**NAMA : RIZKY MUHAMMAD AKMAL**

**NPM : 22552011211**

**KELAS : TIF-RM-22A**

**MATKUL : STRUKTUR DATA**

Big o notation beserta contohnya

**Big-O Notation** adalah cara untuk mengkonversi keseluruhan langkah-langkah suatu algoritma kedalam bentuk **Aljabar**, yaitu dengan menghiraukan konstanta yang lebih kecil dan koefisien yang tidak berdampak besar terhadap keseluruhan kompleksitas permasalahan yang diselesaikan oleh algoritma tersebut.

**O(1) — Constant Time**

**Constant Time** artinya banyaknya input yang diberikan kepada sebuah algoritma, tidak akan mempengaruhi waktu proses (runtime) dari algoritma tersebut.

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

// selalu melakukan 1 langkah

}

let firstEl = getFirst(myArray);

Contoh diatas, terdapat sebuah fungsi untuk mengambil elemen pertama dari sebuah input array. Kita bisa melihat bahwa berapapun jumlah array yang diberikan kepada fungsi tersebut, dia akan selalu melakukan 1 hal, yaitu mengambil elemen pertama. Itu artinya **jumlah input yang diberikan tidak mempengaruhi waktu proses (runtime) dari algoritma tersebut**.

## O(log n) — Logarithmic Time

## ****Logarithmic Time**** artinya ketika kita memberikan input sebesar n terhadap sebuah fungsi, jumlah tahapan yang dilakukan oleh fungsi tersebut berkurang berdasarkan suatu faktor. Salah satu contohnya adalah algoritma ****Binary Search****.

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

## O(n) — Linear Time

## ****Linear Time****adalah ketika runtime dari fungsi kita berbanding lurus dengan jumlah input yang diberikan.

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);

## Kita bisa melihat bahwa **semakin banyak jumlah input yang diberikan, maka waktu proses/**runtime **dari fungsi tersebut akan semakin besar**.

## O(n²) — Quadratic Time

## ****Quadratic Time**** adalah ketika runtime dari fungsi kita adalah sebesar n^2, dimana n adalah jumlah input dari fungsi tersebut. Hal tersebut bisa terjadi karena kita menjalankan ****fungsi linear didalam fungsi linear**** (n\*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) — Exponential Time

**Exponential Time** biasanya digunakan dalam situasi dimana kita tidak terlalu tahu terhadap permasalahan yang dihadapi, sehingga mengharuskan kita mencoba setiap **kombinasi**dan **permutasi**dari semua kemungkinan.