

**“AZƏRBAYCAN HAVA YOLLARI” CJSC NATIONAL AVIATION ACADEMY**

**Individual Work № 2:**

Topic**:**  **Space complexity with examples in Python**

**Subject: Teacher: Mehemmed Shahmaliyev**

**Group: 1459i Student: Aliyeva Ilaha**

**Date: Signature:**

**Baku 2021**

## 1. Introduction

**Space complexity measures the total amount of memory that an algorithm or operation needs to run according to its input size.**

In this tutorial, we’ll see different ways to quantify space complexity. Moreover, we’ll analyze the total space taken via some examples.

Lastly, we’ll discuss how space and time complexity impact each other.

## 2. Notations

There are different notations we can use to express space complexity measurements. The most widely used is [big-O notation](https://www.baeldung.com/cs/big-o-notation), and that’ll be our main focus. Also, we’ll briefly define other common notations.

### 2.1. Big-O Notation

Big-O notation describes an asymptotic upper bound. It represents the algorithm’s scalability and performance.

Simply put, **it gives the worst-case scenario of an algorithm’s growth rate.**We can say that: “the amount of space this algorithm takes will grow no more quickly than this f(x), but it could grow more slowly.”

Let’s see a few examples of expressing space complexity using big-O notation, starting from slowest space growth (best) to fastest (worst):

1. O(1) – constant complexity – takes the same amount of space regardless of the input size
2. O(log n) – logarithmic complexity – takes space proportional to the log of the input size
3. O(n) – linear complexity – takes space directly proportional to the input size
4. O(n log n) – log-linear/quasilinear complexity – also called “linearithmic”, its space complexity grows proportionally to the input size and a logarithmic factor
5. O(n^2) – square/polynomial complexity – space complexity grows proportionally to the square of the input size

### 2.2. Omega Notation – Ω

Omega notation expresses an asymptotic lower bound.

So, **it gives the best-case scenario of an algorithm’s complexity**, opposite to big-O notation. We can say that: “the amount of space this algorithm takes will grow no more slowly than this fix), but it could grow more quickly.”

Let’s analyze a simple example to illustrate why we prefer big-O notation over omega notation. Using Ω, we could say that the richest person on Earth owns at least $10. It is a true sentence but obviously far from being precise.

### 2.3. Theta Notation – θ

Theta notation represents a function that is within lower and upper bounds.

We can say that: “the algorithm’s space takes at least that (lower bound function) amount of space and no more than that (maximum bound function) amount of space”.

## 3. Analyzing Space Complexity of Algorithms

The ability to calculate space complexity is essential in considering an algorithm’s efficiency. In this section, we’ll analyze the space complexity of a few programs of differing difficulty. We’ll measure it using big-O notation.

Above all, 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**.

We’ll use Java for our examples, although, each of them could be easily applied to any technology. To apply them to other programming languages, we can simply replace the Java data types with language-specific ones. Then in our calculations, we’d use the sizes that correspond to the language-specific data types.

Space complexity is the total amount of memory space used by an algorithm/program including the space of input values for execution.  
Often space complexity is confused with auxiliary space. **Auxiliary space** is temporary or extra space used by an algorithm.  
In simple terms  
Space Complexity = Auxiliary space + space used by the input values

Space complexity uses the same notation as time complexity so expect O(n), O(1) etc.  
Let’s look at this simple function using ruby and try to figure out the space complexity

***def sum***

***array = [1,2,3]***

***sum = 0***

***array.each {|i| sum = sum + i}***

***puts sum***

***end***

***sum***