**TUGAS DESAIN & ANALISIS ALGORITMA**

**Klasifikasi Fungsi Berdasarkan Laju Pertumbuhannya**



**Disusun oleh:**

**Adrian Maulana Muhammad** **(06111540000099)**

**Departemen Matematika**

**Fakultas Sains dan Analitika Data**

**Institut Teknologi Sepuluh Nopember**

**Surabaya**

**2020**

1. **Pembukaan**

Pada bidang ilmu komputer dan matematika, terdapat momen dimana dibutuhkan untuk mengetahui kecepatan laju pertumbuhan dari suatu fungsi. Pengetahuan tersebut sangat berpengaruh dalam proses menganalisa seberapa cepat suatu algoritma bekerja untuk memecahkan masalah dengan *input* yang diberikan. Kita tentu ingin mendapatkan algoritma yang optimal, bukan algoritma yang bekerja dengan waktu yang lebih dari semestinya. Kita juga ingin menentukan seberapa optimal algoritma bekerja dengan ukuran input yang terus bertambah. Dengan mengetahui laju pertumbuhan dari fungsi atau estimasi *Big-O*, kita dapat membandingkan efesiensi kerja dari algoritma-algoritma yang diberikan dalam memecahkan suatu masalah. Berikut tokoh-tokoh yang berpengaruh dalam perkembangan analisa laju pertumbuhan fungsi.

Edmund Landau (1877-1938) Paul Gustav Heinrich Bachmann (1837-1920) Donald E. Knuth (Born 1938)

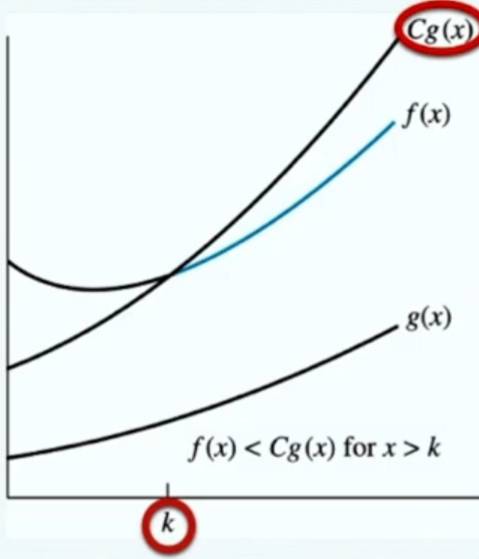
1. **Notasi *Big-O***

Terdapat fungsi *f* dan *g* dengan *input* dan *output* bertipe integer atau bilangan riil. Kita dapat menyatakan bahwa *f*(*x*) adalah *O*(*g*(*x*)), jika terdapat konstan *C* dan *k*. Dengan begitu, maka

… (1)

dengan syarat

Kondisi tersebut dapat dibaca “*f*(*x*) merupakan *big-O* dari *g*(*x*)” atau “*g* secara asimtot mendominasi *f*”’. Konstanta *C* dan *k* merupakan *witnesses* terhadap relasi *f*(*x*)merupakan *O*(*g*(*x*)). Relasi ini hanya membutuhkan sepasang *witnesses* sebagai syarat yang harus terpenuhi*.* Jika dilihat pada ilustrasi, *f*(*x*)merupakan *O*(*g*(*x*)), terdapat daerah pada *f*(*x*)(berwarna biru)yang kurang dari *C·g*(*x*))*.* Fungsi *g*(*x*)dikalikan dengan konstanta *C* bisa dianggap sebagai batas atas dari fungsi *f*(*x*)*.*



Terdapat beberapa hal yang penting tentang notasi *Big-O*. Pertama, jika terdapat sepasang *witnesses* yang ditemukan, maka terdapat tak hingga pasang *witnesses* yang lain. Kita dapat membuat nilai dari konstanta *k* atau *C* menjadi lebih besar, namun kita akan tetap mendapat kondisi pertidaksamaan (1). Setiap pasang *C\** dan *k\**, dimana *C* < *C\** dan *k* < *k\**, juga merupakan sepasang *witnesses,* karena

