

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

**Individual Work № 1:**

## **Topic: What is Time complexity?**

**Subject: Teacher: Mehemmed Shahmaliyev**

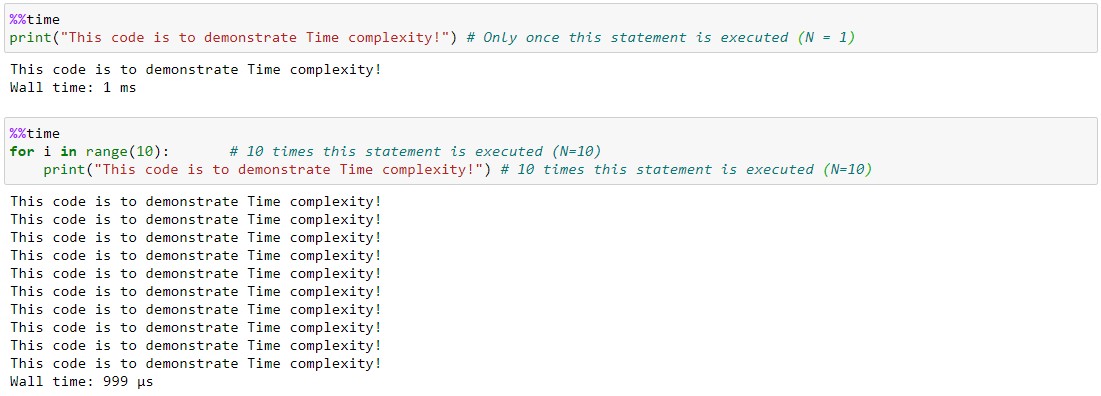
**Group: 1459i Student: Aliyeva Ilaha**

**Date: Signature:**

**Baku 2021**

## **What is Time complexity?**

By definition, the time complexity is the amount of time taken by an algorithm to run, as a function of the length of the input. Here, the length of input indicates the number of operations to be performed by the algorithm. This gives a clear indication of what exactly Time complexity tells us. It is not going to examine the total execution time of an algorithm. Rather, it is going to give information about the variation (increase or decrease) in execution time when the number of operations (increase or decrease) in an algorithm. Yes, as the definition says, the amount of time taken is a function of the length of input only.

To elaborate, Time complexity measures the time taken to execute each statement of code in an algorithm. If a statement is set to execute repeatedly then the number of times that statement gets executed is equal to N multiplied by the time required to run that function each time.

The first algorithm is defined to print the statement only once. The time taken to execute is shown as **0 nanoseconds**. While the second algorithm is defined to print the same statement but this time it is set to run the same statement in FOR loop 10 times. In the second algorithm, the time taken to execute both the line of code – FOR loop and print statement, is **2 milliseconds**. And, the time taken increases, as the N value increases, since the statement is going to get executed N times.

By now, you could have concluded that when an algorithm uses statements that get executed only once, will always require the same amount of time, and when the statement is in loop condition, the time required increases depending on the number of times the loop is set to run. And, when an algorithm has a combination of both single executed statements and LOOP statements or with nested LOOP statements, the time increases proportionately, based on the number of times each statement gets executed.

This leads us to ask the next question, about how to determine the relationship between the input and time, given a statement in an algorithm. To define this, we are going to see how each statement gets an order of notation to describe time complexity, which is called Big**O Notation**.

## **What are the different types of Time complexity notation used?**

As we have seen, Time complexity is given by time as a function of the length of the input. And, there exists a relation between the input data size (n) and a number of operations performed (N) with respect to time. This relation is denoted as Order of growth in Time complexity and given notation O[n] where O is the order of growth and n is the length of the input. It is also called as **‘Big O Notation’**

Big O Notation expresses the run time of an algorithm in terms of how quickly it grows relative to the input ‘n’ by defining the N number of operations that are done on it. Thus, the time complexity of an algorithm is denoted by the combination of all O[n] assigned for each line of function.

There are different types of time complexities used, let’s see one by one:

**1. Constant time – O (1)**

**2. Linear time – O (n)**

**3. Logarithmic time – O (log n)**

**4. Quadratic time – O (n^2)**

**5. Cubic time – O (n^3)**

and many more complex notations like **Exponential time, Quasilinear time, factorial time, etc.** are used based on the type of functions defined.

## **Constant time – O (1)**

An algorithm is said to have constant time with order O (1) when it is not dependent on the input size n. Irrespective of the input size n, the runtime will always be the same. Example as shown:

## **Linear time – O(n)**

An algorithm is said to have a linear time complexity when the running time increases linearly with the length of the input. When the function involves checking all the values in input data, such function has Time complexity with this order O(n).

## **Logarithmic time – O (log n)**

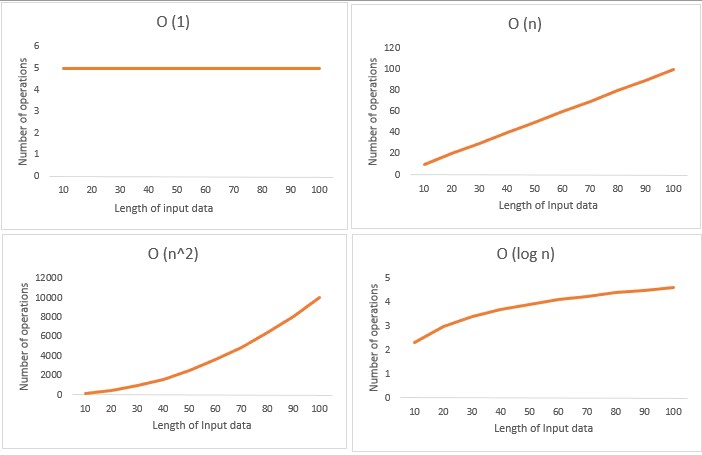
An algorithm is said to have a logarithmic time complexity when it reduces the size of the input data in each step. This indicates that the number of operations is not the same as the input size. The number of operations gets reduced as the input size increases. Algorithms with Logarithmic time complexity are found in binary trees or binary search functions. This involves the search of a given value in an array by splitting the array into two and starting searching in one split. This ensures the operation is not done on every element of the data.

## **Quadratic time – O (n^2)**

An algorithm is said to have a non – linear time complexity where the running time increases non-linearly (n^2) with the length of the input. Generally, nested loops come under this time complexity order where for one loop takes O(n) and if the function involves a loop within a loop, then it goes for O(n)\*O(n) = O(n^2) order.

Similarly, if there are ‘m’ loops defined in the function, then the order is given by O (n ^ m), which are called **polynomial time complexity** functions.

The order of growth for all time complexities are indicated in the graph below:



In this post, we had introduced the basic concepts of Time complexity and the importance of why we need to use it in the algorithm we design. Also, we had seen what are the different types of time complexities used for various kinds of functions, and finally, we learned how to assign the order of notation for any algorithm based on the cost function and the number of times the statement is defined to run.

Given the condition of the big data, the flow of data is increasing unconditionally by every second and designing an effective algorithm to perform a specific task, is needed of the hour. And, knowing the time complexity of the algorithm with given input data size, can help us to plan our resources, process and provide the results efficiently and effectively. Thus, knowing the time complexity of your algorithm, can help you do that and also makes you an effective programmer. Happy Coding!