**Big-O shows** how efficient an algorithm is in the *worst-case scenario* relative to its input size.

To measure the efficiency of an algorithm, we need to consider two things:

* **Time Complexity:** How much time does it take to run completely?
* **Space Complexity:** How much extra space does it require in the process?

Big-O notation captures the upper bound to show how much time or space an algorithm would require in the *worst-case scenario* as the input size grows. It is usually written as:

***f(n) = O(inputSize)f*(*n*)=*O*(*inputSize*)**

## **How is complexity calculated?**

Time complexity is determined by taking two factors into account: the input size and the solution of the algorithm. Here’s a generic way to calculate the complexity:

1. List down all the basic operations in the code
2. Count the number of times each gets executed
3. Sum all the counts to get an equation in terms of *n*

## **Example:** Let’s look at the following code and see how we can calculate its complexity if the input size is equal to n:

|  |
| --- |
| #include <iostream> using namespace std;  int main() {  int sum = 0;  for (int i=0;i<5;i++){  sum = sum+i;  }  cout << "Sum = " << sum; return 0;  } |

Let’s list down all the statements along with their execution count:

|  |  |
| --- | --- |
| **Operations** | **Num of Executions** |
| int sum = 0 | 1 |
| for (int i=0;i<5;i++) | 6 |
| sum = sum+i | 5 |
| cout << "Sum = " << sum | 1 |
| return 0 | 1 |

## **Calculations:**

***1+6+5+1+1*1+6+5+1+1**

Generalizing this notation in terms of input size (n) would form this expression:

***=>1+(n+1) + n + 1+1*=>1+(*n*+1) +*n*+1+1**

After simplifying the above expression, the final time complexity would be:

***2n +4*2*n*+4**

## **How do you estimate the Big-O notation of an algorithm?**

To find Big-O notation, follow two steps:

* Discard the leading constants
* Ignore the lower order terms

After performing the above two steps on the time complexity that we just calculated, we can estimate the Big-O notation as:

***=> 2n+4*=>2*n*+4 *=> n+4*=>*n*+4 *=> n*=>*n* *=> O(n)*=>*O*(*n*)**