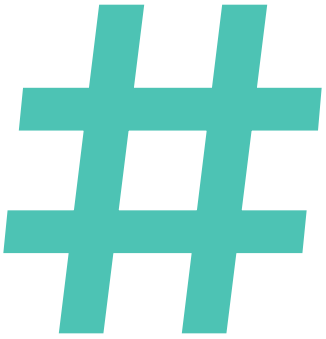

Space Complexity

Step 01 What is the Space Complexity

The space complexity of an algorithm refers to the amount of memory or storage space required by the algorithm to perform its computations. It includes the space used by auxiliary data structures (like stacks, queues, or arrays) and the space needed to store input values during the algorithm's execution.

Step 02 Analyzing Space Complexity of Algorithms

Here is the space complexity of Bubble Sort and Insertion Sort, along with their implementations.

```
01  public class BubbleSort {
02     public static void bubbleSort(int[] arr) {
03  // The variable 'n' has a space complexity of O(1)
04          int n = arr.length;
05          for (int i = 0; i < n - 1; i++){
06  // The loop counter 'i' has a space complexity of O(1)
07              for (int j = 0; j < n - i - 1; j++) {
08  // The loop counter 'j' has a space complexity of O(1)
09                  if (arr[j] > arr[j + 1]) w{
10                      int temp = arr[j];
11  // The variable 'temp' has a space complexity of O(1)
12                      arr[j] = arr[j + 1];
13                      arr[j + 1] = temp;
14                  }
15              }
16          }
17     }
18  }
```

CODE 1 **Bubble Sort**

The space complexity of the Bubble Sort is (O(1)).

```java
public class InsertionSort {
    public static void insertionSort(int[] arr) {
        int n = arr.length;
// The variable ‹n› has a space complexity of O(1)
        for (int i = 1; i < n; i++) {
// The loop counter ‹i› has a space complexity of O(1)
            int key = arr[i];
// The variable ‹key› has a space complexity of O(1)
            int j = i - 1;
// The loop counter ‹j› has a space complexity of O(1)
            while (j >= 0 && arr[j] > key) {
                arr[j + 1] = arr[j];
                j--;
            }
            arr[j + 1] = key;
        }
    }
}
```

CODE 2 Insertion Sort

- The implementation uses variables such as key and j for element comparisons and shifting.
- The sorting is performed directly on the input array without needing additional memory.
- The space complexity of the insertionSort function is O(1).

Step 03 How to Calculate Space Complexity

To calculate the space complexity of an algorithm, we consider the auxiliary space used by the algorithm, which refers to any extra or temporary space it requires. Additionally, we factor in the space used by the input values.

```
Space Complexity = Auxiliary Space + Space used for input values
```

So we can say that space complexity is the combination or sum up of the auxiliary space and the space used by input value, let's determine the space complexity of a program that sums all integer elements in an array.

> **NOTE**
> Auxiliary space is needed for additional data structures or variables created during an algorithms execution. This extra space is used for intermediate computations, temporary storage, recursion stacks, or any other space required for the algorithm to complete its task.

Step 04 How to Enhance Space Complexity

How can we enhance the space complexity:

```
01  public int sum(int[] array) {
02      if (array == null || array.length == 0) {
03          return 0;
04      }
05      return Arrays.stream(array).sum();}
```

CODE 3 Sum Function

The if statement has a space complexity of 1.
The Arrays.stream method has a space complexity of 1.
The sum method has a space complexity of 1.
O(1+1+1).
Which is constant space complexity O(1).

> **NOTE**
> we use the Arrays.stream method to create a stream of the elements in the input array and then calculate the sum of the elements using the sum method. It is easier to read, but it requires the use of the Java Streams API.

Step 05 Examples of Different Space Complexities

```
01   public static int calculateSum(int x, int y) {
02  // 4 bytes for each variable
03          int a = x + y;
04   // 4 bytes
05          return a;
06  // 4 bytes
07  }
08  // 4 + 4 + 4 = 16 bytes
09
```

CODE 4 Constant Space O(1)

a , x and y : is constant so each of them.
The space complexity of the calculateSum function is O(1+1+1).
The space complexity is constantO(1).

```
01  public int[] createArray(int n) {
02  // 4 bytes
03      int[] arr = new int[n];
04  // 4 * n bytes
05      for (int i = 0; i < n; i++) {
06  // bytes
07          arr[i] = i;
08      }
09      return arr;
10  // 4 bytes
11  }
```

CODE 5 Linear Space O(n)

int n: is constant so 1 .
arr: is n so the space complexity is linear n.
i : is constant so 1.
O(1+n+1).
The space complexity is linear O(n).

```java
public int[][] create2DArray(int n) {
// 4 bytes
    int[][] arr = new int[n][n];
// 4 * n * n bytes
    for (int i = 0; i < n; i++) {
// 4 bytes
        for (int j = 0; j < n; j++) {
// 4 bytes
            arr[i][j] = i * n + j;
        }
    }
    return arr;
// 4 bytes
}
```

CODE 6 Quadratic Space

int n: is constant so 1.
arr: size of n x n so, the space complexity of a 2D array is n^2.
i and j: is constant so 1.
The space complexity of create2DArray is O(n^2).