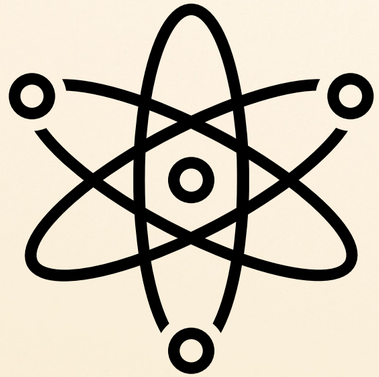
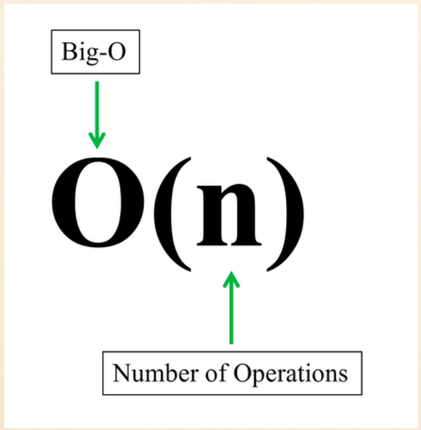
QUICK GUIDE TO TIME COMPLEXITY



1. Let's first understand Big Oh Notation, O

- **Big - O notation, O(N) calculates the worst-case time complexity i.e the maximum time an algorithm will take to execute.**
- **It is the MOST COMMONLY used notation for expressing time complexity.**
- ***Other notations are Omega Notation,(Ω) and Theta Notation,θ.***

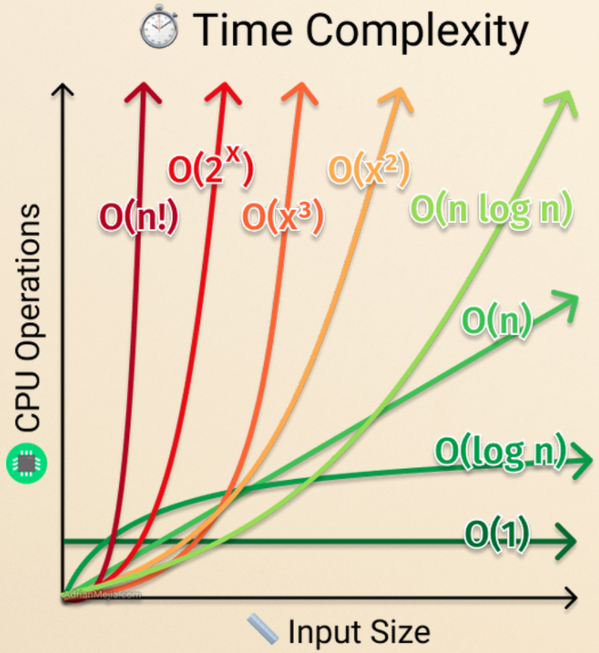
Let's focus on O(n) i.e the most common notation for this guide.

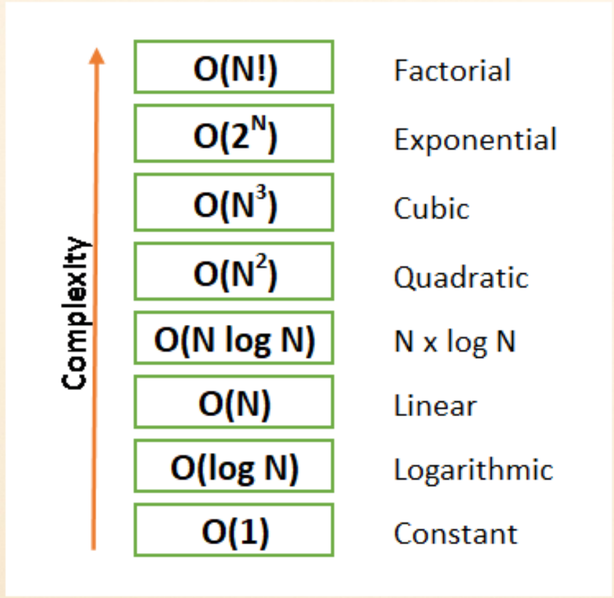


2. Let's begin with 5 common values of Big-O complexity

- O(1)
- O(n)
- O(n^2)
- $O(\log_2 n)$
- $O(n\log_2 n)$

Yes, just understand these 5 first and you will feel much more confident to learn more ~





3. Let's check out these 5 values with examples.

- **O(1)**
  - **When Algo's execution time is not based on the input size, it is said to have a time complexity with order O(1).**
  - **Example - Please note that the runtime doesn't change with input or its size.**

```java
public class Main {
    no usages
    public static void main(String[] args) {
        System.out.println("Hello world!");
        System.out.println("Hello world again");
        System.out.println("Bye for now");
    }
}
```

O/P

```
Hello world!
Hello world again
Bye for now
```

- **O(n)**

  - **When Algo's execution time increases linearly with the length of input, it is said to have a linear time complexity i.e order O(n).**

  - **In Simple words , if your program is traversing through all the values of the input using a for/while loop , then time complexity is O(n).**

