Arrays

Arrays is a kind of data structure that can store a elements of the same type. Arrays stores the elements in a contiguous memory locations.

Array is a collection of variables of the same type.

For example: we want to declare 100 integer variable

Instead of declaring 100 individual variables, such as

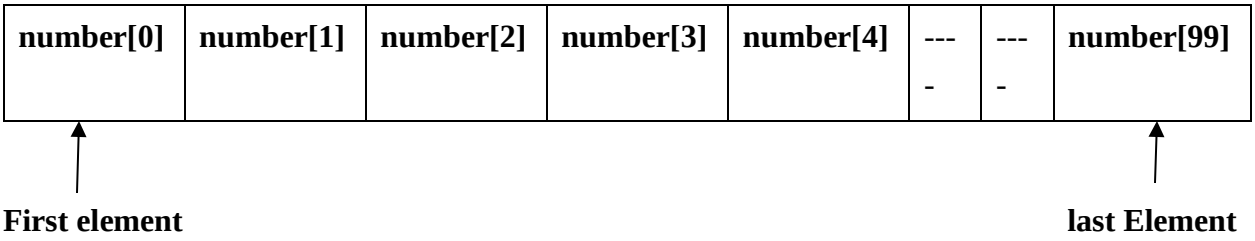
int number0, number1,number2,number3, ... …., number99;

you declare one integer array variable such as

int number[100];

here **number[100]** is an integer array of size 100, means this array can store 100 integer value.

Array indexing starts from index 0 to n-1. Means first integer number store in array number[0], second integer number store in array number[1], third integer number store in array number[2], and so on. Here 100^{th} number store in array numbers[99].



Declaring Arrays

Syntax of declare an array in C:

```
Data_type  arrayName [ arraySize ];
```

This is called a single-dimensional array.

The **arraySize** must be an integer constant greater than zero.

datatype can be any valid C data type.

Example: write a program to create integer array and store 5 integer number and print.

```
#include <stdio.h>
void main()
{       int  num[5];            // array num can store 5 integer value
        num[0] = 20;            // store 20 at array index 0
        num[1] = 11;            // store 11 at array index 1
        num[2] = 20;            // store 20 at array index 2
        num[3] = 33;            // store 33 at array index 3
        num[4] = 55;            // store 55 at array index 55

        printf("num[0]= %d" , num[0]);
        printf("num[1]= %d" , num[1]);
        printf("num[2]= %d" , num[2]);
        printf("num[3]= %d" , num[3]);
        printf("num[4]= %d" , num[4]);

}
```

Output:
```
num[0] = 20;
num[1] = 11;
num[2] = 20;
num[3] = 33;
num[4] = 55;
```

Description : In the above program, "int num[5]" array can store 5 element in a contiguous memory locations from index 0 to 4.

num[0]	num[1]	num[2]	num[3]	num[4]
20	**11**	**20**	**33**	**55**

First element

last Element

Example: write a program to take 5 number from user and store integer number in integer array and print.

```
#include <stdio.h>
int main()
{       int i;
        int  num[5];                    // array num can store 5 integer value
        printf("Enter 5 integer number");
        for(i = 0; i<5; i++ )           // for loop to take 5 integer number from user
        {       scanf("%d", &num[i]);
        }

        printf("\n Entered number is");
        for(i = 0; i<5; i++ )           // for loop to take 5 integer number from user
        {       printf("\n num[%d] =%d", i, num[i]);
        }

return 0;
}
```

Output: Enter 5 integer number
5 // enter by user
4
3
2
1
Entered number is
num[0] = 5;
num[1] = 4;
num[2] = 3;
num[3] = 2;
num[4] = 1;

Another way to Initialize Array

We can also initialize array by this way:

int num[5] = { 54 , 4, 13, 2, 17 };

above integer array num[5], stores five numbers. Where num[0] store 54, num[1] store 4….. , & num[4] store 17.

Advantages of Array-

- collection of similar types of data.
- used to implement other data structure like linked lists,stack,queue, trees, graph etc.
- Create multi dimension array.
- 2 Dimensional array is used to represent a matrix.
- In array, elements can be accessed randomly by using index number.
- All elements of array are stored in the contiguous memory locations therefore no extra memory is needed to store the address of next element like link list..

Multi-dimensional Arrays

C programming language also support multidimensional arrays.

Syntax of multidimensional array declaration:

```
Data_type    array_name[size1][size2]...[sizeN];
```

For example, if we want to creates a three dimensional integer array −

```
int  num[5][10][4];
```

Two-dimensional Arrays

The simplest form of multidimensional array is the two-dimensional array. if we want to declare a two dimensional integer array of size [x][y] (where x is a number of rows and y is a number of columns) you would write something as follows –

```
int a[3][4];
```

In the above line we have created two-dimensional integer array ”a” with 3 rows and 4 columns.

	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]

Initializing Two-Dimensional Arrays

In a C programming language Multidimensional arrays may be initialized by specifying bracketed values for each row. For example we want to create an integer array with 3 rows and each row has 4 columns.

```
int a[3][4] = {
  { 1, 2, 3, 4 } ,   /*  for row index 0 */
  { 5, 6, 7, 8 } ,   /*  for row index 1 */
  { 9, 10, 11, 12}   /*  for row index 2 */
};
```

Above the nested braces indicates the intended rows.

The following initialization is equivalent to the previous example −

```
int a[3][4] = {1,2,3,4,5,6,7,8,9,10,11,12};
```

which indicate the nested braces for intended row, are optional.

