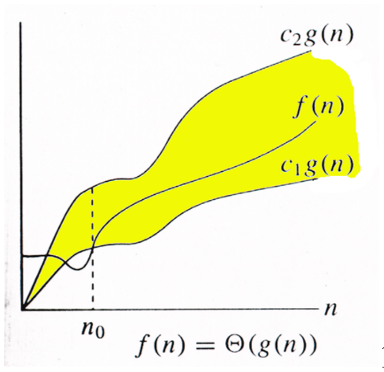
Analysis of Algorithms | Set 3 (Asymptotic Notations)

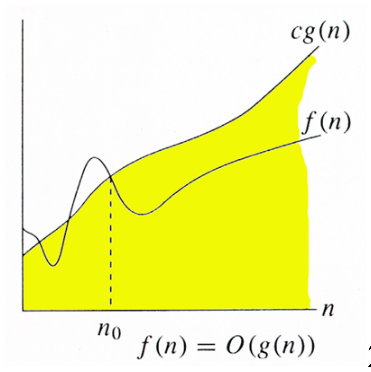
We have discussed Asymptotic Analysis, and Worst, Average and Best Cases of Algorithms. The main idea of asymptotic analysis is to have a measure of efficiency of algorithms that doesn't depend on machine specific constants, and doesn't require algorithms to be implemented and time taken by programs to be compared. Asymptotic notations are mathematical tools to represent time complexity of algorithms for asymptotic analysis. The following 3 asymptotic notations are mostly used to represent time complexity of algorithms.



1) [Tex]\Theta[/Tex] Notation: The theta notation bounds a functions from above and below, so it defines exact asymptotic behavior.
A simple way to get Theta notation of an expression is to drop low order terms and ignore leading constants. For example, consider the following expression.
$3n^3 + 6n^2 + 6000$ = [Tex]\Theta[/Tex](n^3)
Dropping lower order terms is always fine because there will always be a n0 after which [Tex]\Theta[/Tex](n^3) beats [Tex]\Theta[/Tex](n^2) irrespective of the constants involved.
For a given function g(n), we denote [Tex]\Theta[/Tex](g(n)) is following set of functions.

```
[Tex]\Theta[/Tex]((g(n)) = {f(n): there exist positive constants c1, c2 and n0 such that
                 0 <= c1*g(n) <= f(n) <= c2*g(n) for all n >= n0}
```

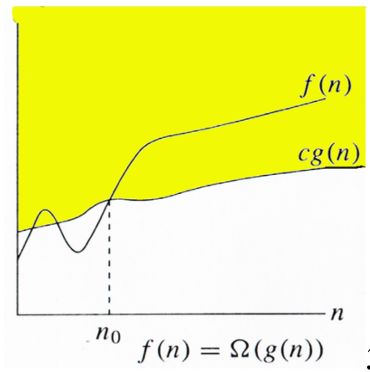
The above definition means, if f(n) is theta of g(n), then the value f(n) is always between c1*g(n) and c2*g(n) for large values of n (n >= n0). The definition of theta also requires that f(n) must be non-negative for values of n greater than n0.



2) Big O Notation: The Big O notation defines an upper bound of an algorithm, it bounds a function only from above. For example, consider the case of Insertion Sort. It takes linear time in best case and quadratic time in worst case. We can safely say that the time complexity of Insertion sort is O(n^2). Note that O(n^2) also covers linear time.
If we use [Tex]\Theta[/Tex] notation to represent time complexity of Insertion sort, we have to use two statements for best and worst cases:
1. The worst case time complexity of Insertion Sort is [Tex]\Theta[/Tex](n^2).
2. The best case time complexity of Insertion Sort is [Tex]\Theta[/Tex](n).

The Big O notation is useful when we only have upper bound on time complexity of an algorithm. Many times we easily find an upper bound by simply looking at the algorithm.

```
O(g(n)) = { f(n): there exist positive constants c and n0 such that
            0 <= f(n) <= cg(n) for all n >= n0}
```



3) [Tex]\Omega[/Tex] Notation: Just as Big O notation provides an asymptotic upper bound on a function, [Tex]\Omega[/Tex] notation provides an asymptotic lower bound.

