Problem Solving Through Programming (18ESCS01)

Unit-3: Arrays & strings

MANJUNATH C R

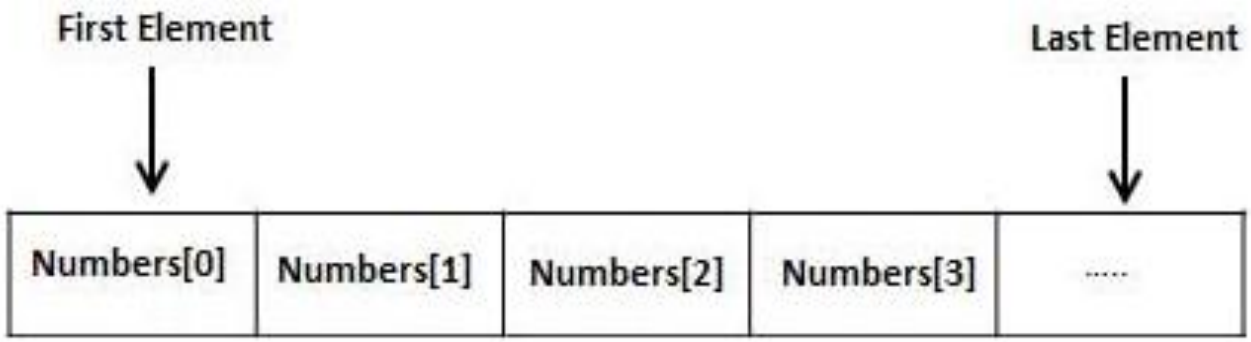
ARRAYS AND STRINGS

- ARRAYS
- DECLARATION OF ARRAYS
- INITIALIZATION OF ARRAYS
- ONE-DIMENTIONAL & TWO-DIMENTIONAL ARRAYS
- STRINGS
- STRING OPERATIONS
- STRING ARRAYS
- SIMPLE OPERATIONS – SORTING & SEARCHING – MATRIX OPERATIONS

MANJUNATH C R

ARRAYS

- Array is a data structure that can store a fixed-size sequential collection of elements of the same type.
- An array is defined as **finite ordered collection of homogenous** data (of same type), stored in contiguous memory locations.
- The lowest address corresponds to the first element and the highest address to the last element.



- Examples where arrays are applicable:
  - Used to store list of Employee or Student names.
  - Used to store marks of students,
  - Or to store list of numbers or characters etc.

MANJUNATH C R

DECLARATION OF ARRAYS

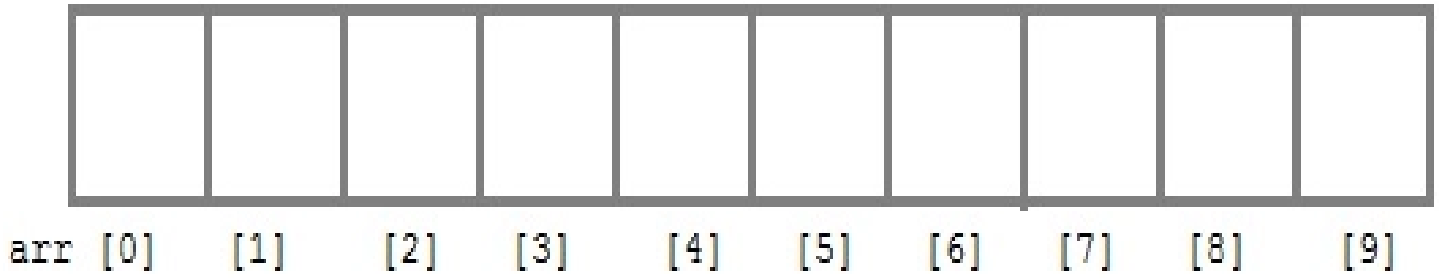
- General form of array declaration is:

```
data-type array_name[array_size];

/* Example of array declaration */

        int arr[10];
```

- **int** -> is the data type,

  arr ->is the name of the array and

  10 -> is the size of array [**0 to size – 1**]. It means array 'arr' can only contain 10 elements of int type.



