Big O Notation

<http://www.apekshit.com/t/46/Big-O-Notation>

Efficiency of the program depends on how much time it takes to execute and provide accurate result. It is required to measure the growth rate of the program. It means, analysing the time with respect to increase in input elements.

Let's try to understand this with the help of example.

You want to find a number from given 5 integer numbers. What are the alternatives to find  a number 41 from given set of numbers  {9, 3, 8, 5, 41}.

**Option 1:**

Iterate over the list of elements and check it one by one till the time you find the matching number.

    It may be possible that you find the number at first place, if you are lucky. It may be possible that you get number in second comparison or may be third or may be fourth or may be fifth or may be not....

    The best case is the one when you get the matching number in first attempt.

    If you try 10 times, 100 times, 1000 times, it is possible that on an average you get matching number in 2nd or 3rd time. That is average case.

    In worst case scenario, you have to check all numbers.

    While comparing the algorithms, it is generally compared based on average and worst case scenario.

    In the example above, what if you want to find a number from the set of 1000 numbers. In worst case, you will have to check all 1000 numbers. It means that the proposed solution has growth rate that depends on the size of the input elements. In mathematical terms if the number of elements is n, it will check n numbers. Number of comparison is increasing with increase in number of elements. Hence it is called linear function.

**Option 2:**

Assume that you have sorted set of elements. You have new sorted set {3, 5, 8, 9, 41} Now what...?

    Start with the middle element and do comparison. If you find the number in first attempt, that is best case. But here middle element is 8. That is not the number you are searching (number 41). Since you know that this is sorted set of elements and hence you just need to check from the middle element to last element. You can simply ignore the other half of the set. Again apply similar logic to the upper half of the set. Continue till you find the elements or reach to a conclusion that number does not exist.

As you can see that in each attempt, number of elements is reduced by half. So in a set of 1000 numbers, the worst case scenario will be like this :

  1. After 1st attempt, you are left with 500 numbers.

  2. After 2nd attempt, you are left with 250 numbers.

  3. After 3rd attempt, you are left with 125 numbers.

  4. After 4th attempt, you are left with 62 numbers.

  5. After 5th attempt, you are left with 31 numbers.

  6. After 6th attempt, you are left with 15 number.

  7. After 7th attempt, you are left with 7 numbers.

  8. After 8th attempt, you are left with 3 number.

  9. After 9th attempt, you are left with 1 numbers.

  10. After 10th attempt, you know the result. Either you found the matching number or you know for sure that number does not exist in the given set of numbers.

As you can see that worst case scenario needs only 10 comparison versus 1000 comparison in Option 1.

In mathematical terms if the number of elements is N, it will check log2N numbers. Hence it is called logarithmic function.

n computer science, this is also referred as Big O notation. It is a measure of growth rate of the function. Various functions are constant O(1), linear O(n), logarithmic O(log n) and quadratic O(n2).

# Constant Time complexity

Constant Time O(1) functions needs fixed amount of time to execute program or algorithm. It does not depend on number of inputs.

# Linear Time complexity

Functions or Programs with Linear Time Complexity ( O(N) complexity ) needs amount of time that directly propotional to the number of input elements.

# Logarithmic Complexity Algorithm

Logarithmic Log2N functions needs amount of time that is in logarthimc propotional to the number of input elements.

# Quadratic Time Complexity

Quadratic N2 functions needs amount of time that is in quadratic propotional to the number of input elements.

Known algorithms having quadratic time complexity are bubble sort algorithm, selection sort algorithm and insertion sort algorithm.

Any algorithm that involves two loops, one outer loop and one inner loop and both loop iterates N number of times where N is the number of elements in a collection is considered as quadratic function.

Assume that you have two boxes with balls inside it. Each ball has unique color in its own box. It might be possible that same color of ball may exist in another box. You want to create a pair of ball ( one from each box ). How many number of pairs you can create ? Position of the ball matters.

Let's name the boxes as box1 and box2. box1 has 3 balls and box2 has 3 balls. We can represent Boxes as array of Strings.

box1 =  {"RED", "GREEN", "YELLOW"};

box2 =  {"PINK" , "WHITE" , "BLACK"};

Possible Pairs are :

{"RED", "PINK"}, {"RED", "WHITE"}, {"RED", "BLACK"}

{"GREEN", "PINK"}, {"GREEN", "WHITE"}, {"GREEN", "BLACK"}

{"YELLOW", "PINK"}, {"YELLOW", "WHITE"}, {"YELLOW", "BLACK"}

As you can see total 9  ( 3\*3) possibilities are there in which you can select pair of balls, one from each box. For each ball ( total 3 ball ) in first box needs 3 selection from the second box. If you have N1 number of balls in box1 and N2 number of balls in box2, you need N1 \* N2 selection.

Programatic implementation of above solution appears as shown below:

**Quadratic Time Complexity Algorithm Example:**

package com.Jwalant.model;

public class Example {

public static void main(String args[]) {

String[] box1 = {"RED", "GREEN", "YELLOW"};

String[] box2 = {"PINK" , "WHITE" , "BLACK"};

for(int i=0; i< box1.length; i++) {

for(int j=0; j< box2.length; j++) {

System.out.println(" Pair is : ("+box1[i]+","+box2[j]+")");

}} }}

As you can see above that for each iteration of the outer loop, you need to complete inner loop.

In mathematical terms if the number of elements is N, it will get N2 selection. Hence it is called quadratic function.