[Tex]\Omega[/Tex] Notation< can be useful when we have lower bound on time complexity of an algorithm. As discussed in the previous post, the best case performance of an algorithm is generally not useful, the Omega notation is the least used notation among all three.

For a given function g(n), we denote by [Tex]\Omega[/Tex](g(n)) the set of functions.

```
[Tex]\Omega[/Tex] (g(n)) = {f(n): there exist positive constants c and n0 such that
                            0 <= cg(n) <= f(n) for all n >= n0}.
```

Let us consider the same Insertion sort example here. The time complexity of Insertion Sort can be written as [Tex]\Omega[/Tex](n), but it is not a very useful information about insertion sort, as we are generally interested in worst case and sometimes in average case.

Exercise:
Which of the following statements is/are valid?
1. Time Complexity of QuickSort is [Tex]\Theta[/Tex](n^2)
2. Time Complexity of QuickSort is O(n^2)
3. For any two functions f(n) and g(n), we have f(n) = [Tex]\Theta[/Tex] (g(n)) if and only if f(n) = O(g(n)) and f(n) = [Tex]\Omega[/Tex](g(n)).
4. Time complexity of all computer algorithms can be written as [Tex]\Omega[/Tex] (1)

References:

Lec 1 | MIT (Introduction to Algorithms)

Introduction to Algorithms 3rd Edition by Clifford Stein, Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest

This article is contributed by **Abhay Rathi**. Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.

Related Topics:

- An interesting time complexity question
- A Problem in Many Binary Search Implementations
- Analysis of Algorithms | Set 4 (Analysis of Loops)
- NP-Completeness | Set 1 (Introduction)
- Static and Dynamic Libraries | Set 1

- The Ubiquitous Binary Search | Set 1
- Reservoir Sampling
- Analysis of Algorithms | Set 2 (Worst, Average and Best Cases)

Like 34 Tweet 4 g+1 1

Writing code in comment? Please use **ideone.com** and share the link here.

46 Comments GeeksforGeeks 1 Login

Recommend 1 Share Sort by Newest

Join the discussion…

Ammar Baig • 2 months ago

You got a little error Aditiya, first you claim that " We write f(n) = Ω(g(n)) if there exist constants c> 0, n0> 0 (which is right),, and then u deny yourself by saying, " such that 0 ≤cg(n) ≤f(n) for all n≥n0 ",, which is also right except for [0≤] thing as we agreed earlier in this statement that c>0, so in this case true relation that will hold for c.g(n) and f(n)is 0<cg(n)≤f(n) for="" all="" n≥n0.="">

• Reply • Share ›

Aditya Goel • 2 months ago

For all those saying 4th statement is false, they are wrong.
We write f(n) = Ω(g(n)) if there exist constants c> 0, n0> 0 such that 0 ≤cg(n) ≤f(n) for all n≥n0

Here g(n) is 1.

0 <= c <= f(n) holds true for any positive constant c and and algorithm in world(which takes f(n) time). Of couse we can provide more precise lower bound but that doesn't make this statement false.

• Reply • Share ›

Ritz • 2 months ago

Guys what is the lower bound for binary search algorithm ? Upper Bound is O(logn) .. Is lower bound big-Omega of a constant ? Plz explain

• Reply • Share ›

karunakar ➦ Ritz • 2 months ago

its omega(1) because , for binary search elements are in sorted order we are search an element it can be found in mid=(low+high)/2 takes constant time so, it is omega(1) or theta(1)

• Reply • Share ›

calvin • 3 months ago

Hi,
I liked the way you analyzed the asymptotic notations, I was reading them in certain book but I couldn't get the difference well but you made it simple and clear . Thank you.

• Reply • Share ›

DPS • 3 months ago

2 nd 3 are valid option !!

