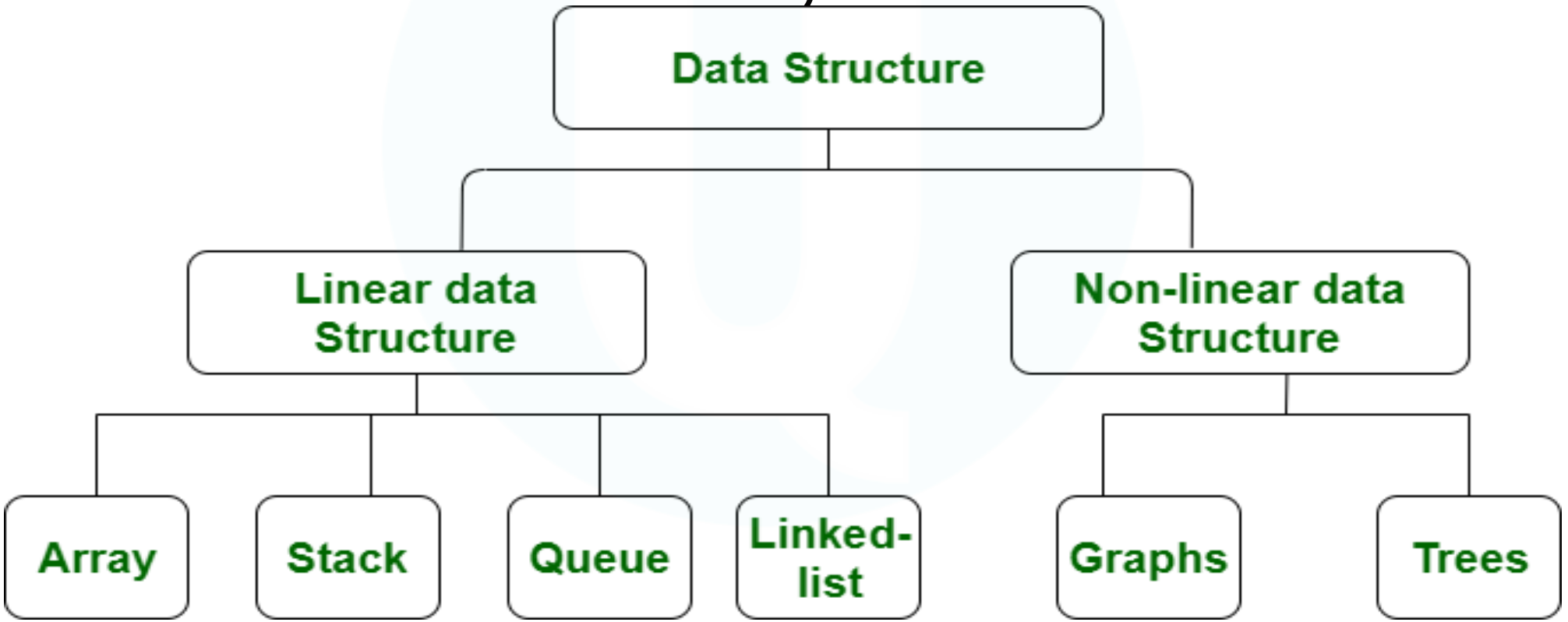
MODULE II

Arrays and Searching Polynomial representation using Arrays, Sparse matrix, Stacks, Queues-Circular Queues, Priority Queues, Double Ended Queues, Evaluation of Expressions. Linear Search and Binary Search

DataStructure:

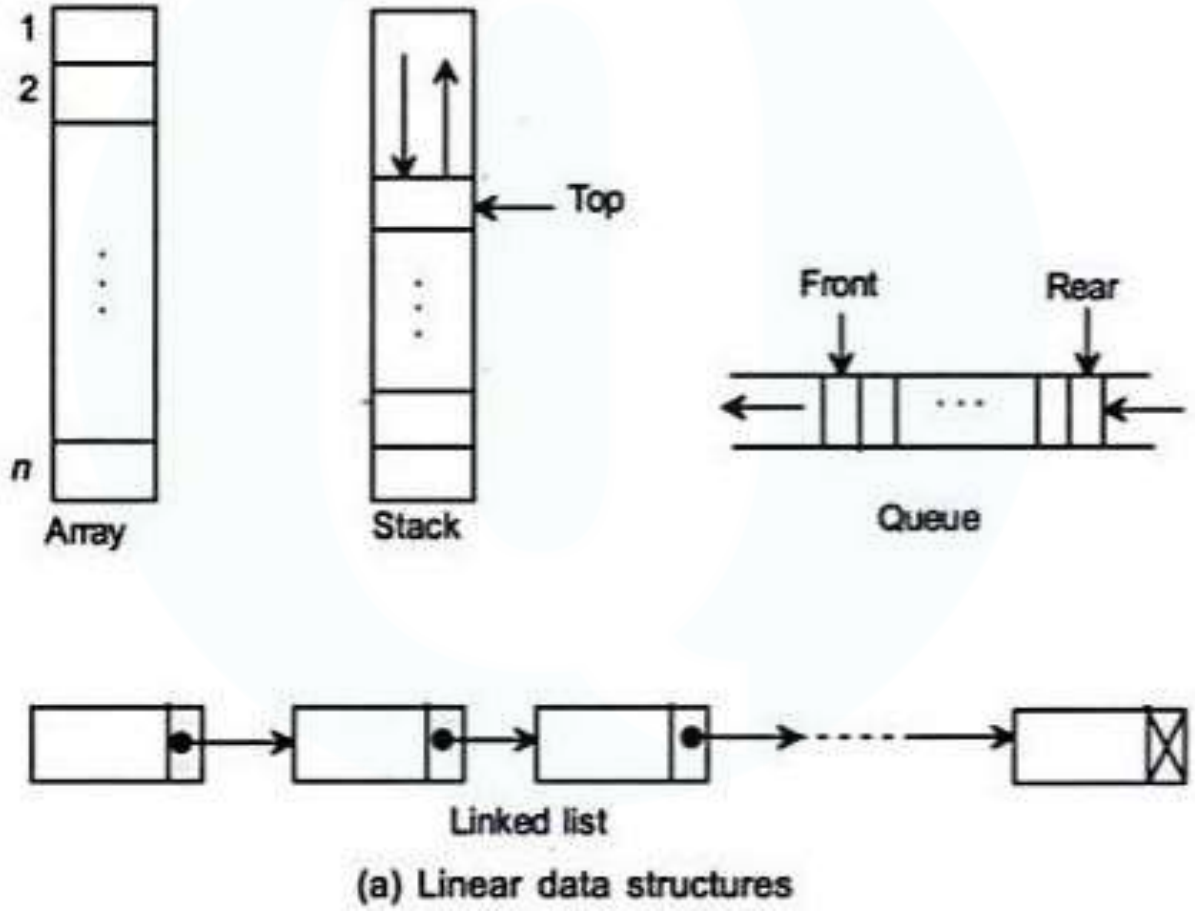
- Data Structure is a way of collecting and organising data in such a way that we can perform operations on these data in an effective way.



Data structure in real life

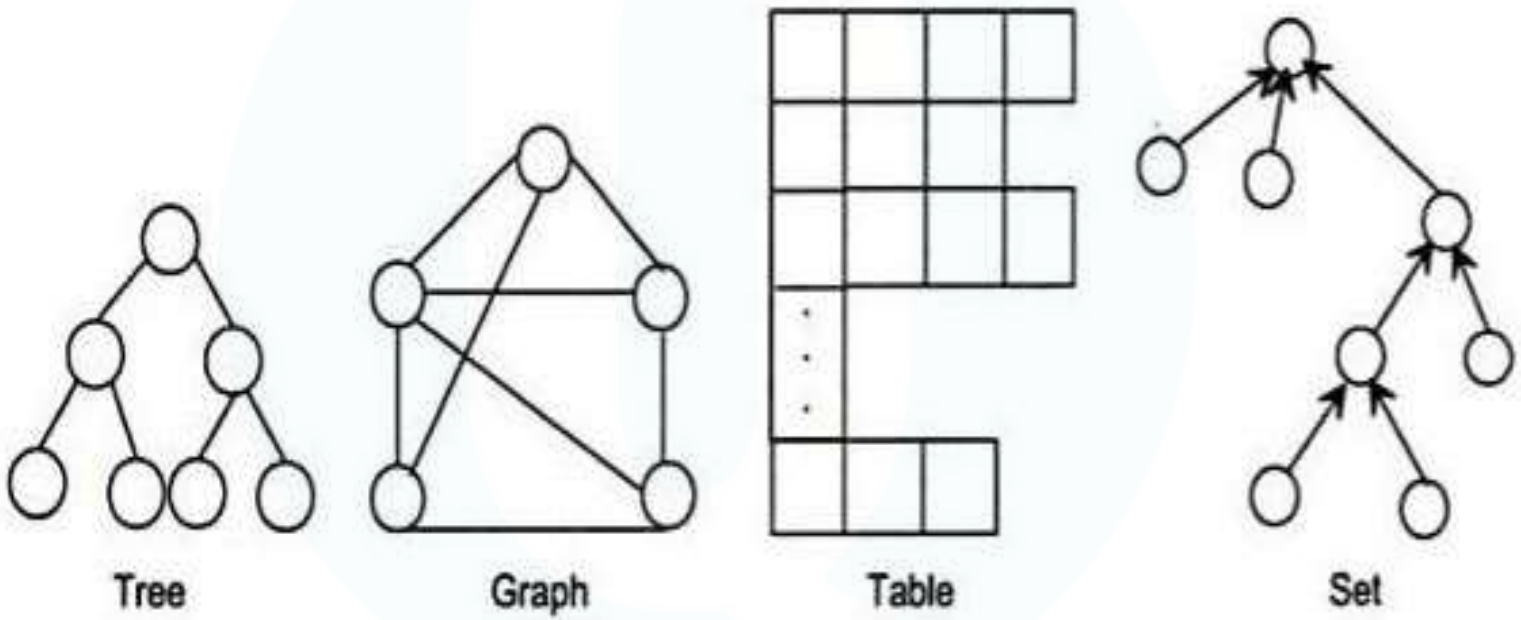
https://www.youtube.com/watch?v=d_XvFOkQz5k&t=193s

- Linear data structure
  - All the elements form a sequence or maintain a linear ordering.



(a) Linear data structures

- Non linear data structure
  - Elements are distributed over a plane.



(b) Non-linear data structures

Array Data Structure

- Arrays are defined as the collection of similar type of data items stored at contiguous memory locations.
- Array is the simplest data structure where each data element can be randomly accessed by using its index number.
- The total number of elements in an array is called **length**. The details of an array are accessed about its position. This reference is called **index** or **subscript**.
- **Element** − Each item stored in an array is called an element.
- **Index** − Each location of an element in an array has a numerical index, which is used to identify the element.

