SPACE COMPLEXITY ANALYSIS

Space complexity refers to the analysis of space that is required for an algorithm. The space here refers to the following two components:

Fixed Components

This is defined as portions of memory that are independent of the input/output of a program. The fixed components refers to the following:

1. Instruction space
2. Space required for simple variables
3. Space required for sorting constants.
4. Space required for a set of variables or aggregates.

Variable Part

This is defined as the portion of the program instances that are dependent on the program input/output.

Examples include referenced variables and stack space.

Therefore, the space complexity:

$$S(n) = fixed\ components + variable\ components.$$

Example 1

Consider the following program segment. What is the space complexity?

<u>Algorithm</u>

```
Begin

    return n;
End
```

Solution:

The space complexity is zero as no variables are stored and processed.

Example 2

Consider the following program segment. What is the space complexity?

Algorithm

```
Begin
    i =0;
    s = 0;
    s = s+i;
    return s;
End
```

Solution

Here two variables are store i.e.,
$i\ holds\ 1\ unit\ of\ Space\ and\ s\ holds\ 1\ unit\ of\ Space$.

Therefore, $S(n) \geq 2$.

As it is constant, we get $\mathbf{O(1)}$

Example 3

Consider the following program segment. What is the space complexity?

Algorithm

sum (x [], n) {

total =0;

for $i \leftarrow 1\ to\ n\ do$

$$total \coloneqq total + x[i\ ]$$

}

Solution

i holds 1 unit of Space (During declaration in for loop).
n holds 1 unit of Space (During passing as an argument).
total holds 1 unit of Space (After declaration in Inner block of Algorithm).

Now for loop will run up to n times so we can say:

$$total = total + x[1 \dots n]$$

i.e.
$x\ is\ an\ array\ holds\ n\ number\ of\ space, depends\ upon\ input\ size.$

Therefore, time complexity of x becomes $n, here.$

And finally we have : 1+1+1+n = 3+n.

If we see in Big O: $\boldsymbol{O}(3+\boldsymbol{n})=\boldsymbol{O}(\boldsymbol{n})$

Example 4

Consider the following program segment. What is the space complexity?

int arr []={1,2,3,4,5,6,7,8};

Solution

Here each array holds 1 unit of space.
arr[0] - 1 unit ,
arr[1] – 1 unit
arr[2] – 1 unit
arr[3] - 1 unit
arr[4] - 1 unit
arr[5] - 1 unit
arr[6] - 1 unit
arr[7]- 1 unit,

thus there is 8 unit of space the array consumes as , it is in constant.

So, we get $\boldsymbol{O}(1)$ $\boldsymbol{space\ complexity}$.

SOME MORE ABOUT SPACE COMPLEXITY:

Space complexity can be divided into two types:

1. Auxiliary Space: The extra space that is taken by an algorithm temporarily to finish its work.

2. Input Space: Space used by input.

$$Space\ Complexity = Input\ Space + Auxiliary\ Space$$

Suppose we have input variable a, b and we have variable c = a+b;

Algorithm:

```
sum (a, b)
  c: = a + b;
return c;
```

So, c is creating an auxiliary space or an extra space to solve the problem where as input space is space taken by input variable a and b.

Again,

Algorithm:

sum (x [], n) {

total =0;

for $i \leftarrow 1\ to\ n\ do$

$$total \coloneqq total + x[i\,]$$

}

Then variable **total is creating an auxiliary space** to compute the given problem whereas **x [] and n are inputs creates input space**.

We can convert the units to bytes i.e., according to the variable's size.

DATA TYPES	SIZE
byte	1 *bytes*
short	2 *bytes*
int	4 *bytes*
long	8 *bytes*
float	4 *bytes*
double	8 *bytes*
boolean	1 *bit*
char	2 *bytes*

Algorithm:

```
int sum (int a, int b) {
return a + b;
}
```

Solution

Therefore, 'a' holds $1\ \boldsymbol{unit} \times 4\ \boldsymbol{bytes\ of\ space} = 4\ \boldsymbol{bytes}$.

And

'b' holds $1\ \boldsymbol{unit} \times 4\ \boldsymbol{bytes\ of\ space} = 4\ \boldsymbol{bytes}$.

Therefore, total space consumed is 8 bytes.

Algorithm:

```
int arr [ ] ={1,2,3,4,5,6,7,8};
```

Solution

arr[0] - $1\ \boldsymbol{unit} \times 4\ \boldsymbol{bytes} = 4\ \boldsymbol{bytes}$,
arr[1] – $1\ \boldsymbol{unit} \times 4\ \boldsymbol{bytes} = 4\ \boldsymbol{bytes}$
arr[2] – $1\ \boldsymbol{unit} \times 4\ \boldsymbol{bytes} = 4\ \boldsymbol{bytes}$
arr[3] - $1\ \boldsymbol{unit} \times 4\ \boldsymbol{bytes} = 4\ \boldsymbol{bytes}$
arr[4] - $1\ \boldsymbol{unit} \times 4\ \boldsymbol{bytes} = 4\ \boldsymbol{bytes}$
arr[5] - $1\ \boldsymbol{unit} \times 4\ \boldsymbol{bytes} = 4\ \boldsymbol{bytes}$
arr[6] - $1\ \boldsymbol{unit} \times 4\ \boldsymbol{bytes} = 4\ \boldsymbol{bytes}$
arr[7]- $1\ \boldsymbol{unit} \times 4\ \boldsymbol{bytes} = 4\ \boldsymbol{bytes}$

Therefore, total of $4 \times 8 = 32\ bytes$.

Algorithm:

sum (x [], n) {

total =0;

for $i \leftarrow 1\ to\ n\ do$

$$total := total + x[i\ ]$$

}

Solution:

$$n \rightarrow 1\ unit \times 4\ bytes = 4\ bytes$$
$$total\ \rightarrow 1\ unit\ \times 4\ bytes = 4\ bytes$$
$$i\ \rightarrow 1\ unit\ \times 4\ bytes = 4\ bytes$$
$$x\ \rightarrow N\ unit\ \times 4\ bytes = 4N\ bytes$$

Total $= 4 + 4 + 4 + 4N = 12 + 4N\ Bytes$.

....