**COMPUTER ALGORITHMS**

**1.2 Asymptotic Notations | Big O | Big Omega | Theta Notations**Asymptotic Notations are the mathematical ways of representing the time complexity.  
  
In the previous topic, we have discussed about the **Priori Analysis**, i.e., the analysis of the algorithm carried out before the execution.   
To determine the better algorithm amongst the other given algorithms, three of the important notations used are:   
i] Big O Notation  
ii] Big Omega Notation  
iii] Theta Notation

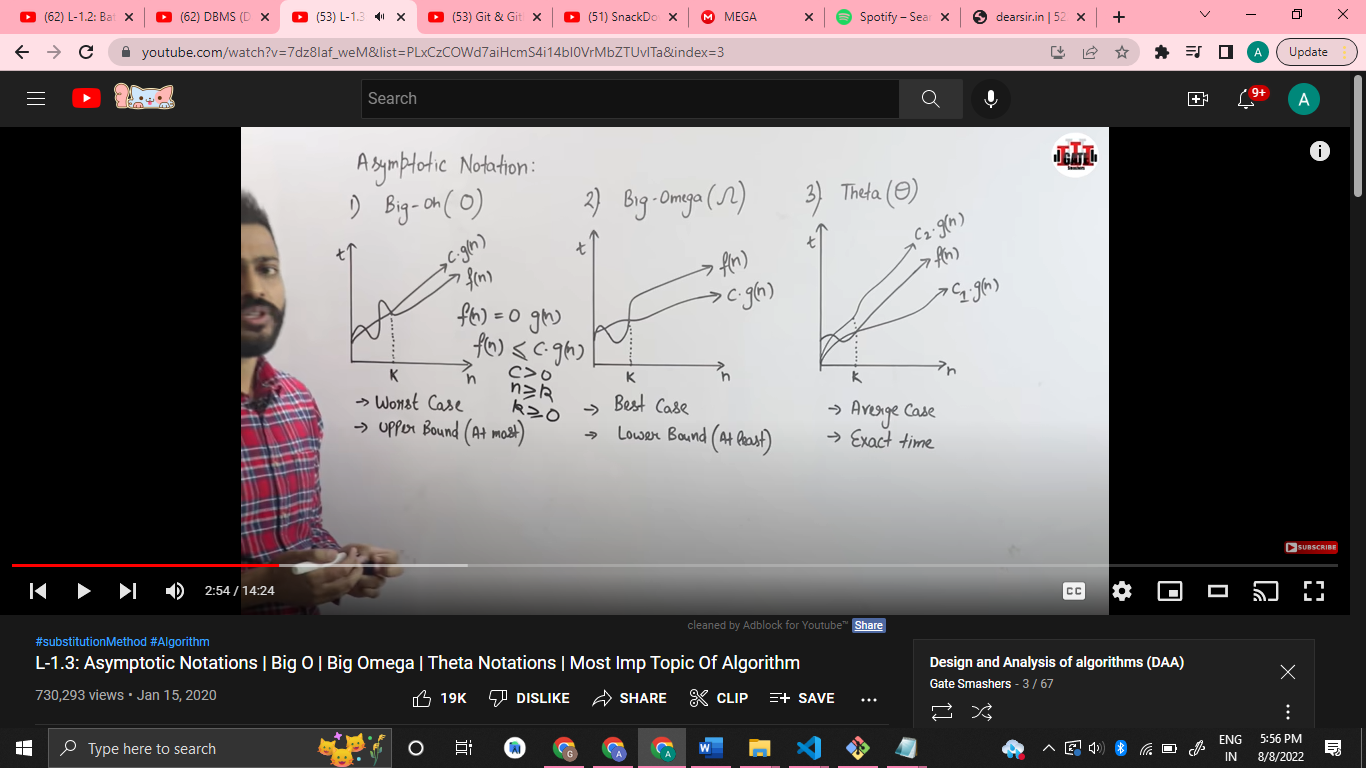
Diagram

Description automatically generated

|  |
| --- |
| n= number of input values. t=time. k= threshold, minimum value required to achieve the order of the f(n). f(n)= problem to be solved. g(n)= f(n) represented in terms of order. |

Iteration 🡺 Number of times a statement is being executed.

