Searching :-

It means to find whether a particular value is present in the array or not. If the value is present in the array then searching is said to be successful and the searching process gives the location of that value in the array.

1. Linear Search
2. Binary Search (for sorted Array)

Linear Search :-

It is call sequential Search

```
main()
{   scanf("%d", &num);
for (i=0; i<n; i++)
    {
      if (arr[i] == num)
       { found = 1;
         pos = i;
      printf("\n %d is found in the array
            at position = %d", num, i);
         break;
    }
}
if (found == 0)
    printf("\n %d Does not Exist
                ", num);
}
```

Binary Search:-

```
main()
{ int arr[10], num, i, n, pos=-1,
  beg, end, mid, found=0;
printf("\n Enter the no of elements)
scanf("%d", &n);
printf("\n Enter the elements:"
  for(i=0; i<n; i++)
    { scanf("%d", &arr[i]);
printf(" Enter the number that
        has to be searched:");
scanf(" &num);
  beg=0, end = n-1;
while (beg ≤ end)
    {
      mid = (beg+end)/2;
  if (arr[mid] == num)
    { printf("%d is present in the
          array at position %d",
          num, mid);
      found = 1;
      break;
    }
```

```
if (arr[mid] > num
        end = mid - 1;
    else if (arr[mid] < num)
        beg = mid + 1;
}
if (beg > end && found == 0)
    printf("\n %d Does not Exist
            in the array", num)
    return 0;
}
```

using function: pos = binary_search(arr, size, item);

```
int binary_search(int arr[], int size,
                  int item)
{ int Low = 0;
  int High = size - 1;
  int MID;
  while (Low <= HIGH)
  { MID = (HIGH + LOW)/2;
    if (arr[MID] == item)
        return MID;
    else
        if (item < arr[MID]
            HIGH = MID - 1;
        else
            Low = MID + 1;
  }
  return (-1);
```

Sorting procedure

Bubble - Sort :-

```
main()
{
  int i, n, temp, j, arr[10];
void bubble_sor( int a[], int n)
  {
    int temp, i, j;
for(i=0; i<n; i++)
  {
  for(j=0; j<n-i-1; j++)
    {
      ef (a[j] > a[j+1])
      {
        temp = a[j];
        a[j] = a[j+1];
        a[j+1] = temp;
      }
    }
  }
}
```

Analyses:-

Here, the first pass requires (n-1) comparisons to fix the highest element to its location, the second pass requires (n-2)... kth pass requires (n-k), and last pass requires only one comparison to be fixed at its proper position.

```c
#define ROW 2
#define COL 5
int main(){
   int i, j, mat[ROW][COL], trans[COL][
   printf("Enter matrix: \n");
   // input matrix
   for(i = 0; i < ROW; i++){
      for(j = 0; j < COL; j++){
         scanf("%d", &mat[i][j]);
      }
   }
   // create transpose
   for(i = 0; i < ROW; i++){
      for(j = 0; j < COL; j++){
         trans[j][i] = mat[i][j];
      }
   }
   printf("\nTranspose matrix: \n");
   // print transpose
   for(i = 0; i < COL; i++){
      for(j = 0; j < ROW; j++){
         printf("%d ", trans[i][j]);
      }
      printf("\n");
   }
   return 0;
}
```

Output

```
Enter matrix:
1 2 3 4 5
5 4 3 2 1

Transpose matrix:
1 5
2 4
3 3
4 2
5 1
```

```c
#include <stdio.h>

int main()
{
   int array[100], search, c, n, count = 0;

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

   printf("Enter %d numbers\n", n);

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

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

   for (c = 0; c < n; c++) {
      if (array[c] == search) {
         printf("%d is present at location
%d.\n", search, c+1);
         count++;
      }
   }
   if (count == 0)
      printf("%d isn't present in the
array.\n", search);
   else
      printf("%d is present %d times in the
array.\n", search, count);

   return 0;
}
```

Matrix multiplication in C

Matrix multiplication in C: We can add, subtract, multiply and divide 2 matrices. To do so, we are taking input from the user for row number, column number, first matrix elements and second matrix elements. Then we are performing multiplication on the matrices entered by the user.

