Big O Basic Concepts:

- **O(1)**: Constant Time
  - Doesn't depend on the size of the data set.
  - Example: Accessing an array element by its index.
- **O(log n)**: Logarithmic Time
  - Splits the data in each step (divide and conquer).
  - Example: Binary search.
- **O(n)**: Linear Time
  - Directly proportional to the data set size.
  - Example: Looping through an array.
- **O(n log n)**: Linearithmic Time
  - Splits and sorts or searches data.
  - Example: Merge sort, quick sort.
- **O(n^2)**: Polynomial Time
  - Nested loops for each power of n.
  - Example: Bubble sort (O(n^2)).

Omega (Ω) – Best Case

- **What it means:** Omega (Ω) describes the best-case scenario for an algorithm.
- **In simple terms:** It tells you the fastest an algorithm can run in the best circumstances.

Theta (Θ) - Average Case

- **In simple terms:** It tells you what to generally expect in terms of time complexity.

Big O (O) - Worst Case

- **What it means:** Big O (O) describes the worst-case scenario for an algorithm.
- **In simple terms:** It tells you the slowest an algorithm can run in the worst circumstances.

Useful Tips

- **Drop Non-Dominant Terms**
  - In **O(n^2 + n)**, focus on **O(n^2)** as it will dominate for large **n**.
- **Drop Constants**
  - **O(2n)** simplifies to **O(n)**.

1. Big O:

- O(n)

```
for(int i=0;i<10;i++)
    System.out.println(i);
```

The above for loop is running n times therefore the time complexity is O(n).

Rule of Simplification:

```
for(int i=0;i<10;i++)
    System.out.println(i);
for(int j=0;j<10;j++)
    System.out.println(j);
```

Here 2 loops are running
$\therefore O(n+n) \Rightarrow O(2n)$
We will always drop the constant.
$\therefore O(n)$

- $O(n^2)$

```
for(int i=0;i<10;i++)
    for(int j=0;j<10;j++)
        System.out.println(j);
```

Here two loops are running one inside another
$\therefore O(n*n) \Rightarrow O(n^2)$

```java
for(int i=0;i<10;i++)
    System.out.println(i);

for(int i=0;i<10;i++)
    for(int j=0;j<10;j++)
        System.out.println(j);
```

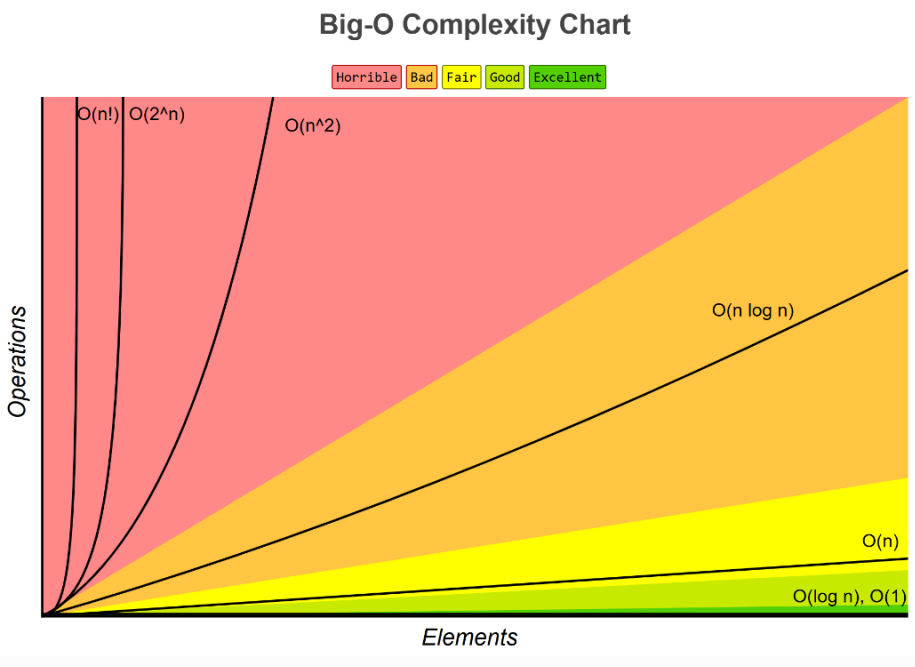
Here the loop runs $O(n + n^2)$ times, we will drop constant n

$\therefore O(n^2)$

- $O(1)$ (Also known as constant complexity)

```java
public class Main {
    public static void main(String[] args) {
        System.out.println(add( a: 3, b: 4));
    }
    static int add(int a, int b){  1 usage
        return a+b;
    }
}
```

The function is being called once therefore $O(1)$



Note: All divide and conquer approach algorithm have $O(\log n)$ complexity. Eg: Binary search.