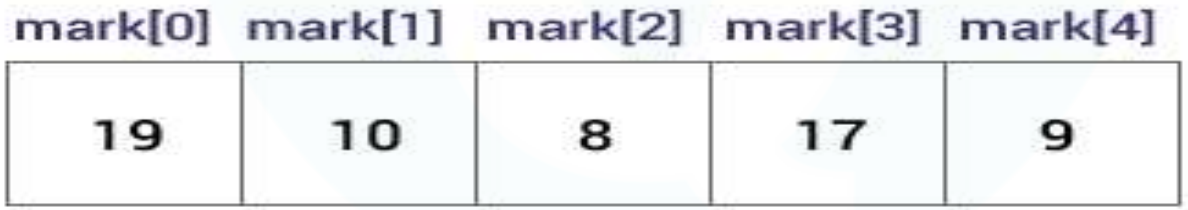
Array Cont...

- Array declaration:

  Syntax: datatype arrayname[size];

  Eg: int a[5];

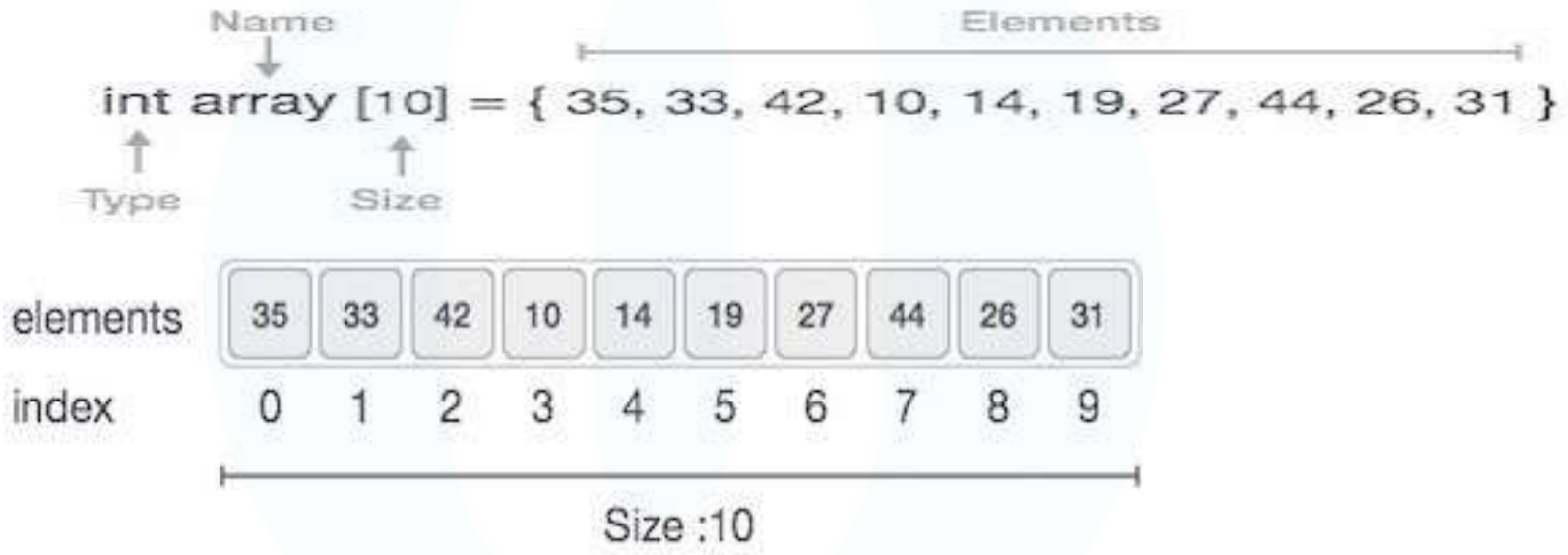
- Array initialization:
  - It is possible to initialize an array during declaration. For example,
  - int mark[] = {19, 10, 8, 17, 9};



- An array is a variable that can store multiple values. For example, if you want to store 100 integers, you can create an array for it.

  int data[100];

Array cont….



- 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 .

- Arrays
  - Linear data structure
  - List of finite no. of homogeneous data elements.
  - It is finite, bcoz it contains fixed no. of elements.
  - It is homogeneous, bcoz the elements of the array are of same data types.
  - This data structure enables us to arrange more than one element- composite data structure
  - The elements are referenced by array index or subscript (A_i)
  - The array elements are stored in successive memory locations
  - The array length= no. of elements in the array
  - Length=UB-LB+1

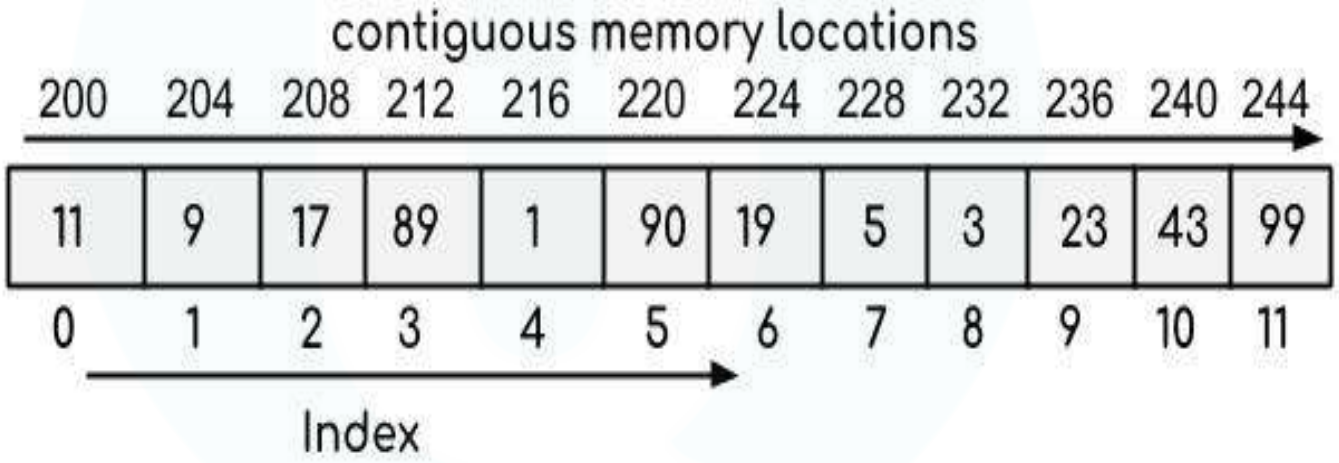
- TERMINOLOGY
  - Size: The no. of elements in an array.
    Also called length or dimension.
  - Type: Kind of data type it is meant for.
  - Base: Address of the memory location where the first element of the array is located
  - Index: Subscript like A_i or A[i]
  - Range of indices: Boundaries of an array
    Upper bound(U) and lower bound(L)
    In C, the range of indices is from 0

- TERMINOLOGY
  - Range of indices: Boundaries of an array
    Upper bound(U) and lower bound(L)
    In C, the range of indices is from 0
  - Index (A[i]) = L + i - 1
  - Size(A) = U – L + 1

Array

Single dimensional array

- Array with only one subscript/index.



Multidimensional array

- Array having more than one subscript variable is called multidimensional array.
- Multidimensional array is also called as matrix.
  1. Two Dimensional Array requires **Two Subscript Variables**
  2. Two Dimensional Array stores the values in the form of matrix.
  3. One Subscript Variable denotes the "**Row**" of a matrix.
  4. Another Subscript Variable denotes the "**Column**" of a matrix.

Declaration and Use of Two Dimensional Array :

int a[3][4];

	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]

WAP to access& print elements in an array

```c
#include <stdio.h>
int main()
{
            int values[5];
           printf("Enter 5 integers: ");
// taking input and storing it in an array
           for(int i = 0; i < 5; ++i)
           {
              scanf("%d", &values[i]);
           }
           printf("Displaying integers: ");
 // printing elements of an array
           for(int i = 0; i < 5; ++i)
           {
               printf("%d\n", values[i]);
           }
           return 0;
}
```

- OUTPUT

```
Enter 5 integers: 1 3 34 0 22
Displaying integers:
1
3
34
0
22
```

Write a C program

1) To get & display n elements in an 1D array.
2) To add elements in an array & display the sum
3) To display n elements with their position

Operations on arrays

- Traversing: 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
- Sort: It is used to arrange the data items in some order(ascending/descending order)
- Merging: It is used to combine the data items of two sorted files into single file in the sorted form

1)Array Traversal

- Accessing & processing each element exactly once
- i. e visit each element
- If we need to print all the elements or to count no. of elements or to find largest or smallest element, the traversal should be done.

- Algorithm: Traverse array
  - Let A be a linear array with Lower bound L and Upper bound U

1. Set i = L
2. while i<= U do
3. Process A[i]
4. i= i+1
5. End while
6. Stop

2)Insertion

- New elements can be added at beginning, end or any given index of array
- Let A be a linear array with N elements. Insert an ITEM in to k^{th} position of A.

1. Start
2. Set i= N
3. Repeat steps 4 &5 while i>=k
4. A[i+1]=A[i] Move i^{th} element downward
5. i=i-1
6. A[k]=ITEM
7. N=N+1
8. stop

- Algorithm: Insertion
  - New elements can be added at beginning, end or any given index of array
  - Let A be a linear array with N elements. Insert an ITEM in to k^{th} position of A.

1. Start
2. Set i= N
3. Repeat steps 4 &5 while i>=k
4. A[i+1]=A[i] Move i^{th} element downward
5. i=i-1
6. A[k]=ITEM
7. N=N+1
8. stop

Watch it

- https://www.youtube.com/watch?v=KELqVT7hjeE

Module 2(cont..)
Array Representation

Polynomial representation using Arrays, Sparse matrix, Applications of array

Polynomial representation using Arrays

- Array: a set of index and value
- Data structure: For each index, there is a value associated with that index.
- Representation: Implemented by using consecutive memory.
- An array is a set of pairs, index and value.
- For each index which is defined, there is a value associated with that index.
- In mathematical terms this a correspondence or a mapping

Polynomial representation using Arrays

- A polynomial is a sum of terms where each term has the form ax^e ,where x is the variable, a is the coefficient and e is the exponent.

A general polynomial $A(x)$ can be written as

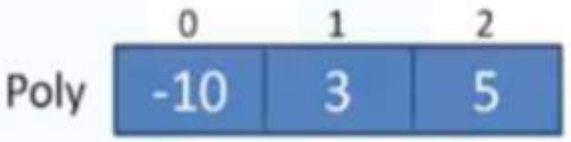
$$a_n x^n + a_{n-1} x^{n-1} + \ldots + a_1 x + a_0$$

where $a_n \neq 0$ and we say that the degree of A is n.

Polynomial representation using Arrays

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

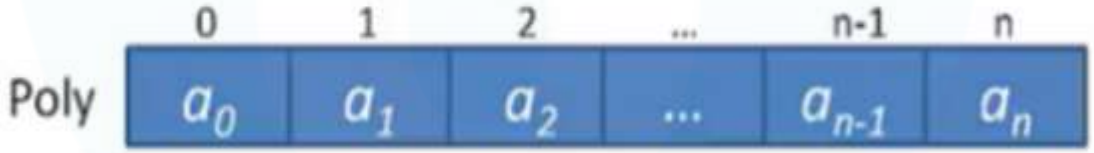
$-10x^0 + 3x^1 + 5x^2$




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

Polynomial representation using Arrays

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 representation using Arrays

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

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

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

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

i= 0	1	2	3
0	3	5	7

j= 0	1	2	3
10	3	5	0

$a_0 = 0 + 10 =$

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

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 representation using Arrays

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

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

0	j= 1	2	3
0	3	5	7

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

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

0	j= 1	2	3
10	3	5	0

$a_1 = 3 + 3 =$

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

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 representation using Arrays

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

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

0	1	i= 2	3
0	3	5	7

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

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

0	1	j=2	3
10	3	5	0

$a_2 = 5 + 5 =$

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

0	1	k=2	3
10	6		

```
i=j=k=2
while (i <= M)
{
C[k] = A[i] +B[j]
 i = i ++; j = j ++, k = k++
}
```

Polynomial representation using Arrays

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

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

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 representation using Array of Structures

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

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

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 representation using Array of Structures

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

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

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

X3=

	k=0
Coefficient	
Exponent	

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 term.