Accessing Elements of Two-Dimensional Array

Create a two-dimensional array of size 3*3 and nested loop is used to handle a two-dimensional array −

```
#include <stdio.h>
int main ()
{

  /* an array with 3 rows and 3 columns*/
  int a[3][3] = { {1,2,3}, {4,5,6}, {7,8,9}};
  int i, j;

  /* output each array element's value */
            for ( i = 0; i < 3; i++ )
            {
                        for ( j = 0; j < 3; j++ )
                        {
                                    printf("a[%d][%d] = %d\n", i,j, a[i][j] );
                        }
            }
  return 0;
}
```

Output:

```
a[0][0] = 1
a[0][1] = 2
a[0][2] = 3
a[1][0] = 4
a[1][1] = 5
a[1][2] = 6
a[2][0] = 7
```

```
a[2][1] = 8
a[2][2] = 9
```

Example: Write a program to create 3*3 matrix and take all elements of matrix as an input from user and print it.

```
#include <stdio.h>
 int main ()
{
  int a[3][3];
  int i, j;
            printf(“Enter 9 elements of 3*3 matrix in sequence”);
            for ( i = 0; i < 3; i++ )          /* input each array element's value */
            {
                        for ( j = 0; j < 3; j++ )
                        {            scanf("%d", &a[i][j] );
                        }
            }
            printf(“Elements of 3*3 matrix is \n”);

            /* output each array element's value */
            for ( i = 0; i < 3; i++ )
            {
                        for ( j = 0; j < 3; j++ )
                        {
                                    printf("%d", a[i][j] );
                        }
            printf("\n");
            }
     return 0;
}
```

Output:

```
Enter 9 elements of 3*3 matrix in sequence
1
2
3
4
5
6
7
8
9
Elements of 3*3 matrix is
1       2       3
4       5       6
7       8       9
```

Example: Write a program to create square matrix (3*3, 4*4, 5*5 matrix etc.) and print the diagonal elements of square matrix.

```
#include <stdio.h>
 int main ()
{
  int a[3][3];
  int i, j;
            printf(“Enter 9 elements of 3*3 matrix in sequence”);
            for ( i = 0; i < 3; i++ )          /* input each array element's value */
            {
                        for ( j = 0; j < 3; j++ )
                        {           scanf("%d", &a[i][j] );
                        }
            }
            printf(“Diagonal Elements of 3*3 matrix is \n”);

            /* output each array element's value */
```

```
            for ( i = 0; i < 3; i++ )
            {
                        for ( j = 0; j < 3; j++ )
                        {           if( i = = j )
                                     {     printf("%d", a[i][j] );  }
                        }
            printf("\n");
            }
     return 0;
}
```

Output:

```
Enter 9 elements of 3*3 matrix in sequence
1
2
3
4
5
6
7
8
9
Elements of 3*3 matrix is
1
      5
            9
```

Addition of Two Matrix

Example: Write a program to add two matrix. Firstly, ask from user order of matrix(number of rows and column). Then take the elements of matrix from user as a input and print the resultant matrix

For example, if a user input order as 3,3, i.e., three rows and three columns and
First matrix

```
1       2       3
4       5       6
7       8       9
Second matrix:
9       8       7
6       5       4
3       2       1
```

then the output of the program (addition of the two matrices) is:

```
10      10      10
10      10      10
10      10      10
```

Matrix addition program in C

```
#include <stdio.h>
 int main()
{
   int m, n, i, j, f[10][10], s[10][10], sum[10][10];

   printf("Enter the number of rows and columns of matrix\n");
   scanf("%d%d", &m, &n);
   printf("Enter the elements of first matrix\n");

              for (i = 0; i < m; i++)
              {             for (j = 0; j < n; j++)
                            {             scanf("%d", &f[i][j]);
                            }
              }
   printf("Enter the elements of second matrix\n");

              for (i = 0; i < m; i++)
```

```
        {       for (j = 0 ; j < n; j++)
                {       scanf("%d", &s[i][j]);
                }
        }
  printf("Sum of matrices= \n");

        for (i = 0; i < m; i++)
        {
                for (j = 0 ; j < n; j++)
                {
                        sum[i][j] = f[i][j] + s[i][j];
                        printf("%d\t", sum[i][j]);
                }
        printf("\n");
        }
 return 0;
}
```

Output:

```
Enter the number of rows and columns of matrix
3                                                   // enter by user
3                                                   // enter by user
Enter the elements of first matrix
1
2
3
4
5
6
7
8
9
Enter the elements of second matrix
9
8
7
6
```

```
5
4
3
2
1
Sum of matrices=
10      10      10
10      10      10
10      10      10
```

Matrix multiplication

Example: Write a program to multiply two matrix. Firstly, ask from user order of matrix(number of rows and column). Then take the elements of matrix from user as a input and print the resultant matrix.

```
#include <stdio.h>
int main()
{
  int m, n, p, q, i, j, k, sum = 0;
  int f[10][10], s[10][10], multiply[10][10];

  printf("Enter number of rows and columns of first matrix\n");
  scanf("%d%d", &m, &n);
  printf("Enter elements of first matrix\n");

            for (i = 0; i < m; i++)
            {           for (j = 0; j < n; j++)
                        {           scanf("%d", &f[i][j]);
                        }
            }
  printf("Enter number of rows and columns of second matrix\n");
  scanf("%d%d", &p, &q);
```

```
  if (n != p)
{
   printf("The matrices can't be multiplied with each other.\n");
  }
else
  {
   printf("Enter elements of second matrix\n");

            for (i = 0; i < p; i++)
            {           for (j = 0; j < q; j++)
                        {           scanf("%d", &s[i][j]);
                        }
            }

            for (i = 0; i < m; i++)
             {
                        for (j = 0; j < q; j++)
                        {
                                    for (k = 0; k < p; k++)
                                    {
                                                sum = sum + f[i][k]*s[k][j];
                                    }
                        multiply[i][j] = sum;
                        sum = 0;
                        }
            }

   printf("Product of the matrices:\n");

            for (i = 0; i < m; i++)
            {
                        for (j = 0; j < q; j++)
                        {           printf("%d\t", multiply[i][j]);
                        }
            printf("\n");
            }
```

```
  }

  return 0;
}
```

