respective index appear in second column of sparse matrix
5. Size of index array : size of total array + 1
6. Assign index[0] = 1 and index[i] = index[i-1] + total[i-1]
7. Now traverse the sparse matrix from second row and consider column elements
8. Location = index[Col_No]
9. $location^{th}$ index of transpose matrix = swapped row from original matrix.
10. Increase index[Col_No] by 1
11. Repeat steps from 7 to 10 for the remaining triplets of original matrix

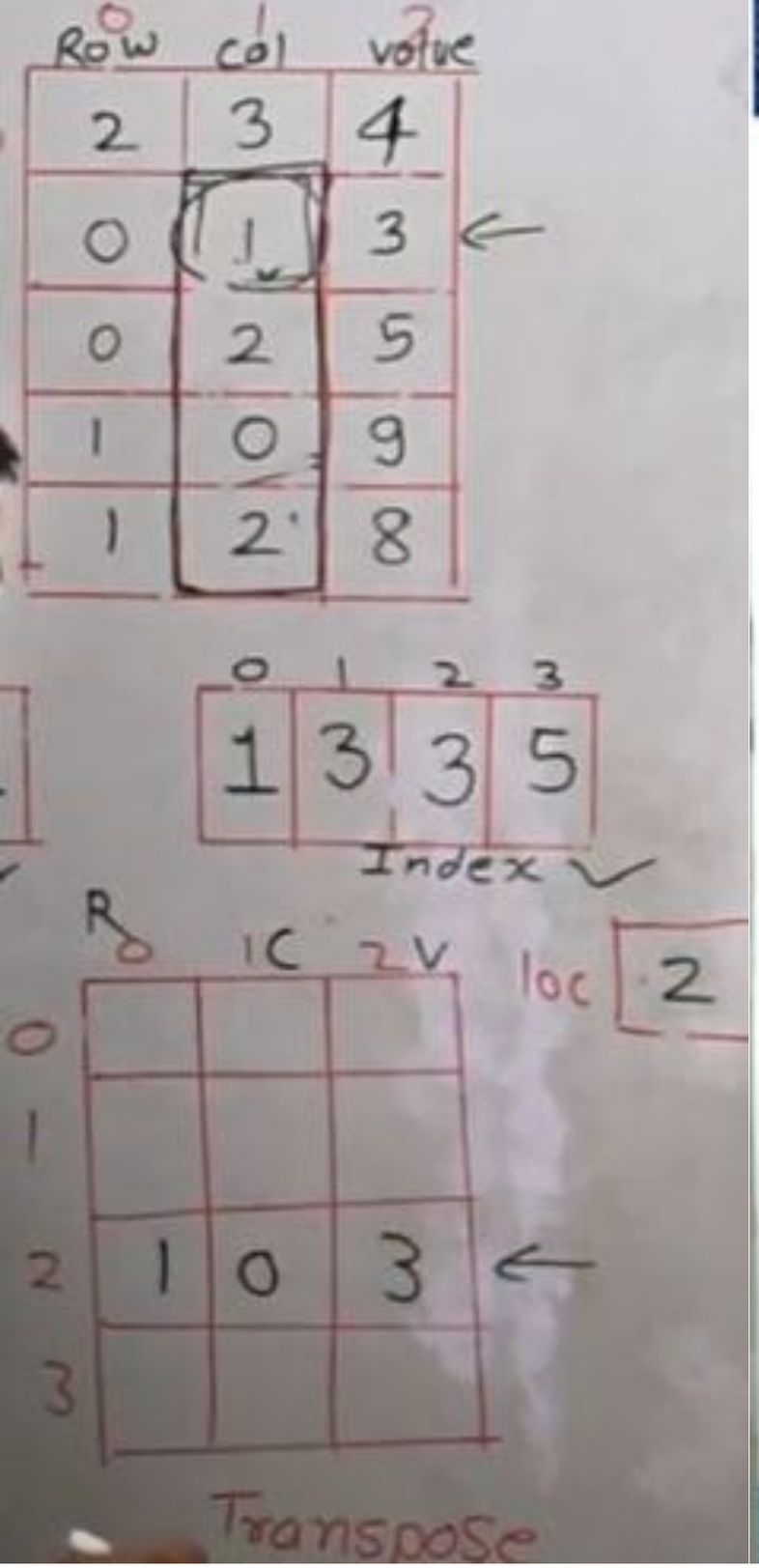


Fast Transpose

- It is very fast but little complex method to find out transpose of sparse matrix
- Time complexity is O(no. of columns + no. on non-zero values)

Steps:

1. Obtained the triplet form of sparse matrix
2. Create 2 one dimensional arrays : total_array and index_array
3. Size of total array is the total number of columns in the original matrix
4. At every index of total, put the number of times respective index appear in second column of sparse matrix
5. Size of index array : size of total array + 1
6. Assign index[0] = 1 and index[i] = index[i-1] + total[i-1]
7. Now traverse the sparse matrix from second row and consider column elements
8. Location = index[Col_No]
9. $location^{th}$ index of transpose matrix = swapped row from original matrix.
10. Increase index[Col_No] by 1
11. Repeat steps from 7 to 10 for the remaining triplets of original matrix

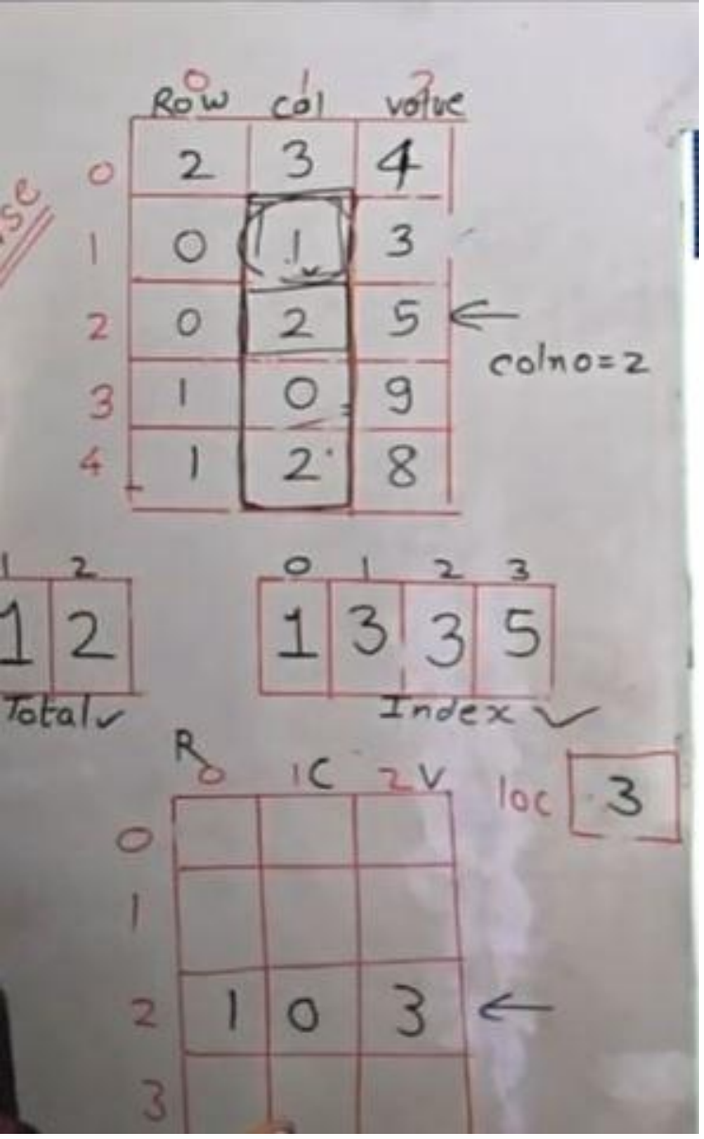


Fast Transpose

- It is very fast but little complex method to find out transpose of sparse matrix
- Time complexity is O(no. of columns + no. on non-zero values)

Steps:

1. Obtained the triplet form of sparse matrix
2. Create 2 one dimensional arrays : total_array and index_array
3. Size of total array is the total number of columns in the original matrix
4. At every index of total, put the number of times respective index appear in second column of sparse matrix
5. Size of index array : size of total array + 1
6. Assign index[0] = 1 and index[i] = index[i-1] + total[i-1]
7. Now traverse the sparse matrix from second row and consider column elements
8. Location = index[Col_No]
9. $location^{th}$ index of transpose matrix = swapped row from original matrix.
10. Increase index[Col_No] by 1
11. Repeat steps from 7 to 10 for the remaining triplets of original matrix

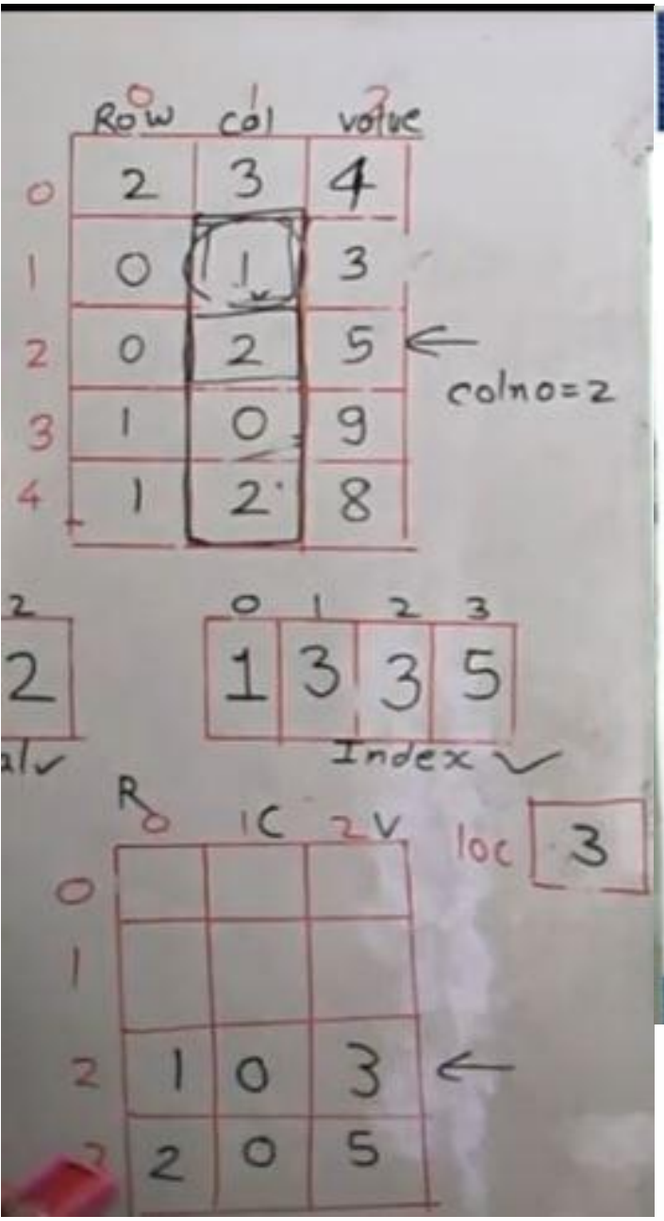


Fast Transpose

- It is very fast but little complex method to find out transpose of sparse matrix
- Time complexity is O(no. of columns + no. on non-zero values)

Steps:

1. Obtained the triplet form of sparse matrix
2. Create 2 one dimensional arrays : total_array and index_array
3. Size of total array is the total number of columns in the original matrix
4. At every index of total, put the number of times respective index appear in second column of sparse matrix
5. Size of index array : size of total array + 1
6. Assign index[0] = 1 and index[i] = index[i-1] + total[i-1]
7. Now traverse the sparse matrix from second row and consider column elements
8. Location = index[Col_No]
9. $location^{th}$ index of transpose matrix = swapped row from original matrix.
10. Increase index[Col_No] by 1
11. Repeat steps from 7 to 10 for the remaining triplets of original matrix