Polynomial representation using Array of Structures

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

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

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	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 representation using Array of Structures

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

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

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 representation using Array of Structures

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

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

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	

```
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 representation using Array of Structures

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

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

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
Coefficient	7	5	5	
Exponent	4	3	2	

```
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 representation using Array of Structures

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

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

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 the next term.

Polynomial representation using Array of Structures

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

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

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
k = k + 1
```

Polynomial representation using Array of Structures

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

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

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 representation using Array of Structures

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

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

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 representation using Array of Structures

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

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

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

Sparse Matrix

- In computer programming, a matrix can be defined with a 2-dimensional array.
- Any array with 'm' columns and 'n' rows represent a m X n matrix.
- There may be a situation in which a matrix contains **more number of ZERO values than NON-ZERO values**.
- Such matrix is known as sparse matrix.

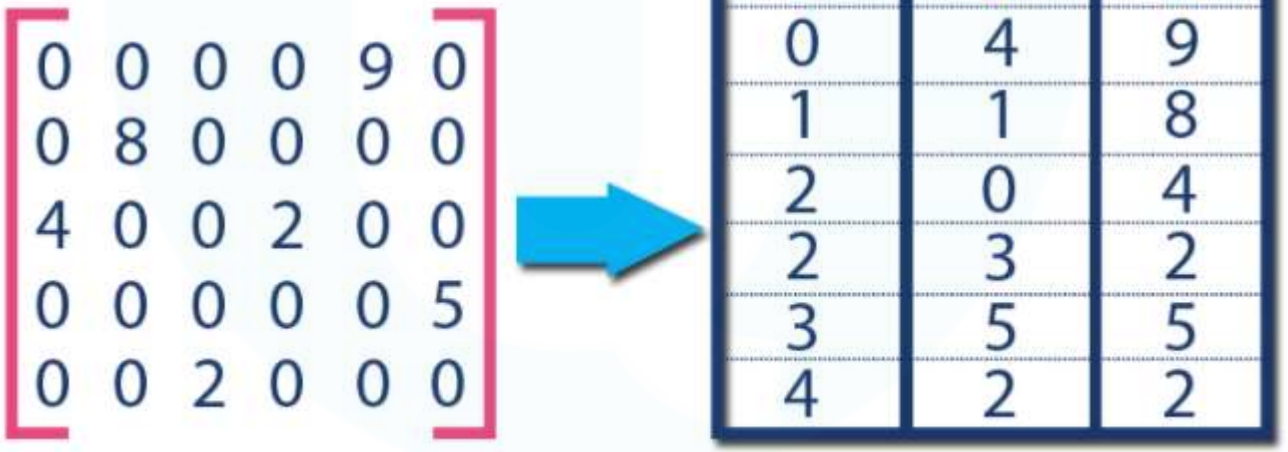
Sparse Matrix

- **Sparse matrix is a matrix which contains very few non-zero elements.**
- When a sparse matrix is represented with a 2-dimensional array, we waste a lot of space to represent that matrix.
- For example, consider a matrix of size 100 X 100 containing only 10 non-zero elements.

Triplet Representation (Array Representation)

- In this representation, we consider only non-zero values along with their row and column index values.
- In this representation, the 0^{th} row stores the total number of rows, total number of columns and the total number of non-zero values in the sparse matrix.
- For example, consider a matrix of size 5 X 6 containing 6 number of non-zero values.
- This matrix can be represented as shown in the figure

Sparse Matrix



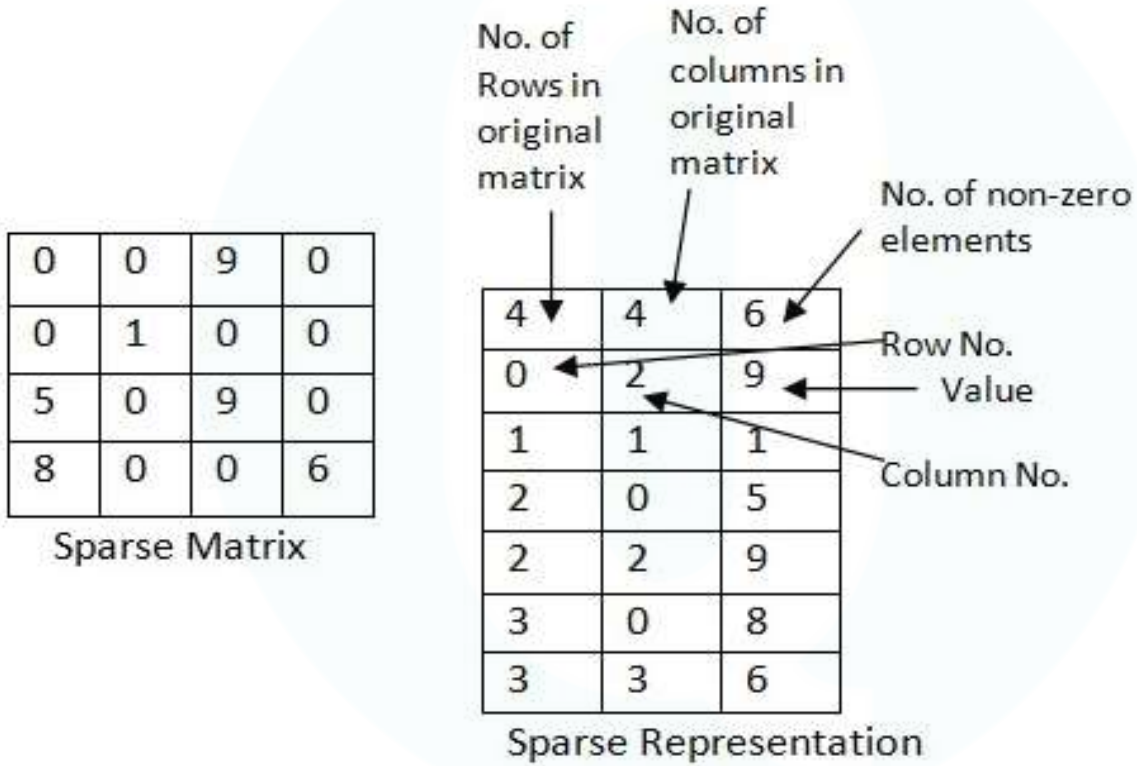
$$\begin{bmatrix} 0 & 0 & 0 & 0 & 9 & 0 \\ 0 & 8 & 0 & 0 & 0 & 0 \\ 4 & 0 & 0 & 2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 5 \\ 0 & 0 & 2 & 0 & 0 & 0 \end{bmatrix}$$

Rows	Columns	Values
5	6	6
0	4	9
1	1	8
2	0	4
2	3	2
3	5	5
4	2	2

Sparse Matrix

- In above example matrix, there are only 6 non-zero elements (those are 9, 8, 4, 2, 5 & 2) and matrix size is 5 X 6.
- We represent this matrix as shown in the above image. Here the first row in the right side table is filled with values 5, 6 & 6 which indicates that it is a sparse matrix with 5 rows, 6 columns & 6 non-zero values.
- The second row is filled with 0, 4, & 9 which indicates the non-zero value 9 is at the 0th-row 4th column in the Sparse matrix.
- In the same way, the remaining non-zero values also follow a similar pattern.

Sparse Matrix



0	0	9	0
0	1	0	0
5	0	9	0
8	0	0	6

Sparse Matrix

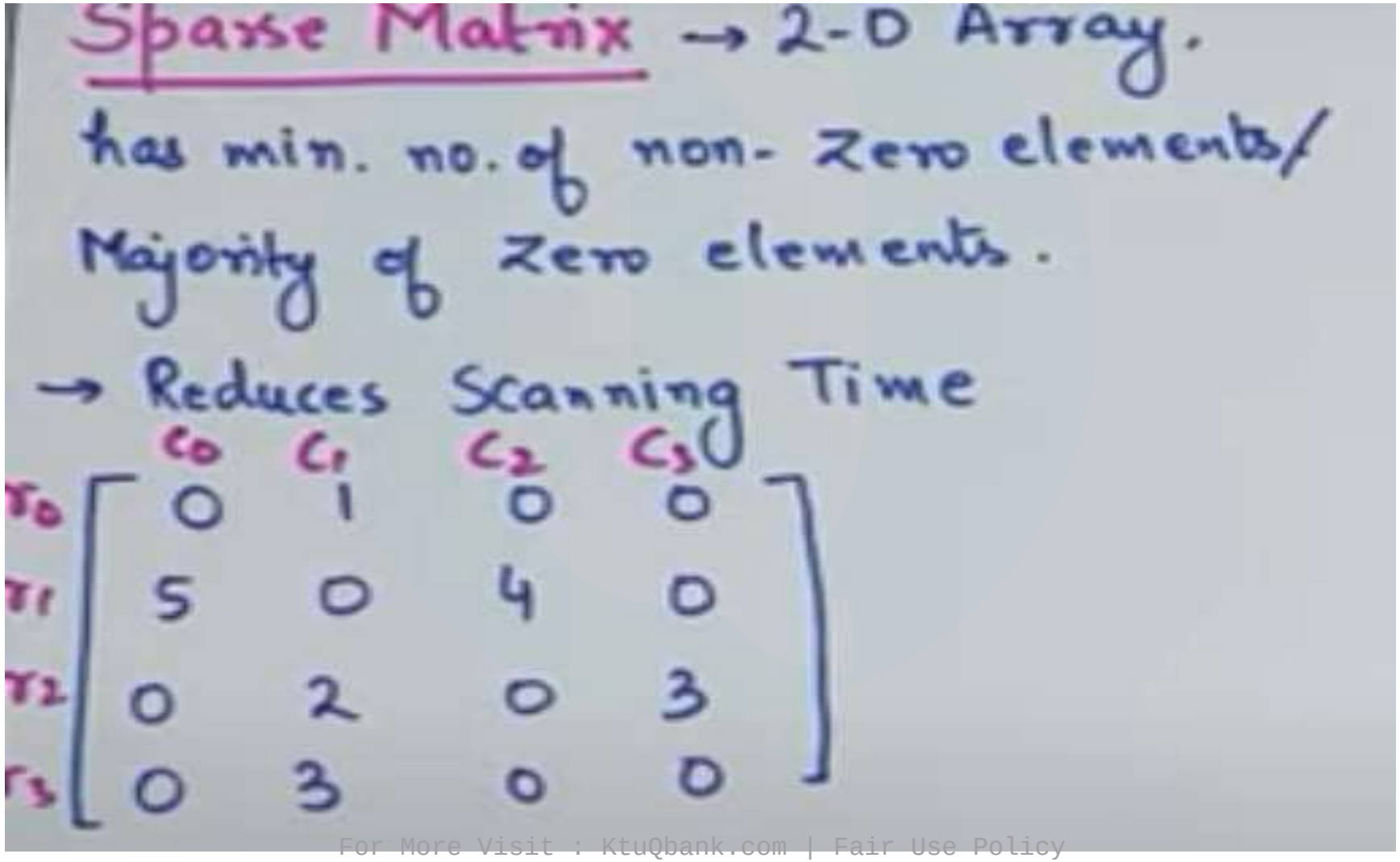
4	4	6
0	2	9
1	1	1
2	0	5
2	2	9
3	0	8
3	3	6

Sparse Representation

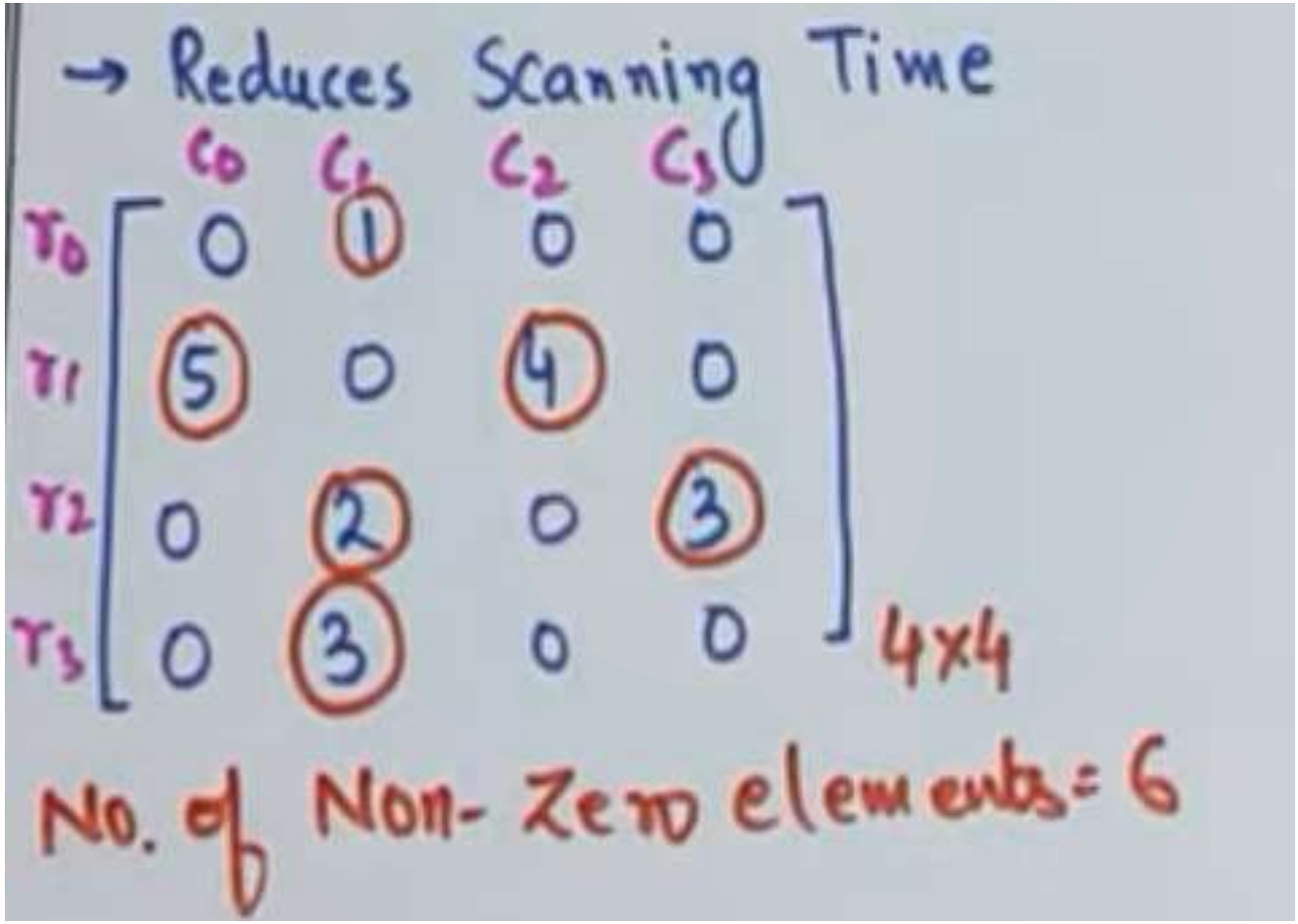
Why to use Sparse Matrix instead of simple matrix ?

- **Storage:** There are lesser non-zero elements than zeros and thus lesser memory can be used to store only those elements.
- **Computing time:** Computing time can be saved by logically designing a data structure traversing only non-zero elements..

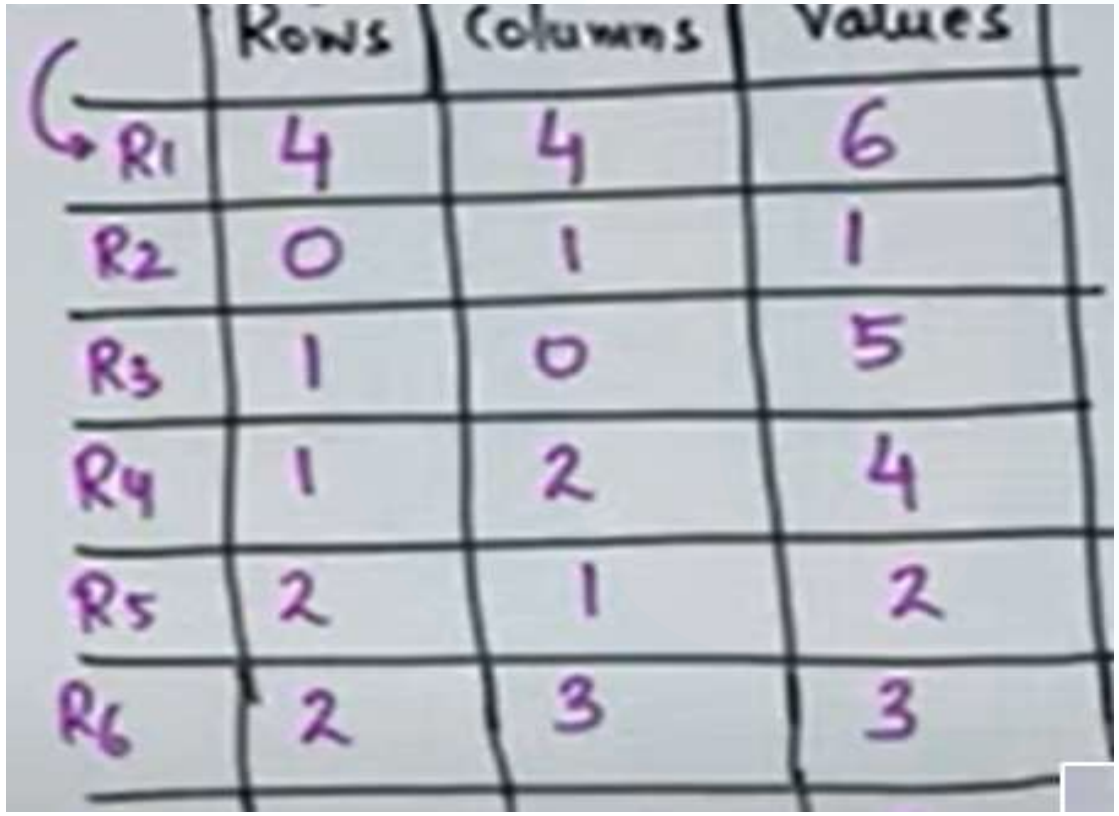
Sparse Matrix



Sparse Matrix



Sparse Matrix



	Rows	Columns	Values
R1	4	4	6
R2	0	1	1
R3	1	0	5
R4	1	2	4
R5	2	1	2
R6	2	3	3