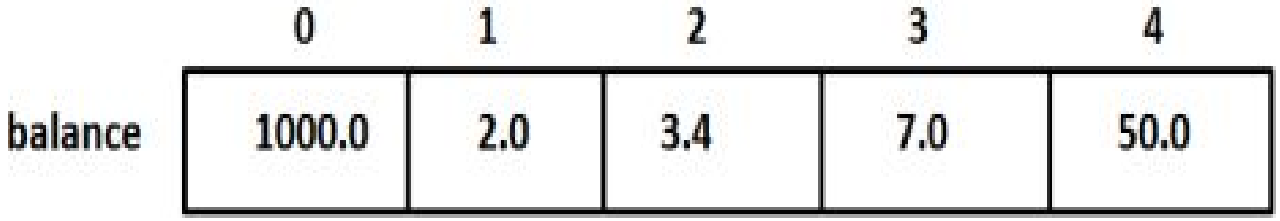
MANJUNATH C R

INITIALIZATION OF ARRAYS

- We can initialize an array in 'C' either one by one or using a single statement as:

```
int a[5] = {10, 20, 30, 40, 50};

float balance[5] = {1000.0, 2.0, 3.4, 7.0, 50.0};
```

- The number of values between braces { } cannot be larger than the number of elements that we declare for the array between square brackets [ ].



INITIALIZATION OF ARRAYS (CONTD.)

- An array can be initialized at either **compile time** or at **run time**.

- **Example for compile time:**

```
#include<stdio.h>
void main( )
{
 int i;
 int arr[ ] = {2, 3, 4}; //Compile time array initialization
 for(i = 0 ; i < 3 ; i++)
    {
      printf("%d\t", arr[i]);
     }
 }
```

Output:

2 3 4

MANJUNATH C R

INITIALIZATION OF ARRAYS (CONTD.)

- **Example for run time:**

```
#include<stdio.h>
void main( )
{
 int arr[3];  int i, j;
 printf("Enter array element");
 for(i = 0; i < 3; i++)
    {
     scanf("%d", &arr[i]);   //Run time array initialization
     }
 for(j = 0; j < 3; j++)
    {
     printf("%d\n", arr[j]);
     }
}
```

Output:

2 3 4

MANJUNATH C R

EXAMPLE – 1: DISPLAY ELEMENTS ELEMENT'S

```
#include <stdio.h>
int main ( )
{
int n[ 10 ];  /* n is an array of 10 integers */
int i, j;   /* initialize elements of array n to 0 */
for ( i = 0; i < 10; i++ )
   {
   n[ i ] = i + 100;   /* set element at location i to i + 100 */
   }   /* output each array element's value */
for (j = 0; j < 10; j++ )
   {
   printf("Element[%d] = %d\n", j, n[j] );
   } return 0;
}
```

Output:

```
Element[0] = 100
Element[1] = 101
Element[2] = 102
Element[3] = 103
Element[4] = 104
Element[5] = 105
Element[6] = 106
Element[7] = 107
Element[8] = 108
Element[9] = 109
```


MANJUNATH C R

Different ways of initialization of Arrays

(a) Initializing all specified memory locations

(b) Partial array initialization

(c) Initialization without size

(d) Array initialization with a string

MANJUNATH C R

(A) INITIALIZING ALL SPECIFIED MEMORY LOCATION

- **Arrays can be initialized at the time of declaration when their initial values are known in advance.**

- **Array elements can be initialized with data items of type int, char etc.**

- **Ex:- int a[5] = {10, 15, 1, 3, 20};**

a[0]	a[1]	a[2]	a[3]	a[4]
10	**15**	**1**	**3**	**20**
1000	1002	1004	1006	1008

- During compilation, 5 contiguous memory locations are reserved by the compiler for the variable **'a' and all these locations are initialized as shown in figure.**