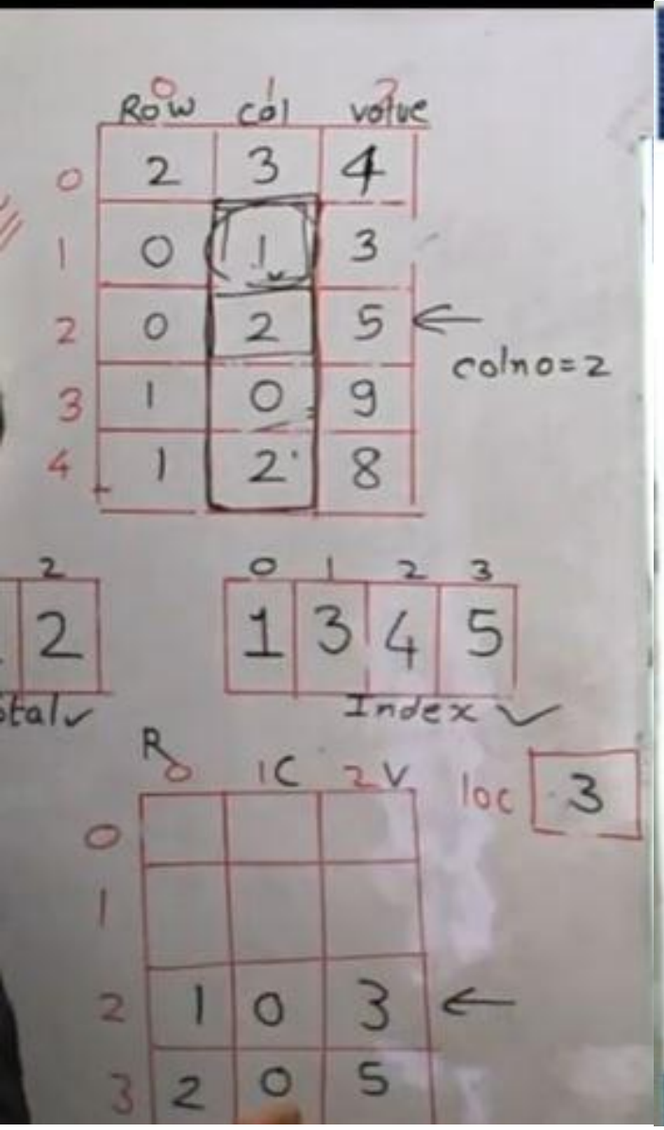


Fast Transpose

- It is very fast but little complex method to find out transpose of sparse matrix
- Time complexity is O(no. of columns + no. on non-zero values)

Steps:

1. Obtained the triplet form of sparse matrix
2. Create 2 one dimensional arrays : total_array and index_array
3. Size of total array is the total number of columns in the original matrix
4. At every index of total, put the number of times respective index appear in second column of sparse matrix
5. Size of index array : size of total array + 1
6. Assign index[0] = 1 and index[i] = index[i-1] + total[i-1]
7. Now traverse the sparse matrix from second row and consider column elements
8. Location = index[Col_No]
9. $location^{th}$ index of transpose matrix = swapped row from original matrix.
10. Increase index[Col_No] by 1
11. Repeat steps from 7 to 10 for the remaining triplets of original matrix

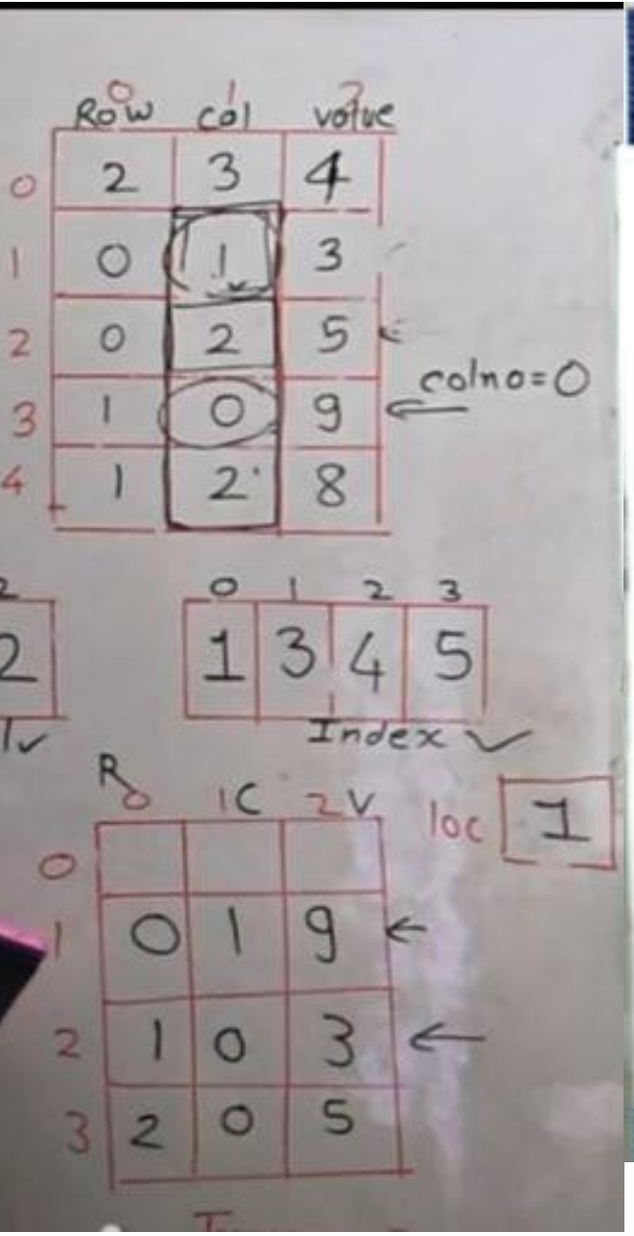


Fast Transpose

- It is very fast but little complex method to find out transpose of sparse matrix
- Time complexity is O(no. of columns + no. on non-zero values)

Steps:

1. Obtained the triplet form of sparse matrix
2. Create 2 one dimensional arrays : total_array and index_array
3. Size of total array is the total number of columns in the original matrix
4. At every index of total, put the number of times respective index appear in second column of sparse matrix
5. Size of index array : size of total array + 1
6. Assign index[0] = 1 and index[i] = index[i-1] + total[i-1]
7. Now traverse the sparse matrix from second row and consider column elements
8. Location = index[Col_No]
9. $location^{th}$ index of transpose matrix = swapped row from original matrix.
10. Increase index[Col_No] by 1
11. Repeat steps from 7 to 10 for the remaining triplets of original matrix

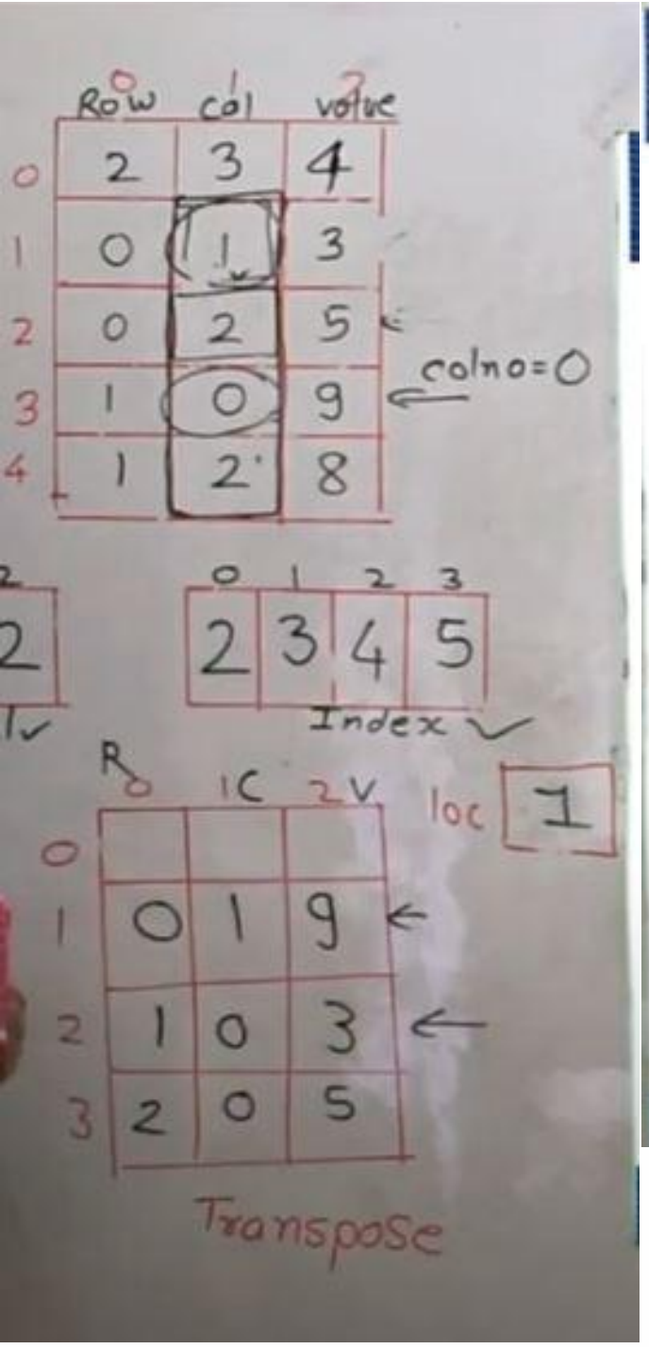


Fast Transpose

- It is very fast but little complex method to find out transpose of sparse matrix
- Time complexity is O(no. of columns + no. on non-zero values)

Steps:

1. Obtained the triplet form of sparse matrix
2. Create 2 one dimensional arrays : total_array and index_array
3. Size of total array is the total number of columns in the original matrix
4. At every index of total, put the number of times respective index appear in second column of sparse matrix
5. Size of index array : size of total array + 1
6. Assign index[0] = 1 and index[i] = index[i-1] + total[i-1]
7. Now traverse the sparse matrix from second row and consider column elements
8. Location = index[Col_No]
9. $location^{th}$ index of transpose matrix = swapped row from original matrix.
10. Increase index[Col_No] by 1
11. Repeat steps from 7 to 10 for the remaining triplets of original matrix

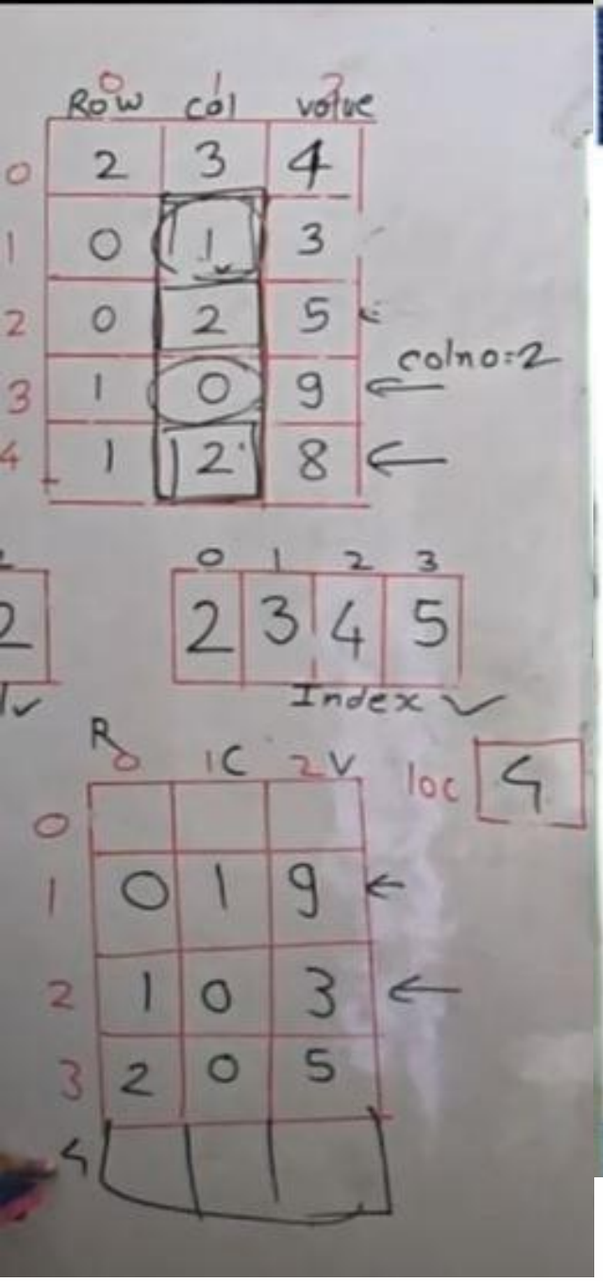


Fast Transpose

- It is very fast but little complex method to find out transpose of sparse matrix
- Time complexity is O(no. of columns + no. on non-zero values)