```c
#include <stdio.h>
#define MAX 20

void read_matrix(int a[10][10], int row, int column);
void print_sparse(int b[MAX][3]);
void create_sparse(int a[10][10], int row, int column, int b[MAX][3]);

int main()
{
    int a[10][10], b[MAX][3], row, column;
    printf("\nEnter the size of matrix (rows, columns): ");
    scanf("%d%d", &row, &column);

    read_matrix(a, row, column);
    create_sparse(a, row, column, b);
    print_sparse(b);
    return 0;
}
```

```c
void read_matrix(int a[10][10], int row, int column)
{
    int i, j;
    printf("\nEnter elements of matrix\n");
    for (i = 0; i < row; i++)
    {
        for (j = 0; j < column; j++)
        {
            printf("[%d][%d]: ", i, j);
            scanf("%d", &a[i][j]);
        }
    }
}
```

```c
void create_sparse(int a[10][10], int row, int column, int b[MAX][3])
{
    int i, j, k;
    k = 1;
    b[0][0] = row;
    b[0][1] = column;
    for (i = 0; i < row; i++)
    {
        for (j = 0; j < column; j++)
        {
            if (a[i][j] != 0)
            {
                b[k][0] = i;
                b[k][1] = j;
                b[k][2] = a[i][j];
                k++;
            }
        }
        b[0][2] = k - 1;
    }
}
```

```c
void print_sparse(int b[MAX][3])
{
    int i, column;
    column = b[0][2];
    printf("\nSparse form - list of 3 triples\n\n");
    for (i = 0; i <= column; i++)
    {
        printf("%d\t%d\t%d\n", b[i][0], b[i][1], b[i][2]);
    }
}
```

```
Enter the size of matrix (rows, columns): 3 4

Enter elements of matrix
[0][0]: 6
[0][1]: 0
[0][2]: 0
[0][3]: 0
[1][0]: 0
[1][1]: 1
[1][2]: 0
[1][3]: 0
[2][0]: 0
[2][1]: 0
[2][2]: 0
[2][3]: 5