Output:

```
Enter number of rows and columns of first matrix
3
3
Enter elements of first matrix
2
2
2
2
2
2
2
2
2
Enter number of rows and columns of second matrix
3
3
Enter elements of second matrix
2
2
2
2
2
2
2
2
2
Product of the matrices:
12      12      12
12      12      12
12      12      12
```

MATRIX TRANSPOSE

Example: Write a program to create 3*3 matrix and take all elements of matrix as an input from user and find the transpose of the matrix & print it.

```
#include <stdio.h>
 int main ()
{
  int a[3][3];
  int i, j;
            printf(“Enter 9 elements of 3*3 matrix in sequence”);
            for ( i = 0; i < 3; i++ )          /* input each array element's value */
            {
                        for ( j = 0; j < 3; j++ )
                        {             scanf("%d", &a[i][j] );
                        }
            }

      for(i=0; i<r; ++i)       // Finding the transpose of matrix a
       {  for(j=0; j<c; ++j)
         {
             transpose[j][i] = a[i][j];
         }
       }
            printf(“Transpose matrix is \n”);


            for ( i = 0; i < 3; i++ )
            {
                        for ( j = 0; j < 3; j++ )
                        {
                                    printf("%d", a[i][j] );                    //print  element of matrix
                        }
            printf("\n");
            }
     return 0;
```

```
}
```

Output:

```
Enter 9 elements of 3*3 matrix in sequence
1
2
3
4
5
6
7
8
9
Transpose matrix is
1	4	7
2	5	8
3	6	9
```

String and Character Array

String is a sequence of characters that is treated as a single data item and terminated by null character `'\0'`.

Remember that C language does not support strings as a data type.

A **string** is actually one-dimensional array of characters in C language.

For example: The string "hello world" contains 12 characters including `'\0'` character which is automatically added by the compiler at the end of the string.

Declaring and Initializing a string variables

There are different ways to initialize a character array variable.

char name[13] = “StudyTonight”; // valid character array initialization

char name[6] = {‘h’, ‘e’, ‘l’, ‘l’, ‘o’, ‘\0’ }; // valid character array initialization

Remember that when you initialize a character array by listing all of its characters separately then you must supply the `'\0'` character explicitly.

Some examples of illegal initialization of character array are,

char name[4] = “Study”; // invalid character array initialization

char name[5];

name = “Study”; // invalid character array initialization

String Input and Output

Input function `scanf()` can be used with **%s** format specifier to read a string input from the terminal. But there is one problem with `scanf()` function, it terminates its input on the first white space it encounters.

However, C supports a format specification known as the **edit set conversion code %[..]** that can be used to read a line containing a variety of characters, including white spaces.

```
#include<stdio.h>
#include<string.h>
void main()
{
   char str[20];
   printf("Enter a string :");
   scanf("%[^\n]", &str);  //scanning the whole string, including the white spaces
   printf("%s", str);
   getch();
}
```

Output : Enter a string : I am student
I am student

Another method to read character string with white spaces from terminal is by using the gets() function.

```
#include<stdio.h>
#include<string.h>
void main()
```

```
{
   char str[20];
   printf("Enter a string :");
   gets(str);                    //scanning the whole string, including the white spaces
   printf("%s", str);
   getch();
}
```
Output : Enter a string : I am student
I am student

```
// C program to read strings

#include<stdio.h>
 void main()
{
   // declaring string
   char str[50];

   // reading string
   scanf("%s",str);

   // print string
   printf("%s",str);
 }
```

Output : Hello
Hello

Passing strings to function:

As strings are character arrays, so we can pass strings to function in a same way we pass an array to a function. Below is a sample program to do this:

```
Example: C program to illustrate how to pass string to functions.

#include<stdio.h>
void printStr();
void main()
{
    char str[] = "GeeksforGeeks";

    printStr(str);      //  passing string to function printStr()
      getch();
}
void printStr(char str[])
{
    printf("String is : %s",str);
}
```

Output:

String is : GeeksforGeeks

gets() and puts()

You can use **gets()** function to read a line of string. And, you can use **puts()** to display the string.

```
#include <stdio.h>
void main()
{
   char name[30];
   printf("Enter name: ");
   gets(name);     // read string
   printf("Name: ");
   puts(name);    // display string
   getch();
}
```

```
Enter name: Tom Hanks
Name: Tom Hanks
```

String Handling Functions

The following are the most commonly used string handling functions.

