Data Structure and Algorithm

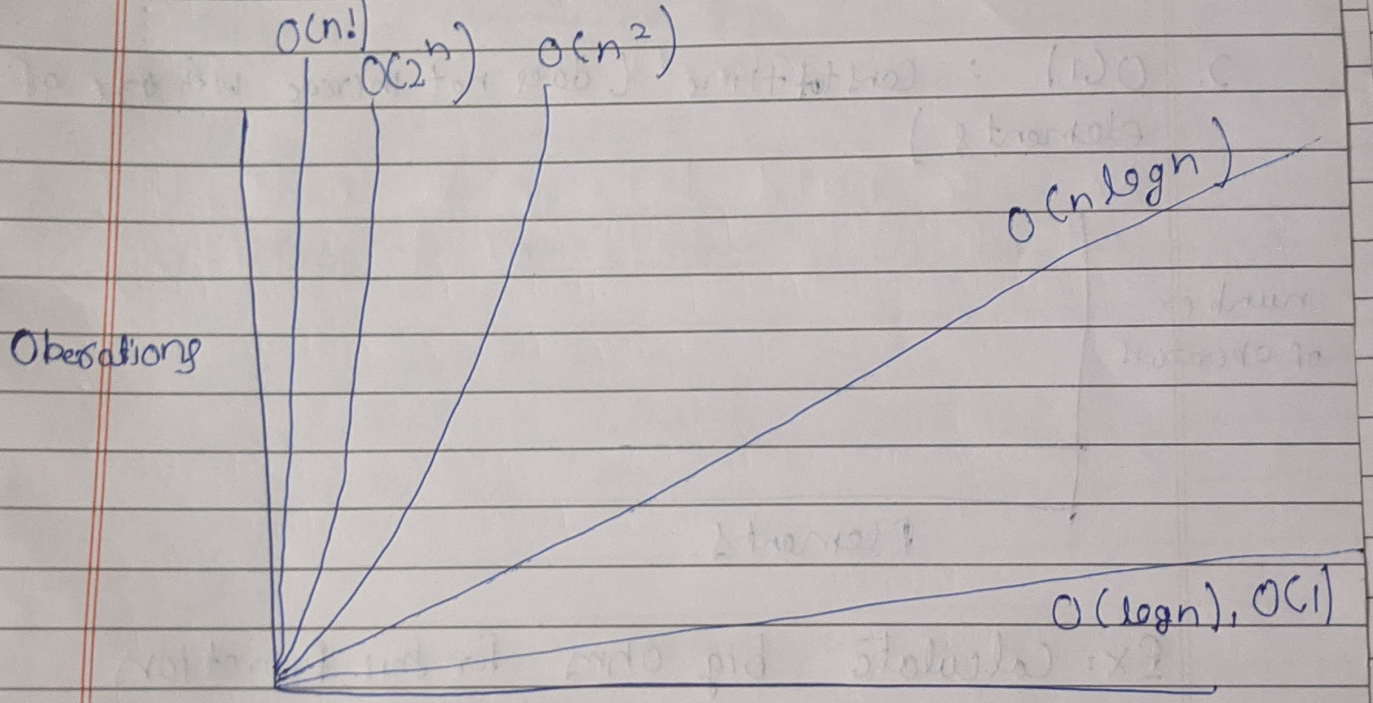
Date ___/___/___
Page ___

- Time Complexity :

→ Asymtotic analysis .

- Good Code :

1. Readable : (easily understand by other developers)
2. Scalable :



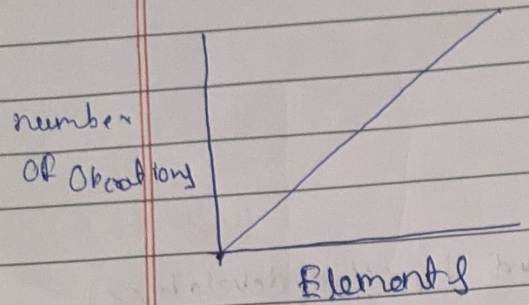
Big Ohm chart

→ This chart basically shows when elements are increasing bigger and bigger input, how much algo slow down .

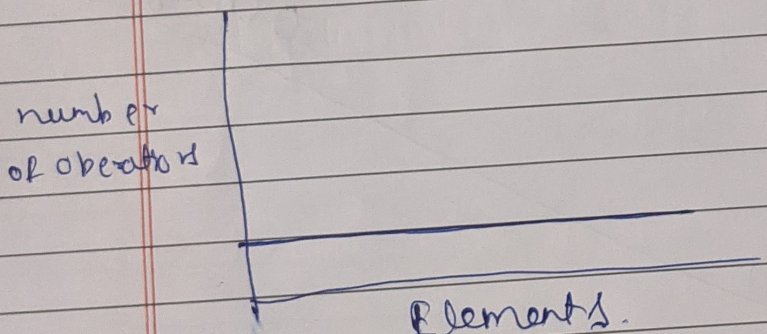
→ The less it slow down or slower it ~~down~~ slow down the better it is.

Time

1. O(n) : Linear (Increases with size of elements)



2. O(1) : Constant time (does not increase with size of elements)



Ex: Calculate big Ohm for this function

```
int IAmFunction (int ele) {
    int a = 2;          //O(1)
    a = 3 + 5;          //O(1)
    for (int i=0; i<ele; i++) {//O(n)
    Cout<< "I am Vikrant";//O(n)
    Cout << 5;//O(n)
    }
    return 5;//O(1)
}
```

Time Complexity = O(3 + 3n)

- How to Simplify the Time Complexity into one terms?

Rules:

1. Worst Case
2. Remove Constants
3. Different terms for inputs
4. Drop Non dominants

Example of third rule

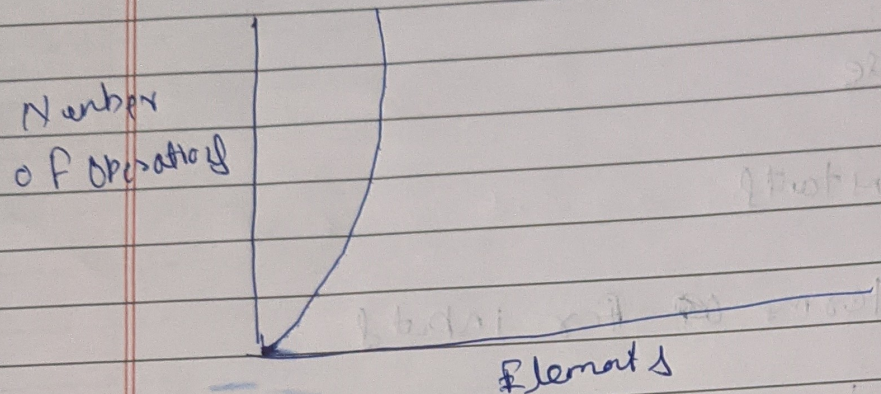
```
void CombineBoxesTwice (boxes, boxes2) {
boxes.forEach (function (boxes) {
  console.log (boxes);
});

boxes2.forEach (function (boxes) {
  console.log (boxes);
});
}
```

Time Complexity: O(n+m)

Why? because arrays size can be different.

3. $O(n^2)$: Quadratic Time



We take nested loops in * (multiplication) and individually looks as + (addition)