* Big O Notation
* Worst Case
* Upper Bound (At Most)

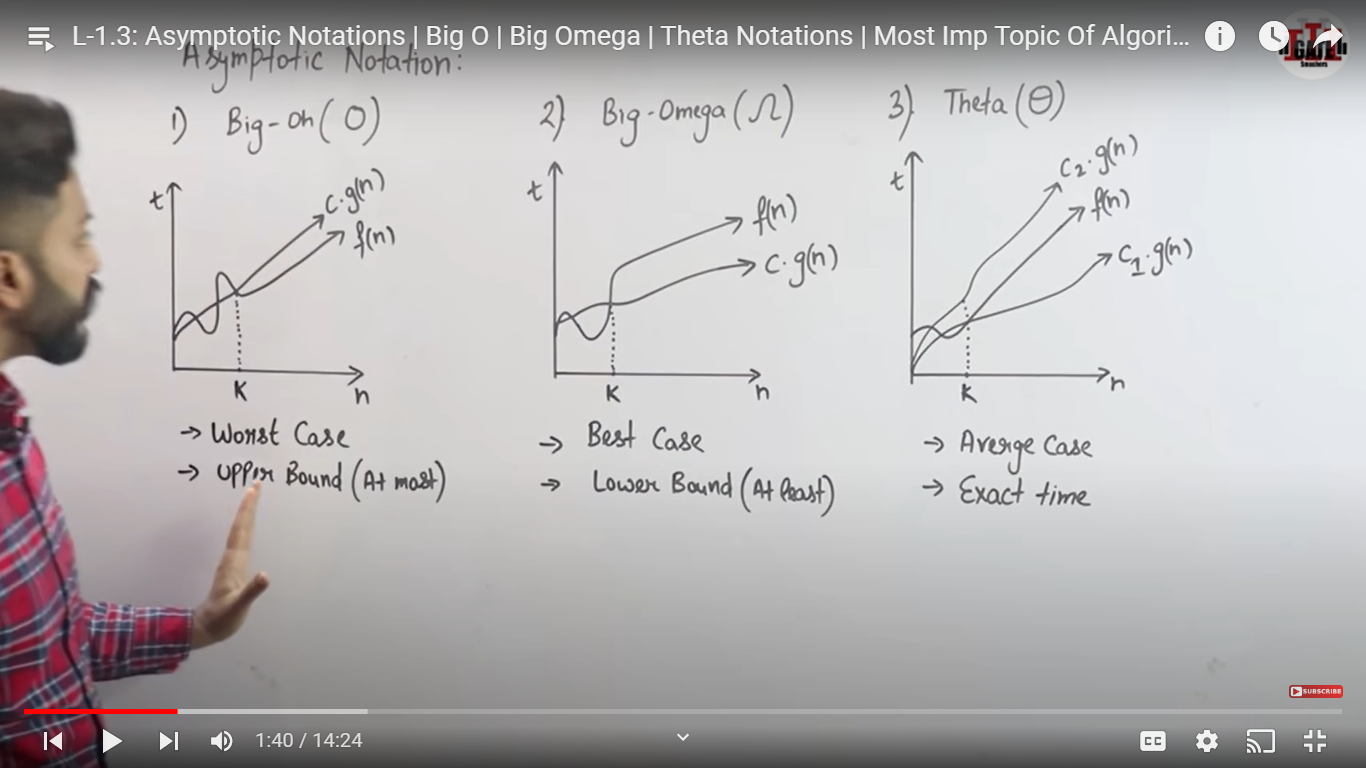


|  |
| --- |
| Required Conditions: 1] f(n) = O g(n) 2] f(n) ≤ c\*g(n) 3] c > 0 4] n ≥ k 5] k ≥ 0 |

Big O Notation represents the upper bound, that is the larger value. So, it mostly represents the maximum (at most) time taken to complete the execution.

|  |
| --- |
| Consider, **f(n)=2n² + n**;  Here, the upper bound indicates the greater value between the two terms mentioned in the equation.  According to the given conditions,  f(n) ≤ c\*g(n); if f(n) is being represented through the order of the g(n). Thus,  2n² + n ≤ c\* g(n); So, as per the above equation, **n²** is the dominating term as compared to **n**; since it would always deliver a greater value. It is advised to choose the least value closest to the upper bound. Thus, **n²** is the least upper bound.  Therefore, 2n² + n ≤ c\*g(n²); 🡺c=constant that can take non-negative values.  2n² + n ≤ c\* n²; c>=3; 2n² + n ≤ 3n²;  n²-n≥0;  n(n-1) ≥ 0; n≥1, since n>0.  Thus, for all the values of the inputs **n**, n ≥1 will always hold, with c ≥ 3. |