Sr.No.	Function & Purpose
1	**strcpy(s1, s2);** Copies string s2 into string s1.
2	**strcat(s1, s2);** Concatenates string s2 onto the end of string s1.
3	**strlen(s1);** Returns the length of string s1.
4	**strcmp(s1, s2);**

	Returns 0 if s1 and s2 are the same; less than 0 if s1<s2; greater than 0 if s1>s2.	
5	**strrev();**	It is used to reverse the given string expression.
6	**strstr(s1, s2);**	Returns a pointer to the first occurrence of string s2 in string s1.

some of the above-mentioned functions −

```
#include <stdio.h>
#include <string.h>
void main () {

   char str1[12] = "Hello";
   char str2[12] = "World";
   char str3[12];
   int  len ;


   strcpy(str3, str1);     /* copy str1 into str3 */
   printf("strcpy result :  %s", str3 );

   strcat( str1, str2);    /* concatenates str1 and str2 */
   printf("\n strcat result:   %s", str1 );

   /* total lenghth of str1 after concatenation */
   len = strlen(str1);
   printf("\n string length :  %d", len );

}
```

Output:

strcpy result : Hello
strcat result: HelloWorld
string length : 10

Example: Use of strrev() and strcmp() function.

```
#include <stdio.h>
#include <string.h>
void main ()
{  char str1[12] = "Hello";
   char str2[12] = "World";

    printf("strrev  result :  %s", strrev(str1) );                    // strrev()  is used to reverse the given string

   printf("\n strcmp result:   %d", strcmp( str1, str1) );      // strcmp() is used for  string comparison
   printf("\n strcmp result:   %d", strcmp( str1, str2) );
getch();
}
```

Output: strrev result : olleH
strcmp result: 0
strcmp result: 1

Example: Program to count vowels, consonants and space.

```c
#include <stdio.h>

int main()
{
    char line[150];
    int i, vowels, consonants, spaces;

    vowels =  consonants = spaces = 0;

    printf("Enter a line of string: ");
    scanf("%s", line);

    for(i=0; line[i]!='\0'; ++i)
    {
     if(line[i]=='a' || line[i]=='e' || line[i]=='i' || line[i]=='o' ||
     line[i]=='u' || line[i]=='A' || line[i]=='E' || line[i]=='I' ||
     line[i]=='O' || line[i]=='U')
     {
            ++vowels;
      }
     else{ if((line[i]>='a'&& line[i]<='z') || (line[i]>='A'&& line[i]<='Z'))
```

```
            {
                        ++consonants;
            }
            else
            {          if (line[i]==' ')
                       {       ++spaces;
                        }
              }
     }
    printf("Vowels: %d",vowels);
    printf("\nConsonants: %d",consonants);
    printf("\nWhite spaces: %d", spaces);

    return 0;
}
```

```
Enter a line of string: adfslkj lkj lk
Vowels: 1
Consonants: 11
White spaces: 2
```

Example: WAP to take a string from the user and find the entered string is palindrome or not.

```
#include <stdio.h>
 void   main()
{
  char text[100];
  int begin, end, length = 0;
   printf(“ Enter a string : ”);
   gets(text);
   length = strlen(text) -1 ;
   end = length - 1;
                        for (begin = 0; begin < end ; begin++)
                        {
                                if (text[begin] != text[end])
                                {
                                        printf("Not a palindrome.\n");
                                         break;
                                }
                          end  - - ;
                        }
  if (begin == end)
  printf("Palindrome.");
 }
```

Output: Enter a string : nitin

Palindrome.

Example : Using loop to calculate the length of string.

```
// C program to find the length of string
#include <stdio.h>
#include <string.h>

int main()
{
    char Str[1000];
    int i;

    printf("Enter the String: ");
    scanf("%s", Str);

    for (i = 0; Str[i] != '\0'; ++i);

    printf("Length of Str is %d", i);

    return 0;
}
```

Output:

```
Enter the String: Geeks

Length of Str is 5
```

WAP to reverse a string.

```
#include <stdio.h>
int main()
{
  char s[1000], r[1000];
  int begin, end, count = 0;

  printf("Input a string\n");
  gets(s);

  // Calculating string length

  while (s[count] != '\0')
    count++;

  end = count - 1;

  for (begin = 0; begin < count; begin++) {
    r[begin] = s[end];
    end--;
  }

  r[begin] = '\0';
```

```
  printf("%s\n", r);

  getch();

  return 0;
}
```

Time and Space Complexity

Sometimes, there are more than one way to solve a problem. We need to learn how to compare the performance different algorithms and choose the best one to solve a particular problem. While analyzing an algorithm, we mostly consider time complexity and space complexity.

Time complexity of an algorithm quantifies the amount of time taken by an algorithm to run as a function of the length of the input.

Space complexity of an algorithm quantifies the amount of space or memory taken by an algorithm to run as a function of the length of the input.

Each of the operation in computer take approximately constant time. Let each operation takes c time. The number of lines of code executed is actually depends on the value of x. During analyses of algorithm, mostly we will consider worst case scenario, i.e., when x is not present in the array A. In the worst case, the **if** condition will run N times where N is the length of the array A. So in the worst case, total execution time will be (N∗c+c). N∗c for the **if** condition and c for the **return** statement (ignoring some operations like assignment of i).

As we can see that the total time depends on the length of the array A. If the length of the array will increase the time of execution will also increase.

Linear search

Or

Write a c code to initialize an array and search a particular element in that array. How can the break statement help to increase the efficiency if this code.

Linear search is used to find whether a given number is present in an array and if it is present then at what location it occurs.

It is also known as sequential search.

