Time Complexity

Apa itu Time Complexity

- *suatu cara sederhana untuk mengetahui berapa lama waktu yang dibutuhkan untuk menjalankan suatu algoritma dengan input tertentu (n).*

Big-O Notation → *O(n)*

O(n)

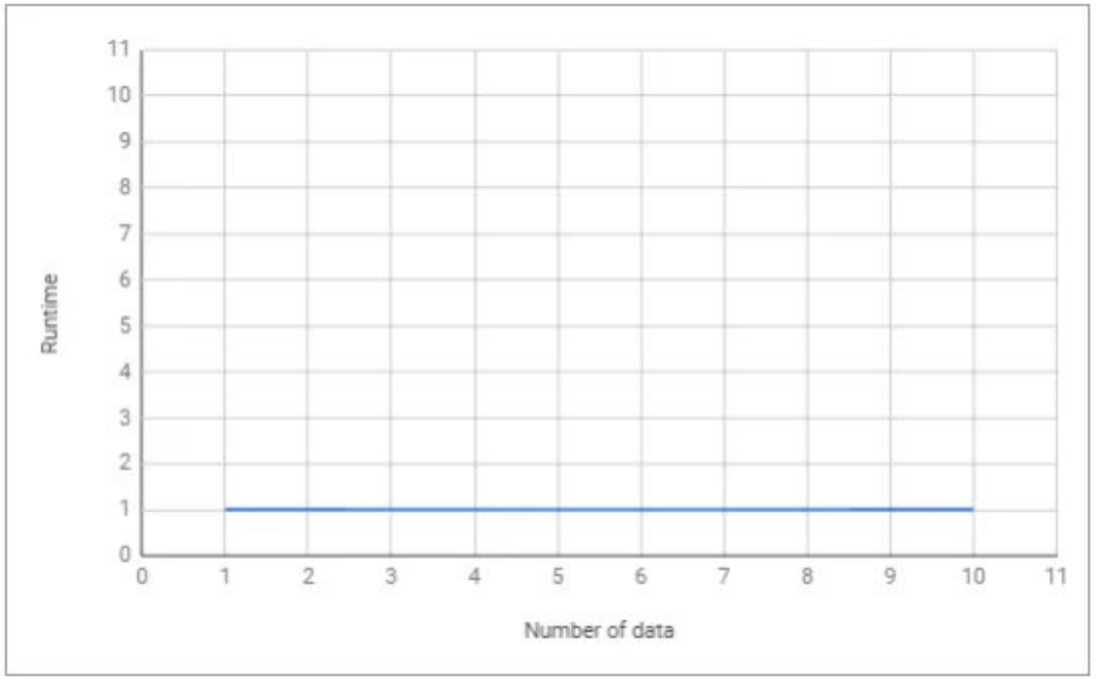
```
Regular       Big-O
2             O(1)    --> It's just a constant number
2n + 10       O(n)    --> n has the largest effect
5n^2          O(n^2) --> n^2 has the largest effect
```

**Fastest growing*

O(1)

O(1) — Constant Time: Given an input of size n, it only takes a single step for the algorithm to accomplish the task.

```
let myArray = [1, 5, 0, 6, 1, 9, 9, 2];
function getFirst(input){

   return input[0]; // selalu melakukan 1 langkah

}

let firstEl = getFirst(myArray);
```



O(n)

