**LAPORAN PRAKTIKUM**

**WORKSHEET 3**

****

**Oleh:**

Muhamad Ilham Habib (140810180018)

Kelas B

**FAKULTAS MATEMATIKA DAN ILMU PENGETAHUAN ALAM**

**UNIVERSITAS PADJADJARAN**

**JATINANGOR**

**2019**

1. **Tujuan**
2. Mahasiswa paham mengenai perhitungan worst case.
3. Mahasiswa paham cara menghitung Big-O Notation.
4. Mahasiswa paham menghitung Big-Ω dan Big-Θ.
5. Mahasiswa paham menghitung aturan kompleksitas waktu asimtotik.
6. **Landasan Teori**

II.1 Dalam analisis algoritma kita selalu mengutamakan perhitungan worst case dengan alasan sebagai berikut:

* 1. Worst-case running time merupakan *upper bound* (batas atas) dari running time untuk input apapun. Hal ini memberikan jaminan bahwa algoritma yang kita jalankan tidak akan lebih lama lagi dari *worst-case.*
  2. Untuk beberapa algoritma, *worst-case* cukup sering terjadi. Dalam beberapa aplikasi pencarian, pencarian info yang tidak ada mungkin sering dilakukan.
  3. Pada kasus average-case umumnya lebih sering seperti worst-case. Contoh: misalkan kita secara random memilih angka dan mengimplementasikan insertion sort, *average-case = worst-case* yaitu fungsi kuadratik dari *n*.

II.2 Big-O Notation adalah cara untuk mengkonversi keseluruhan langkah-langkah suatu algoritma kedalam bentuk Aljabar, yaitu denganmenghiraukan konstanta yang lebih kecil dan koefisien yang tidakberdampak besar terhadap keseluruhan kompleksitas permasalahan yang diselesaikan oleh algoritma tersebut.

1. *Worst-case* dihitung dengan *Big-O Notation.*
2. *T(n) = O(f(n))* artinya *T(n)* berorde paling besar *f(n)* bila terdapat konstanta *C* dan *n0* sehingga *T(n) ≤ C.f(n)*, untuk *n ≥ n0.*
3. Dalam pembuktian *Big-O Notation,* perlu dicari nilai *n0* dan *C* sehingga terpenuhi kondisi *T(n) ≤ C.f(n)*.

II.3 Big-O Notation polinomial berderajat *n* digunakan untuk memperkirakan kompleksitas dengan mengabaikan suku berorde rendah.

1. Teorema 1

*T(n) = amnm +am-1nm-1+a1n+a0* adalah polinom berderajat *m* maka *T(n) = O(nm)*

1. Teorema 2

Misalkan *T1(n)=O(f(n)) dan T2(n) = O(g(n))*, maka

* + - 1. *T1(n)+T2(n)=O(max(f(n), g(n))* atau *T1(n)+T2(n) = O(f(n)+g(n))*
      2. *T1(n).T2(n)=O(f(n)).O(g(n))=O(f(n).g(n))*
      3. *O(c.f(n))=O(f(n)), c* adalah konstanta
      4. *f(n) =O(f(n))*

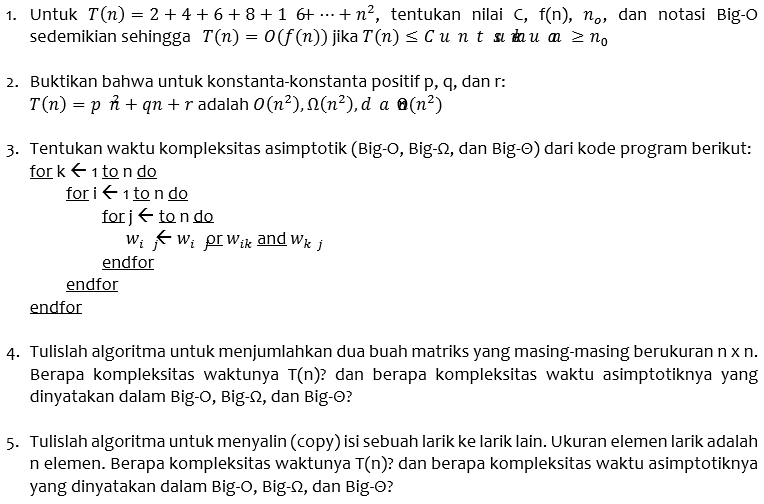
II.4 Aturan Menentukan Kompleksitas Waktu Asimptptik

1. Jika kompleksitas waktu T(n) dari algoritma sudah dihitung, maka kompleksitas waktu asimptotiknya dapat langsung ditentukan dengan mengambil suku yang mendominasi fungsi T dan menghilangkan koefisiennya (sesuai TEOREMA 1)
2. Kita bisa langsung menggunakan notasi Big-O, dengan cara: Pengisian nilai (assignment), perbandingan, operasi aritmatika (+, -, / ,\* , div, mod), read, write, pengaksesan elemen larik, memilih field tertentu dari sebuah record, dan pemanggilan function/void membutuhkan waktu O(1)

II.5 Notasi Big-O hanya menyediakan batas atas (upper bound) untuk perhitungan kompleksitas waktu asimptotik, tetapi tidak menyediakan batas bawah (lower bound). Untuk itu, lower bound dapat ditentukan dengan Big-Ω Notation dan Big-θ Notation.

1. *T(n)=Ω(f(n))*, artinya *T(n)* berorde paling kecil *f(n)* bila terdapat konstanta *C* dan *n* sehingga *T(n) ≥ C.(f(n))*, dengan syarat nilai *c* dan *n* positif.
2. *T(n)=Θ(h(n))*, artinya *T(n)* berorde sama dengan *h(n)* Jika *T(n)=O(h(n))* dan *T(n)=Ω(g(n))*.

C1f(n) ≤ T(n) ≤ C2f(n), dengan syarat nilai *c* dan *n* positif

1. **Worksheet 3**