MANJUNATH C R

(B) PARTIAL ARRAY INITIALIZATION

- **If the number of values to be initialized is less than the size of the array, then the elements will be initialized to zero automatically.**

- **Ex:- int a[5] = {10, 15};**

a[0]	a[1]	a[2]	a[3]	a[4]
10	**15**	**0**	**0**	**0**
1000	1002	1004	1006	1008

- Eventhough compiler allocates 5 memory locations, using this declaration statement; the compiler initializes first two locations with 10 and 15, the next set of memory locations are automatically initialized to 0's by compiler as shown in figure.

MANJUNATH C R

(C) INITIALIZATION WITHOUT SIZE

- Consider the declaration along with the initialization:

- **Ex:- char b[ ] = {'C','O','M','P','U','T','E','R'};**

b[0]	b[1]	b[2]	b[3]	b[4]	B[5]	b[6]	b[7]
C	**O**	**M**	**P**	**U**	**T**	**E**	**R**
1000	1001	1002	1003	1004	1005	1006	1007

- In this declaration, eventhough we have not specified exact number of elements to be used in array 'b', the array size will be set of the total number of initial values specified.
- So, the array size will be set to 8 automatically. The array 'b' is initialized as shown in figure.

(D) ARRAY INITIALIZATION WITH A STRING

- Consider the declaration with string initialization.

- **Ex:- char b[ ] = "COMPUTER";**

b[0]	b[1]	b[2]	b[3]	b[4]	B[5]	b[6]	b[7]	b[8]
C	**O**	**M**	**P**	**U**	**T**	**E**	**R**	**\0**
1000	1001	1002	1003	1004	1005	1006	1007	1008

- Eventhough the string "COMPUTER" contains 8 characters, because it is a string, it always ends with null character.
- So, the array size is 9 bytes (i.e., string length 1 byte for null character).

ONE-DIMENTIONAL & TWO-DIMENTIONAL ARRAYS

(a) One-dimensional arrays:

- A **one-dimensional array** (or **single dimension array**) is a type of linear **array**. An array with a single subscript is known as one dimensional array.

- Accessing its elements involves a **single** subscript which can either represent a row or column index.
- Syntax:

Data type array_name[array_size];

- Example: The **array** can contain 10 elements of any value available to the various data type is **:**

int a[10];

float a[10];

double a[10.0];

MANJUNATH C R

ONE-DIMENTIONAL & TWO-DIMENTIONAL ARRAYS

(b) Two-dimensional arrays:

- An array consisting of two subscripts is known as two-dimensional array.
- These are often known as array of the array.
- In two dimensional arrays, the array is divided into **rows** and **columns**.
- Syntax: **Data type array_name[array_size][column_size];**
- Example: **int a[3][3];** where first index value shows the number of the rows and second index value shows the number of the columns in the array.

MANJUNATH C R

MULTI-DIMENTIONAL ARRAYS

(b) Multi-dimensional arrays:

- In multidimensional arrays the array is divided into rows and columns, mainly applicable for 3-D array, 4-D arrays., etc.

- Syntax:

Data type array_name[size1][size2][size3].....[sizeN];

- Example: **int a[2][3][4];**

float a[5][4][5][3];

MANJUNATH C R

STRINGS

- In C language a string is group of characters (or) array of characters, which is terminated by **delimiter '\0' (null).**
- C uses variable-length delimited strings in programs.
- C does not support string as a data type. It allows us to represent strings as character arrays.
- Strings **output** is given as: **printf("%s", S);**
- Strings as **input** is given as: **scanf("%s", S);**

MANJUNATH C R

DECLARATION OF STRINGS

- In C, a string variable is any valid C variable name and is always declared as an array of characters.

- **Syntax:-**

```
char string_name[size];
```

- The **size** determines the number of characters in the string name.

