## Algorithm Analysis

Suppose **X** is an algorithm and **n** is the size of input data, the time and space used by the algorithm X are the two main factors, which decide the efficiency of X.

* **Time Factor** − Time is measured by counting the number of key operations such as comparisons in the sorting algorithm.
* **Space Factor** − Space is measured by counting the maximum memory space required by the algorithm.

The complexity of an algorithm **f(n)** gives the running time and/or the storage space required by the algorithm in terms of **n** as the size of input data.

## Space Complexity

Space complexity of an algorithm represents the amount of memory space required by the algorithm in its life cycle. The space required by an algorithm is equal to the sum of the following two components −

* A fixed part that is a space required to store certain data and variables, that are independent of the size of the problem. For example, simple variables and constants used, program size, etc.
* A variable part is a space required by variables, whose size depends on the size of the problem. For example, dynamic memory allocation, recursion, stack space, etc.

Space complexity S(P) of any algorithm P is S(P) = C + SP(I), where C is the fixed part and S(I) is the variable part of the algorithm, which depends on instance characteristic

## Time Complexity

Time complexity of an algorithm represents the amount of time required by the algorithm to run to completion. Time requirements can be defined as a numerical function T(n), where T(n) can be measured as the number of steps, provided each step consumes constant time.

For example, addition of two n-bit integers takes **n** steps. Consequently, the total computational time is T(n) = c ∗ n, where c is the time taken for the addition of two bits. Here, we observe that T(n) grows linearly as the input size increases.

## Asymptotic Analysis

Asymptotic analysis of an algorithm refers to defining the mathematical boundation/framing of its run-time performance. Using asymptotic analysis, we can very well conclude the best case, average case, and worst case scenario of an algorithm.

Asymptotic analysis refers to computing the running time of any operation in mathematical units of computation. For example, the running time of one operation is computed as *f*(n) and may be for another operation it is computed as *g*(n2). This means the first operation running time will increase linearly with the increase in **n** and the running time of the second operation will increase exponentially when **n** increases. Similarly, the running time of both operations will be nearly the same if **n** is significantly small.

Usually, the time required by an algorithm falls under three types −

* **Best Case** − Minimum time required for program execution.
* **Average Case** − Average time required for program execution.
* **Worst Case** − Maximum time required for program execution.

## Asymptotic Notations

Following are the commonly used asymptotic notations to calculate the running time complexity of an algorithm.

* Ο Notation (Big Oh) Worst Case
* Ω Notation(Omega) Best Case
* θ Notation(Theta) Average Case

