Space Complexity

```
Algorithm to find sum of two varaible
int sum(int num1, int num2)
{
        int res = num1 + num2;
        return res;
}
```

Input space	:	num1, num2	:	2 units
Auxilliary space	:	res	:	1 unit

Total space = Input space + Auxilliary space
= 2 + 1
= 3 units

Irrespective of type and value of data, space required is constant space (k)

Total space α constant space
Total space α k
Total space α K * 1

Total space = 1

To indicate complexities, we need to use one notataion ie 'order of' notation o()

Space complexity = o(1)

Space Complexity

```
Algorithm to find sum of array elements
int sumofarray(int arr[], int size)
{
        int sum = 0;
        for(int i = 0 ; i < size ; i++)
                sum += arr[i]
        return sum;
}
```

size = n

Input space : n unit

Auxilliary space : size, sum, i : 3 units

Total space α Input space + Auxilliray

Total space α n + 3

if n >>>>>>>3

Total space α n

Space complexity = o(n)

Space complexity : o(1), o(n), o(n^2), o(n^3),....

```
Algorithm to display matix
void display_matrix(int mat[][], int row, int col)
{
        int i,j;
        for(i = 0 ; i < row ; i++)
        {
                for(j = 0 ; j < col ; j++)
                        printf("%d", mat[i][j]);
                printf("\n");
        }
}
```

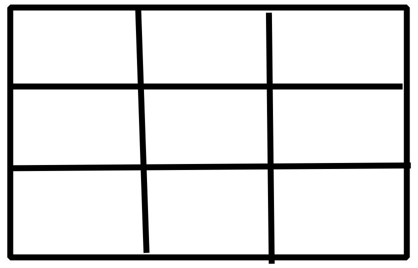
size n X n

Input space : n^2 units
Auxilliray space : row, col, i, j : 4 units

Total space α Input space + Auxilliary space
Total space α n^2 + 4
if n >>>>>>
Total space α n^2

Space complexity = o(n^2)

Space Complexity



3 X 3 = 9 = 9 units

size m X n

Input space : m * n units
Auxilliray space : row, col, i, j : 4 units

Total space α Input space + Auxilliary space
Total space α m * n + 4
if m,n >>>>>>
Total space α m*n

Space complexity = o(m*n)

Time Complexity

```
statement;
```

Time = 1 unit
Time complexity = o(1)

```
for(int i = 0 ; i < n ; i++)
{
        statements;
}
```

```
for(int i = n ; i >= 0 ; i--)
{
        statements;
}
```

Condition will be checked = n + 1 times
statement executes = n times

Total time = n + 1+ n
Total time = 2n + 1
if n >>>>>>
Total time = 2n
Total time α n
Time complexity = o(n)

Time Complexity

```
for(int i = 0 ; i < n ; i++)
{
        for(int j = 0 ; j < n ; j++)
        {
                statements
        }
}
```

Outer condition checked	: n times
Inner condition checked	: n * n times
statements will execute	: n * n times

Total time = n + n^2 + n^2
= n + 2n^2
= 2n^2

Total time α n^2

Time complexity = o(n^2)

Time Complexity

```
for(int i = n ; i > 0 ; i=i/2)
{
        statement;
}
```

i = n, n/2, n/4, n/8, ...
i = n/2^0, n/2^1, n/2^2, n/2^3,......n/2^itr

for i=1, last time condition will be true

n/2^itr = 1
2^itr = n
log 2^itr = log n
itr log 2 = log n
itr = log n / log 2
itr = (1/log 2) log n
itr α log n

Time complexity = o(log n)

Time complexity : o(log log n), o(log n), o(n log n), o(1), o(n), o(n^2), o(n^3) ,......