Setelah mempelajari topik **Analisis Algoritma** di kuliah SDA, ada beberapa kompetensi yang perlu Anda kuasai:

- Menentukan kompleksitas waktu (Big-Oh) dari beberapa algoritma (logaritmik, linier, kuadratik, kubik, dsb).
- Memikirkan versi yang lebih efisien dari solusi (algoritma) yang sudah Anda implementasikan karena terkadang sebuah permasalahan dapat diselesaikan dengan beberapa algoritma yang mempunyai kompleksitas waktu yang berbeda-beda.
- Menghitung prediksi running time sebuah algoritma

## Problem 1. Konsep Big-Oh

- Misal, **f(N)** adalah sebuah fungsi yang menyatakan banyaknya *basic operation* sebuah algoritma jika diberikan **ukuran input N**. Sebuah algoritma ternyata mempunyai fungsi **f(N) = $4N^3$ + 1000N**. Berapa kompleksitas (asimtotik) algoritma tersebut dalam notasi Big-Oh ?
- Sebuah algoritma mempunyai kompleksitas waktu **O($N^3$)**. Dengan kata lain, algoritma tersebut mempunyai **orde kubik (fungsi running time algoritma tumbuh dalam orde kubik)**. Apa maksudnya ?
- Mengapa notasi Big-Oh **hanya** memperhatikan suku paling dominan ?
- Mengapa notasi Big-Oh tidak memperhatikan koefisien pada suku paling dominan ?
- Jadi, Apa yang direpresentasikan oleh notasi Big-Oh ?

## Problem 2. Menghitung Running Time

Selection Sort vs Quick Sort

Di kuliah DDP, Anda sudah mempelajari ide dari algoritma Selection Sort dan Quick Sort. Anda juga sudah mengetahui bahwa algoritma Quick Sort terbukti lebih efisien dibandingkan Selection Sort. Anda juga sudah mengamati bahwa algoritma Selection Sort mempunyai kompleksitas **O($N^2$)**, dan nanti Anda akan

mempelajari bahwa kompleksitas dari Quick Sort adalah **O(N log N)**. Jelas bahwa orde **O(N log N) < O($N^2$)**.

Untuk menghitung running time di Java, Anda bisa menggunakan **System.nanoTime()**. Di soal ini, Anda akan diajak untuk melakukan eksperimen kecil untuk membandingkan running time antara kedua algoritma tersebut jika diberikan beberapa ukuran input array (panjang array). Array akan di-generate secara random.

Gunakan kode **Problem2.java** untuk melengkapi tabel berikut:

| **Panjang Array** | **Selection Sort (miliseconds)** | **Quick Sort (miliseconds)** |
|---|---|---|
| 10 | | |
| 100 | | |
| 10000 | | |
| 100000 | | |

Dari hasil eksperimen, Apa yang dapat Anda simpulkan ? dan Apa yang Anda pelajari ?

## Problem 3. Prediksi Running Time

Problem 2 sebelumnya menyuruh Anda untuk menghitung *running time* yang sesungguhnya menggunakan mesin/komputer. Misal, Anda sudah mengetahui running time yang sesungguhnya untuk ukuran input **N = 100**. Anda juga bisa memprediksi running time untuk, misal **N = 1000**, dengan menggunakan kalkulasi tangan jika diketahui kompleksitasnya dalam notasi Big-Oh.

Untuk soal-soal terkait prediksi *running time*, Silakan Anda kerjakan soal nomor 1 dan 2 pada **Latihan 2 analisis algoritma**.

## Problem 4. Simple Problem

Diberikan sebuah matriks berukuran **N x N**, sebuah method mampu menghitung jumlah **semua elemen pada bagian diagonal**. Berikut adalah implementasi dari method tersebut ?

```
public static int sumDiagonal(int[][] m) {
      int sum = 0;
      for (int i = 0; i < m.length; i++) {
            for (int j = 0; j < m[i].length; j++) {
                  if (i == j) {
                        sum += m[i][j];
                  }
            }
      }
      return sum;
}
```

- Berapakah kompleksitas dari algoritma di atas dalam notasi Big-Oh ?
- Apakah algoritma tersebut sudah efisien ? jika belum efisien, buatlah versi lain yang lebih efisien dan tentukan pula kompleksitasnya !

## Problem 5. Banyaknya Bits

Angka 3 direpresentasikan dengan 2 digit: 11. Angka 8 direpresentasikan dengan 4 digit biner: 1000, dan 13 juga direpresentasikan dengan 4 digit biner: 1101. Angka 23 direpresentasikan dengan 5 digit biner: 10111. Berikut adalah method yang digunakan untuk menghitung berapa banyak digit biner yang dibutuhkan untuk merepresentasikan sebuah **bilangan bulat positif**.

```
public static int numBiner(int n) {
       ...
}
```

Implementasikan method numBiner tersebut, dan tentukan berapa kompleksitas dari method tersebut dalam notasi Big-Oh !

## Problem 6. Perkalian Matriks

Di persoalan sebelumnya, Anda belajar untuk mengimplementasikan **ADT Matrix** yang merepresentasikan sebuah matriks dengan operasi-operasinya. Anda juga sudah mengimplementasikan method untuk melakukan perkalian matriks. Sekarang, jika matriks **pertama berukuran N x M** dan **matriks kedua berukuran M x K**, berapakah kompleksitas dari algoritma perkalian matriks tersebut ?

Jika Anda belum menyelesaikan persoalan ADT Matrix pada latihan sebelumnya, Anda bisa mulai dengan mengimplementasikan method perkalian matriks berikut:

```
//m1 : N x M, m2 : M x K
public static int[][] mult(int[][] m1, int[][] m2) {

      //kembalikan matriks baru hasil perkalian m1 dan m2

}
```

Berapakah kompleksitas algoritma tersebut, seandainya matriks pertama dan matriks kedua mempunyai dimensi yang sama, yaitu **N x N** ?

## Problem 7. More on Big-Oh

Di kelas, Anda pernah diberikan persoalan untuk mencari pasangan bilangan bulat **a** dan **b**, dimana **a < b < N** dan **$(a^2 + b^2 + 1)/(ab)$** adalah bilangan bulat. Dimana **N** adalah input yang merupakan sebuah bilangan bulat (ketika di kelas, **N = 1000**). Salah satu solusinya adalah:

```
for (int a = 1; a < N; a++) {
    for (int b = a + 1; b < N; b++) {
        if ((a * a + b * b + 1) % (a * b) == 0) {
             System.out.println (a + " " + b);
        }
    }
}
```

Berapakah kompleksitas dari solusi tersebut dalam notasi Big-Oh ?

**Problem 8. Problem Solving**

Buatlah program yang dapat menjawab permasalahan berikut !

$\mathbf{A^5 + B^5 + C^5 + D^5 + E^5 = F^5}$

Dimana, $\mathbf{0 < A \leq B \leq C \leq D \leq E \leq F \leq 75}$

Hanya ada satu solusi ! yaitu ketika **A**, **B**, **C**, **D**, **E**, dan **F** bernilai berapa ? ☺

Berapakah kompleksitas algoritma dari solusi program yang Anda usulkan ?

Apakah Anda yakin solusi Anda itu adalah versi yang paling efisien ? ☺

**Problem 9. Orde yang Lebih Besar ?**

Urutkan dari yang orde paling kecil ke orde paling besar !

$O(N)$, $O(N!)$, $O(N \log N)$, $O(N \log \log N)$, $O(N \log^2 N)$, $O(N \log (N^2))$, $O(2^N)$