Sparse form - list of 3 triples

3    4    3
0    0    6
1    1    1
2    3    5
```

Applications of array

- Arrays are used to implement other data structures, such as **lists, heaps, hash tables, deques, queues and stacks.**
- Arrays are used to implement mathematical
    - **Vectors**
    - **Matrices**
    - **Records**.
- Arrays are used to **Store List of values**
- Arrays are used to implement **Search Algorithms**
- Arrays are used to implement **Sorting Algorithms**
- Arrays are also used to implement **CPU Scheduling Algorithms**

ARRAY OPERATIONS

PART 2

Operations on arrays

- Traversing: 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
- Sort: It is used to arrange the data items in some order(ascending/descending order)
- Merging: It is used to combine the data items of two sorted files into single file in the sorted form

1)Array Traversal

- Accessing & processing each element exactly once
- i. e visit each element
- If we need to print all the elements or to count no. of elements or to find largest or smallest element, the traversal should be done.

- Algorithm: Traverse array
  - Let A be a linear array with Lower bound L and Upper bound U

1. Set i = L
2. while i<= U do
3. Process A[i]
4. i= i+1
5. End while
6. Stop

2)INSERTION

Algorithm: Insertion

- New elements can be added at beginning, end or any given index of array
- Let A be a linear array with N elements,indices from 1 to N. Insert an ITEM in to k^{th} position of A.

1. Start
2. Set i= N
3. Repeat steps 4 &5 while i>=k
4. A[i+1]=A[i] Move i^{th} element downward
5. i=i-1
6. A[k]=ITEM
7. N=N+1
8. stop

Watch it: insertion

- https://www.youtube.com/watch?v=KELqVT7hjeE

3)Deletion:

- Algorithm: Deletion
  - Let A be a linear array with N elements.
  - Delete the k^{th} element from the array 'A' and assigns it to ITEM

1. Start
2. Set ITEM= A[k]
3. Repeat for i=k to N-1
4. A[i]=A[i+1]
5. i=i+1
6. Set N=N-1
7. Stop

Watch it:Deletion

- https://www.youtube.com/watch?v=CZYR2v8rYLA&t=105s

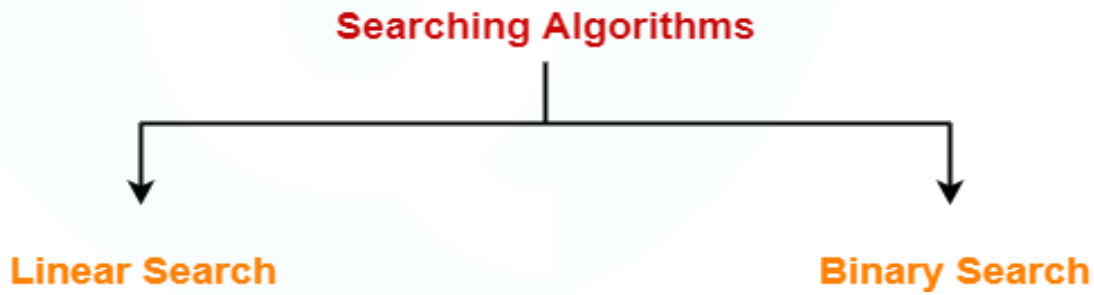
Watch it:Searching

- https://www.youtube.com/watch?v=-lY4_THb2wM

4)Searching

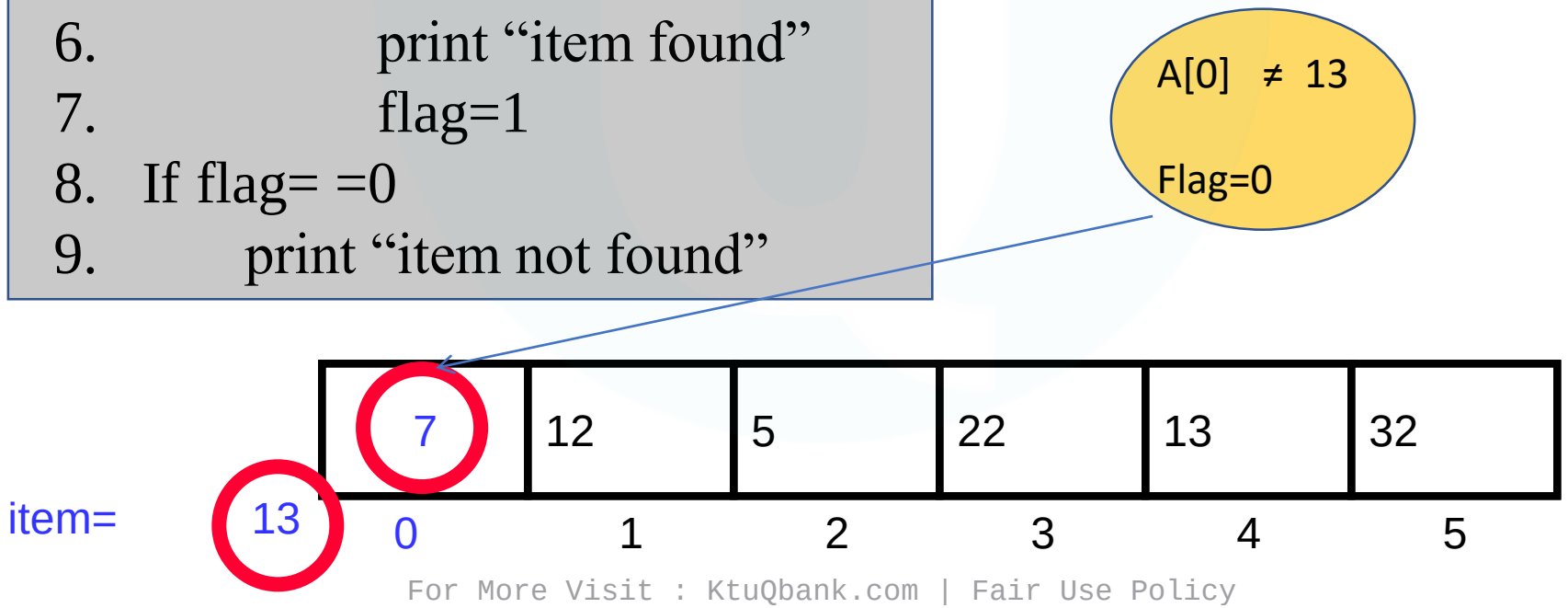
- Search whether ITEM is present in array or not
- Finding the location LOC of ITEM in array A
- Two types

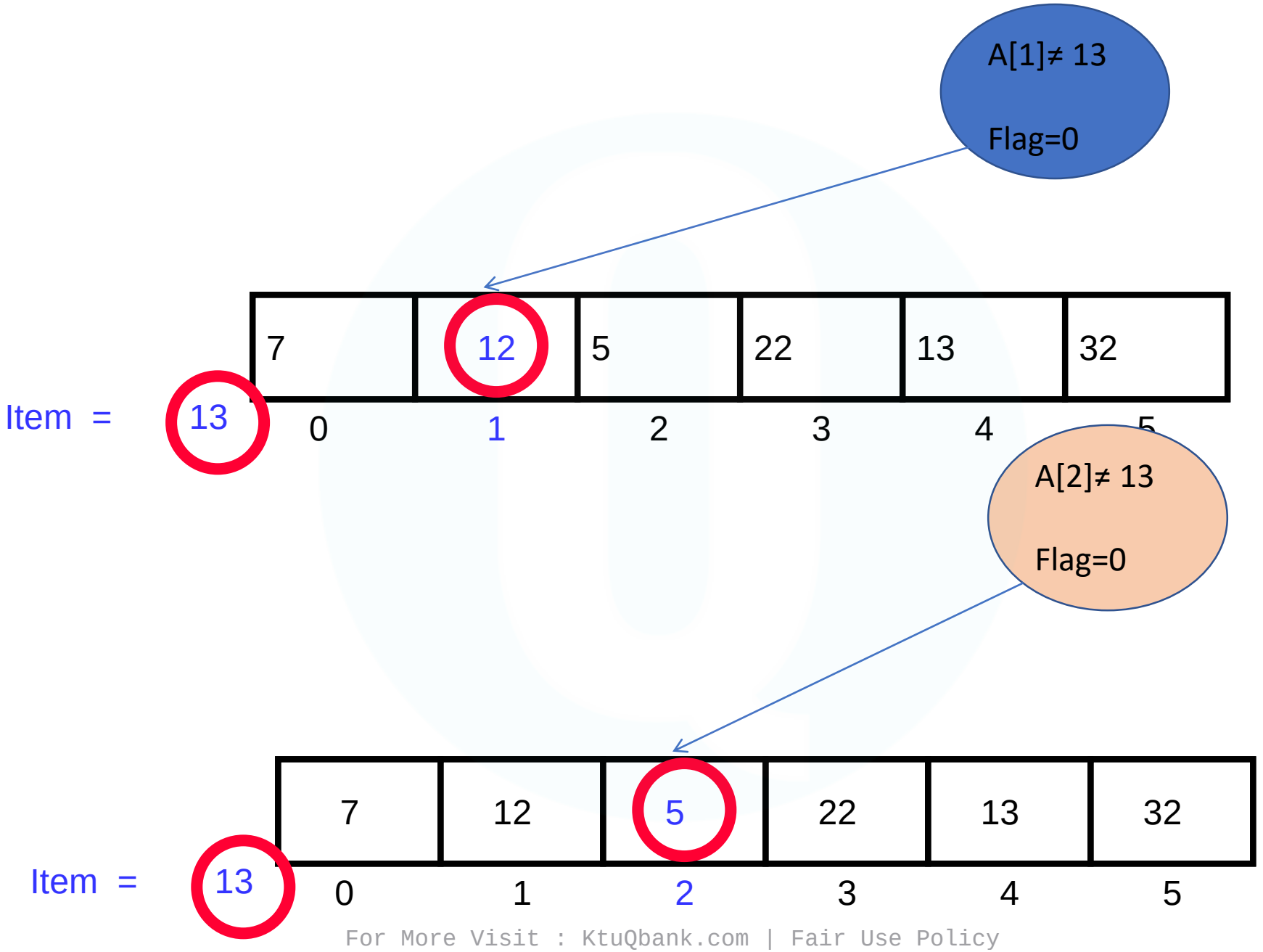
1. Linear Search: Small array,unsorted array
2. Binary Search: Large array,Sorted array

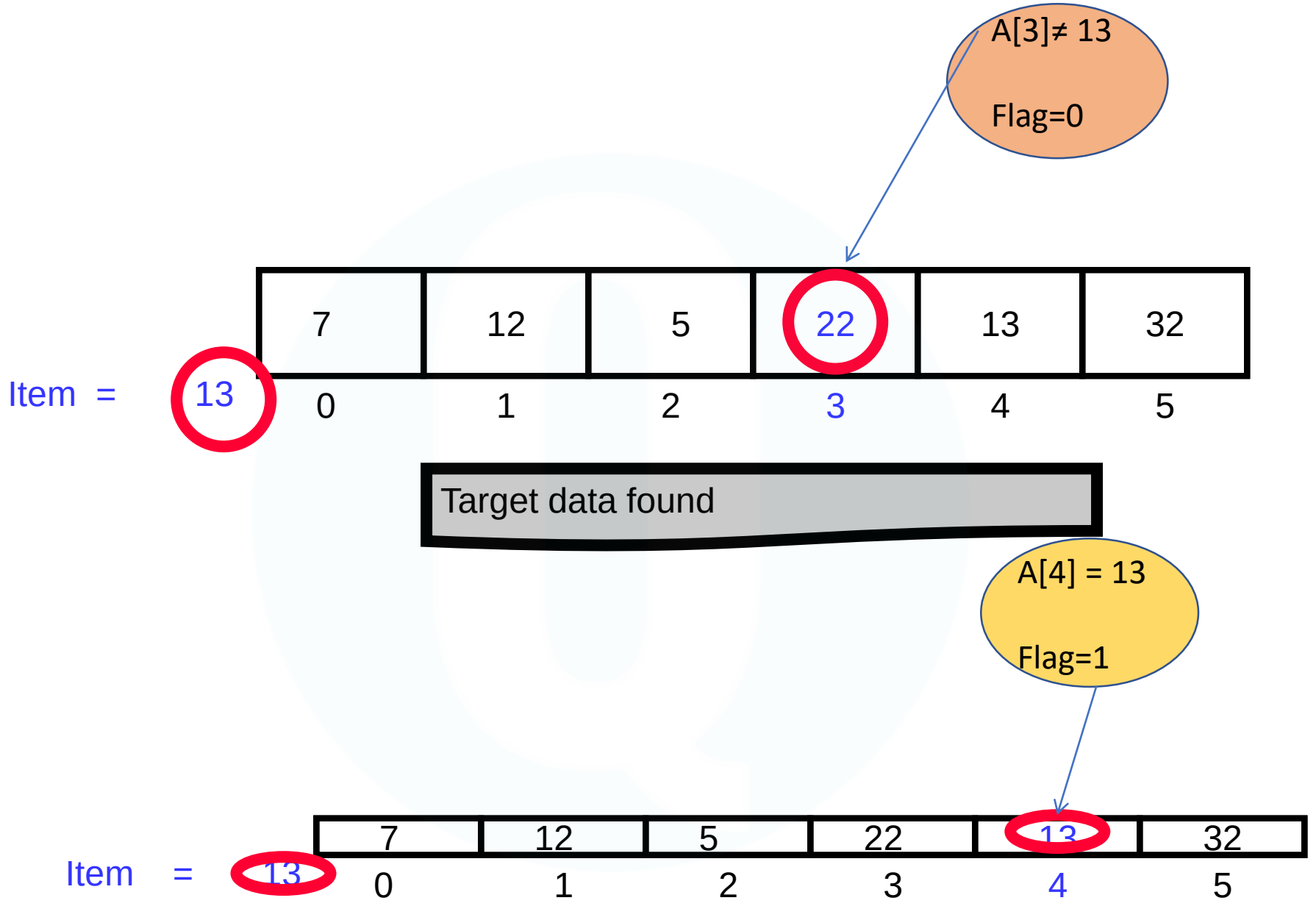


Let A be an array with N elements and we want to search for ITEM in A.