- **Example:-**

```
char city[10];
char name[30];
```

INITIALIZATION OF STRINGS

- There are several methods to initialize values for string variables.

- Example 1: **char str[6] = "HELLO";**

H	E	L	L	O	\0

- Example 2: **char month[ ] = "JANUARY";**

J	A	N	U	A	R	Y	\0

- Example 3: **char city[8] = "NEWYORK";**

N	E	W	Y	O	R	K	\0

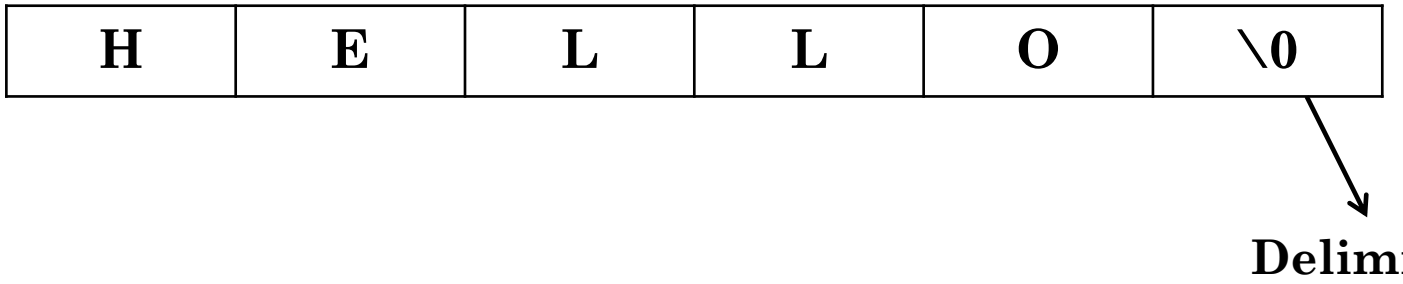
- Example 4:

char city[8] = {'N', 'E', 'W', 'Y', 'O', 'R', 'K', '\0'};

STRINGS STORED IN MEMORY

- In C a string is stored in an array of characters and terminated by '\0' (null).

- Ex:



- A string is stored in array, the name of the string is a pointer to the beginning of the string.
- The character requires only one memory location. If we use one-character string it requires two locations:
  - **a character ->** H
  - **One-character string ->** H \0
  - **Empty String ->** \0

DIFFERENCE BETWEEN STRINGS & ARRAY

- The difference between array and string is shown below:

- **STRING:**

H	E	L	L	O	\0

- **ARRAY:**

H	E	L	L	O

- Because strings are variable-length structure, we must provide enough space for maximum-length string to store and one byte for delimiter.

MANJUNATH C R

Example Program for Strings

```
#include <stdio.h>
int main ( )
{
char greeting[6] = {'H', 'E', 'L', 'L', 'O', '\0'};
printf("Greeting message: %s\n", greeting );
return 0;
}
```

Output:

Greeting message: HELLO

STRING OPERATIONS

(a) strlen(s1) – Returns the length of string s1.

(b) strcpy(s1, s2) – Copies string s2 into string s1.

(c) strcmp(s1, s2) – Returns 0 if s1 and s2 are the same,
less than 0 if s1 < s2,
greater than 0 if s1 > s2

(d) strcat(s1, s2) – Concatenates string s2 onto the end of string s1.

MANJUNATH C R

STRING OPERATIONS – EXAMPLE PROGRAM

```
#include <stdio.h>
#include <string.h>
int main ( )
{
char str1[12] = "Hello";
char str2[12] = "World";
char str3[12];
int len ;
 strcpy(str3, str1); printf("strcpy( str3, str1) : %s\n", str3 );
        /* copy str1 into str3 */
strcat( str1, str2); printf("strcat( str1, str2): %s\n", str1 );
        /* concatenates str1 and str2 */
len = strlen(str1); printf("strlen(str1) : %d\n", len );
        /* total length of str1 after concatenation */
return 0;
}
```

Output:

strcpy(str3, str1) : Hello
strcat(str1, str2): HelloWorld
strlen(str1) : 10

MANJUNATH C R

STRING ARRAYS

- We have array of integers, array of floating point numbers, etc.. similarly we have array of strings also.

