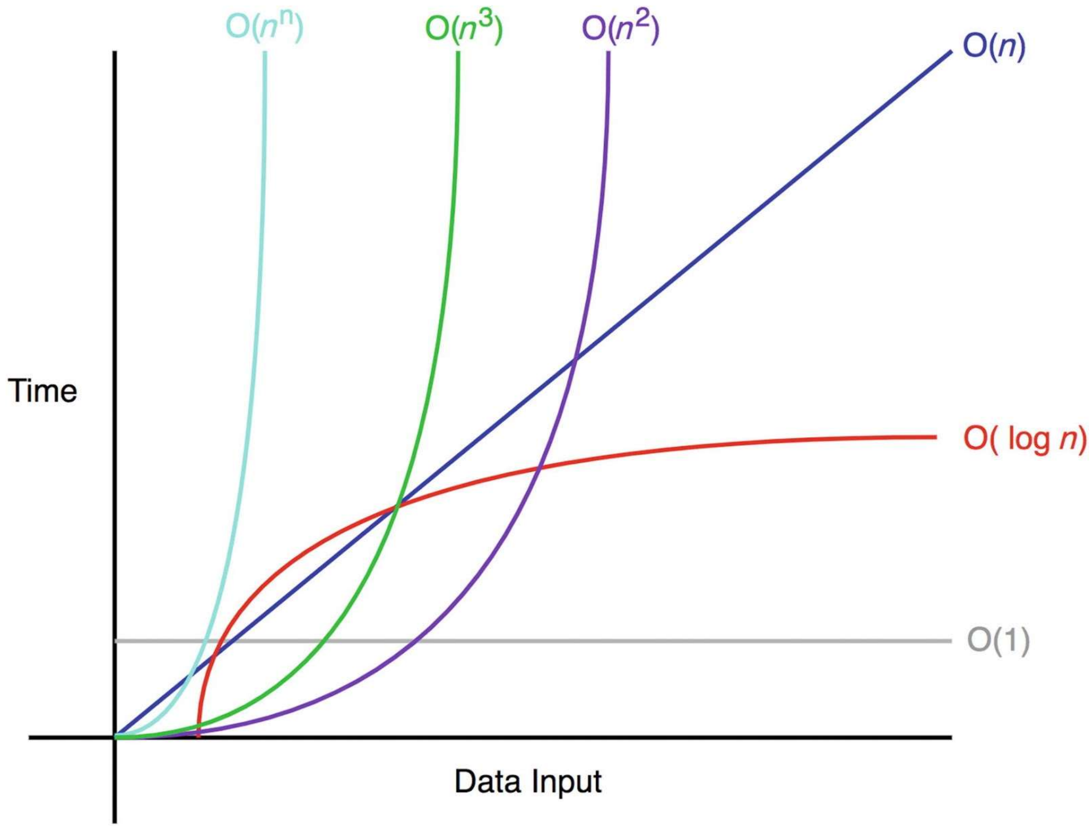
Graphical representation of various runtime complexities:



Complexities are also known as :-

c

complexity	O(1)	O(log n)	O(n)	O(n^a)	O(a^n)
called as	constant	logarithmic	lenier	polinomial	exponential

Few log formula that will become handy :

- $\log_a 1 = 0$
- $\log_a a = 1$
- $\log_a(x*y) = \log_a x + \log_a y$
- $\log_a(x/y) = \log_a x - \log_a y$
- $\log_a x^p = p \log_a x$

Log in Mathematics is generally to base 10. But, in Computer Science, log is to base 2, by default.

We can remember the log in the below simple manner. Since the base is 2, we get the log value of a particular number by calculating the number of steps required bring the number to 1, by dividing it by 2.

Let's take an example:

Say input is 8. We can divide 8 with 2, for 3 times to make its value to be 1.