```
1.  Start
2.  Read the ITEM to be searched
3.  Set flag=0
4.  Repeat for i=0 to N
5.          if  A[i]= =ITEM
6.                  print “item found”
7.                  flag=1
8.  If flag= =0
9.          print “item not found”
```






A[3]≠ 13
Flag=0
7
12
5
22
13
32
Item =
13
0
1
2
3
4
5
Target data found
A[4] = 13
Flag=1
7
12
5
22
13
32
Item =
13
0
1
2
3
4
5

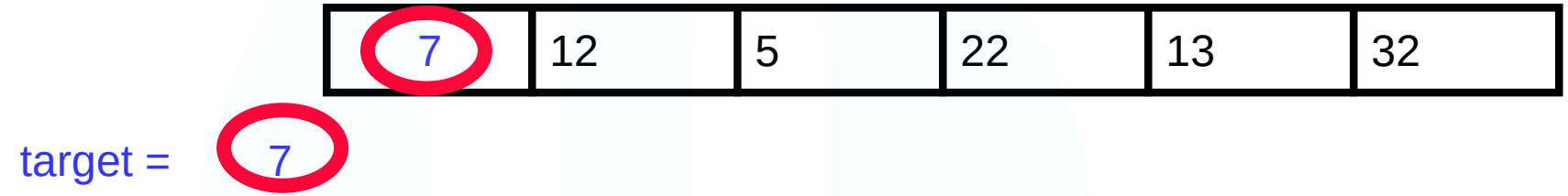
Linear Search: Complexity

- How long will our search take?
- Best case,worst case,average case.
- In the **best case**, the target value is in the first element of the array.Complexity is **O(1)**
- In the **worst case**, the target value is in the last element of the array. Complexity is **O(n).**
- In the **average case**,the target value will be in the middle of the array.So the search takes an amount of time proportional to half the length of the array – also proportional to the length of the array – **O(*n*)** again!

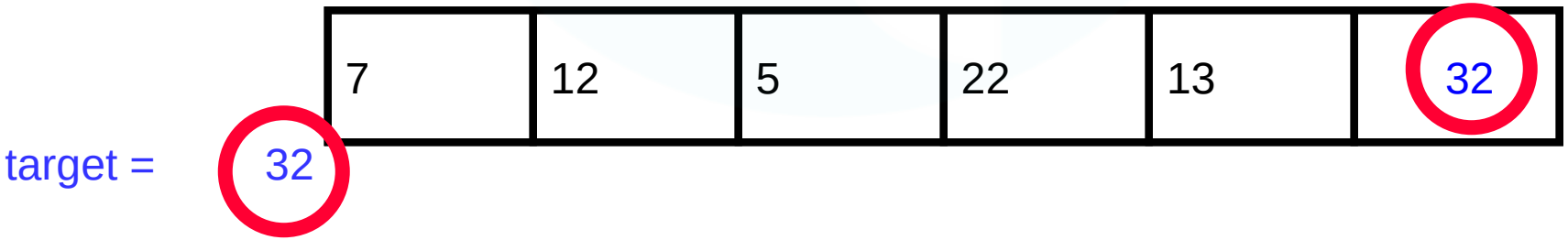
Linear Search Analysis:

Best case =O(1)
Average case =O(n)
Worst case=O(n)

Best Case: match with the first item



Worst Case: match with the last item (or no match)



Binary Search

The general term for a smart search through sorted data is a ***binary search***.

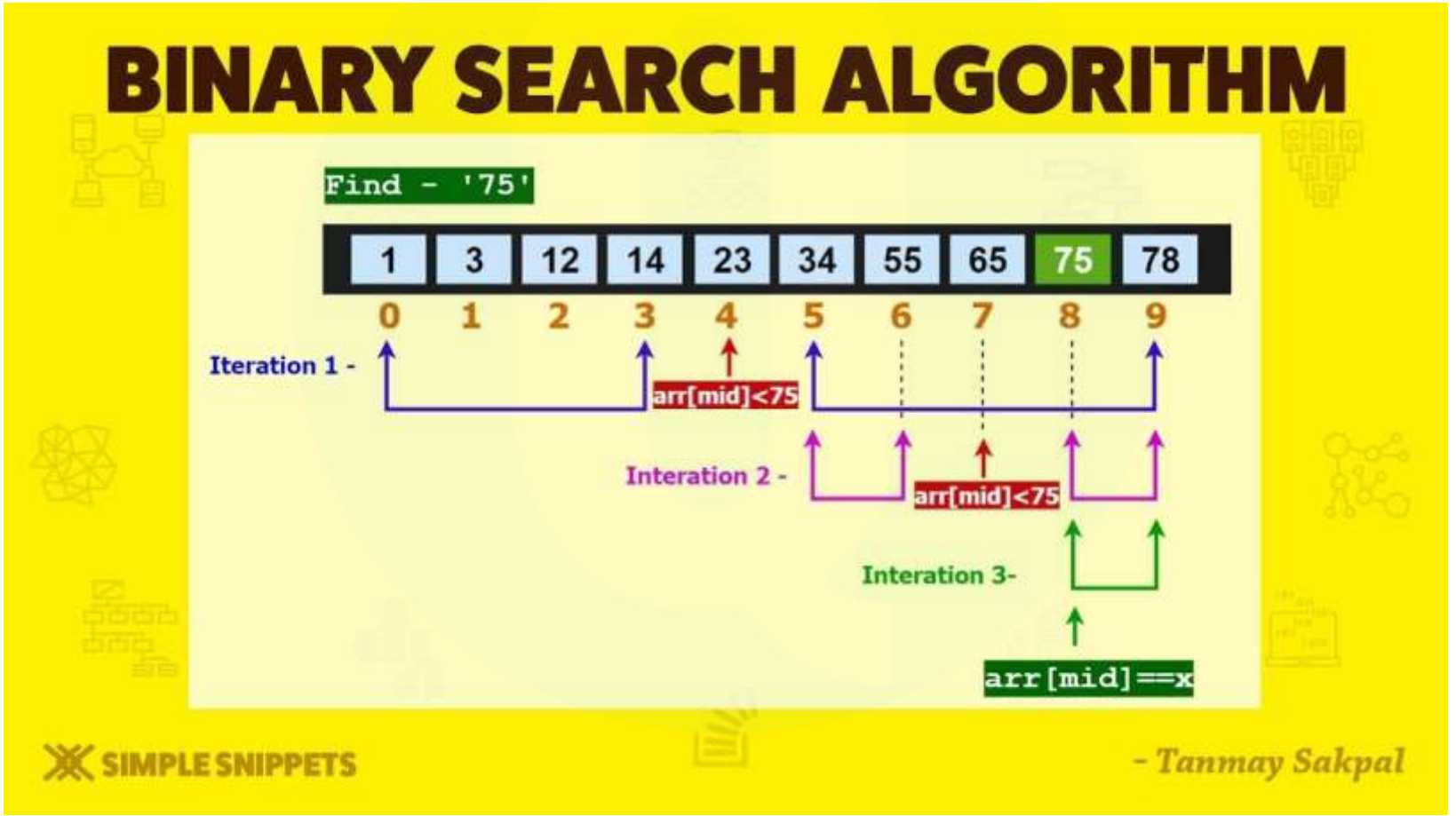
1. The initial search region is the whole array.
2. Look at the data value in the middle of the search region.
3. If you've found your target, stop.
4. If your target is less than the middle data value, the new search region is the lower half of the data.
5. If your target is greater than the middle data value, the new search region is the higher half of the data.
6. Continue from Step 2.

BINARY SEARCH

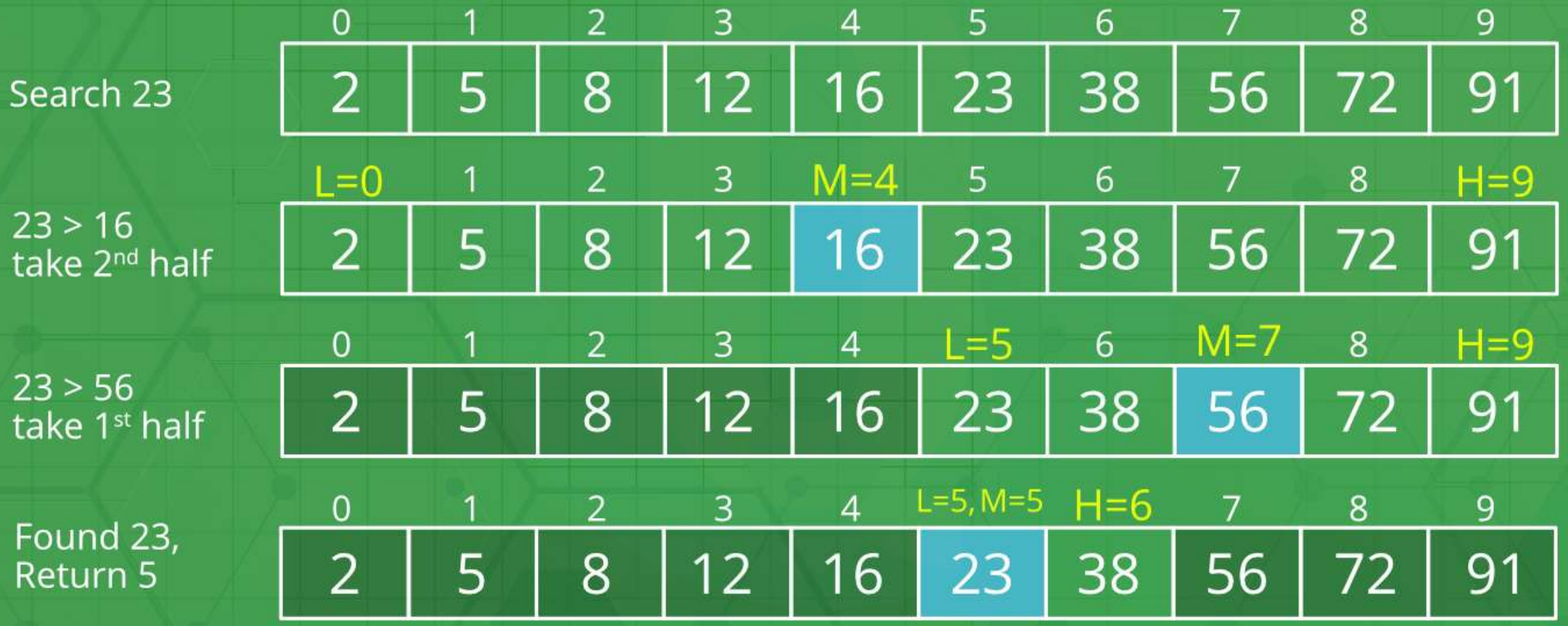
- Algorithm:

1. Input an array A of n elements an “item” to be sorted.
2. Set low:=0,high:=n,mid:=(low+high)/2.
3. Repeat step 4 & 5 while (low<=high) & (A[mid]!=item)
4. If(item<A[mid])
   high=mid-1
   else
   low=mid+1
5. mid=(low+high)/2.
6. If(A[mid]==item)
   Print”the item is found”
   Else
   Print”item not found”
7. Exit

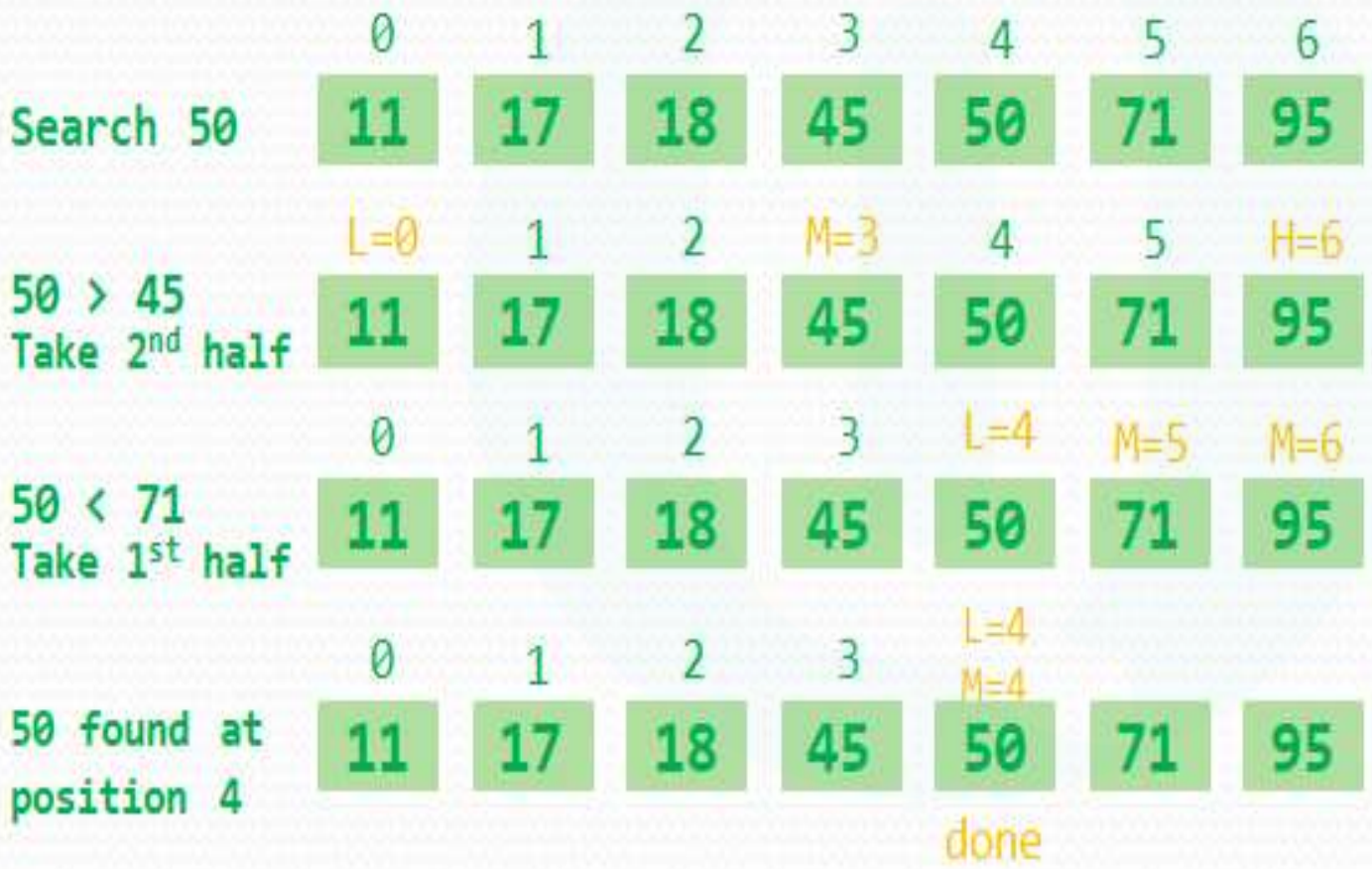
Example of binary search



Binary Search



Binary Search



Item to be searched = 23

Step 1 →

1	5	7	8	13	19	20	23	29
0	1	2	3	4	5	6	7	8

a [mid] = 13
13 < 23
beg = mid + 1 = 5
end = 8
mid = (beg + end)/2 = 13 / 2 = 6

Step 2 →

1	5	7	8	13	19	20	23	29
0	1	2	3	4	5	6	7	8

a [mid] = 20
20 < 23
beg = mid + 1 = 7
end = 8
mid = (beg + end)/2 = 15 / 2 = 7