Steps:

1. Obtained the triplet form of sparse matrix
2. Create 2 one dimensional arrays : total_array and index_array
3. Size of total array is the total number of columns in the original matrix
4. At every index of total, put the number of times respective index appear in second column of sparse matrix
5. Size of index array : size of total array + 1
6. Assign index[0] = 1 and index[i] = index[i-1] + total[i-1]
7. Now traverse the sparse matrix from second row and consider column elements
8. Location = index[Col_No]
9. $location^{th}$ index of transpose matrix = swapped row from original matrix.
10. Increase index[Col_No] by 1
11. Repeat steps from 7 to 10 for the remaining triplets of original matrix

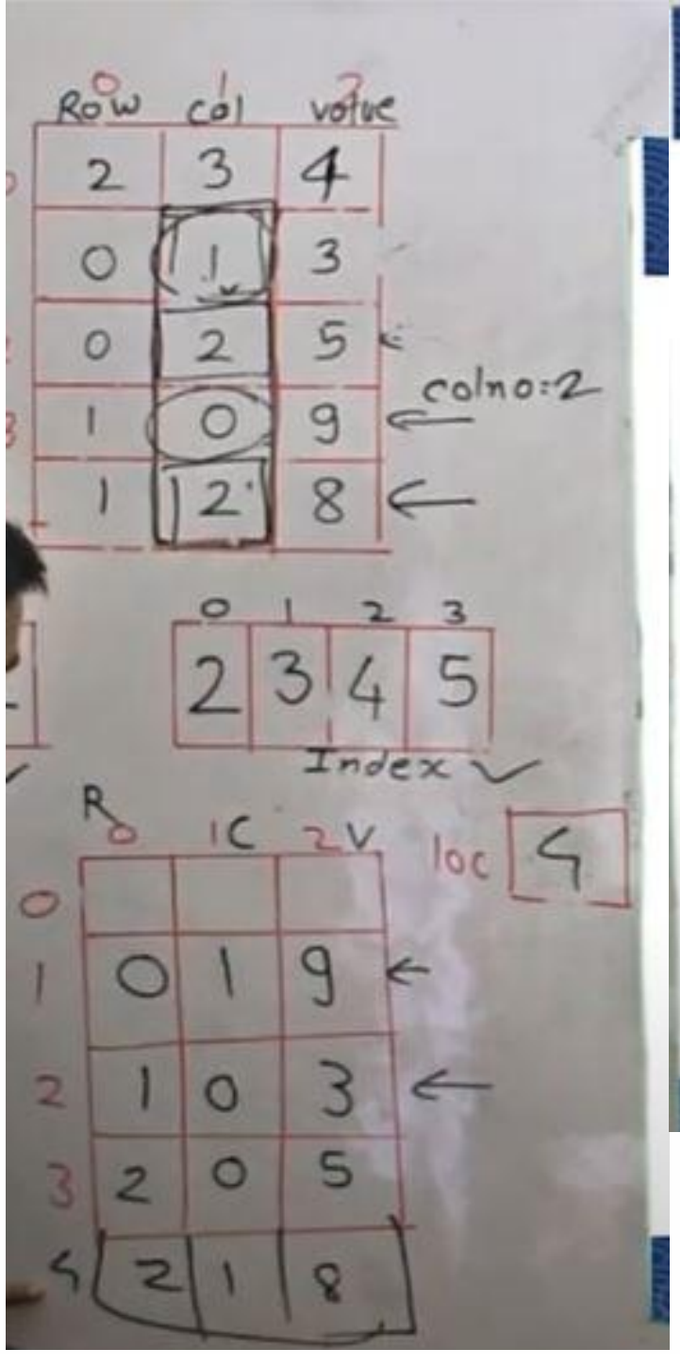


Fast Transpose

- It is very fast but little complex method to find out transpose of sparse matrix
- Time complexity is O(no. of columns + no. on non-zero values)

Steps:

1. Obtained the triplet form of sparse matrix
2. Create 2 one dimensional arrays : total_array and index_array
3. Size of total array is the total number of columns in the original matrix
4. At every index of total, put the number of times respective index appear in second column of sparse matrix
5. Size of index array : size of total array + 1
6. Assign index[0] = 1 and index[i] = index[i-1] + total[i-1]
7. Now traverse the sparse matrix from second row and consider column elements
8. Location = index[Col_No]
9. $location^{th}$ index of transpose matrix = swapped row from original matrix.
10. Increase index[Col_No] by 1
11. Repeat steps from 7 to 10 for the remaining triplets of original matrix

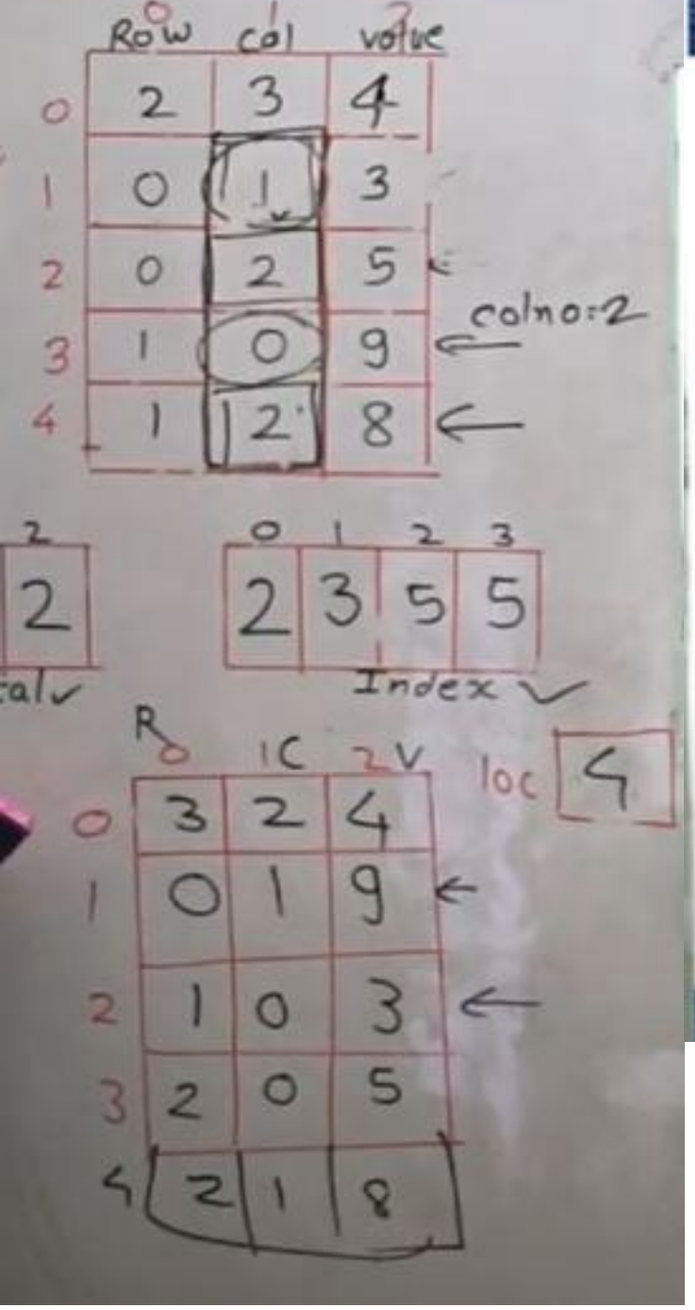


Fast Transpose

- It is very fast but little complex method to find out transpose of sparse matrix
- Time complexity is O(no. of columns + no. on non-zero values)

Steps:

1. Obtained the triplet form of sparse matrix
2. Create 2 one dimensional arrays : total_array and index_array
3. Size of total array is the total number of columns in the original matrix
4. At every index of total, put the number of times respective index appear in second column of sparse matrix
5. Size of index array : size of total array + 1
6. Assign index[0] = 1 and index[i] = index[i-1] + total[i-1]
7. Now traverse the sparse matrix from second row and consider column elements
8. Location = index[Col_No]
9. $location^{th}$ index of transpose matrix = swapped row from original matrix.
10. Increase index[Col_No] by 1
11. Repeat steps from 7 to 10 for the remaining triplets of original matrix



Fast Transpose

- It is very fast but little complex method to find out transpose of sparse matrix
- Time complexity is O(no. of columns + no. on non-zero values)

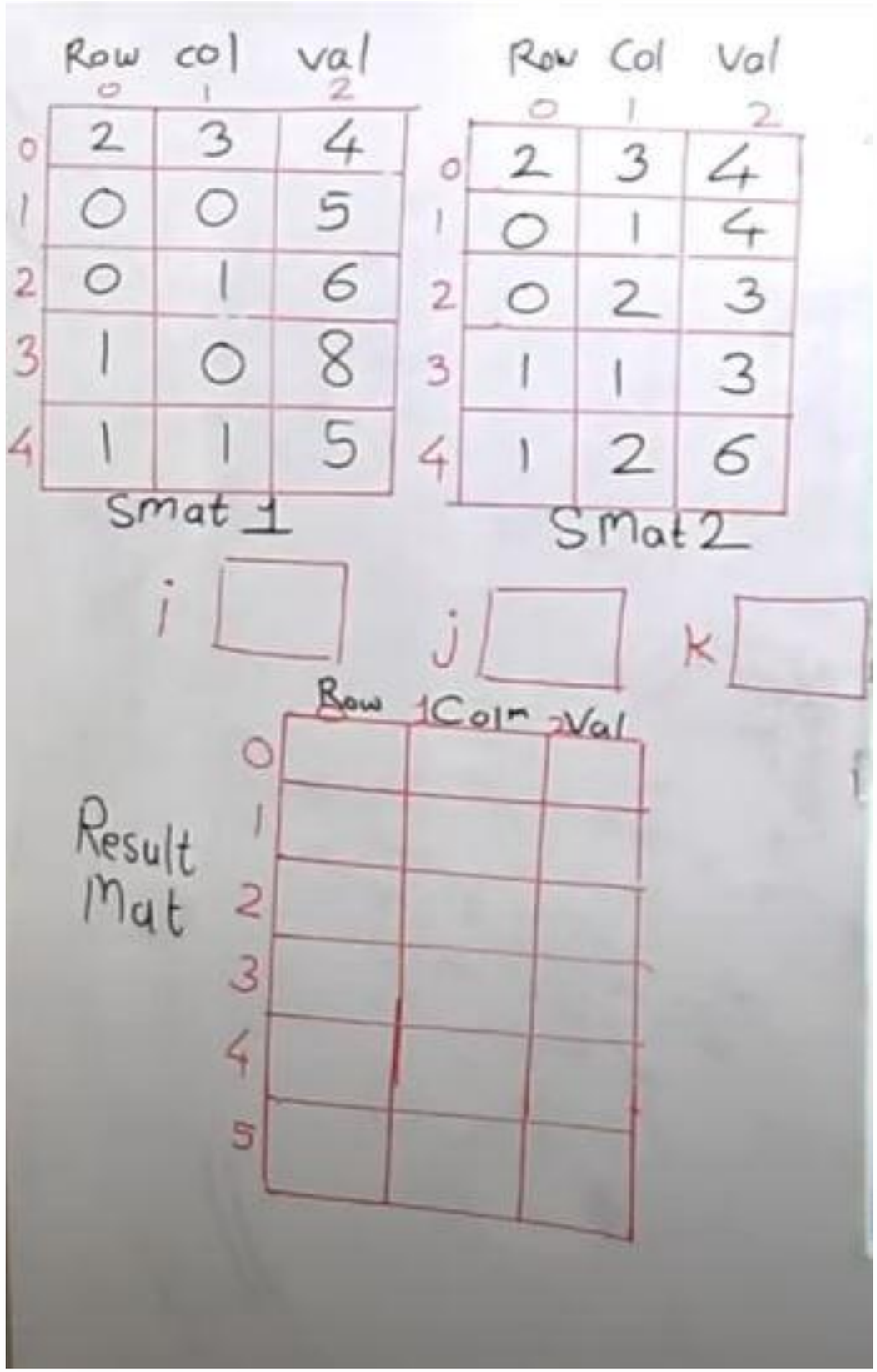
Steps:

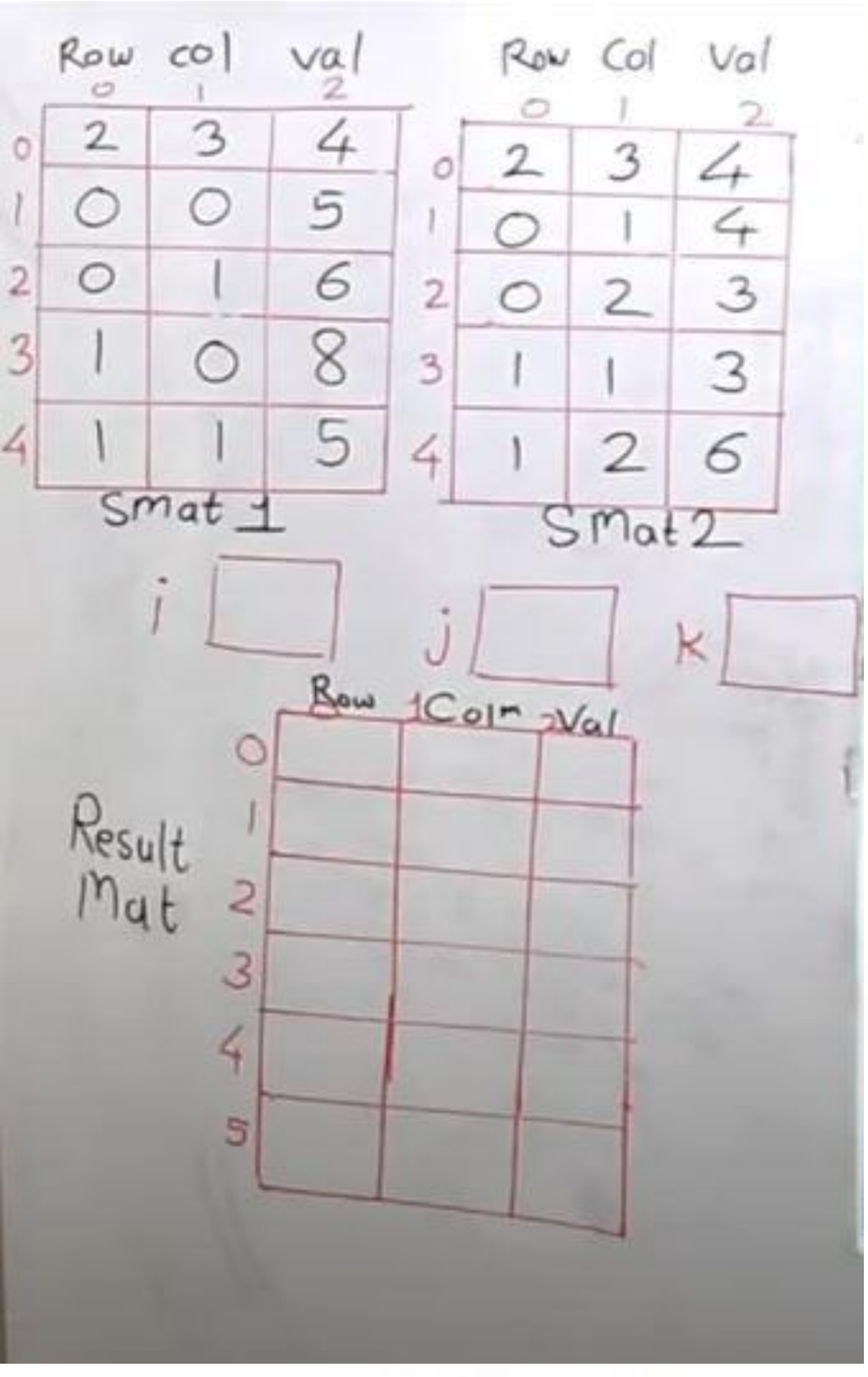
1. Obtained the triplet form of sparse matrix
2. Create 2 one dimensional arrays : total_array and index_array
3. Size of total array is the total number of columns in the original matrix
4. At every index of total, put the number of times respective index appear in second column of sparse matrix
5. Size of index array : size of total array + 1
6. Assign index[0] = 1 and index[i] = index[i-1] + total[i-1]
7. Now traverse the sparse matrix from second row and consider column elements
8. Location = index[Col_No]
9. $location^{th}$ index of transpose matrix = swapped row from original matrix.
10. Increase index[Col_No] by 1
11. Repeat steps from 7 to 10 for the remaining triplets of original matrix

Addition of Sparse Matrices

Steps:

1. Obtained the triplet form of both sparse matrices
2. Create new triplet to store result as result matrix
3. Copy number of rows and column from any sparse matrix to result matrix
4. Let i,j,k be the indices of sparse matrices 1,2,3 respectively
5. Initialize i,j,k to 1
6. Traverse both matrices from second row


Row col val
0 1 2
0 2 3 4
1 0 0 5
2 0 1 6
3 1 0 8
4 1 1 5
Smat 1
Row Col Val
0 1 2
0 2 3 4
1 0 1 4
2 0 2 3
3 1 1 3
4 1 2 6
Smat 2
i
j
k
Row Colm Val
Result Mat
0
1
2
3
4
5



Addition of Sparse Matrices

Steps:

1. Obtained the triplet form of both sparse matrices
2. Create new triplet to store result as result matrix
3. Copy number of rows and column from any sparse matrix to result matrix
4. Let i,j,k be the indices of sparse matrices 1,2,3 respectively
5. Initialize i,j,k to 1
6. Traverse both matrices from second row

Addition of Sparse Matrices

Steps:

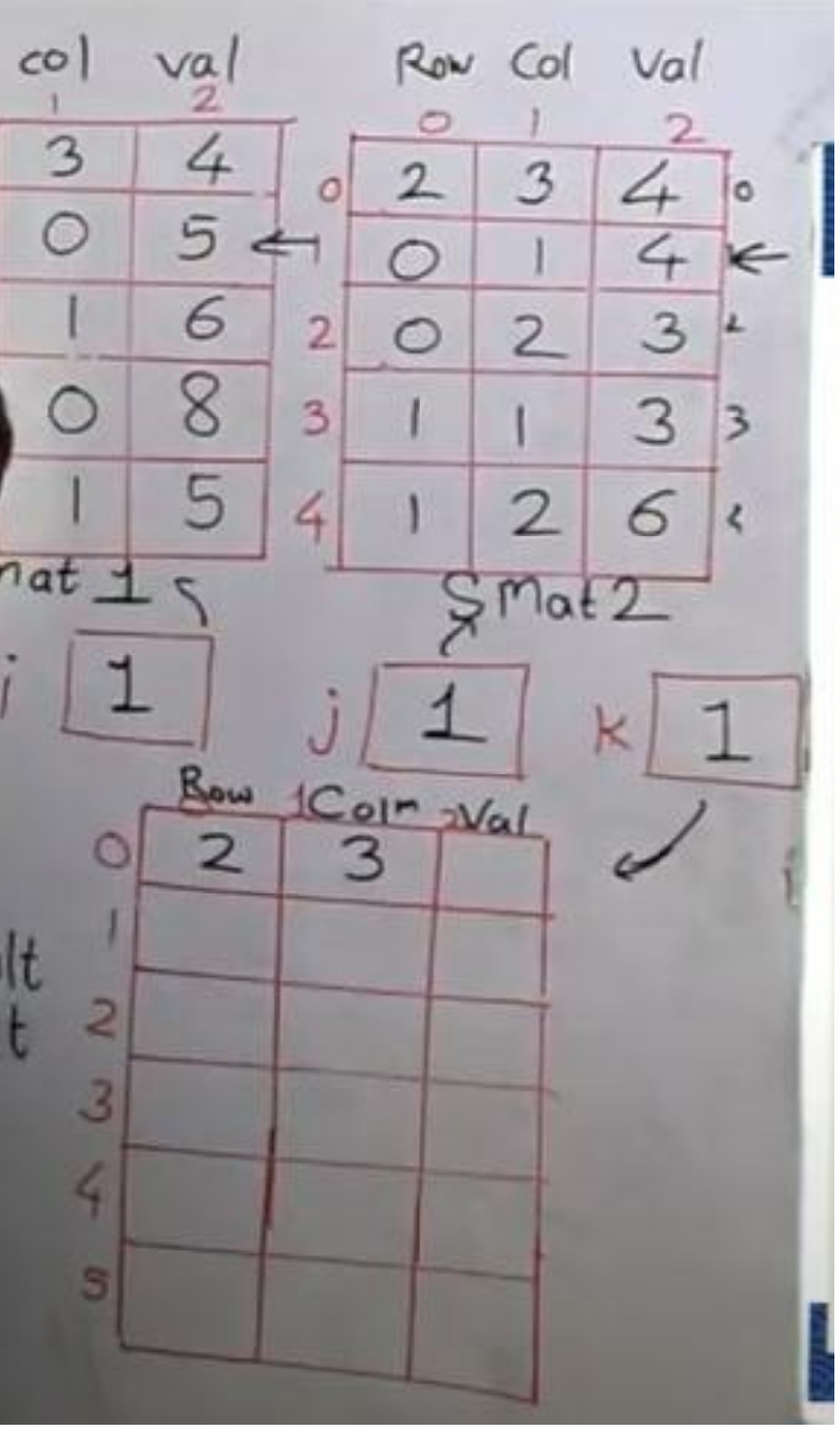
1. Obtained the triplet form of both sparse matrices
2. Create new triplet to store result as result matrix
3. Copy number of rows and column from any sparse matrix to result matrix
4. Let i,j,k be the indices of sparse matrices 1,2,3 respectively
5. Initialize i,j,k to 1
6. Traverse both matrices from second row

Addition of Sparse Matrices

Steps:

6. **If**(row number of matrix 1 == row number of matrix2)
   {
   if(column number of matrix 1 == column number of matrix2)
   make the addition of non zero values and store it into result matrix by incrementing all indices
   else
   which ever has less column value copy that to result matrix by incrementing respective indices
   }
   Else
   compare rows of both sparse matrices and which ever has less row value copy that to result matrix by incrementing respective indices

7. Repeat steps 6 till the end of any matrix's triplet
8. Copy the remaining term of sparse matrix (if any) to resultant matrix

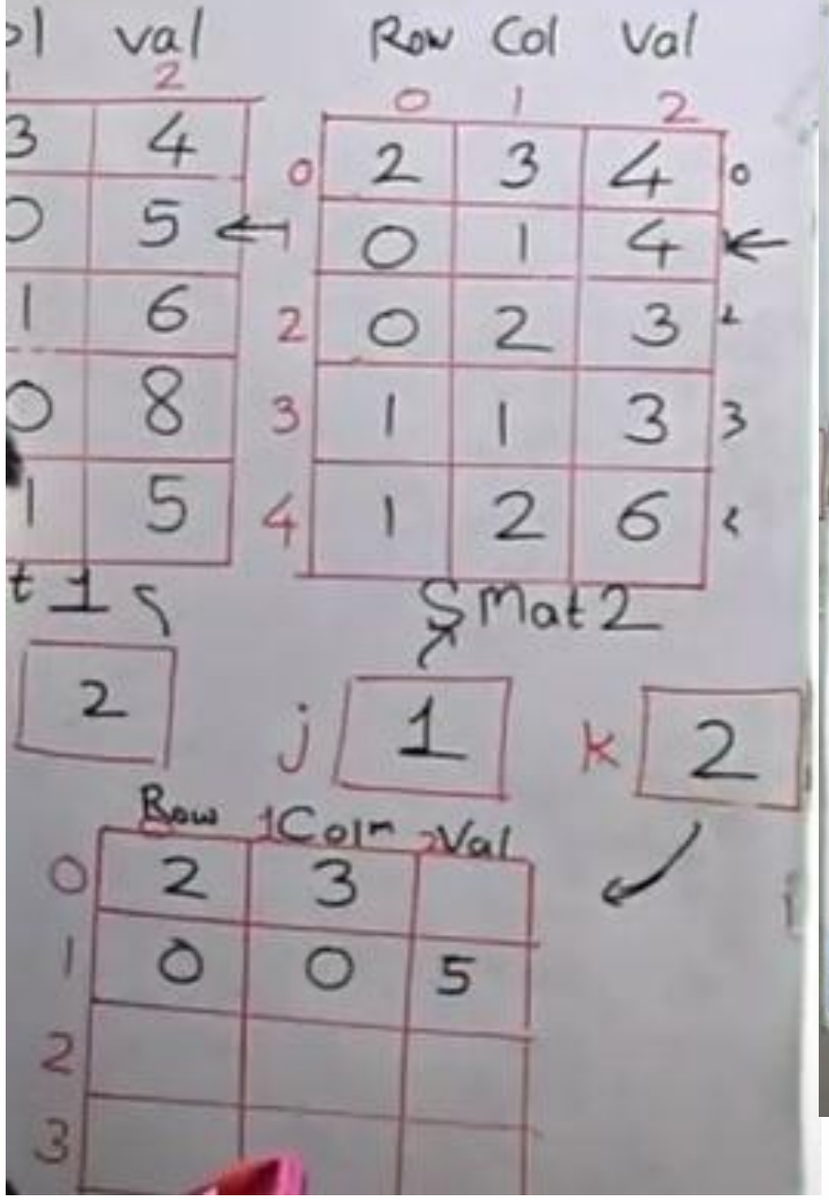


Addition of Sparse Matrices

Steps:

6. **If**(row number of matrix 1 == row number of matrix2)
   {
   if(column number of matrix 1 == column number of matrix2)
   make the addition of non zero values and store it into result matrix by incrementing all indices
   else
   which ever has less column value copy that to result matrix by incrementing respective indices
   }
   Else
   compare rows of both sparse matrices and which ever has less row value copy that to result matrix by incrementing respective indices

7. Repeat steps 6 till the end of any matrix's triplet
8. Copy the remaining term of sparse matrix (if any) to resultant matrix



Addition of Sparse Matrices

Steps:

6. **If**(row number of matrix 1 == row number of matrix2)
   {
   if(column number of matrix 1 == column number of matrix2)
   make the addition of non zero values and store it into result matrix by incrementing all indices
   else
   which ever has less column value copy that to result matrix by incrementing respective indices
   }
   Else
   compare rows of both sparse matrices and which ever has less row value copy that to result matrix by incrementing respective indices

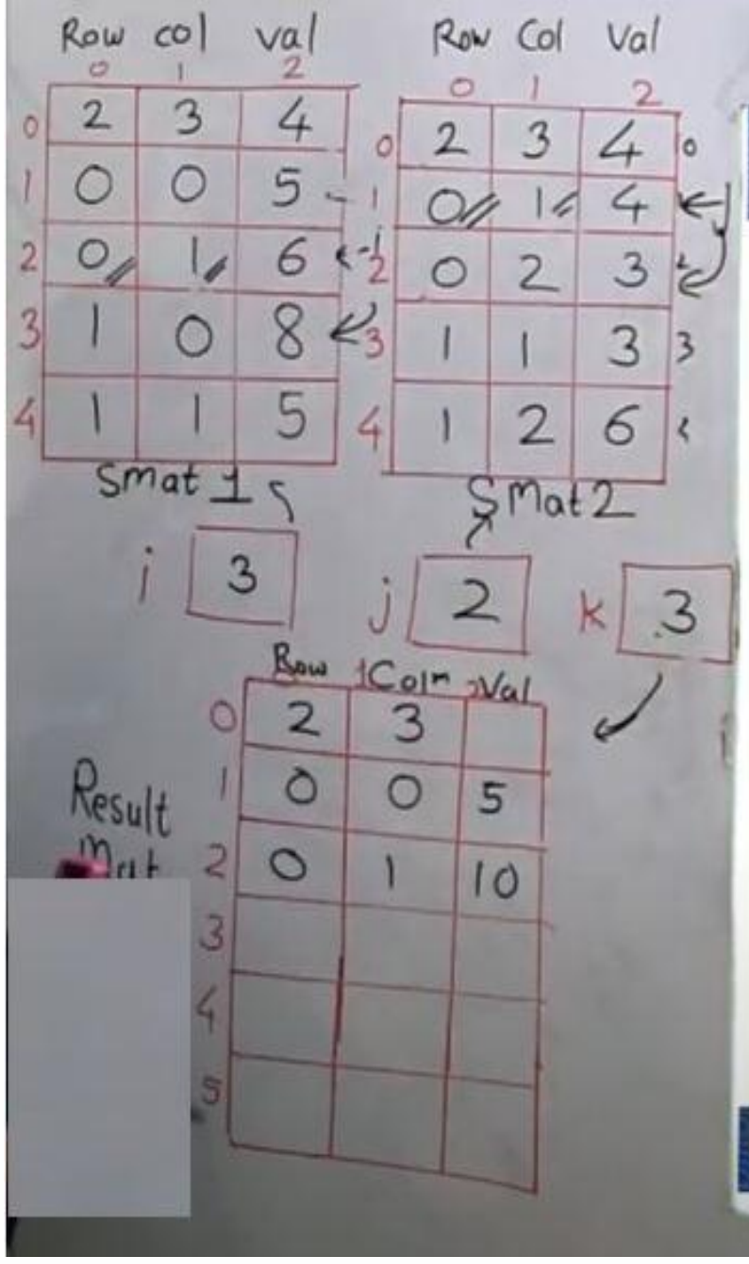
7. Repeat steps 6 till the end of any matrix's triplet
8. Copy the remaining term of sparse matrix (if any) to resultant matrix

Addition of Sparse Matrices

Steps:

6. **If**(row number of matrix 1 == row number of matrix2)
   {
   if(column number of matrix 1 == column number of matrix2)
   make the addition of non zero values and store it into result matrix by incrementing all indices
   else
   which ever has less column value copy that to result matrix by incrementing respective indices
   }
   Else
   compare rows of both sparse matrices and which ever has less row value copy that to result matrix by incrementing respective indices

7. Repeat steps 6 till the end of any matrix's triplet
8. Copy the remaining term of sparse matrix (if any) to resultant matrix



Addition of Sparse Matrices

Steps:

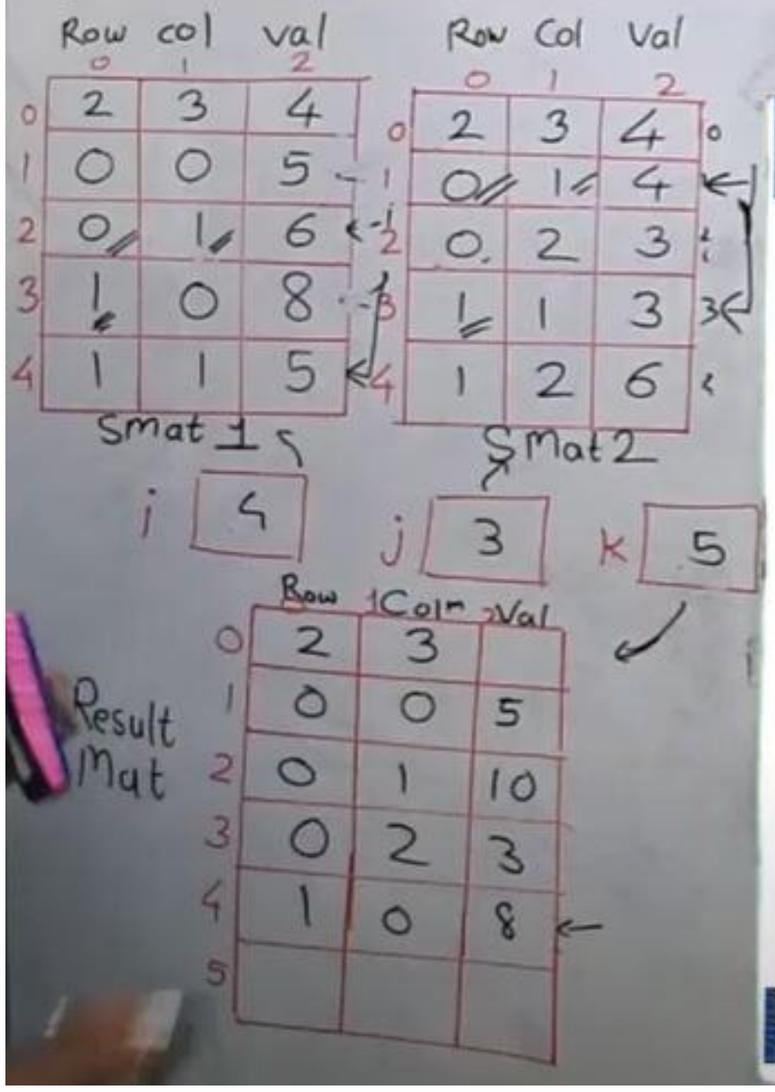
6. **If**(row number of matrix 1 == row number of matrix2)
   {
   if(column number of matrix 1 == column number of matrix2)
   make the addition of non zero values and store it into result matrix by incrementing all indices
   else
   which ever has less column value copy that to result matrix by incrementing respective indices
   }
   Else
   compare rows of both sparse matrices and which ever has less row value copy that to result matrix by incrementing respective indices
7. Repeat steps 6 till the end of any matrix's triplet
8. Copy the remaining term of sparse matrix (if any) to resultant matrix

Addition of Sparse Matrices

Steps:

6. **If**(row number of matrix 1 == row number of matrix2)
   {
   if(column number of matrix 1 == column number of matrix2)
   make the addition of non zero values and store it into result matrix by incrementing all indices
   else
   which ever has less column value copy that to result matrix by incrementing respective indices
   }
   Else
   compare rows of both sparse matrices and which ever has less row value copy that to result matrix by incrementing respective indices

7. Repeat steps 6 till the end of any matrix's triplet
8. Copy the remaining term of sparse matrix (if any) to resultant matrix

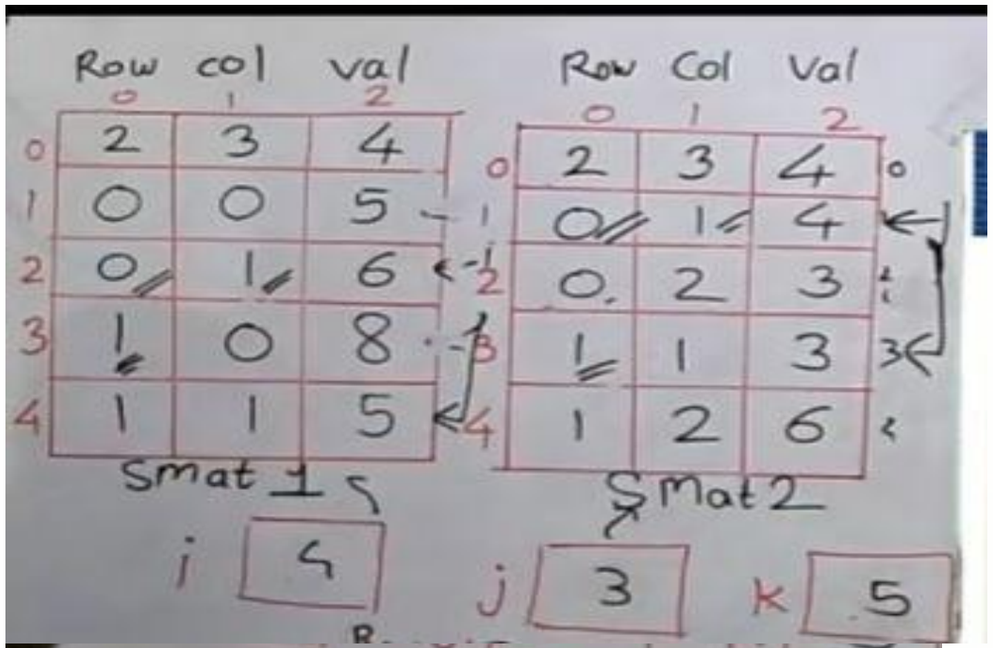


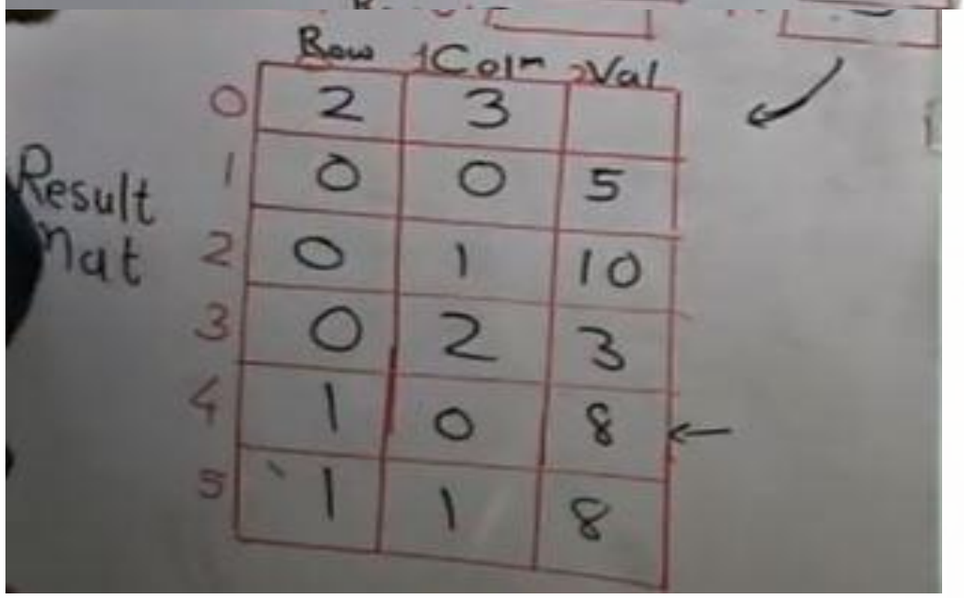
Addition of Sparse Matrices

Steps:

6. **If**(row number of matrix 1 == row number of matrix2)
   {
   if(column number of matrix 1 == column number of matrix2)
   make the addition of non zero values and store it into result matrix by incrementing all indices
   else
   which ever has less column value copy that to result matrix by incrementing respective indices
   }
   Else
   compare rows of both sparse matrices and which ever has less row value copy that to result matrix by incrementing respective indices

7. Repeat steps 6 till the end of any matrix's triplet
8. Copy the remaining term of sparse matrix (if any) to resultant matrix



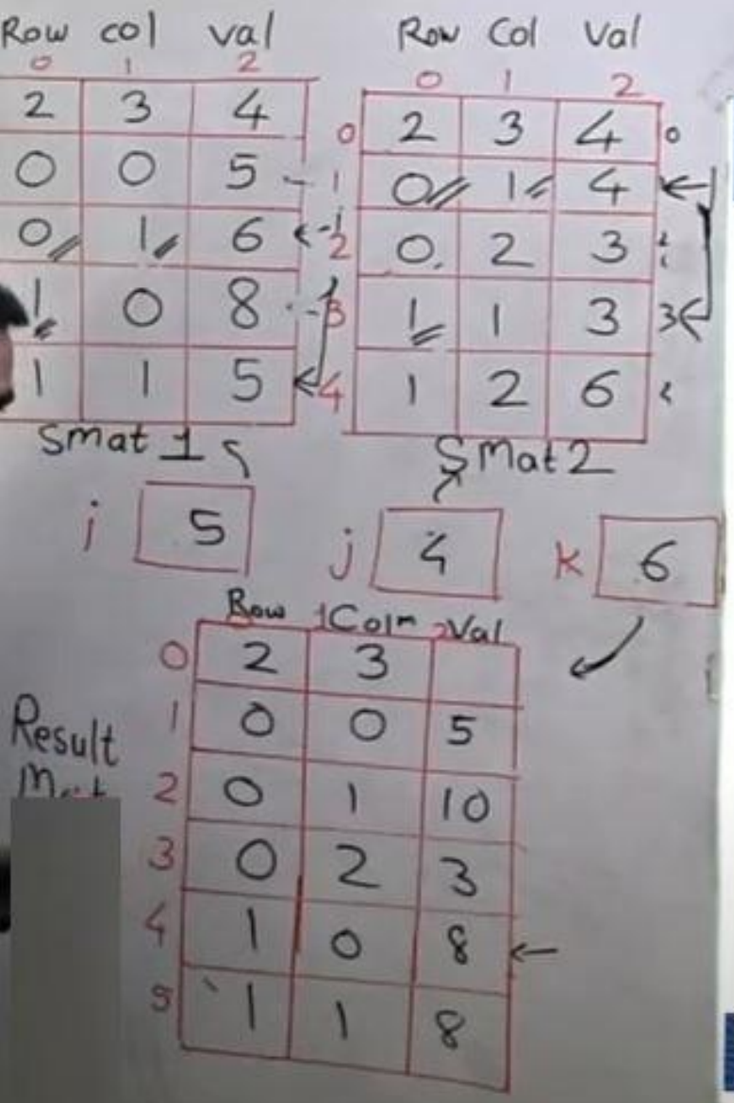


Addition of Sparse Matrices

Steps:

6. **If**(row number of matrix 1 == row number of matrix2)
   {
   if(column number of matrix 1 == column number of matrix2)
   make the addition of non zero values and store it into result matrix by incrementing all indices
   else
   which ever has less column value copy that to result matrix by incrementing respective indices
   }
   Else
   compare rows of both sparse matrices and which ever has less row value copy that to result matrix by incrementing respective indices

7. Repeat steps 6 till the end of any matrix's triplet
8. Copy the remaining term of sparse matrix (if any) to resultant matrix

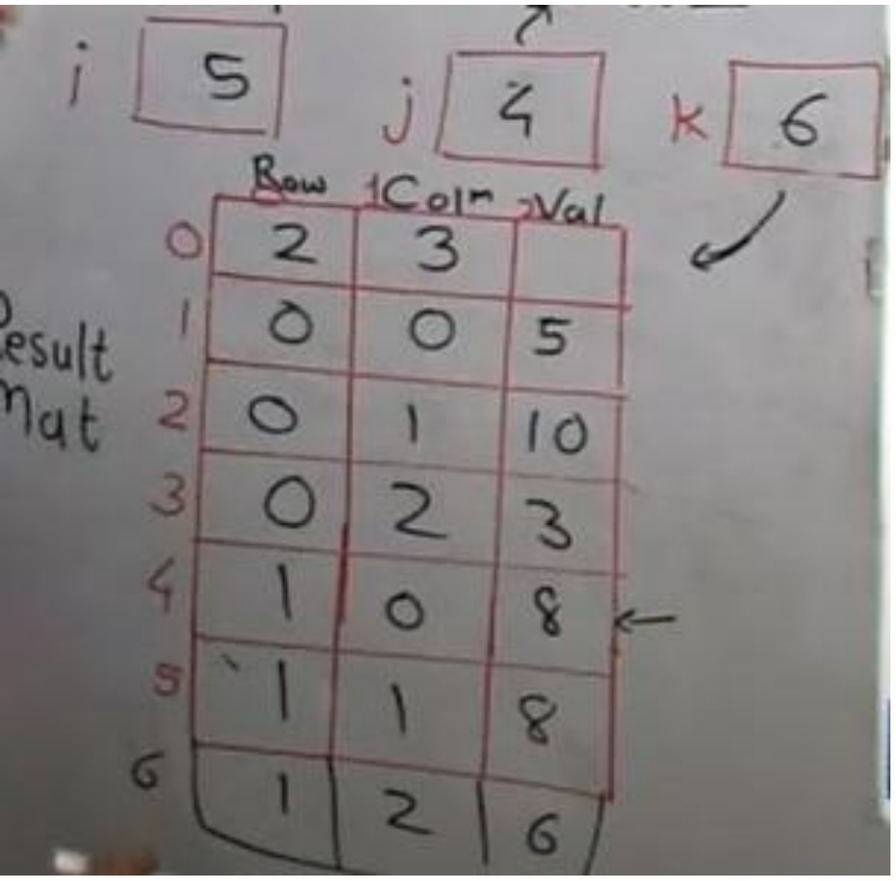


Addition of Sparse Matrices

Steps:

6. **If**(row number of matrix 1 == row number of matrix2)
   {
   if(column number of matrix 1 == column number of matrix2)
   make the addition of non zero values and store it into result matrix by incrementing all indices
   else
   which ever has less column value copy that to result matrix by incrementing respective indices
   }
   Else
   compare rows of both sparse matrices and which ever has less row value copy that to result matrix by incrementing respective indices

7. Repeat steps 6 till the end of any matrix's triplet
8. Copy the remaining term of sparse matrix (if any) to resultant matrix

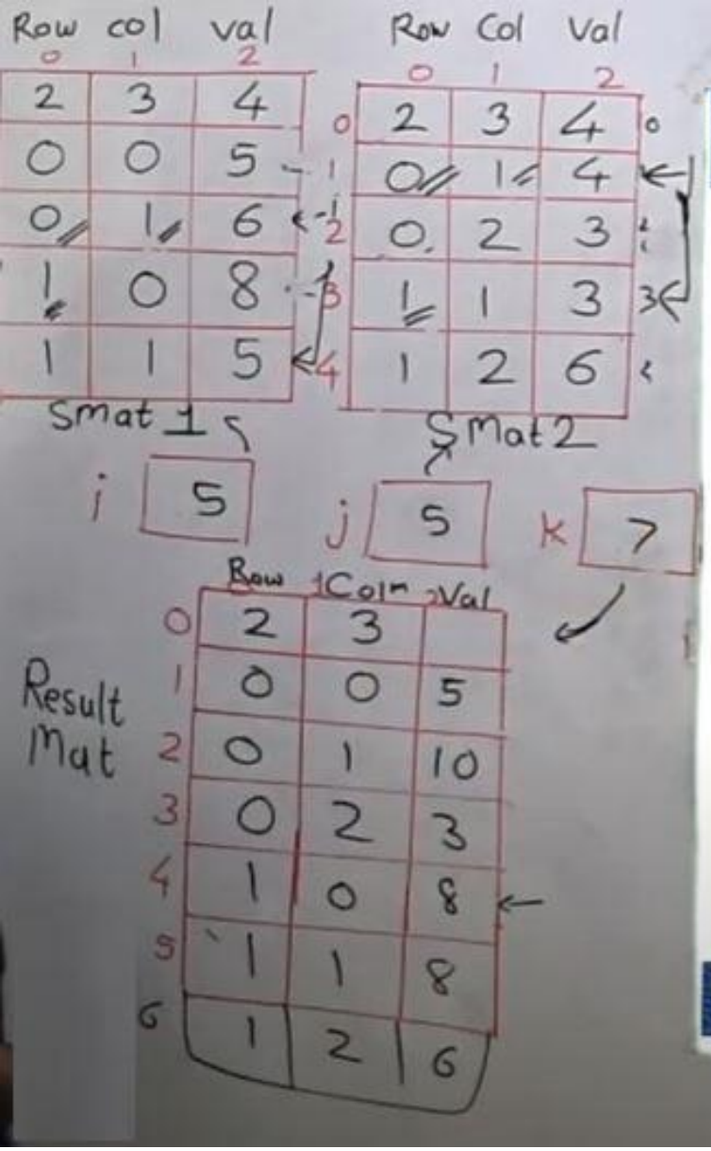


Addition of Sparse Matrices

Steps:

6. **If**(row number of matrix 1 == row number of matrix2)
   {
   if(column number of matrix 1 == column number of matrix2)
   make the addition of non zero values and store it into result matrix by incrementing all indices
   else
   which ever has less column value copy that to result matrix by incrementing respective indices
   }
   Else
   compare rows of both sparse matrices and which ever has less row value copy that to result matrix by incrementing respective indices

7. Repeat steps 6 till the end of any matrix's triplet
8. Copy the remaining term of sparse matrix (if any) to resultant matrix