In matrix multiplication *first matrix one row element is multiplied by second matrix all column elements*.

Let's try to understand the matrix multiplication of **2*2 and 3*3** matrices by the figure given below:

$$A = \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} \quad B = \begin{pmatrix} 5 & 6 & 7 \\ 8 & 9 & 10 \end{pmatrix}$$

Multiplication of two matrixes:

$$A * B = \begin{pmatrix} 1*5 + 2*8 & 1*6 + 2*9 & 1*7 + 2*10 \\ 3*5 + 4*8 & 3*6 + 4*9 & 3*7 + 4*10 \end{pmatrix}$$

$$A * B = \begin{pmatrix} 21 & 24 & 27 \\ 47 & 54 & 61 \end{pmatrix}$$

```c
#include<stdio.h>
#include<stdlib.h>
int main(){
int a[10][10],b[10][10],mul[10][10],r,c,i,j,k;
system("cls");
printf("enter the number of row=");
scanf("%d",&r);
printf("enter the number of column=");
scanf("%d",&c);
printf("enter the first matrix element=\n");
for(i=0;i<r;i++)
{
for(j=0;j<c;j++)
{
scanf("%d",&a[i][j]);
}
}
```

```
printf("enter the second matrix element=\n");
for(i=0;i<r;i++)
{
for(j=0;j<c;j++)
{
scanf("%d",&b[i][j]);
}
}

printf("multiply of the matrix=\n");
for(i=0;i<r;i++)
{
for(j=0;j<c;j++)
{
mul[i][j]=0;
for(k=0;k<c;k++)
{
mul[i][j]+=a[i][k]*b[k][j];
}
}
}
```

```
printf("multiply of the matrix=\n");
for(i=0;i<r;i++)
{
for(j=0;j<c;j++)
{
mul[i][j]=0;
for(k=0;k<c;k++)
{
mul[i][j]+=a[i][k]*b[k][j];
}
}
}
//for printing result
for(i=0;i<r;i++)
{
for(j=0;j<c;j++)
{
printf("%d\t",mul[i][j]);
}
printf("\n");
}
return 0;
}
```

```
enter the number of row=3
enter the number of column=3
enter the first matrix element=
1 1 1
2 2 2
3 3 3
enter the second matrix element=
1 1 1
2 2 2
3 3 3
multiply of the matrix=
6 6 6
12 12 12
18 18 18
```

Bubble Sort Example

To discuss the bubble sort let us consider an array that has the following elements:

```
A[] = {30, 52, 29, 87, 63, 27, 18, 54}
```

Pass 1:

(a) Compare 30 and 52. Since30 < 52, then no swapping is done.

(b) Compare 52 and 29. Since 52 > 29, swapping is done. 30, **29, 52**, 87, 63, 27, 19, 54

(c) Compare 52 and 87. Since 52 < 87, no swapping is done.

(d) Compare, 87 and 63. Since, 87 > 83, swapping is done. 30, 29, 52, **63, 87**, 27, 19, 54

(e) Compare 87 and 27. Since 87 > 27, swapping is done. 30, 29, 52, 63, **27, 87**, 19, 54

(f) Compare 87 and 19. Since 87 > 19, swapping is done. 30, 29, 52, 63, 27, 19, 87, 54

(g) Compare 87 and 54. Since 87 > 54, swapping is done. 30, 29, 52, 63, 27, 19, **54, 87**

Observe that after the end of the first pass, the largest element is placed at the highest index of the array. All the other elements are still unsorted.