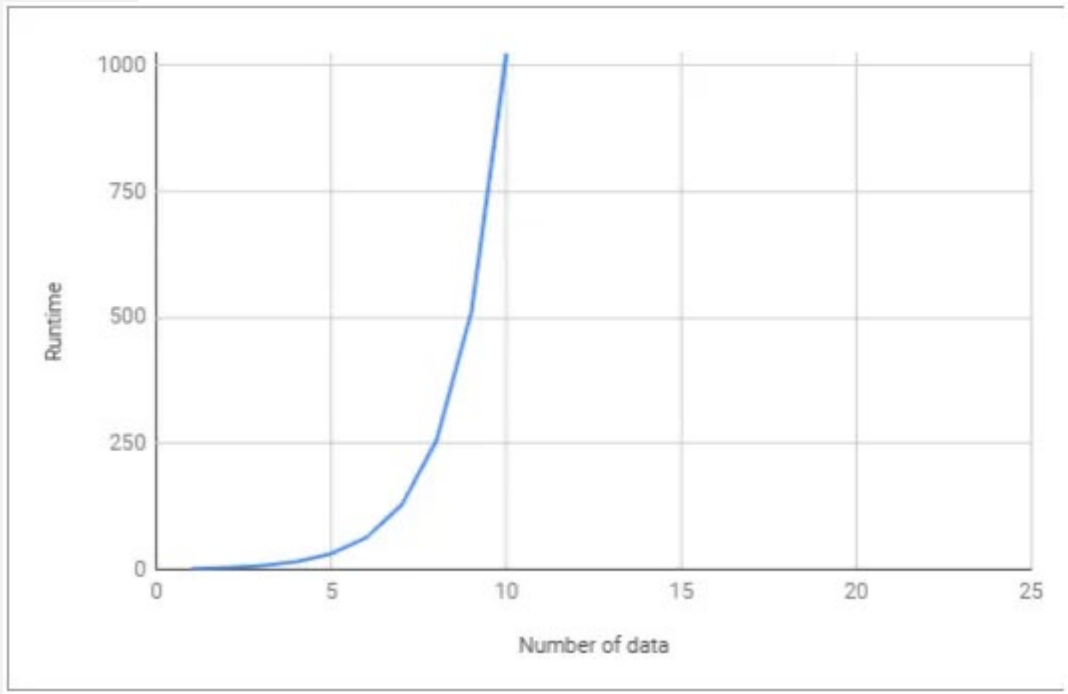
O(n) — Linear Time: Given an input of size n, the number of of steps required is directly related (1 to 1)

```
let myArray = [1, 5, 0, 6, 1, 9, 9, 2];
function getMax(input){
    var max = 0;

    for (var i=0; i<input.length; i++){

        if (max < input[i])
            max = input[i];

    }
    return max;

}

let maxNumber = getMax(myArray);
```

$O(n^2)$

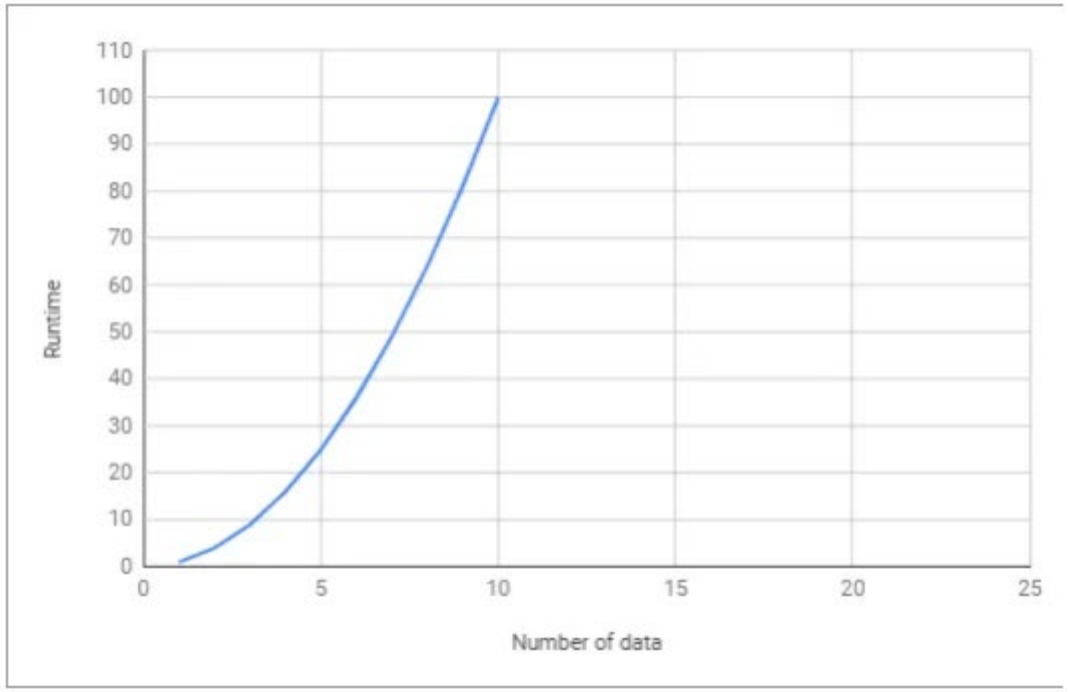
$O(n^2)$ — Quadratic Time: Given an input of size n, the number of steps it takes to accomplish a task is square of n.

```
let myArray = [1, 5, 0, 6, 1, 9, 9, 2];
function sort(input){
    var sortedArray = [];

    for (var i=0; i<input.length; i++){ // O(n)

        let min = input[i];

        for (var j=i+1; i<input.length; i++){ // O(n)

            if (input[i] < input[j])
                min = input[j];

        }

        sortedArray.push(min);

    }

    return sortedArray;

}

let sortedArray = sort(myArray);
```