• Reply • Share ›

po • 4 months ago

1,2,3 are valid 4 in invalid
1--->time complexity of quick sort best case theta(nlogn) and worst case is O(n^2) we can say that theta(n^2)

• Reply • Share ›

anshika • 4 months ago

2. and 3. are valid!!

• Reply • Share ›

abc • 7 months ago

Suppose B(n) and W(n) are respectively the best case and worst case asymptotic running times for sorting an array of size n using Quick Sort. Consider the two statements:(i) B(n) is O(W(n))(ii) B(n) is Theta(W(n)). (Select ONE answer)

A. Both (i) and (ii) are true

B. (i) is true but (ii) is false

C. (i) is false but (ii) is true

D. Both (i) and (ii) are false

6 • Reply • Share ›

still learning abc • 5 months ago

it should be B

• Reply • Share ›

dk abc • 5 months ago

C.

• Reply • Share ›

Chandresh Singh abc • 5 months ago

c.

• Reply • Share ›

aswinii • 7 months ago

can anyone provide a link to learn enter concepts of COMPLEXITY?

3 • Reply • Share ›

dk aswinii • 5 months ago

http://www.geeksforgeeks.org/

2 • Reply • Share ›

guest • 8 months ago

4) Suppose T(n) = 2T(n/2) + n, T(0) = T(1) = 1
Which one of the following is false.
a) T(n) = O(n^2)
b) T(n) = (nLogn)
c) T(n) = (n^2)
d) T(n) = O(nLogn)

Answer (c)

i think c and D... what you say

3 • Reply • Share ›

Adauta Garcia Ariel guest • 4 months ago

I think c is false, b may be false, but D is true.

• Reply • Share ›

Anurag Panwar guest • 5 months ago

I think the answer is b and c because the answer is not in asymptotic notation. That means that they are the exact answer which is not true. T(n) = O(nlgn) is right because running time of above recursion is Theta(nlogn) which can be written as O(nlgn)

1 • Reply • Share ›

esjeh • 8 months ago

everyone here is concerned with the last question. i want to clarify my doubt that big omega always helps in finding the greatest lower bound of any function.so i think last question is invalid.
correct me if im wrong?

2 • Reply • Share ›

Praveen Kumar Tomar • 9 months ago

Should we be more interestedi in worst case analysis or average case analysis ?

• Reply • Share ›

Techie Me Praveen Kumar Tomar • 2 months ago

Worst Case analysis is the best thing to look for. At least it gives an upper bound on the running time.

However, there is a slight disagreement when we talk about real world problems. Most of the time the inputs on which we run an algorithm will not be the worst possible input. So, it will always complete before the worst running time.

If you are looking for an order in which we should be interested in, there is none specified. It all depends upon your need.

To understand more about this watch my video https://www.youtube.com/watch?...

• Reply • Share ›

Kartik Praveen Kumar Tomar • 9 months ago

Worst case analysis is done usually, because it is not feasible to do average case analysis most of the time. See http://www.geeksforgeeks.org/a... for details.

• Reply • Share ›

Praveen Kumar Tomar • 9 months ago

I have observed most of the times we use big O notation , why shouldn't we use big theta notation more often because it seems to represent the average case situations

• Reply • Share ›

Kartik Praveen Kumar Tomar • 9 months ago

Because Big O notation covers as complexities. We can always write worst case complexity in big O. For example, it is technically correct to say that the time complexity of insertion sort is O(n*n), but not correct to say that the time complexity of insertion sort is theta(n*n)

2 • Reply • Share ›

Kartik Kartik • 9 months ago

Please read the first line as "Because Big O notation covers complexities of all cases"

1 • Reply • Share ›

Vaibhav Jain • 10 months ago

Can anybody tell me that what is "3n3 + 6n2 + 6000" or f(n) in the above article and how it is calculated for any algorithm (if there is any proposal to do so)?