1. 8/2=4
2. 4/2=2
3. 2/2=1

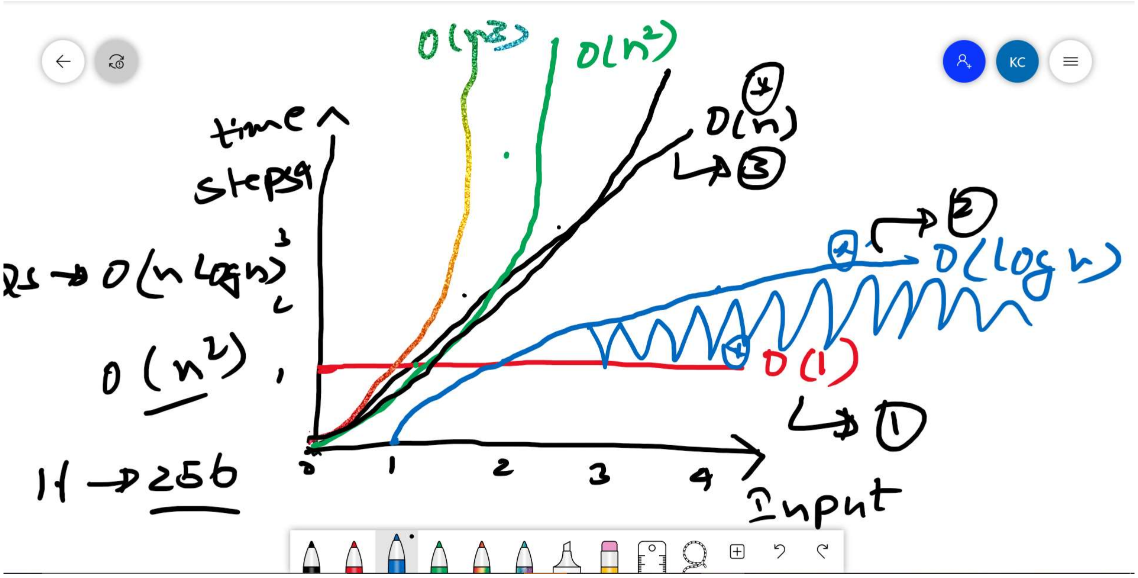
Now, say input is 16, we need 4 steps.

1. 16/2=8
2. 8/2=4
3. 4/2=2
4. 2/2=1

Log values for first 25 numbers:

$\log_2(x)$	Notation	Value
$\log_2(1)$	lb(1)	0
$\log_2(2)$	lb(2)	1
$\log_2(3)$	lb(3)	1.584963
$\log_2(4)$	lb(4)	2
$\log_2(5)$	lb(5)	2.321928
$\log_2(6)$	lb(6)	2.584963
$\log_2(7)$	lb(7)	2.807355
$\log_2(8)$	lb(8)	3
$\log_2(9)$	lb(9)	3.169925
$\log_2(10)$	lb(10)	3.321928
$\log_2(11)$	lb(11)	3.459432
$\log_2(12)$	lb(12)	3.584963
$\log_2(13)$	lb(13)	3.70044
$\log_2(14)$	lb(14)	3.807355
$\log_2(15)$	lb(15)	3.906891
$\log_2(16)$	lb(16)	4
$\log_2(17)$	lb(17)	4.087463
$\log_2(18)$	lb(18)	4.169925
$\log_2(19)$	lb(19)	4.247928
$\log_2(20)$	lb(20)	4.321928
$\log_2(21)$	lb(21)	4.392317
$\log_2(22)$	lb(22)	4.459432
$\log_2(23)$	lb(23)	4.523562
$\log_2(24)$	lb(24)	4.584963
$\log_2(25)$	lb(25)	4.643856

Class Explanation:-



Linear Search vs Binary Search:

Linear search: Searching for an element, one element at a time without skipping any item.

Binary Search: Cut down your search to half as soon as you find middle of a sorted list.

```
public class LinearSearch {
     public static void main(String[] args) {
          int[] arr = { 10, 20, 30, 40, 50 };
          boolean result = linearSearch(arr, 166);
          System.out.println(result);
     }
     static boolean linearSearch(int[] arr, int x) {
          for (int i = 0; i < arr.length; i++) {
               if (arr[i] == x) {
                    return true;
               }
          }
          return false;
     }
}
```

```java
public class BinarySearch {
	public static void main(String[] args) {
		int[] arr = { 10, 20, 30, 40, 50 };
		boolean result = binarySearch(arr, 50, 0, arr.length - 1);
		System.out.println(result);
	}
	private static boolean binarySearch(int[] arr, int x, int low, int high) {
		while (low <= high) {
			int mid = (low + high) / 2;
			if (arr[mid] == x) {
				return true;
			} else if (arr[mid] < x) {
				low = mid + 1;
			} else if (arr[mid] > x) {
				high = mid - 1;
			}
		}
		return false;
	}
}
```