**Omega Notation**

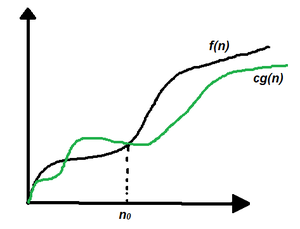
This article will discuss Big - Omega Notation represented by a Greek letter (Ω).

**Definition:** Let g and f be the function from the set of natural numbers to itself. The function f is said to be Ω(g), if there is a constant c > 0 and a natural number n0 such that c\*g(n) ≤ f(n) for all n ≥ n0

**Mathematical Representation:**

Ω(g) = {f(n): there exist positive constants c and n0 such that 0 ≤ c\*g(n) ≤ f(n) for all n ≥ n0}   
Note: Ω (g) is a set

**Graphical Representation:**

Graphical Representation

In simple language, Big - Omega(Ω) notation specifies the asymptotic (at the extreme) lower bound for a function f(n).

Follow the steps below to calculate Big - Omega(Ω) for any program:

1. Break the program into smaller segments.
2. Find the number of operations performed for each segment(in terms of the input size) assuming the given input is such that the program takes the least amount of time.
3. Add up all the operations and simplify it, let's say it is f(n).
4. Remove all the constants and choose the term having the least order or any other function which is always less than f(n) when n tends to infinity, let say it is g(n) then, Big - Omega (Ω) of f(n) is Ω(g(n)).

**Example:**Consider an example to print all the possible pairs of an array. The idea is to run two nested loops to generate all the possible pairs of the given array.

The pseudo-code is as follows:

int print(int a[], int n)

{

for (int i = 0; i < n; i++)

{

for (int j = 0; j < n; j++)

{

if(i != j)

cout << a[i] << " "

<< a[j] << "\n";

}

}

}

Below is the implementation of the above approach:

C++Java

// C++ program for the above approach

#include <bits/stdc++.h>

using namespace std;

// Function to print all possible pairs

int print(int a[], int n)

{

for (int i = 0; i < n; i++) {

for (int j = 0; j < n; j++) {

if (i != j)

cout << a[i] << " " << a[j] << "\n";

}

}

}

// Driver Code

int main()

{

// Given array

int a[] = { 1, 2, 3 };

// Store the size of the array

int n = sizeof(a) / sizeof(a[0]);

// Function Call

print(a, n);

return 0;

}

**Output**

1 2

1 3

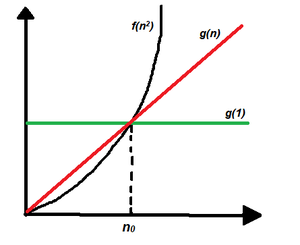
2 1

2 3

3 1

3 2

In this example, it is evident that the print statement gets executed n2 times therefore if the running time vs n graph is plotted a parabolic graph will be obtained, f(n2). Now linear functions g(n), logarithmic functions g(log n), constant functions g(1) all are less than a parabolic function when the input range tends to infinity therefore, the worst-case running time of this program can be Ω(log n), Ω(n), Ω(1), or any function g(n) which is less than n2when n tends to infinity. See the below graphical representation:



**When to use Big - Ω notation:**Big - Ω notation is the least used notation for the analysis of algorithms because it can make a correct but imprecise statement over the performance of an algorithm. Suppose a person takes 100 minutes to complete a task, using Ω notation, it can be stated that the person takes more than 10 minutes to do the task, this statement is correct but not precise as it doesn't mention the upper bound of the time taken. Similarly, using Ω notation we can say that the worst-case running time for the [binary search](https://www.geeksforgeeks.org/binary-search/) is Ω(1), which is true because we know that binary search would at least take constant time to execute.

Mark as Read

Report An Issue