• Reply • Share ›

Vaibhav Bajpai Vaibhav Jain • 10 days ago

The time complexity of an algorothm is a measure of its efficiency in terms of time and space, where time comes as a result of number of key operations/instructions executed by the code.

Now what is a key operation ?

A key operation in a algorithm/code will be executed repeatedly in a single execution/run of that code;

f(n) or T(n) or 3(n^3) + 6(n^2) + 6000, all are same, and these can be called as cost functions, because they majorly contribute to the total time cost for the algorithm.

Every term in this function represents an instruction, like 3(n^3) represents an instruction which will run the number of times equivalent to the value of 3(n^3), where n is the size of the input value. Similarly, 6(n^2) again represents an instruction which will run the number of times equivalent to the value of 6(n^2), and 6000 represents one or more instructions(whose execution doesn't depend on the value of input) and these will be executed 6000 times in total in a single run of that code.

Bundling these terms together as 3(n^3) + 6(n^2) + 6000 tells the total no of key opearations that the code will perform. And hence we calculate the time complexity from this cost function.

• Reply • Share ›

ak • a year ago

Which of the following statements is/are valid?
1. Time Complexity of QuickSort is (n^2)
Ans : INVALID because it is Theta(nlogn)

2. Time Complexity of QuickSort is O(n^2)
Ans: Valid
3. For any two functions f(n) and g(n), we have f(n) = (g(n)) if and only if f(n) = O(g(n)) and f(n) =(g(n)).

Ans: Valid

4. Time complexity of all computer algorithms can be written as Omega(1).
Ans:VALID

It depends on how complex the algorithm is
O(1) represents a constant running time which is not possible.
Linear Search : O(n)
Binary Search O(nlogn)
QuickSort O(n2)
Matrix Multiplication O(n^3)
BFS/DFS V+Elog(V)

3 • Reply • Share ›

DS+Algo=Placement ak • 9 months ago

I think 4th one is invalid

• Reply • Share ›

kar DS+Algo=Placement • 9 months ago
yes 4th is invalid . omega(1) means in best case , any algorithm will perform in constant time , which is not true .
• Reply • Share ›

Aish ak • 9 months ago
How is the 4th one correct ?
• Reply • Share ›

Rahul Ramesh ak • a year ago
You have said that Time complexity of Quick sort is Theta(nlogn)??? Isn't it wrong?
• Reply • Share ›

Albut Rahul Ramesh • a year ago
It should be theta(nlogn) and theta(n^2) both same as in insertion sort.
• Reply • Share ›

Marsha Donna • a year ago
so does this mean that
big O notation corresponds to worst case time complexity,
big omega notation corresponds to best case time complexity,

big theta notation corresponds to average case time complexity???
12 • Reply • Share ›

ak Marsha Donna • a year ago
YES
• Reply • Share ›

darubramha Marsha Donna • a year ago
Somewhat correct.

big O notation corresponds to worst case time complexity, generally it means "less than or equal to", i.e if T(n) [time complexity] = O(n) then it means, T(n) can have maximum cn value, -> T(n) <= cn.

big omega notation corresponds to best case complexity, it is similar to ">=" as explained in the previous case.

But, big theta notation, tells us that the time complexity of the algorithm is asymptotically bounded by the given function, which means if T(n) = theta(n), the time complexity can never be worse than c1n or can never be better than c2n, where c1,c2 are constants. It does not mean the average case, it corresponds more towards "=", i.e c1n <= T(n) <= c2n

3 • Reply • Share ›

Ankur Teotia Marsha Donna • a year ago

i have the same doubt , someone please clarify.
it seems to be the case but no where i see this explicitly mentioned.

2 • Reply • Share ›

noob • 2 years ago

1. Time Complexity of QuickSort is (n^2) -----> NO
2. Time Complexity of QuickSort is O(n^2) -----> YES
3. For any two functions f(n) and g(n), we have f(n) = (g(n)) if and only if f(n) = O(g(n)) and f(n) = (g(n)). -----> YES
4. Time complexity of all computer algorithms can be written as (1) -----> YES

11 • Reply • Share ›

Suvarna noob • a year ago

Can You please tell us that How are you calculating time complexity ?
I mean is it based upon worst case or any of the rest one ?
Can you please brief it ?