- **Collection of strings is represented using array of strings.**

- **Declaration:-**

```
Char arr[row][col];
```

where, arr - name of the array,

row - represents number of strings,

col - represents size of each string.

- **Initialization:-**

```
char arr[row][col] = {list of strings};
```

STRING ARRAYS - EXAMPLE

- **Example:-**

```
char city[5][10] = { "DELHI", "CHENNAI", "BANGALORE",
                     "HYDERABAD", "MUMBAI" };
```

D	E	L	H	I	\0				
C	H	E	N	N	A	I	\0		
B	A	N	G	A	L	O	R	E	\0
H	Y	D	E	R	A	B	A	D	\0
M	U	M	B	A	I	\0			

- **In the above storage representation, memory is wasted due to the fixed length for all strings.**

MANJUNATH C R

SIMPLE OPERATIONS –SEARCHING

- Searching Algorithms are designed to check for an element or retrieve an element from any data structure where it is stored.

- **Searching Techniques:**
  - **Linear Search**
  - **Binary Search**

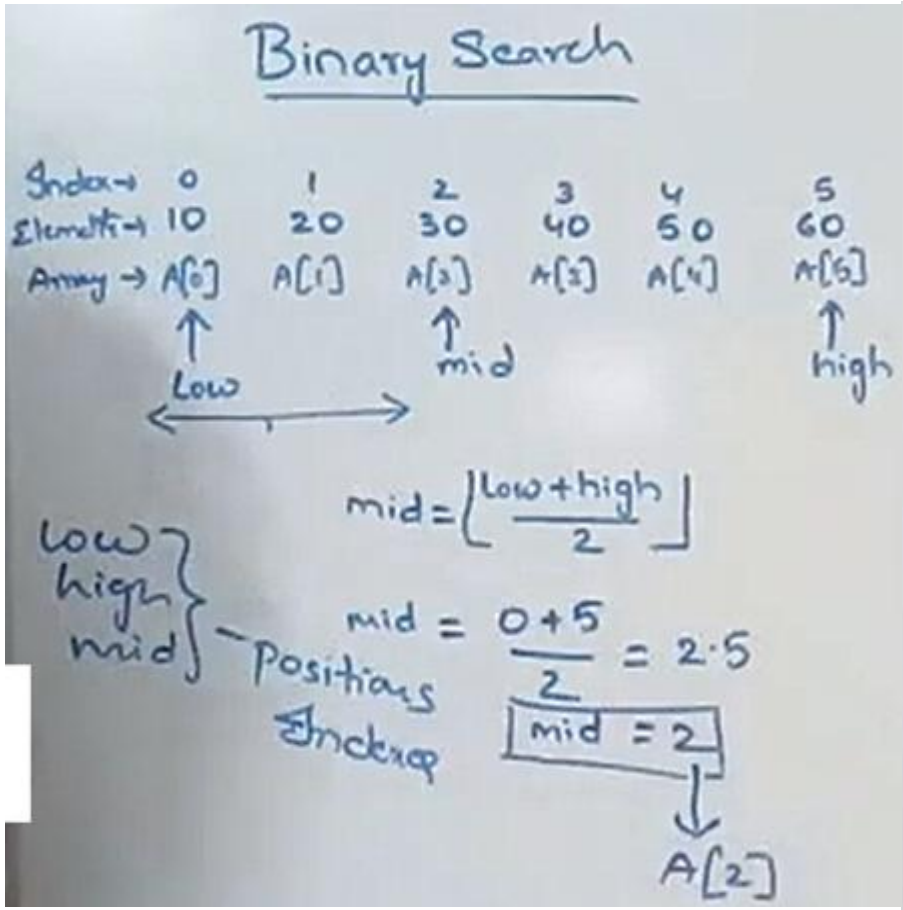
LINEAR SEARCH

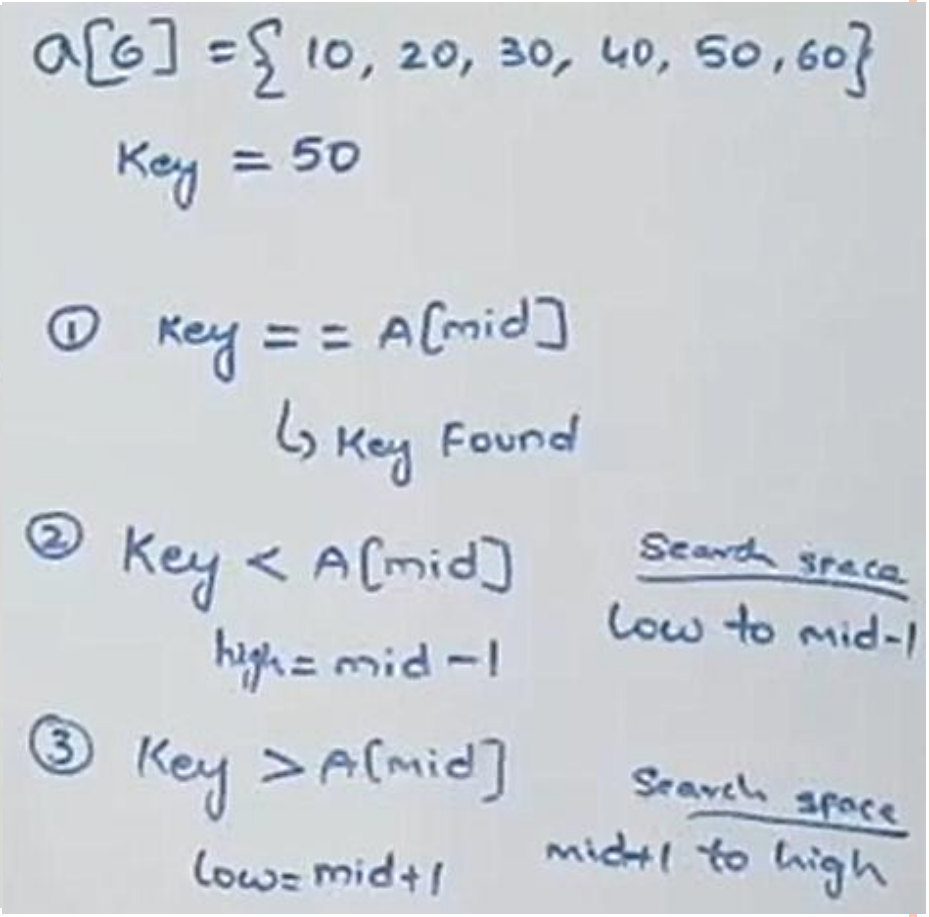
- Linear search is a very simple and basic search algorithm.
- A linear search, also known as a sequential search, is a method of finding an element within a list.
- It checks each element of the list sequentially until a match is found or the whole list has been searched.
- The time required to search an element using a linear search algorithm depends on the size of the list.
- The time complexity of a **linear search is O(n)**.