Step 3 →

1	5	7	8	13	19	20	23	29
0	1	2	3	4	5	6	7	8

a [mid] = 23
23 = 23
loc = mid

Binary Search Analysis:

- **Best Case : O(1)**
- **Worst Case & Average case : $O(Log_2N)$**

BASIS FOR COMPARISON	LINEAR SEARCH	BINARY SEARCH
Time Complexity	O(N)	O($\log_2 N$)
Best case time	First Element O(1)	Center Element O(1)
Prerequisite for an array	No required	Array must be in sorted order
Worst case for N number of elements	N comparisons are required	Can conclude after only $\log_2 N$ comparisons
Can be implemented on	Array and Linked list	Cannot be directly implemented on linked list
Insert operation	Easily inserted at the end of list	Require processing to insert at its proper place to maintain a sorted list.
Algorithm type	Iterative in nature	Divide and conquer in nature
Usefulness	Easy to use and no need for any ordered elements.	Anyhow tricky algorithm and elements should be organized in order.
Lines of Code	Less	More

Stack

Using Arrays and Linked lists

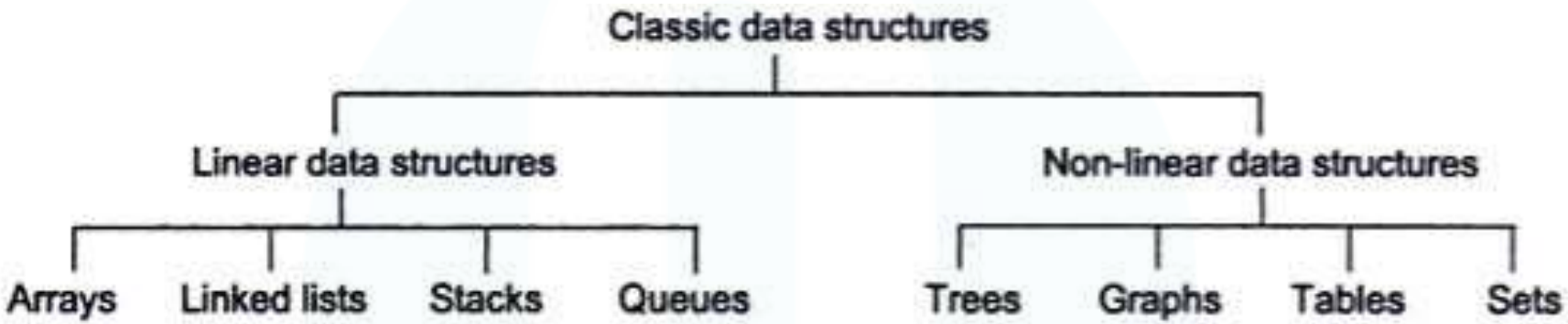


Fig. 1.2 Classification of classic data structures.

Stack

- A stack is a linear data structure, in which items are added or removed only at one end.
- It is named stack as it behaves like a real-world stack, for example – a deck of cards or a pile of plates, etc.

Stack

- Definition:
  - A stack is an ordered collection of homogeneous data elements where the insertion and deletion operations take place only at one end called *top* of the stack.

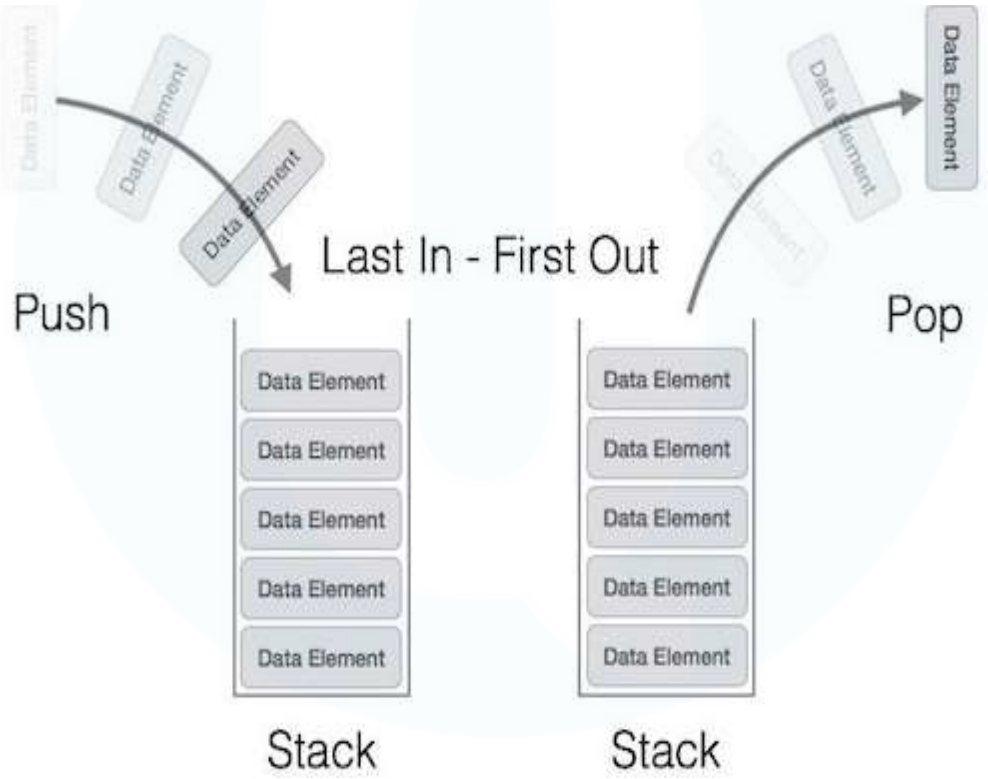
- Two basic operations of stack:
  - PUSH -> Insert an element at the top of stack
  - POP-> Delete an element from the top of stack

- LIFO
  - In stack elements are arranged in Last –In-First-Out manner
  - So it is also called LIFO lists.

Stack

- A stack is a Last In, First Out (LIFO) data structure
- Anything added to the stack goes on the “top” of the stack
- Anything removed from the stack is taken from the “top” of the stack
- Things are removed in the reverse order from that in which they were inserted

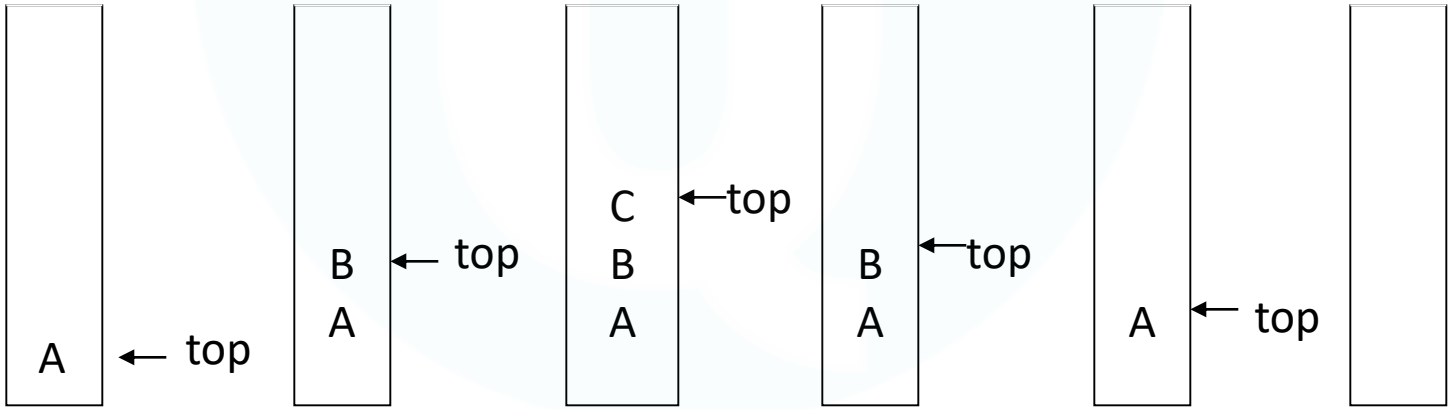
Stack


Data Element
Data Element
Data Element
Last In - First Out
Push
Pop
Data Element
Data Element
Data Element
Data Element
Data Element
Stack
Data Element
Data Element
Data Element
Data Element
Data Element
Stack

Stack operations

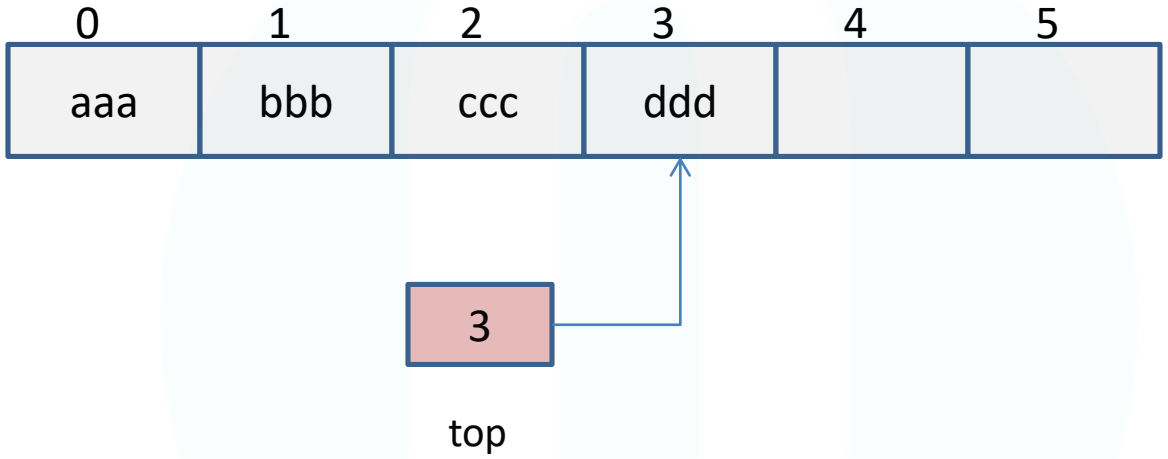
- Placing a data item on the top is called “pushing”, while removing an item from the top is called “popping” it.