It is straightforward and works as follows: We keep on comparing each element with the element to search until it is found or the list ends.

```
#include <stdio.h>
int main()
{
 int array[100], search, c, n;

 printf("Enter number of elements in array  \n");
 scanf("%d", &n);

 printf("Enter %d integer(s)  \n", n);

                    for (c = 0; c < n; c++)
                     scanf("%d", &array[c]);

 printf("Enter a number to be search\n");
 scanf("%d", &search);

                   for (c = 0; c < n; c++)
                    {
                            if (array[c] == search)    /* If required element is found */
                           {
                            printf("%d is present at location %d.\n", search, c+1);
                             break;
                             }
                   }
 if (c == n)
  printf("%d isn't present in the array.\n", search);

 getch();
}
```

Output: Enter number of elements in array
5
Enter 5 integer(s)
44
55
66
33

99
Enter a number to be search
33
33 is present at location 4

In worst case, "n "number of comparisons will be done(one loop run n times) .so time complexity of liner search is O(n)

binary search

Binary search algorithm applies to a sorted array for searching an element. The search starts with comparing the element to be search with the middle element of the array. If value matches then the position of the element is returned.

If the value of element to be search is less than the middle element of the array then the second half of the array is discarded and the search continues by dividing the first half.

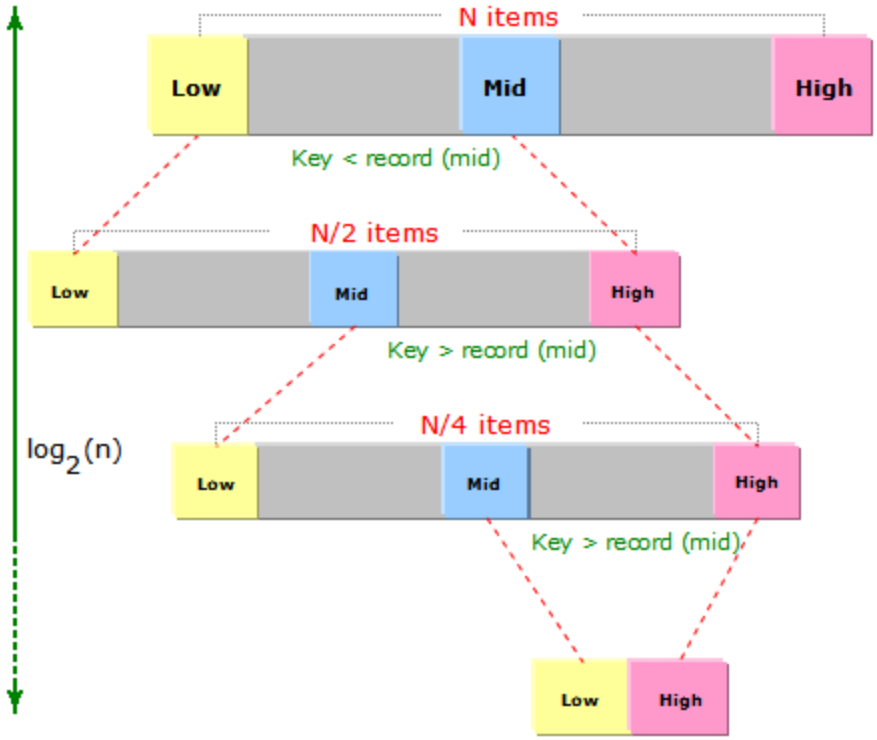
The process is the same when the value of element to be search is greater than the middle element, only, in this case, the first half of the array is discarded before continuing with the search. The iteration repeats until a match for the target element is found.

It can only be used for sorted arrays, but it's fast as compared to linear search.

If you wish to use binary search on an array which isn't sorted, then you must sort it using some sorting technique say merge sort and then use the binary search algorithm to find the desired element in the list.

If the element to be searched is found then its position is printed.

The code below assumes that the input numbers are in *ascending*order.



Algorithm

```
function binary_search(A, n, T):
    L := 0
    R := n - 1
    while L <= R:
        m := floor((L + R) / 2)
        if A[m] < T:
            L := m + 1
        else if A[m] > T:
            R := m - 1
        else:
            return m
    return unsuccessful
```

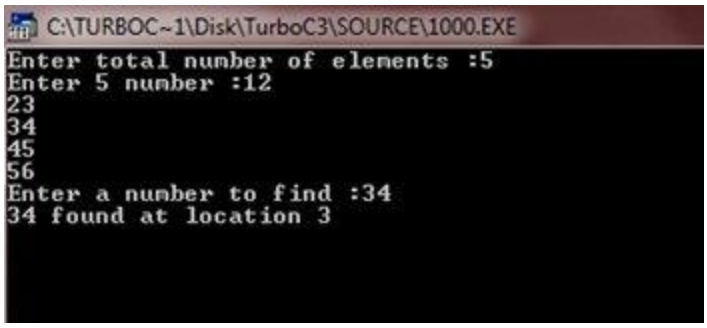
```
/* C Program - Binary Search */

#include<stdio.h>
#include<conio.h>
void main()
```

```
{
	clrscr();
	int n, i, arr[50], search, first, last, middle;
	printf("Enter total number of elements :");
	scanf("%d",&n);
	printf("Enter %d number :", n);
	for (i=0; i<n; i++)
	{
		scanf("%d",&arr[i]);
	}
	printf("Enter a number to find :");
	scanf("%d", &search);
	first = 0;
	last = n-1;
	middle = (first+last)/2;
	while (first <= last)
	{
		if(arr[middle] < search)
		{
			first = middle + 1;

		}
		else
		{	if(arr[middle] == search)
			{
			printf("%d found at location %d\n", search, middle+1);
			break;
			}
			else
			{
			 last = middle - 1;
			}
		}
		middle = (first + last)/2;
	}
	if(first > last)
	{
	printf("Not found! %d is not present in the list.", search);
	}
	getch();
}
```

