ALGORITHMS INTERVIEW QUESTIONS & ANSWERS PART 1

FAQ

1) What is the difference between Lower bound and Tight bound?

Big O is the upper bound, while Omega is the lower bound. Theta requires both Big O and Omega, so that's why it's referred to as a tight bound (it must be both the upper and lower bound).

For example, an algorithm taking *Omega(n log n)* takes at least *n log n* time, but has no upper limit. An algorithm taking T*heta(n log n)* is far preferential since it takes at least *n log n (Omega n log n)* and no more than *n log n (Big O n log n).*

2) What does it mean if an operation is O(n!) ?

O(n!) an algorithm that "tries everything," since there are (proportional to) *n!* possible combinations of n elements that might solve a given problem. It means to do something for all possible permutations (possible ways in which a set or number of things can be ordered or arranged) of the *N* elements. Traveling salesman is an example of this, where there are *N!* ways to visit the nodes, and the brute force solution is to look at the total cost of every possible permutation to find the optimal one.

3) What are some algorithms which we use daily that has *O(1)* , *O(n log n)* and *O(log n)* complexities?

** O(1) time **

- Accessing Array Index (*int a = ARR[5];)*
- Inserting a node in Linked List
- Pushing and Poping on Stack
- Insertion and Removal from Queue
- Finding out the parent or left/right child of a node in a tree stored in Array
- Jumping to Next/Previous element in Doubly Linked List

FAQ

O(n) time

In a nutshell, all Brute Force Algorithms, or Noob ones which require linearity, are based on *O(n)* time complexity

- Traversing an array
- Traversing a linked list
- Linear Search
- Deletion of a specific element in a Linked List (Not sorted)
- Comparing two strings
- Checking for Palindrome
- Counting/Bucket Sort and here too you can find a million more such examples....

O(log n) time

- Binary Search
- Finding largest/smallest number in a binary search tree
- Certain Divide and Conquer Algorithms based on Linear functionality
- Calculating Fibonacci Numbers - Best Method The basic premise here is NOT using the complete data, and
- reducing the problem size with every iteration

O(n log n) time

The factor of 'log n' is introduced by bringing into consideration Divide and Conquer. Some of these algorithms are the best optimized ones and used frequently.

- Merge Sort
- Heap Sort
- Quick Sort
- Certain Divide and Conquer Algorithms based on optimizing *O(n2)* algorithms

FAQ

O(n2) time

These ones are supposed to be the less efficient algorithms if their *O(n log n)* counterparts are present. The general application may be Brute Force here.

-
- Bubble Sort
- Insertion Sort
- Selection Sort
- Traversing a simple 2D array