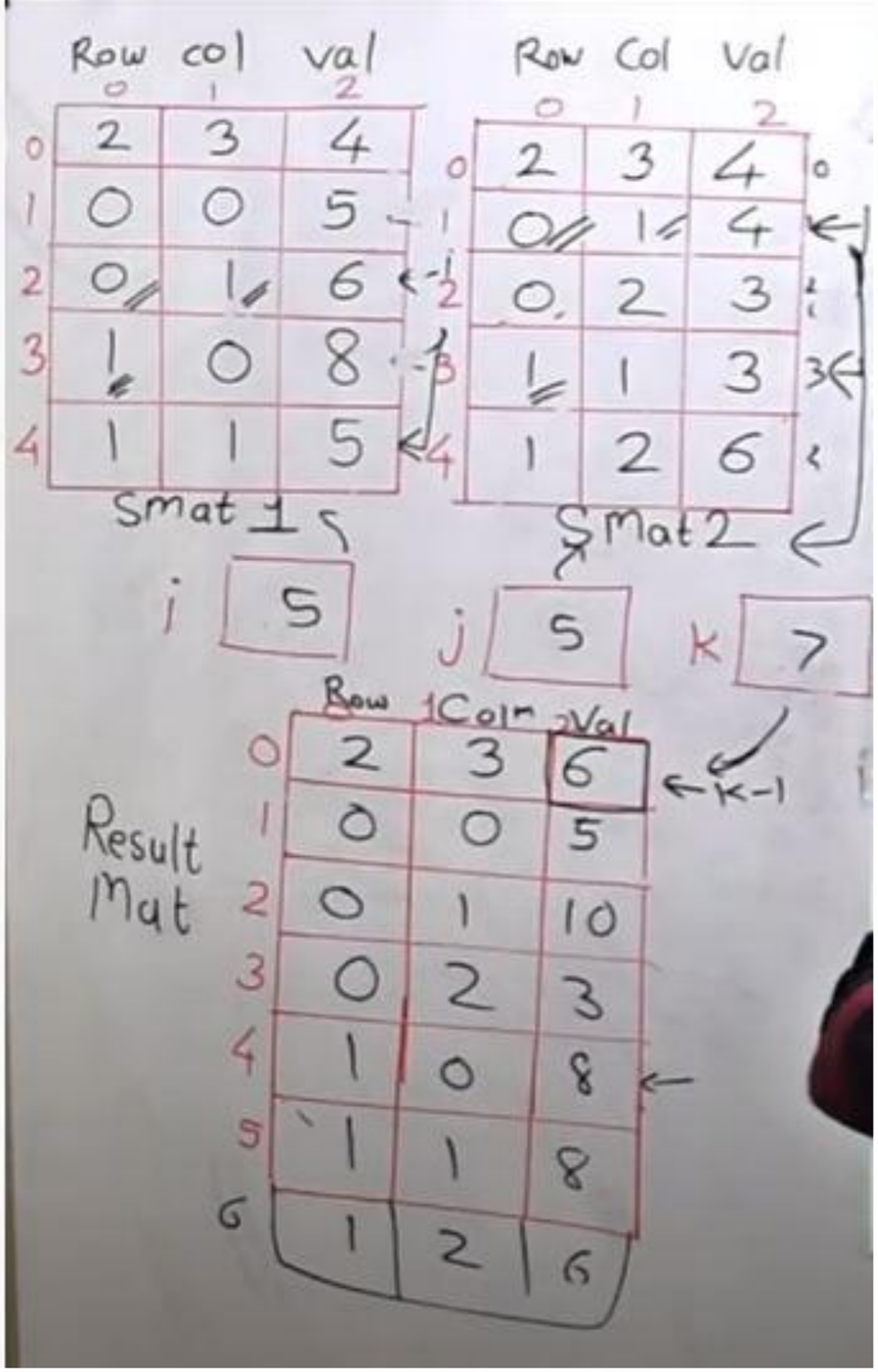


Addition of Sparse Matrices

Steps:

6. **If**(row number of matrix 1 == row number of matrix2)
   {
   if(column number of matrix 1 == column number of matrix2)
   make the addition of non zero values and store it into result matrix by incrementing all indices
   else
   which ever has less column value copy that to result matrix by incrementing respective indices
   }
   Else
   compare rows of both sparse matrices and which ever has less row value copy that to result matrix by incrementing respective indices

7. Repeat steps 6 till the end of any matrix's triplet
8. Copy the remaining term of sparse matrix (if any) to resultant matrix



Addition of Sparse Matrices

Steps:

6. **If**(row number of matrix 1 == row number of matrix2)
   {
   if(column number of matrix 1 == column number of matrix2)
   make the addition of non zero values and store it into result matrix by incrementing all indices
   else
   which ever has less column value copy that to result matrix by incrementing respective indices
   }
   Else
   compare rows of both sparse matrices and which ever has less row value copy that to result matrix by incrementing respective indices

7. Repeat steps 6 till the end of any matrix's triplet
8. Copy the remaining term of sparse matrix (if any) to resultant matrix

String Manipulation Using Array

- *It is usually formed from the character set of the programming language*

- *The value n is the length of the character string S*

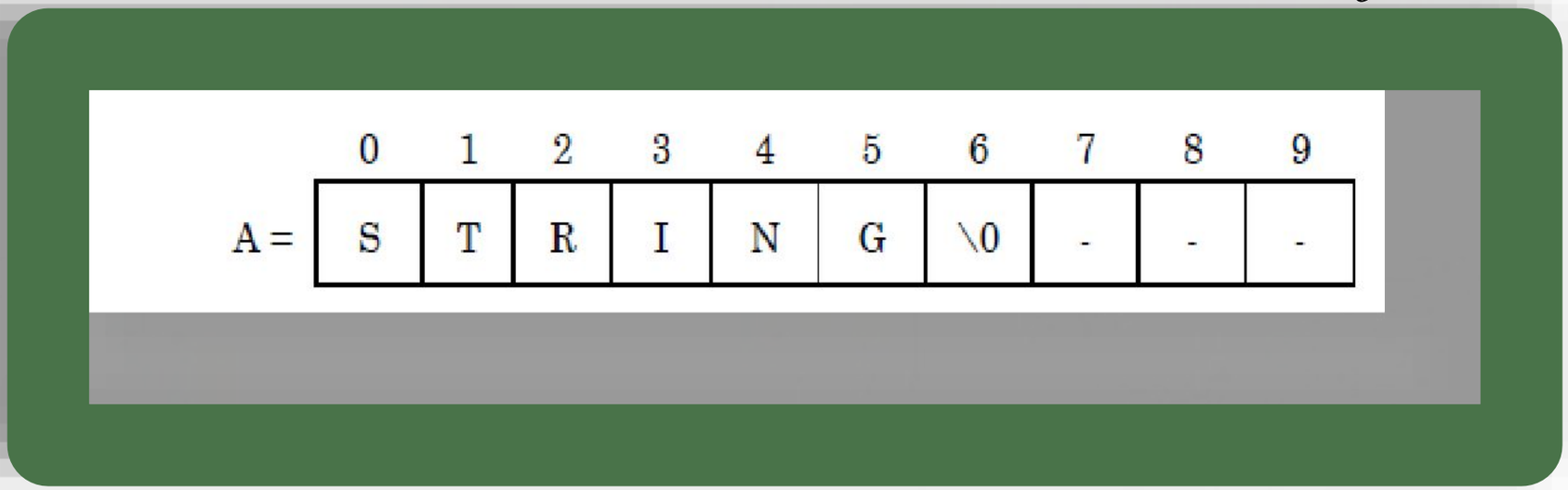
- *If n = 0 then S is called a null string or empty string*

- *Basically a string is stored as a sequence of characters in one- dimensional character array say A.*

char A[10] ="STRING" ;

Each string is terminated by a special character that is null character '\0'.

This null character indicates the end or termination of each string.



- ***There are various operations that can be performed on the string:***

  - *To find the length of a string*
  - *To concatenate two strings*
  - *To copy a string*
  - *To reverse a string*
  - *String compare*
  - *Palindrome check*
  - *To recognize a sub string*

3.9.2 Reversing a String

Q. Give pseudo C/C++ code to reverse the string.

SPPU - May 18, 3 Marks

Q. Write pseudo C/C++ code for reversing a string and state its time complexity.

SPPU - Dec. 18, 4 Marks

Pseudo C algorithm to reverse a string

Example:

Input string = "INDIA"

Output string = "AIDNI"

- Position i at the beginning and j at the end of the string.

I N D I A \0

↑ ↑

i = 0 j = 4

Interchange i^{th} character with the j^{th} character

- i = i + 1 and j = j – 1 and continue interchanging i^{th} character with the j^{th} character as long as i < j

A N D I I \0

↑ ↑

i = 1 j = 3

Loop terminates as the condition i < j fails.

A I D N I \0

↑↑

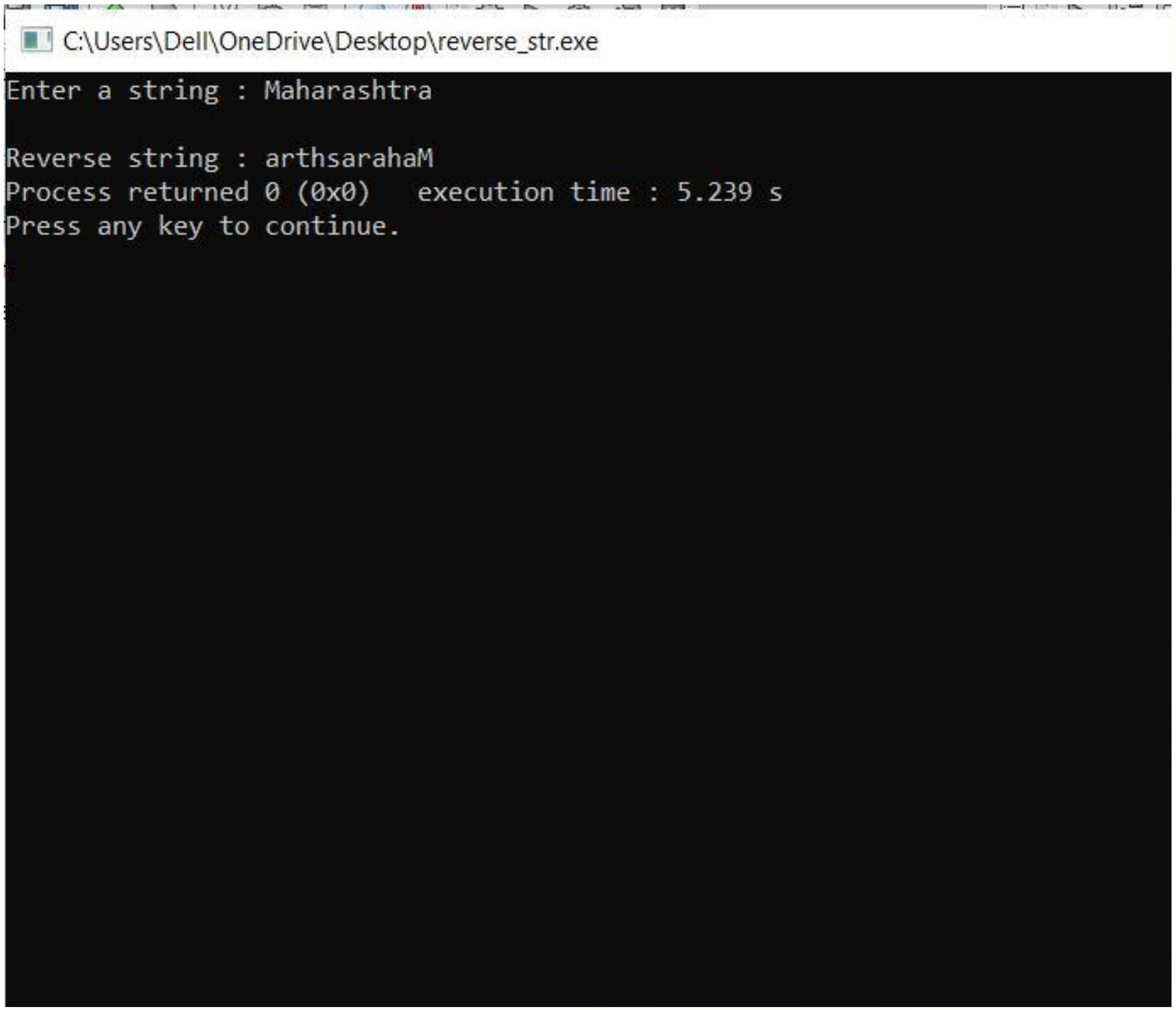
i = 2

j = 2

Time Complexity = O(n)