OUTPUT

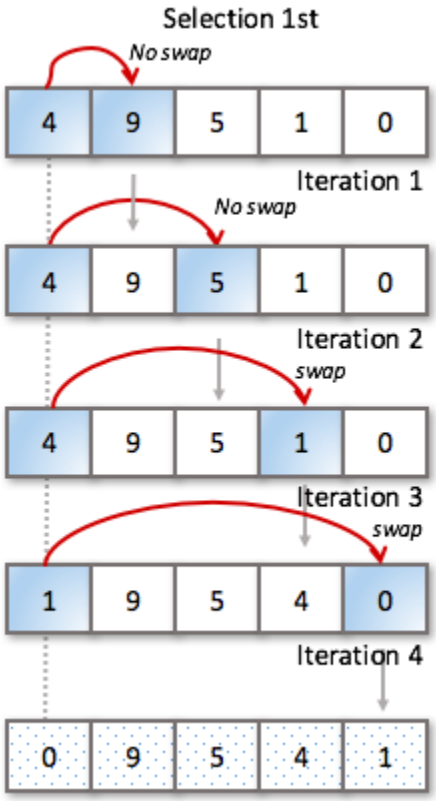


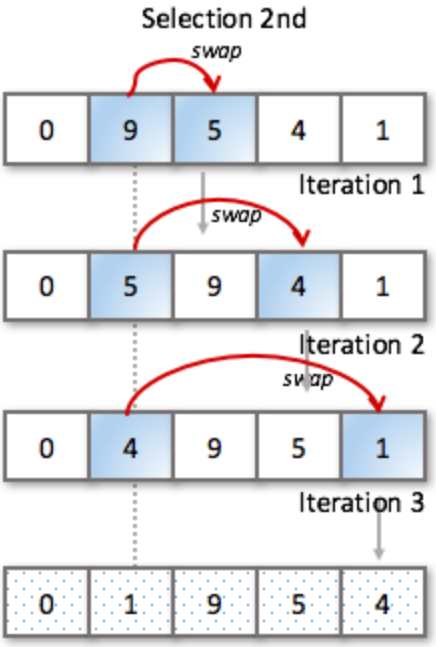
Time complexity of binary search is O(lon n).

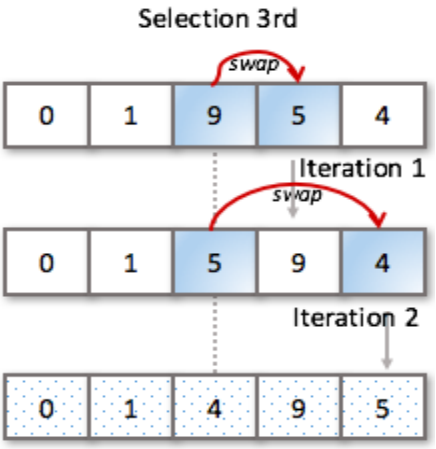
Selection Sort

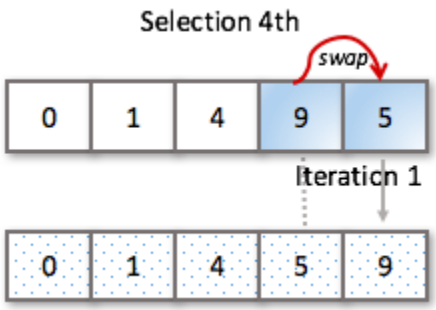
A *Selection Sort* is a Sorting algorithm which finds the smallest element in the array and swaps with the first element then this Sorting algorithm which finds the 2nd smallest element in the array and swaps with the second element and continues until the entire array is sorted.

Steps and Iterations (diagrams)




Selection 2nd
swap
0 9 5 4 1
Iteration 1
swap
0 5 9 4 1
Iteration 2
swap
0 4 9 5 1
Iteration 3
0 1 9 5 4


Selection 3rd
swap
0 1 9 5 4
Iteration 1
swap
0 1 5 9 4
Iteration 2
0 1 4 9 5


Selection 4th
swap
0 1 4 9 5
Iteration 1
0 1 4 5 9

Algorithm Selection Sort

1. SELECTION_SORT(a[n] , i , j)
2. Repeat for i=0 to i<n
   Initialize array a[i]
3. Repeat step 4 for i=0 to i < n
4. Repeat for j= i+ 1 to j < n
5. If a[i] > a[j]
   Than Swap(a[i] , a[j])
6. End If
7. End Inner Loop
8. End Outer Loop

```
#include<stdio.h>
main()
{
   int a[10],i,j,temp,n;
   //clear();
   printf("\n Enter the max no.of Elements to Sort: \n");
   scanf("%d",&n);
   printf("\n Enter the Elements : \n");
   for(i=0; i<n; i++)
   {
      scanf("%d",&a[i]);
   }
   for(i=0; i<n--1; i++)
      for(j=i+1; j<n; j++)
      {
         if(a[i]>a[j])
         {
            temp=a[i];
            a[i]=a[j];
            a[j]=temp;
         }
      }
   for(i=0; i<n; i++)
   {
      printf("%d\t",a[i]);
   }
   getch();
}
```

OUTPUT:

```
Enter the max no.of Elements to Sort:
5
```

```
 Enter the Elements :
45
21
89
78
99
21     45     78     89     99
```

In worst case, "n*n=n^2 "number of swap will be possible (two loop, each loop will be run at most n times) .so time complexity of liner search is O(n^2).