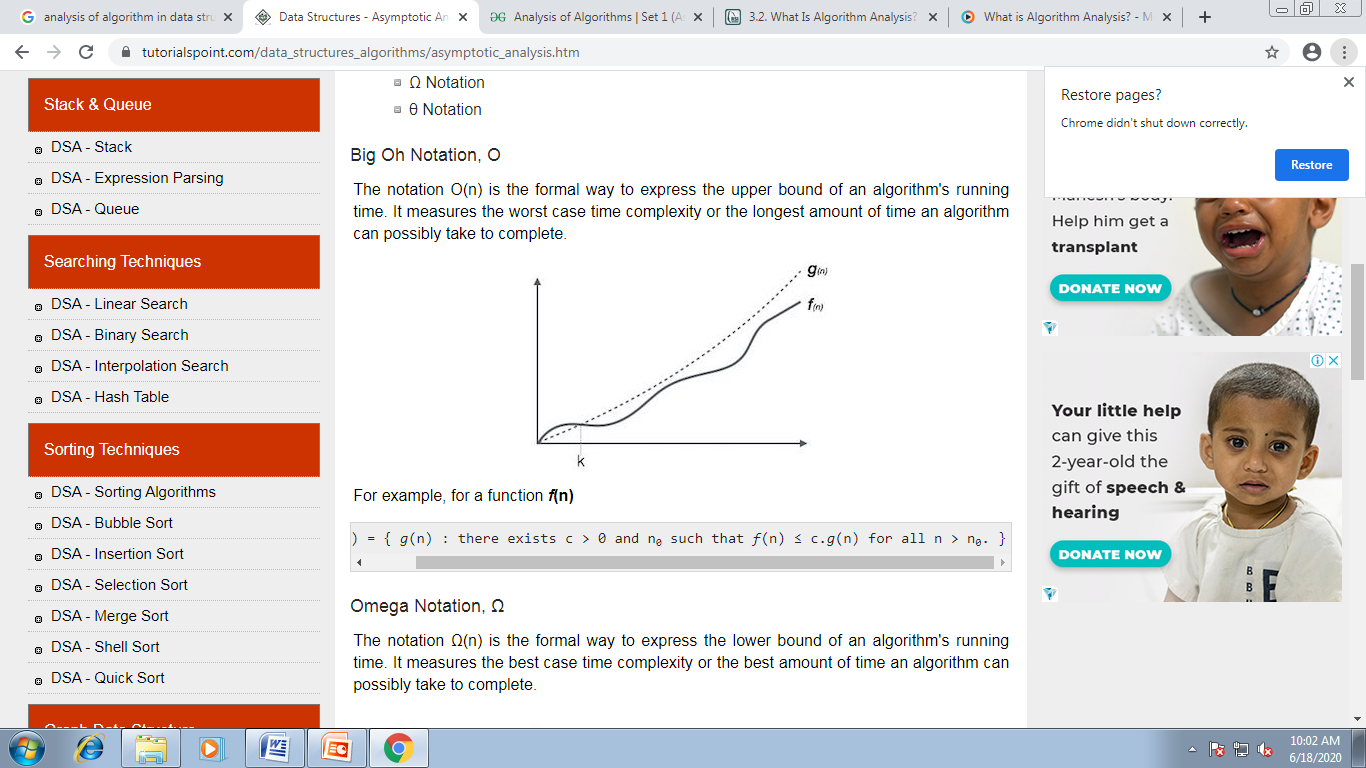
### Big Oh Notation, Ο

The notation Ο(n) is the formal way to express the upper bound of an algorithm's running time. It measures the worst case time complexity or the longest amount of time an algorithm can possibly take to complete.

For example, for a function ***f*(n)**

Ο(*f*(n)) = { *g*(n) : there exists c > 0 and n0 such that *f*(n) ≤ c.*g*(n) for all n > n0. }

**Graphical Representation-**



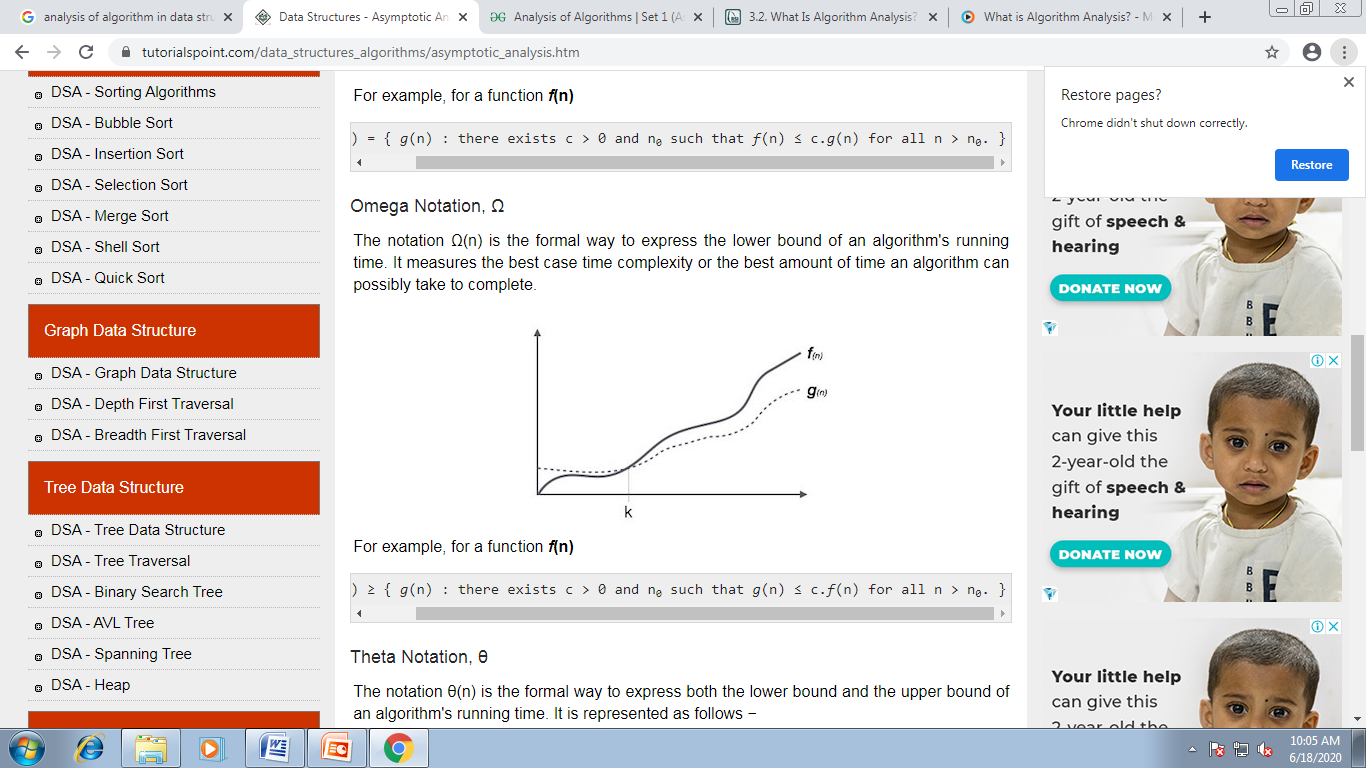
### Omega Notation, Ω

The notation Ω(n) is the formal way to express the lower bound of an algorithm's running time. It measures the best case time complexity or the best amount of time an algorithm can possibly take to complete.

