Space Complexity

Space Complexity:

The term Space Complexity is misused for Auxiliary Space at many places. Following are the correct definitions of Auxiliary Space and Space Complexity.

Auxiliary Space is the extra space or temporary space used by an algorithm.

The space Complexity of an algorithm is the total space taken by the algorithm with respect to the input size. Space complexity includes both Auxiliary space and space used by input.

For example, if we want to compare standard sorting algorithms on the basis of space, then Auxiliary Space would be a better criterion than Space Complexity. Merge Sort uses O(n) auxiliary space, Insertion sort, and Heap Sort use O(1) auxiliary space. The space complexity of all these sorting algorithms is O(n) though.

Space complexity is a parallel concept to time complexity. If we need to create an array of size n, this will require O(n) space. If we create a two-dimensional array of size n*n, this will require $O(n^2)$ space.

In recursive calls stack space also counts.

Example :

Python

```python
def add(n):
  if(n<=0):
    return 0

  return n + add(n-1)
```

```
Here each call add a level to the stack :

1.  add(4)
2.    -> add(3)
3.      -> add(2)
4.        -> add(1)
5.          -> add(0)
```

```
Each of these calls is added to call stack and takes up actual memory.
So it takes O(n) space.
```

However, just because you have n calls total doesn't mean it takes O(n) space.

Look at the below function :

Python

```python
def addSequence(n):
    sum=0
    for i in range(0,n):
        sum+=pairSum(i,i+1)

    return sum

def pairSum(x,y):
    return x+y
```

There will be roughly O(n) calls to pairSum. However, those calls do not exist simultaneously on the call stack, so you only need O(1) space.

Note: It's necessary to mention that space complexity depends on a variety of things such as the programming language, the compiler, or even the machine running the algorithm.