Bubble Sort

Bubble sort is a simple sorting algorithm. This sorting algorithm is comparison-based algorithm in which each pair of adjacent elements is compared and the elements are swapped if they are not in order. This algorithm is not suitable for large data sets as its average and worst case complexity are of O(n^2) where **n** is the number of items.

How Bubble Sort Works?

Bubble sort is one of the simplest sorting algorithms. The two adjacent elements of an array are checked and swapped if they are in wrong order and this process is repeated until we get a sorted array. The steps of performing a bubble sort are:

- Compare the first and the second element of the array and swap them if they are in wrong order.
- Compare the second and the third element of the array and swap them if they are in wrong order.
- Proceed till the last element of the array in a similar fashion.
- Repeat all of the above steps until the array is sorted.

This will be more clear by the following visualizations.

Initial array

16	19	11	15

First iteration

16	19	11	15
16	19	11	15

SWAP

16	11	19	15
16	11	19	15

SWAP

16	11	15	19

Second iteration

16	11	15	19

SWAP

11	16	15	19
11	16	15	19

SWAP

11	15	16	19
11	15	16	19

Third iteration

11	15	16	19

11	15	16	19
11	15	16	19

No swap → Sorted → break the loop

Algorithm

```
procedure bubbleSort(A : list of sortable items )
    n = length(A)
    repeat
        swapped = false
        for i = 1 to n-1 inclusive do
            /* if this pair is out of order */
            if A[i-1] > A[i] then
                /* swap them and remember something changed */
                swap( A[i-1], A[i] )
                swapped = true
            end if
        end for
    until not swapped
end procedure
```

```
#include<stdio.h>

void main()

 {

  int i,a[20],n,flag=1,t;

  printf("enter the number of elements");

  scanf("%d",&n);

  printf("enter the elements");
```

```
for(i=0;i<n;i++)
 scanf("%d",&a[i]);
while(flag)
 {
  flag=0;
  for(i=0;i<n-1;i++)
   if(a[i]>a[i+1])
    {
     t=a[i];
     a[i]=a[i+1];
     a[i+1]=t;
     flag=1;
    }
 }

printf("\nsorted number\n");
for(i=0;i<n;i++)
 printf("%d\t",a[i]);
 getch();
}
```

Output

enter the number of elements : 6
enter the elements
77
4
14
9
22
88

sorted number: 4 9 14 22 77 88

In worst case, "n*n=n^2 "number of swap will be possible (two loop, each loop will be run at most n times) .so time complexity of liner search is O(n^2).

Insertion Sort

Insertion sort is the sorting mechanism where the sorted array is built having one item at a time. The array elements are compared with each other sequentially and then arranged simultaneously in some particular order. The analogy can be understood from the style we arrange a deck of cards. This sort works on the principle of inserting an element at a particular position, hence the name Insertion Sort.

Insertion Sort works as follows:

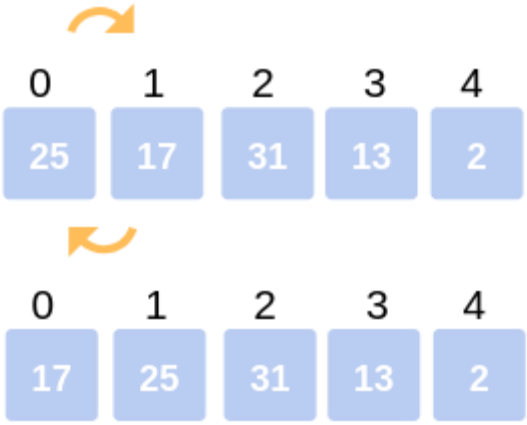
1. The first step involves the comparison of the element in question with its adjacent element.
2. And if at every comparison reveals that the element in question can be inserted at a particular position, then space is created for it by shifting the other elements one position to the right and inserting the element at the suitable position.
3. The above procedure is repeated until all the element in the array is at their apt position.

Let us now understand working with the following example:

Consider the following array: 25, 17, 31, 13, 2

First Iteration: Compare 25 with 17. The comparison shows 17< 25. Hence swap 17 and 25.
The array now looks like:

17, 25, 31, 13, 2



First Iteration

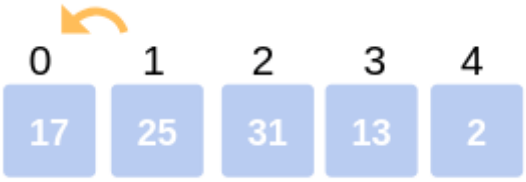
Second Iteration: Begin with the second element (25), but it was already swapped on for the correct position, so we move ahead to the next element. Now hold on to the third element (31) and compare with the ones preceding it.

Since 31> 25, no swapping takes place.

Also, 31> 17, no swapping takes place and 31 remains at its position.

The array after the Second iteration looks like:

17, 25, 31, 13, 2



Second Iteration

Third Iteration: Start the following Iteration with the fourth element (13), and compare it with its preceding elements.
Since 13< 31, we swap the two.