(e) Compare 87 and 63. Since 87 > 63, swapping is done. 30, 29, 52, 63, **27, 87**, 19, 54

(f) Compare 87 and 19. Since 87 > 19, swapping is done. 30, 29, 52, 63, 27, 19, 87, 54

(g) Compare 87 and 54. Since 87 > 54, swapping is done. 30, 29, 52, 63, 27, 19, **54, 87**

Observe that after the end of the first pass, the largest element is placed at the highest index of the array. All the other elements are still unsorted.

Pass 2:

(a) Compare 30 and 29. Since 30 > 29, swapping is done. **29, 30**, 52, 63, 27, 19, 54, 87

(b) Compare 30 and 52. Since 30 < 52, no swapping is done.

(c) Compare 52 and 63. Since 52 < 63, no swapping is done.

(d) Compare 63 and 27. Since 63 > 27, swapping is done. 29, 30, 52, **27, 63**, 19, 54, 87

(e) Compare 63 and 19. Since 63 > 19, swapping is done.
29, 30, 52, 27, **19, 63**, 54, 87

(f) Compare 63 and 54. Since 63 > 54, swapping is done.
29, 30, 52, 27, 19, **54, 63**, 87

Observe that after the end of the second pass, the second largest element is placed at the second highest index of the array. All the other elements are still unsorted.

Pass 3:

(a) Compare 29 and 30. Since 29 < 30, no swapping is done.

(b) Compare 30 and 52. Since 30 < 52, no swapping is done.

(c) Compare 52 and 27. Since 52 > 27, swapping is done.
29, 30, **27, 52**, 19, 54, 63, 87

(d) Compare 52 and 19. Since 52 > 19, swapping is done.
29, 30, 27, **19, 52**, 54, 63, 87

(e) Compare 52 and 54. Since 52 < 54, no swapping is done.

Observe that after the end of the third pass, the third largest element is placed at the third highest index of the array. All the other elements are still unsorted.

(b) Compare 30 and 52. Since $30 < 52$, no swapping is done.

(c) Compare 52 and 27. Since $52 > 27$, swapping is done.
29, 30, **27, 52**, 19, 54, 63, 87

(d) Compare 52 and 19. Since $52 > 19$, swapping is done.
29, 30, 27, **19, 52**, 54, 63, 87

(e) Compare 52 and 54. Since $52 < 54$, no swapping is done.

Observe that after the end of the third pass, the third largest element is placed at the third highest index of the array. All the other elements are still unsorted.

Pass 4:

(a) Compare 29 and 30. Since $29 < 30$, no swapping is done.

(b) Compare 30 and 27. Since $30 > 27$, swapping is done.
29, **27, 30**, 19, 52, 54, 63, 87

(c) Compare 30 and 19. Since $30 > 19$, swapping is done.
29, 27, **19, 30**, 52, 54, 63, 87

(d) Compare 30 and 52. Since $30 < 52$, no swapping is done.

Observe that after the end of the fourth pass, the fourth largest element is placed at the fourth highest index of the array. All the other elements are still unsorted.

Pass 5:

(a) Compare 29 and 27. Since $29 > 27$, swapping is done.
27, 29, 19, 30, 52, 54, 63, 87

(b) Compare 29 and 19. Since $29 > 19$, swapping is done.
27, **19, 29**, 30, 52, 54, 63, 87

(c) Compare 29 and 30. Since $29 < 30$, no swapping is done.

Observe that after the end of the fifth pass, the fifth largest element is placed at the fifth highest index of the array. All the other elements are still unsorted.

Pass 6:

(a) Compare 27 and 19. Since $27 > 19$, swapping is done.

19, 27, 29, 30, 52, 54, 63, 87

(b) Compare 27 and 29. Since $27 < 29$, no swapping is done.

Observe that after the end of the sixth pass, the sixth largest element is placed at the sixth largest index of the array. All the other elements are still unsorted.

Pass 7:

(a) Compare 19 and 27. Since $19 < 27$, no swapping is done.

Observe that the entire list is sorted now.