MANJUNATH C R

BINARY SEARCH

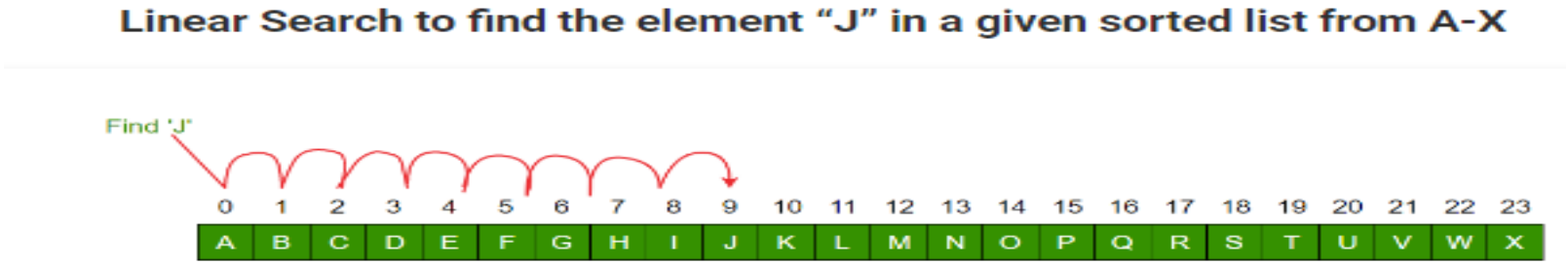
- **Binary search** is the most popular Search algorithm.
- It is efficient and also one of the most commonly used techniques that is used to solve problems.
- The time complexity of a **linear search is O(log n)**.

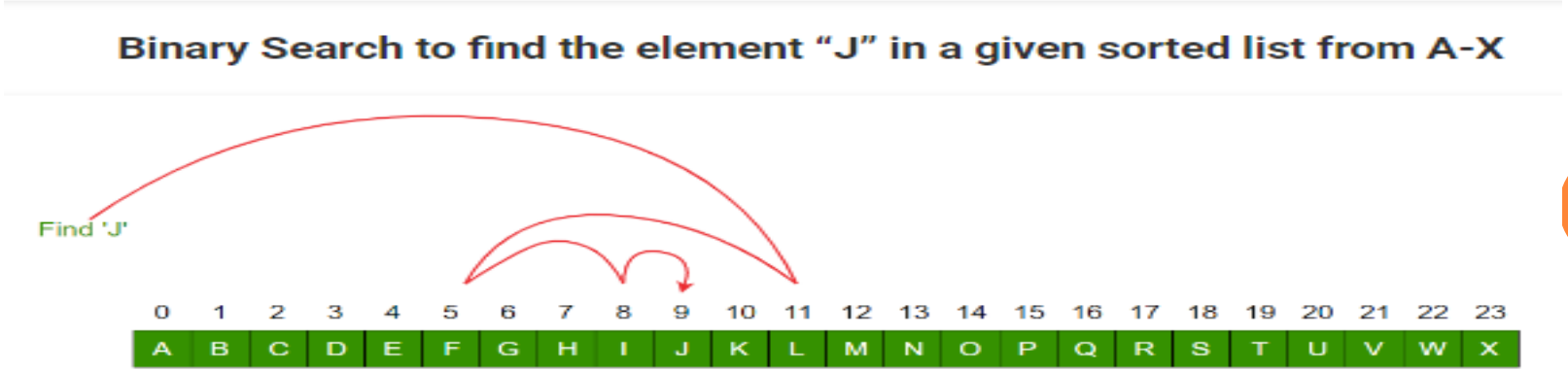




DIFFERENCE BETWEEN LINEAR AND BINARY SEARCH

- Input data needs to be sorted in Binary Search and not in Linear Search
- Linear search does the sequential access whereas Binary search access data randomly.
- Time complexity of linear search -O(n) , Binary search has time complexity O(log n).
- Linear search performs equality comparisons and Binary search performs ordering comparisons





MANJUNATH C R

SIMPLE OPERATIONS – SORTING

- A Sorting Algorithm is used to **rearrange a given array** or list elements according to a comparison operator on the elements.

- The comparison operator is used to decide the new order of element in the respective data structure.

- **Sorting Techniques:**
  - **Insertion Sort**
  - **Bubble Sort**

SIMPLE OPERATIONS –MATRIX OPERATIONS

- **Matrix operations:**
  - **Matrix Addition / Matrix Subtraction**
  - **Matrix Multiplication**

MANJUNATH C R