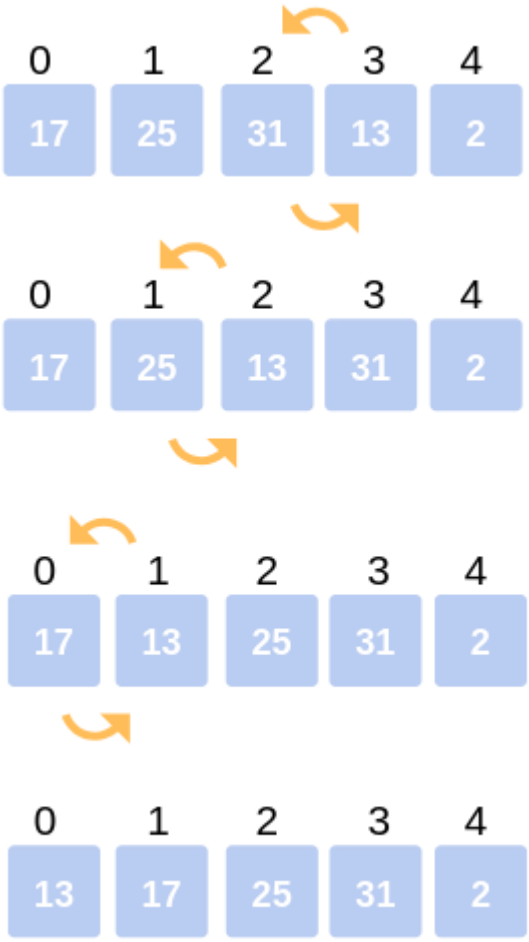
Array now becomes: 17, 25, 13, 31, 2.

But there still exist elements that we haven't yet compared with 13. Now the comparison takes place between 25 and 13. Since, 13 < 25, we swap the two.

The array becomes **17, 13, 25, 31, 2**.
The last comparison for the iteration is now between 17 and 13. Since 13 < 17, we swap the two.

The array now becomes **13, 17, 25, 31, 2**.

Third Iteration



Fourth Iteration: The last iteration calls for the comparison of the last element (2), with all the preceding elements and make the appropriate swapping between elements.
Since, 2< 31. Swap 2 and 31.

Array now becomes: 13, 17, 25, 2, 31.

Compare 2 with 25, 17, 13.

Since, 2< 25. Swap 25 and 2.

13, 17, 2, 25, 31.
Compare 2 with 17 and 13.

Since, 2<17. Swap 2 and 17.

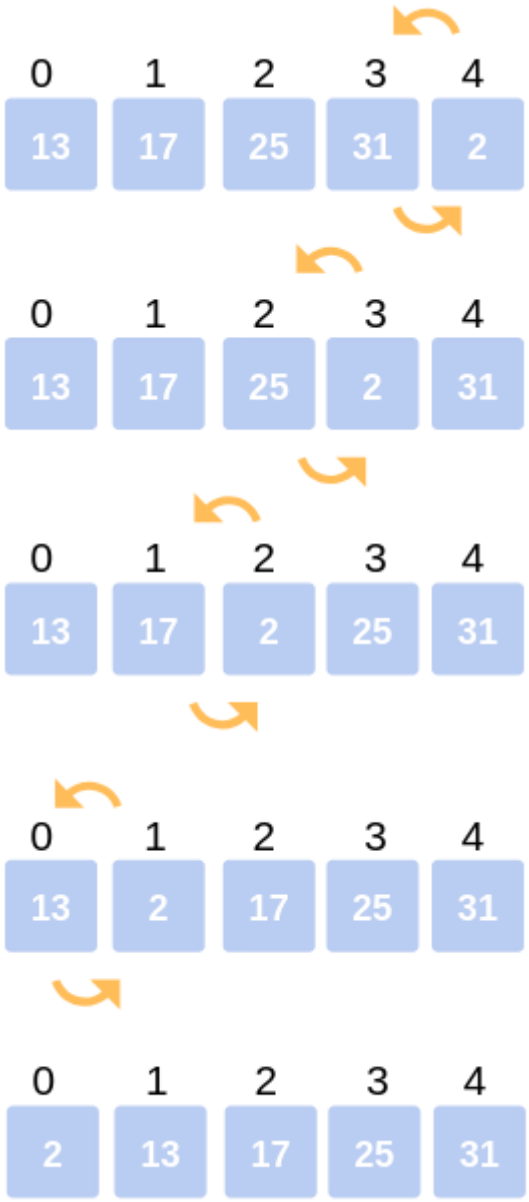
Array now becomes:

13, 2, 17, 25, 31.
The last comparison for the Iteration is to compare 2 with 13.

Since 2< 13. Swap 2 and 13.

The array now becomes:

2, 13, 17, 25, 31.
This is the final array after all the corresponding iterations and swapping of elements.



Fourth Iteration

Algorithm

```
INSERTION-SORT(A)
   for i = 1 to n
         key ← A [i]
         j ← i – 1
          while j > = 0 and A[j] > key
                  A[j+1] ← A[j]
                  j ← j – 1
         End while
         A[j+1] ← key
```

```
  End for
```

Program

```
#include<stdio.h>
void main()
{
  int a[20],n,i,j,key;
  printf("enter the number of elements");
  scanf("%d",&n);
  printf("enter the elements");
  for(i=0;i<n;i++)
   scanf("%d",&a[i]);

  for(j=1;j<n;j++)
   {
     key=a[j];
     i=j-1;
      while(i>=0 && a[i]>key)
        {
          a[i+1]=a[i];
          i--;
         }
       a[i+1]=key;
    }
 printf("\nsorted number:\n");
  for(i=0;i<n;i++)
```

```
   printf("%d\t",a[i]);
getch();
}
```

```
Output

   enter the number of elements:  6
    enter the elements
    77
    4
    14
    9
    22
    88
    sorted number:  4    9    14    22    77    88
```

In worst case, "n*n=n^2 "number of swap will be possible (two loop, each loop will be run at most n times) .so time complexity of liner search is O(n^2).