What Are Big O Notations?

For any problem in the world, there will be number of possible solutions. Computer science world is also no different. In computer science field, everyday you come across thousands of problems. There also exist number of possible solutions to those problems. In computer science, we call solutions as **algorithms**. Your task is to find out the best possible algorithm so that it should solve a given problem in less time by using less memory space. In such situations, **Big O Notations** comes into picture.

In simple terms, **Big O notations** tell us how good is the algorithm in terms of execution time and memory space used. They tell us how an algorithm performs when input data is increased.

**Big O notations** consist of O and an expression enclosed within ( ). Expression within ( ) represents the number of computations a particular algorithm has to perform to solve a given problem.

The symbols **O(N)**, **O(N^2)**, **O(log N)** etc are some examples of Big O Notations. Let’s discuss these notations one by one.

**O(1) :**

O(1) represents an algorithm which has to perform exact one computation to solve a problem. O(1) represents an algorithm which takes same time to execute for any size of input data. In other words, O(1) denotes an algorithm which performance is not affected by the size of the input data. For example, below algorithm checks whether the first element of an array is zero or not. It always checks the first element of an array. It doesn’t bother about the remaining elements. It performs only one computation for any size of input data.

[?](http://javaconceptoftheday.com/big-o-notations-tutorials/)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | boolean isFirstElementZero(int[] inputData)  {      if(inputData[0] == 0)      {          return true;      }        return false;  } |

**O(N) :**

O(N) represents an algorithm which has to perform **‘N’** computations to solve a problem. Where **‘N’** is the number of elements in input data. That means algorithm will take more time as number of elements in input data increases. In the other words, O(N) denotes an algorithm whose performance is directly proportional to size of the input data. i.e O(N) denotes an algorithm whose execution time increases linearly as input data grows.

The below algorithm performs maximum N computations (In worst case)  to search for an element.

[?](http://javaconceptoftheday.com/big-o-notations-tutorials/)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12 | boolean searchForElement(int[] inputData, int n)  {      for (int i = 0; i < inputData.length; i++)      {          if(inputData[i] == n)          {              return true;          }      }        return false;  } |

**O(N^2) :**

O(N^2) denotes an algorithm which has to perform **N^2** computations to solve a problem. Where **N** is the number of elements in input data. In the other words, O(N^2) represents an algorithm whose performance is directly proportional to square of the size of the input data. The algorithms which have nested iterations over the input data will give O(N^2) performance. Further deeper iterations over the input data will give O(N^3), O(N^4) and so on.

[?](http://javaconceptoftheday.com/big-o-notations-tutorials/)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15 | boolean findDuplicates(int[] inputData)  {      for (int i = 0; i < inputData.length; i++)      {          for (int j = 0; j < inputData.length; j++)          {              if(inputData[i] == inputData[j] && i!=j)              {                  return true;              }          }      }        return false;  } |

**O(log N) :**

O(log N) represents an algorithm whose number of computations grows linearly as input data grows exponentially. i.e O(log N) represents the algorithms whose execution time grows linearly as input data grows exponentially. That means if an algorithm takes 1 second to compute 10 elements, then it will take 2 seconds to compute 100 elements, 3 seconds to compute 1000 elements and so on. The best example for O(log N) is binary search algorithm which uses divide and conquer rule to search for an element.

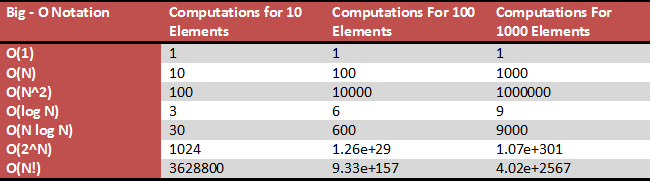
**O(2N) :**

O(2N) represents an algorithm whose execution time is doubled for every extra element in the input data. For example, if an algorithm takes 4 seconds to compute 2 elements, then it will take 8 seconds to compute 3 elements, 16 seconds for 4 elements, 32 seconds for 5 elements and so on.

**O(N!) :**

O(N!) represents an algorithm which has to perform **N!** computations to solve a problem. Where **N** is the number of elements in the input data. For example, if an algorithm takes 2 seconds to compute 2 elements, then it will take 6 seconds to compute 3 elements, 24 seconds to compute 4 elements and so on.

There are many Big O notations used to measure the performance of an algorithm. Below is the list of some of the most used Big O notations and their performance comparisons against different sizes of the input data.



**Note** : All Big O notations measure the performance of algorithms considering the worst case of the input data.