Write a program to reverse string.

```
#include<iostream>

#include<string.h>

using namespace std;

int main ()

{
   char str[50], temp;

   int i, j;

   cout << "Enter a string : ";

   Cin>>str;

   j = strlen(str) - 1;

   for (i = 0; i < j; i++,j--)

   {
      temp = str[i];

      str[i] = str[j];

      str[j] = temp;

   }

   cout << "\nReverse string : " << str;

   Return 0;

}
```

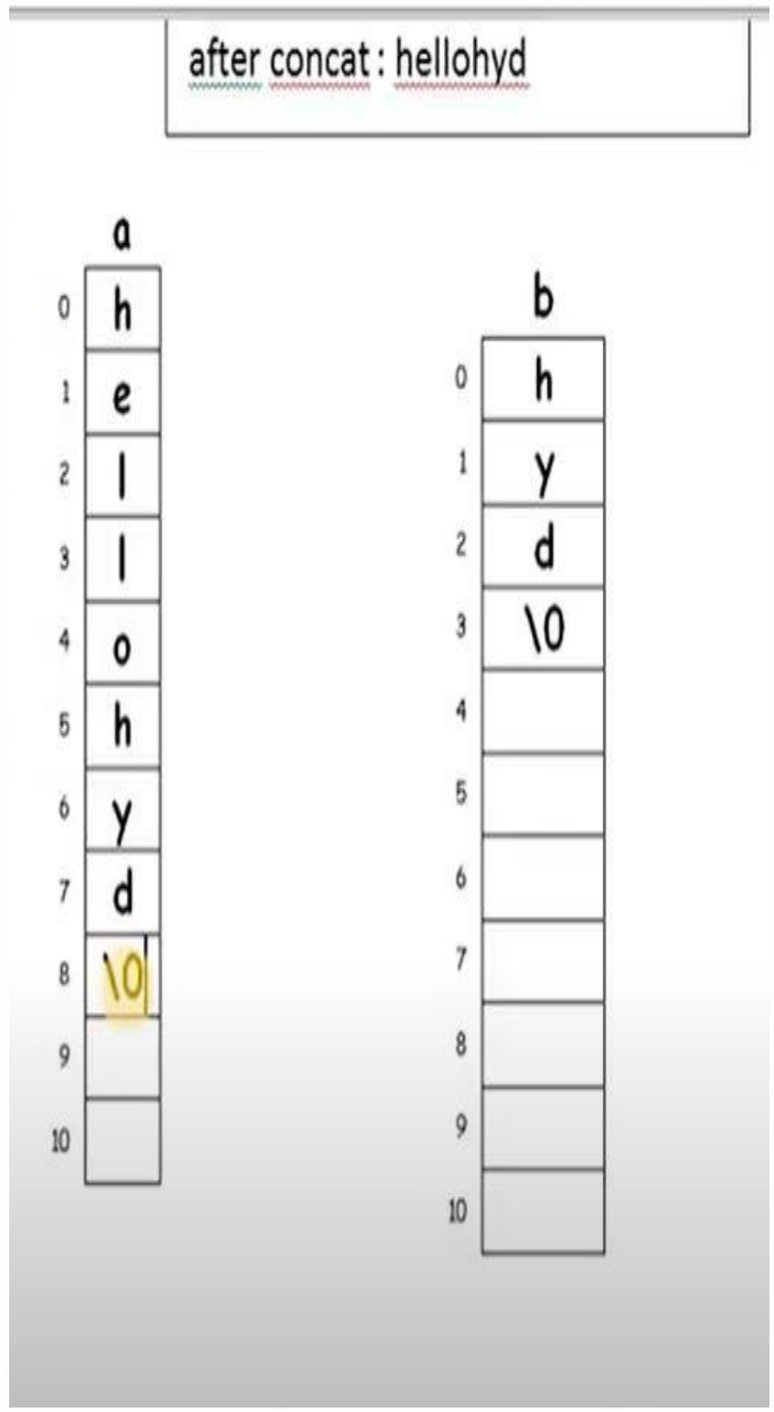

C:\Users\Dell\OneDrive\Desktop\reverse_str.exe
Enter a string : Maharashtra
Reverse string : arthsarahaM
Process returned 0 (0x0)   execution time : 5.239 s
Press any key to continue.

/* C++ Program - Concatenate String using inbuilt function */

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

void main()

{

clrscr();
char str1[50], str2[50];

cout<<"Enter first string : ";

Cin>>str1;

cout<<"Enter second string : ";

 cin>>str2;

strcat(str1, str2);
cout<<"String after concatenation is  "<<str1;

getch();

}
```

/* C++ Program - Concatenate String without using inbuilt function */

```
#include <iostream>
using namespace std;
int main()
 {
 char str1[100] = "hello";
char str2[100] = "hyd";
 int i,j;
cout<<"String 1: "<<str1<<endl;
cout<<"String 2: "<<str2<<endl;
for(i = 0; str1[i] != '\0'; ++i);
j=0;
while(str2[j] != '\0')
{
str1[i] = str2[j];
i++;   j++;
 }
str1[i] = '\0';
cout<<"String after concatenation: "<<str1;
return 0;
}
```



"C:\Users\Dell\OneDrive\Desktop\ConcatenateStringwithoutusinginbuilt function .exe"

```
String 1: hello
String 2: hyd
String after concatenation: hellohyd
Process returned 0 (0x0)   execution time : 0.766 s
Press any key to continue.
```

To get the length of a C-string string, strlen() function is used.

```
#include <iostream>
#include <cstring>

using namespace std;

int main()

{

char str[] = "C++ Programming is awesome";

// you can also use str.length()

cout << "String Length = " << strlen(str);

return 0;

}
```

Find length of string without using strlen function

```
#include<iostream>

using namespace std;

int main()

{
char str[] = "Apple";

int count = 0;

while (str[count] != '\0')

count++;

cout<<"The string is "<<str<<endl; cout <<"The length of the string is "<<count<<endl;

return 0;

}
```

/* C++ Program - Compare Two String */

```
#include<iostream.h>

#include<string.h>

void main()

 {

 clrscr();
char str1[100], str2[100];

cout<<"Enter first string : ";

gets(str1);

cout<<"Enter second string : ";

gets(str2);

if(strcmp(str1, str2)==0)

{

cout<<"Both the strings are equal";

 }

else

{ cout<<"Both the strings are not equal";
 } getch(); }
```

String is palindrom or not

```
#include <iostream>

#include <string.h>

using namespace std;

int main()

{

char str1[20], str2[20];

int i, j, len = 0, flag = 0;
cout << "Enter the string : ";

gets(str1);


len = strlen(str1) - 1;
for (i = len, j = 0; i >= 0 ; i--, j++)

str2[j] = str1[i];

if (strcmp(str1, str2))

flag = 1;


if (flag == 1)
cout << str1 << " is not a palindrome";

else

cout << str1 << " is a palindrome";

return 0;

}
```

SIMPLE TRANSPOSE OF MATRIX IN C++

```
#include <iostream>

using namespace std;

int main()

{

int a[10][10], trans[10][10], r, c, i, j;
cout << "Enter rows and columns of matrix: ";

cin >> r >> c;// Storing element of matrix


cout << endl << "Enter elements of matrix: " <<
endl;
for(i = 0; i < r; ++i)

for(j = 0; j < c; ++j)


{

cout << "Enter elements a" << i + 1 << j + 1 << ":
";

cin >> a[i][j];

} // Displaying the matrix a[][]

cout << endl << "Entered Matrix: " << endl;

for(i = 0; i < r; ++i)

for(j = 0; j < c; ++j)
{ cout << " " << a[i][j];

if(j == c - 1)


cout << endl << endl; }

// Finding transpose of matrix a[][] and storing it
infor(i = 0; i < r; ++i)

for(j = 0; j < c; ++j)

{

trans[j][i]=a[i][j];

}

// Displaying the transpose
cout << endl << "Transpose of Matrix: " << endl;

for(i = 0; i < c; ++i)


for(j = 0; j < r; ++j)

{
cout << " " << trans[i][j];

if(j == r - 1)


cout << endl << endl;

}
return 0;
}
```

MULTIPLICATION OF TWO POLYNOMIAL

```
#include<math.h>
#include<stdio.h>
#include<conio.h>
#define MAX 17
void init(int p[]);
void read(int p[]);
void print(int p[]);
void add(int p1[],int p2[],int p3[]);  void
multiply(int p1[],int p2[],int p3[]);

/*Polynomial is stored in an array, p[i] gives coefficient
of x^i .
  a polynomial 3x^2 + 12x^4 will be represented as
(0,0,3,0,12,0,0,….)
*/

void main()
{
int p1[MAX],p2[MAX],p3[MAX];
int option;
do
{
printf(“nn1 : create 1’st polynomial”);
printf(“n2 : create 2’nd polynomial”);
printf(“n3 : Add polynomials”);
printf(“n4 : Multiply polynomials”);
printf(“n5 : Quit”);
printf(“nEnter your choice :”);
scanf(“%d”,&option);
switch(option)
{
case  1:read(p1);break;
case  2:read(p2);break;
case 3:add(p1,p2,p3);
      printf(“n1’st polynomial -> “);
      print(p1);
      printf(“n2’nd polynomial -> “);
      print(p2);
      printf(“n Sum = “);
      print(p3);
      break;
case 4:multiply(p1,p2,p3);
      printf(“n1’st polynomial -> “);
      print(p1);
      printf(“n2’nd polynomial -> “);
      print(p2);
      printf(“n Product = “);
      print(p3);
      break;
}
}while(option!=5);
}
```

MULTIPLICATION OF TWO POLYNOMIAL

```
void read(int p[])
{
int n, i, power,coeff;
init(p);
printf("n Enter number of terms :");
scanf("%d",&n);
/* read n terms */
for (i=0;i<n;i++)
{    printf("nenter a term(power    coeff.)");
scanf("%d%d",&power,&coeff);
p[power]=coeff;
}
}
void print(int p[])
{
int i;
for(i=0;i<MAX;i++)
if(p[i]!=0)
printf("%dX^%d ",p[i],i);
}
void add(int p1[], int p2[], int p3[])
{
   int i;
   for(i=0;i<MAX;i++)
p3[i]=p1[i]+p2[i];
}
```

```
void     multiply(int p1[], int p2[], int
p3[])
{
int i,j;
init(p3);
for(i=0;i<MAX;i++)
for(j=0;j<MAX;j++)
  p3[i+j]=p3[i+j]+p1[i]*p2[j];
}
void init(int p[])
{
   int i;
   for(i=0;i<MAX;i++)
p[i]=0;
}
```

*******END******

FIND SIMPLE TRANSPOSE OF MATRIX

```
#include <iostream>

using namespace std;

int main()

{

    int a[10][10], trans[10][10], r, c, i, j;


    cout << "Enter rows and columns of matrix: ";

    cin >> r >> c;


//Storing element of matrix enter by user in array a[][].

    cout << endl << "Enter elements of matrix: " << endl;

    for(i = 0; i < r; ++i)


    for(j = 0; j < c; ++j)

    {

        cout << "Enter elements a" << i + 1 << j + 1 << ":
";

        cin >> a[i][j];

    }
```

```
// Displaying the matrix a[][]

    cout << endl << "Entered Matrix: " << endl;

    for(i = 0; i < r; ++i)

        for(j = 0; j < c; ++j)

        {

            cout << " " << a[i][j];

            if(j == c - 1)


                cout << endl << endl;

        }


    // Finding transpose of matrix a[][] and storing it
in array trans[][].

    for(i = 0; i < r; ++i)


        for(j = 0; j < c; ++j)


        {

            trans[j][i]=a[i][j];

        }
```

FIND SIMPLE TRANSPOSE OF MATRIX

```
// Displaying the transpose,i.e, Displaying array
trans[][].
    cout << endl << "Transpose of Matrix: " <<
endl;
    for(i = 0; i < c; ++i)
        for(j = 0; j < r; ++j)
        {
            cout << " " << trans[i][j];
            if(j == r - 1)
                cout << endl << endl;
        }

    return 0;
}
```

THANK YOU !!!!!