The **space complexity** of an algorithm or a computer program is the amount of memory space required to solve an instance of the computational problem as a function of characteristics of the input. It is the memory required by an algorithm until it executes completely.

Similar to time complexity, space complexity is often expressed asymptotically in big O notation.

Whenever a solution to a problem is written some memory is required to complete. For any algorithm memory may be used for the following:

1. Variables (This include the constant values, temporary values)
2. Program Instruction
3. Execution

Space complexity *is the amount of memory used by the algorithm (including the*

input values to the algorithm) to execute and produce the result.

Sometime **Auxiliary Space** is confused with Space Complexity. But Auxiliary Space is the extra space or the temporary space used by the algorithm during it's execution.

Space Complexity = Auxiliary Space + Input space

Memory Usage while Execution

While executing, algorithm uses memory space for three reasons:

1. **Instruction Space**

   It's the amount of memory used to save the compiled version of instructions.

2. **Environmental Stack**

   Sometimes an algorithm(function) may be called inside another

algorithm(function). In such a situation, the current variables are pushed onto the system stack, where they wait for further execution and then the call to the inside algorithm(function) is made.

For example, If a function A()calls function B() inside it, then all th variables of the function A() will get stored on the system stack

temporarily, while the function B() is called and executed inside the funciton A().

3. **Data Space**

   Amount of space used by the variables and constants.

But while calculating the **Space Complexity** of any algorithm, we usually consider only **Data Space** and we neglect the **Instruction**

Space and **Environmental Stack**.

Calculating the Space Complexity

For calculating the space complexity, we need to know the value of memory used by different type of datatype variables, which generally varies for different operating systems, but the method for calculating the space complexity remains the same.

Now let's learn how to compute space complexity by taking a few examples:

```
{
    int z = a + b + c;
    return(z);
}
```

In the above expression, variables a, b, c and z are all integer types, hence they will take up 4 bytes each, so total memory requirement will be $(4(4) + 4) = 20$ bytes, this

additional 4 bytes is for **return value**. And because this space requirement is fixed for the above example, hence it is called **Constant Space Complexity**.

Let's have another example, this time a bit complex one,

```
// n is the length of array a[]
int sum(int a[], int n)
```

```
{
   int x = 0;     // 4
bytes for x
   for(int i = 0; i <
n; i++)  // 4 bytes
for i
   {
        x  = x + a[i];

   }
   return(x);
}
```

. In the above code, 4*n bytes of space is required for the array a[] elements.

. 4 bytes each for x, n, I and the return value.

Hence the total memory requirement will be (4n + 12), which is increasing linearly with the increase in the input value n, hence it is called as **Linear Space Complexity.**

Similarly, we can have quadratic and other complex

space complexity as well, as the complexity of an algorithm increases.

But we should always focus on writing algorithm code in such a way that we keep the space complexity **minimum**.