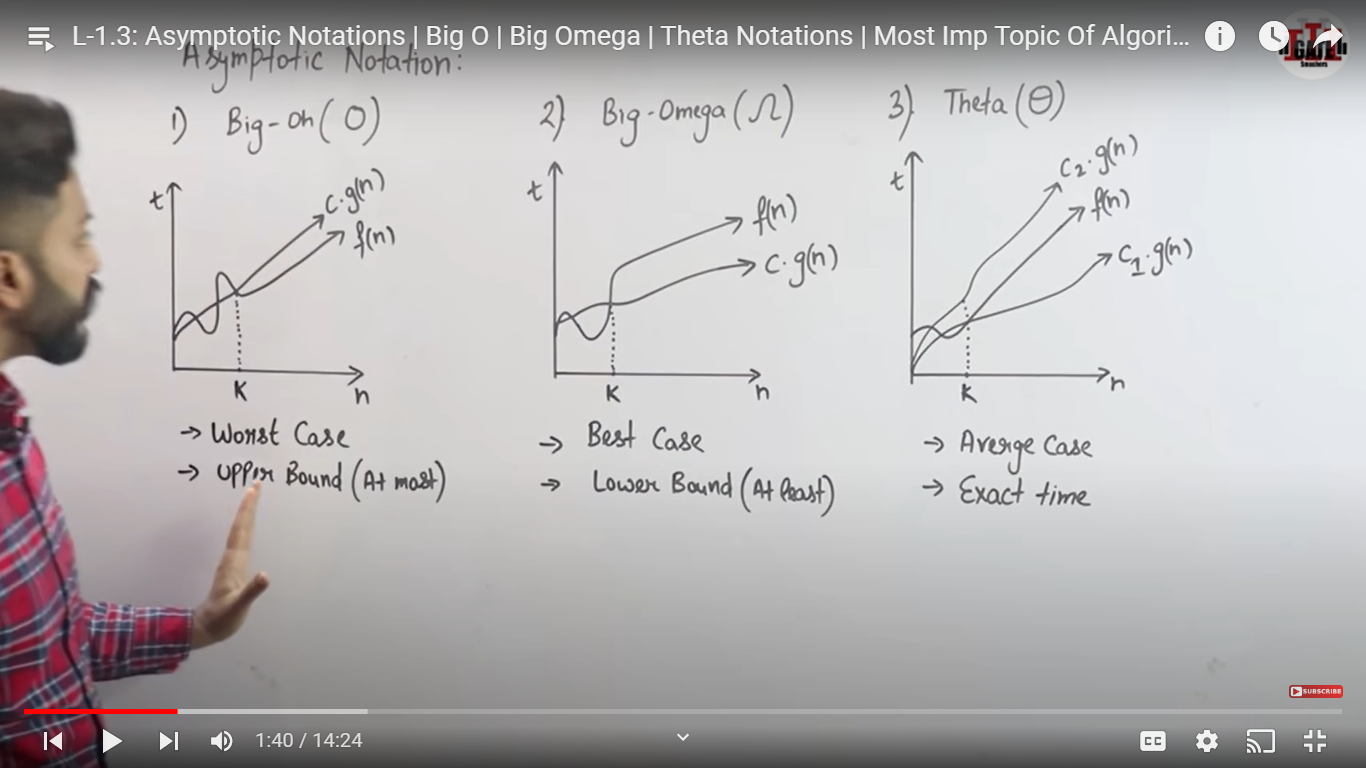
* Big Omega Notation
* Best Case
* Lower Bound (At least)  
    
  Big Omega Notation represents the lower bound, that is the lower value. So, it mostly represents the minimum (at least) time taken to complete the execution.



|  |
| --- |
| Required Conditions: 1] f(n) ≥ Ω g(n) 2] f(n) ≥ c\*g(n) 3] c>0  4] k ≤ n  5] k>0 |

|  |
| --- |
| Consider, **f(n)=2n² + n**;  Here, the lower bound indicates the smaller value between the two terms mentioned in the equation.  According to the given conditions,  f(n) ≤ c\*g(n); if f(n) is being represented through the order of the g(n). Thus,  2n² + n ≥ c\* g(n); So, **n²** and **n,** either of the two can take the value of the g(n), such that the value of the constant **c** should be carefully such that it satisfies the second requirement. It is advised to choose the greatest value closest to the lower bound. Thus, **n²** or **n**, either of them is worthy enough to be the lower bound.  Here, **n²** is considered to represent the g(n). Therefore, 2n² + n ≥ c\*g(n²); 🡺c=constant that can take non-negative values.  2n² + n ≥ c\* n²; c ≤ 2; 2n² + n ≥ 2n²;  n≥0;  n≥ 0; n≥0, since n>0.  Thus, for all the values of the inputs **n**, n ≥0 will always hold, with c ≤ 2. |

* Theta Notation  
    
  Theta Notation represents average value. So, it mostly represents the exact time taken to complete the execution.
* Average Case
* Exact time



|  |
| --- |
| Required Conditions:  1] c1.g(n1) ≤ f(n) ≤ c2.g(n2)  2] c1>0 and c2>0 3] k>0 |

|  |
| --- |
| Consider, **f(n)=2n² + n**; According to the given requirement, c1.g(n1) ≤ f(n) ≤ c2.g(n2); 2n² ≤ 2n² + n ≤ 3n²; |

* Little O Notation

|  |
| --- |
| Required Conditions: 1] f(n) = O g(n) 2] f(n) < c\*g(n) 3] c > 0 4] n ≥ k 5] k ≥ 0 |

* Little Omega Notation

|  |
| --- |
| Required Conditions: 1] f(n) > Ω g(n) 2] f(n) > c\*g(n) 3] c>0  4] k ≤ n  5] k>0 |