- *push* and *pop* are the primary stack operations.
- An element in the stack is termed as ITEM.
- The maximum no. of elements that a stack can accommodate is termed SIZE.

Last In First Out



Array representation of stack

- Stack can be represented using a linear array.
- There is a pointer called TOP to indicate the top of the stack



- Overflow: If we try to insert a new element in the stack top(push) which is already full, then the situation is called stack overflow
- Underflow: If we try to delete an element(pop) from an empty stack, the situation is called stack underflow

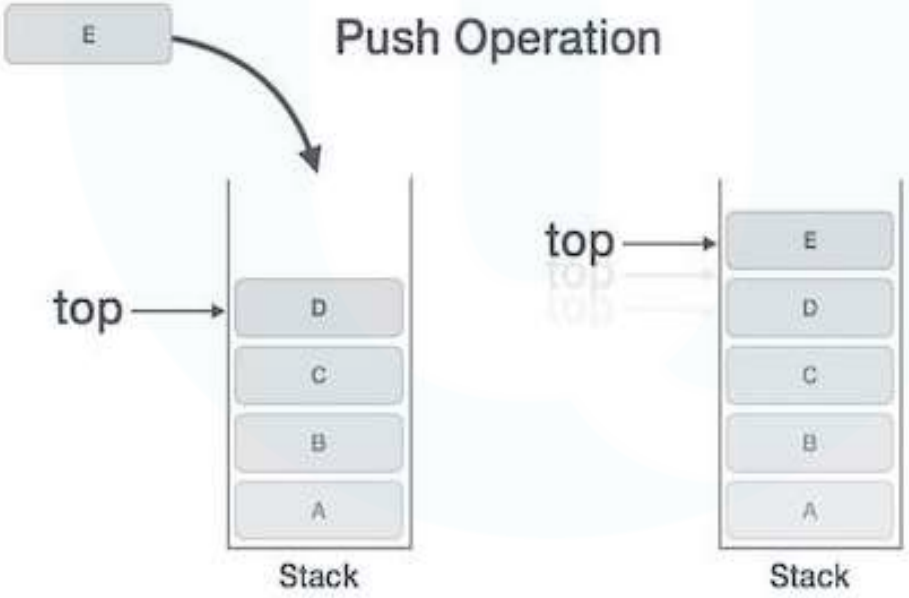
Basic Operations

- **push()** – Pushing (storing) an element on the stack.
- **pop()** – Removing (accessing) an element from the stack.
- **peek()** – get the top data element of the stack, without removing it.
- **isFull()** – check if stack is full.
- **isEmpty()** – check if stack is empty.

Push Operation

The process of putting a new data element onto stack is known as a Push Operation. Push operation involves a series of steps −

- **Step 1** − Checks if the stack is full.
- **Step 2** − If the stack is full, produces an error and exit.
- **Step 3** − If the stack is not full, increments **top** to point next empty space.
- **Step 4** − Adds data element to the stack location, where top is pointing.
- **Step 5** − Returns success.



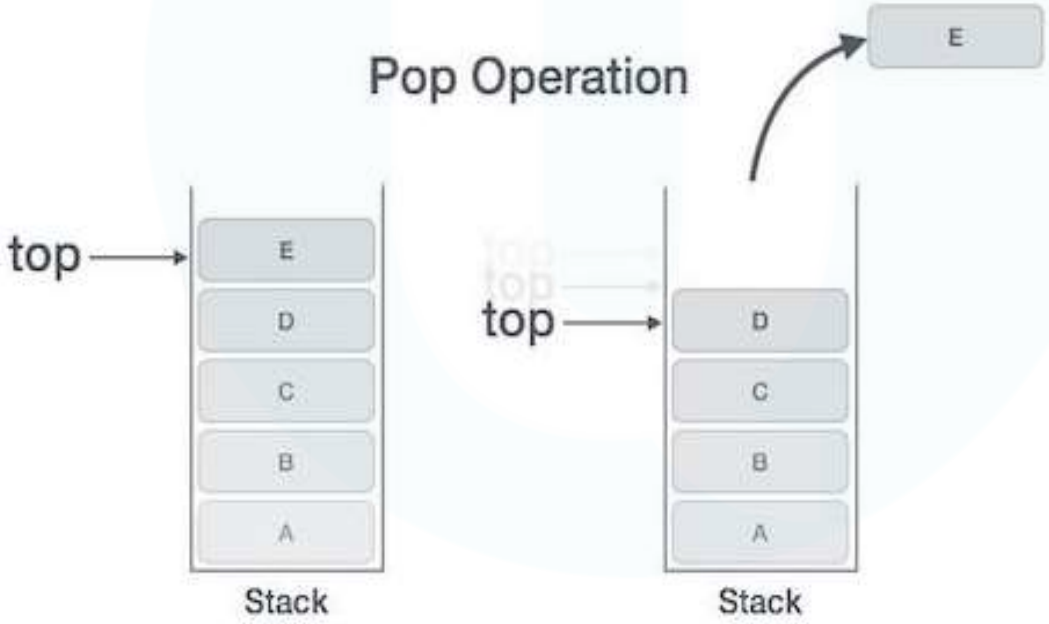
Stack using array

- Algorithm: PUSH()
- Let A be an array with Maximum size as MAXSIZE. Initially, top=-1

1. Start
2. If top< MAXSIZE
3. top=top+1
4. Read the element to be inserted on stack top as “item”
5. A[top]=item
6. else
7. print “OVERFLOW”
8. exit

A Pop operation may involve the following steps –

- **Step 1** – Checks if the stack is empty.
- **Step 2** – If the stack is empty, produces an error and exit.
- **Step 3** – If the stack is not empty, accesses the data element at which **top** is pointing.
- **Step 4** – Decreases the value of top by 1.
- **Step 5** – Returns success.



Stack using array

- Algorithm: POP()

1. Start
2. If top= -1
3. print “UNDERFLOW” [Check for Underflow]
4. Else
5. item=A[top] //Assign the element to be deleted as “item”
6. top=top-1
7. Print the deleted item
8. exit

Applications of stack

- **Reversing an array**
- A B C D
- Pushing to stack A B C D
- Popping from stack D C B A
- **Undo operations**
- **Infix to prefix,infix to postfix expressions**
- **Tree Traversal**
- **Evaluation of postfix expressions**