  - **Example - Please note that the runtime depends directly on the input size.**

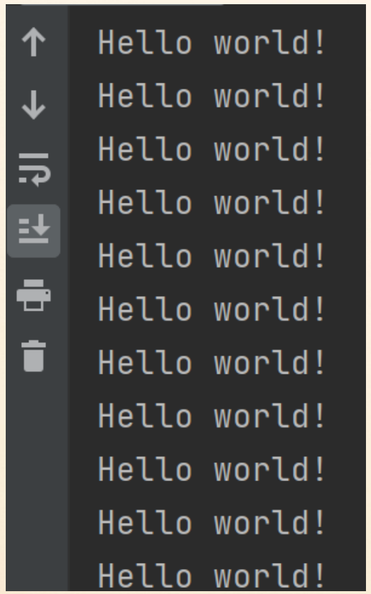
```java
public static void main(String[] args) {
    int n = 5;
    for (int i = 0; i <= n; i++) {
        System.out.println("Hello world!");
    }
}
```

O/P

- O(n^2)
  - When Algo's execution time increases non-linearly O(n^2) with the input size, it is said to have a time complexity with order O(n^2).
  - Simply put, 2 nested for/while loops fill into this category i.e *O(n) * O(n) = O(n^2)*
  - Example - Please note that the runtime depends non-linearly on input size.

```java
public static void main(String[] args) {
    int n = 5;
    int m = 2;
    for (int i = 0; i <= n; i++) {
        for(int j = 0 ; j <= m ; j++)
        System.out.println("Hello world!");
    }
}
```

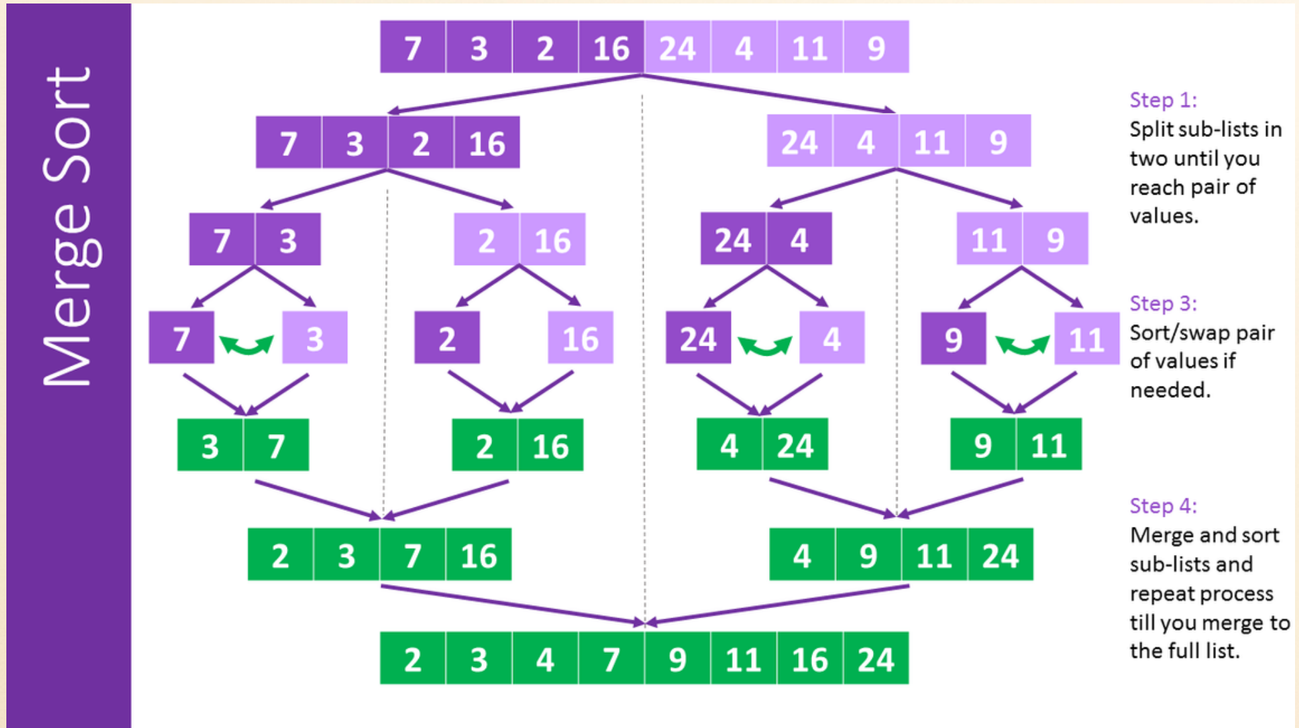
O/P



- O(log2 n)
  - When Algo's execution time increase but less proportionally with the input size, it is said to have a time complexity with order O(log2 n)
  - Divide & Conquer algorithms such as binary search have time complexity O(log2 n)
  - Binary trees & binary search functions are examples of such algorithms

```
public int binaySearch(int arr[], int l, int r, int x) {
    if (r >= l) {
        int mid = l + (r - l) / 2;
        if (arr[mid] == x)
            return mid;
        if (arr[mid] > x)
            return binarySearch(arr, l, toIndex: mid - 1, x);
        return binarySearch(arr, fromIndex: mid + 1, r, x);
    }
    return -1;
}
```

- **O(nlogn)**
  - **When Algo's execution time is linearithmic, it is said to have a time complexity with order O(nlogn).**
  - **It performs slightly slower than O(n) but better than O(n^2).**
  - **Example - Merge sort algorithm.**



4. Time-Complexity Of Sorting & Searching Algorithms

Algorithms	Best Case	Average Case	Worst Case
Linear Search	O(1)	O(n)	O(n)
Binary Search	O(1)	O(log n)	O(log n)
Bubble Sort	O(n)	O(n^2)	O(n^2)
Selection Sort	O(n^2)	O(n^2)	O(n^2)
Merge Sort	O(nlogn)	O(nlogn)	O(nlogn)

@sahilpuri01

@sahilpuri1212

@DecodingWithSahil