For example, for a function ***f*(n)**

Ω(*f*(n)) ≥ { *g*(n) : there exists c > 0 and n0 such that *g*(n) ≤ c.*f*(n) for all n > n0. }

**Graphical Representation-**

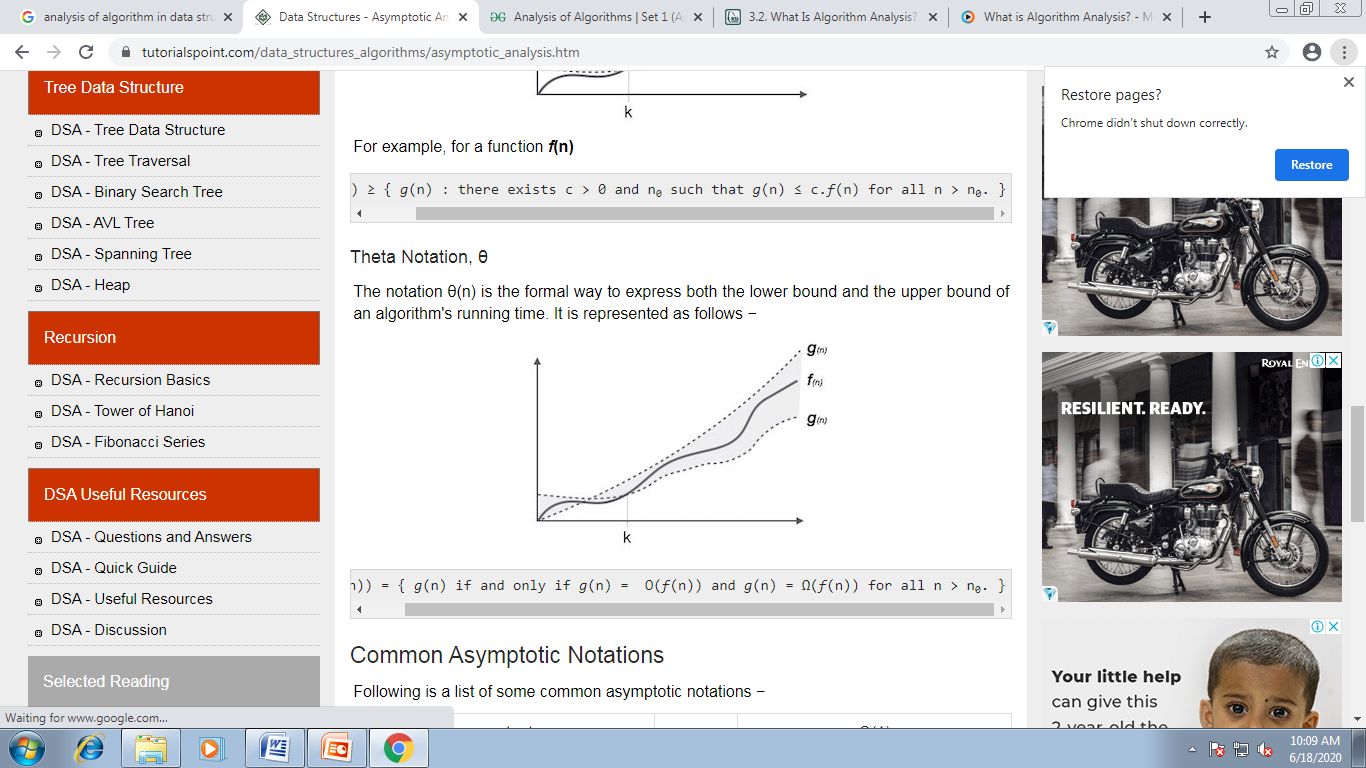


### Theta Notation, θ

The notation θ(n) is the formal way to express both the lower bound and the upper bound of an algorithm's running time.

For example, for a function ***f*(n) and g(n)**

θ(*f*(n)) = { *g*(n) if and only if *g*(n) = Ο(*f*(n)) and *g*(n) = Ω(*f*(n)) for all n > n0. }



## Common Asymptotic Notations-