$O(2^n)$

O(2^n) — Exponential Time: Given an input of size n, the number of steps it takes to accomplish a task is a constant to the n power (pretty large number).

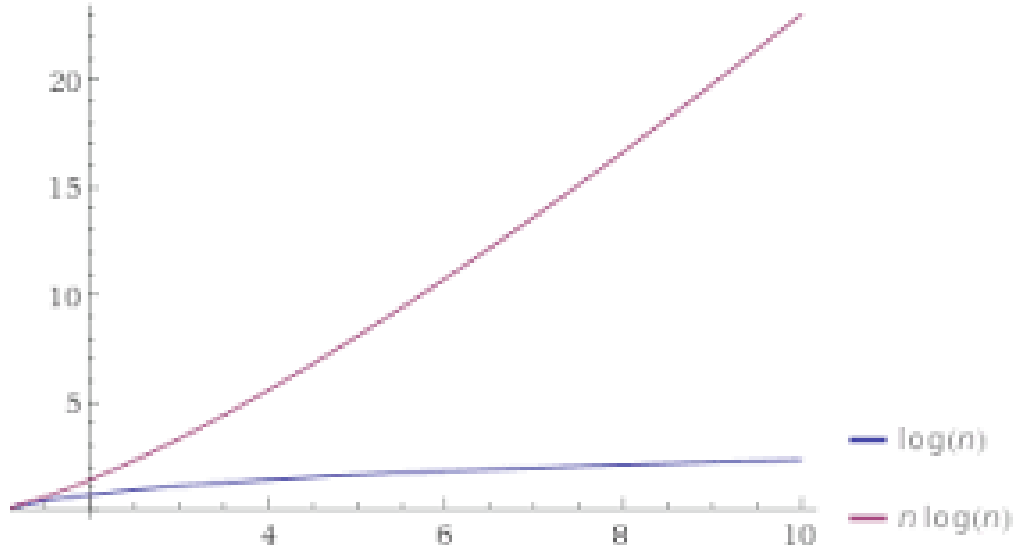


O(log n)

O(log n) — Logarithmic time: given an input of size n, the number of steps it takes to accomplish the task are decreased by some factor with each step.

```
let sortedArray = [11, 24, 30, 43, 51, 61, 73, 86];
function isExists(number, array){
    var midPoint = Math.floor( array.length /2 );
    if( array[midPoint] === num) return true;
    let isFirstHalf = false;
    if( array[midPoint] < num ) isFirstHalf = true;
    else if( array[midpoint] > num ) isFirstHalf = false;
    if (array.length == 1) return false;
    else {
        // memanggil fungsi yang sama dengan mengeleminiasi setengah
dari input array
        if (isFirstHalf)
            return isExists(number, getFirstHalf(array));
        else
            return isExists(number, getSecondHalf(array));
    }
}
isExists (24, sortedArray); // return true
isExists (27, sortedArray); // return false
```

***Contoh : Binary Search** adalah algoritma yang kita gunakan dalam mencari posisi nilai dari suatu array dengan cara 'mengeliminasi' setengah dari array input untuk mempercepat proses pencarian.*



Lets count

Big - O Notation	Computations for 10 Elements	Computations For 100 Elements	Computations For 1000 Elements
O(1)	1	1	1
O(N)	10	100	1000
O(N^2)	100	10000	1000000
O(log N)	3	6	9
O(N log N)	30	600	9000
O(2^N)	1024	1.26e+29	1.07e+301
O(N!)	3628800	9.33e+157	4.02e+2567