Dosen :

Muhammad Reksa Ariansyah, M.Kom.



NPM : 22552011116

NAMA : DIDHAN ADITYA HUDA

KELAS : TIF 22A RM

MATA KULIAH : STRUKTUR DATA

**PROGRAM STUDI TEKNIK INFORMATIKA SEKOLAH TINGGI TEKNOLOGI BANDUNG 2022**

**Pengertian Big-O Notation**

**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.

Mari kita liat contoh dibawah ini:

Regular Big-O2 O(1) --> It's just a constant number2n + 10 O(n) --> n has the largest effect5n^2 O(n^2) --> n^2 has the largest effect

Sederhananya, semua contoh yang ada diatas mengatakan bahwa “***kita hanya akan melihat faktor yang memiliki dampak paling besar terhadap nilai yang dihasilkan oleh algoritma tersebut***”.

Terdapat beberapa macam **time complexity**, diantaranya:

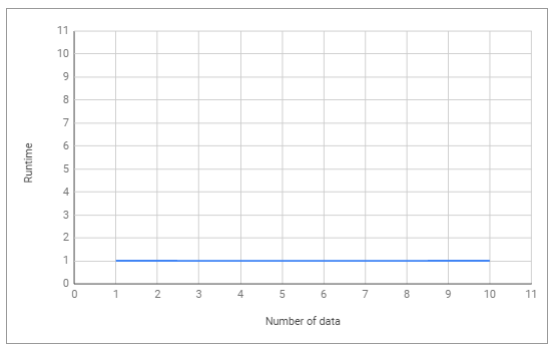
**O(1) — Constant Time**

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

**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**.



Constant Time

**O(log n) — Logarithmic Time**

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.

**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**.

**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.

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 trueisExists (27, sortedArray); // return false

*Note: Fungsi rekursif biasanya Logarithmic*

**O(n) — Linear Time**

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

**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**.



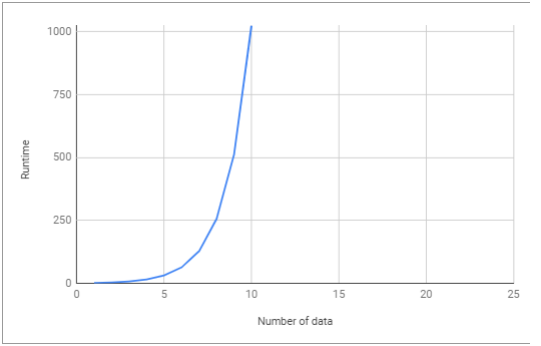
Linear Time

**O(n²) — Quadratic Time**

O(n²) — Quadratic Time: Given an input of size n, the number of steps it takes to accomplish a task is square of n.

**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);

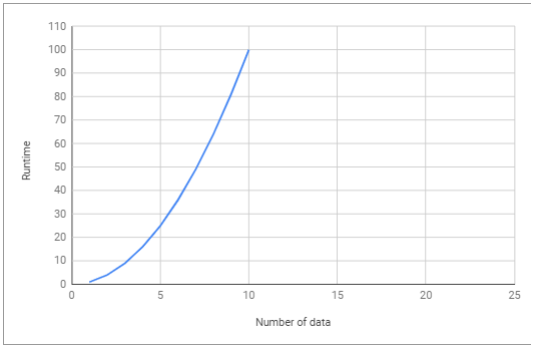


Quadratic Time

**O(2^n) — Exponential Time**

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).

**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.



***Referensi*** : https://medium.com/bee-solution-partners/penjelasan-sederhana-tentang-time-complexity-dan-big-o-notation-4337ba275cfe