• Reply • Share ›

gansai Suvarna • 5 months ago

In the exercise it is mentioned whether it is theta or BigO.
1. Theta(n^2) -- average case complexity.
2. O(n^2) -- worst case complexity

• Reply • Share ›

Chaithanya Kanumolu noob • a year ago

I am confused with 4th point. Since best case for insertion sort is Theta(n) right? In that case how can we write it as Theta (1)?

3 • Reply • Share ›

ibn noob • a year ago

Why was question 4 Yes?

• Reply • Share ›

Utkarsh Gupta ibn • a year ago

If f(n) = O(n) then it means f(n) can be less than or equal to order of n.
If f(n) = Θ(n) then it means f(n) is equal to the order of n.
If f(n) = Ω(n) then it means f(n) is greater than or equal to order of n.
So if you see Ω(1), it means that this is the smallest lower bound for any algorithm. Therefore for any algorithm f(n) = Ω(1). But it can be even more

precise that is f(n) = Ω(n) or Ω(n^2) etc for different different conditions.

3 • Reply • Share ›

mog ibn • a year ago

bcoz oemga always provide a lower bound on running time of algorithm.so minimum running time would be omega(1),it cant be less than omega(1).....so we can write time complexity of all algorithms as omega(1).

• Reply • Share ›

Sourabh mog • a year ago

For a sorting algorithm, best case can not be better than omega(n) as you have to check all n elements before saying that it is sorted, so I guess answer is NO!

2 • Reply • Share ›

rohit Sourabh • a year ago

1<n ==""> yes

• Reply • Share ›

Sourabh rohit • a year ago

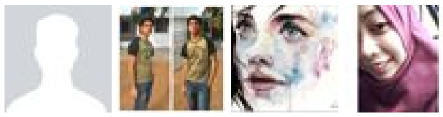
Well you got me there, I saw this answer again today and thought about it again and IMO omega(1) is true for any algorithm (its a lower bound after, not a tight one though)

• Reply • Share ›

- GeeksforGeeks

  Like

  97,207 people like GeeksforGeeks.

  Facebook social plugin
-
- Interview Experiences
  - Advanced Data Structures
  - Dynamic Programming
  - Greedy Algorithms
  - Backtracking
  - Pattern Searching
  - Divide & Conquer
  - Mathematical Algorithms
  - Recursion
  - Geometric Algorithms
-

- ## Popular Posts

  - All permutations of a given string
  - Memory Layout of C Programs
  - Understanding “extern” keyword in C
  - Median of two sorted arrays
  - Tree traversal without recursion and without stack!
  - Structure Member Alignment, Padding and Data Packing
  - Intersection point of two Linked Lists
  - Lowest Common Ancestor in a BST.
  - Check if a binary tree is BST or not
  - Sorted Linked List to Balanced BST

- Follow @GeeksforGeeks

- ## Recent Comments

  - lt_k

    i need help for coding this function in java...

    Java Programming Language · 1 hour ago

  - Piyush

    What is the purpose of else if (recStack[*i])...

    Detect Cycle in a Directed Graph · 1 hour ago

  - Andy Toh

    My compile-time solution, which agrees with the...

    Dynamic Programming | Set 16 (Floyd Warshall Algorithm) · 1 hour ago

  - lucy

    because we first fill zero in first col and...

    Dynamic Programming | Set 29 (Longest Common Substring) · 1 hour ago

  - lucy

    @GeeksforGeeks i don't n know what is this long...

    Dynamic Programming | Set 28 (Minimum insertions to form a palindrome) · 2 hours ago

  - manish

    Because TAN is not a subsequence of RANT. ANT...

    Given two strings, find if first string is a subsequence of second · 2 hours ago

-