dengan berapapun nilai dari kondisi

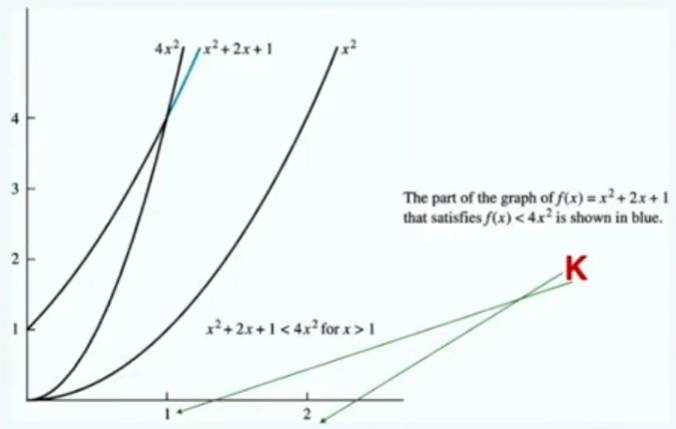
> *k*

Mungkin anda akan mengira “*f*(*x*) *= O*(*g*(*x*))*”* mempunyai makna yang sama dengan “*f*(*x*)merupakan *O*(*g*(*x*))*”,* namun hal tersebut adalah suatu kesalahan terhadap penggunaan operator sama dengan. Makna dari notasi *Big-O* adalah terdapat suatu kondisi ketidaksamaan terhadap nilai dari relasi *f* dan *g,* untuk nilai *x* yang besar. Pada penulisan “*f*(*x*)ϵ *O*(*g*(*x*))*”* bisa dianggap benar, karena *O*(*g*(*x*))merepresentasikan himpunan dari fungsi yang merupakan *O*(*g*(*x*))*.* Secara umum, biasanya kita tidak memperdulikan tanda absolut dari nilai (pemutlakan dari nilai positif atau negatif), dikarenakan kita akan selalu berhadapan dengan fungsi yang menggunakan nilai positif. Sebagai contoh, kita ingin menunjukan fungsi merupakan :

* Dengan dan , maka:

  + dengan *C =* 4 dan *k =* 1 sebagai *witnesses* untuk menunjukan *f(x)* adalah .
* Alternatif lainnya, dengan *x >* 2, , dan , maka:
  + dengan *C =* 3 dan *k =* 2 sebagai *witnesses.*

Jika kita lihat pada ilustrasi sebagai berikut



Maka bisa ditarik beberapa kesimpulan:

* dan merupakan relasi *f*(*x*)adalah *O*(*g*(*x*)) dan *g*(*x*)adalah *O*(*f*(*x*))*.* Atau dengan kata lain, kedua fungsi tersebut berada pada order yang sama.
* Jika *f*(*x*)adalah *O*(*g*(*x*))dan *h*(*x*)lebih besar daripada *g*(*x*)untuk semua bilangan riil yang positif, maka *f*(*x*) merupakan *O*(*h*(*x*)). Karena:
  + untuk *x > k* dan jika untuk semua *x*, maka jika *x > k.* Oleh karena itu *f*(*x*)adalah *O*(*h*(*x*))*.*
* Dalam banyak penerapan, tujuannya adalah untuk memilih fungsi *g*(*x*) di *O*(*g*(*x*))dengan sekecil mungkin (tentunya dengan perkalian terhadap konstanta).

Kita ambil contoh lain, yaitu kita akan tunjukan bahwa bukan merupakan *O*(*n*)*.* Pertama, dengan konstanta *C* dan *k* sebagai pasangan *witnesses*, maka didapat , dengan *n > k.* Selanjutnya, kita akan bagi kedua sisi dengan *n,* didapatkan untuk semua *n > k.* Didapatkan suatu kontradiksi, dimana seharusnya *n* bisa berlaku untuk *n > k,* tidak dibatasi oleh konstanta *C.*

1. **Estimasi *Big-O* pada Beberapa Fungsi Penting**

Selanjutnya kita akan melakukan estimasi terhadap beberapa fungsi penting menggunakan notasi *Big-O*. Pertama, kita akan melakukan estimasi terhadap fungsi *sum* dari urutan *n* pertama pada integer positif.

didapatkan:

maka adalah *O(),* dengan *C =* 1 dan *k =* 1.

Selanjutnya, kita akan estimasi terhadap fungsi *factorial*

didapatkan:

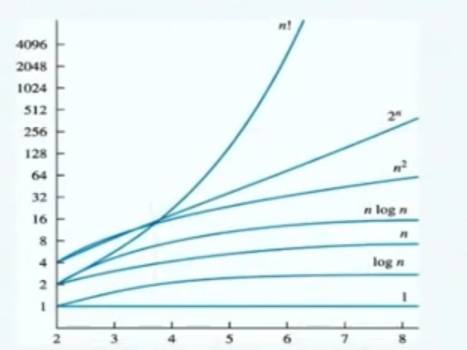
maka *n!* adalah dengan *C =* 1 dan *k =* 1.

Selanjutnya, kita akan estimasi terhadap fungsi log *n.* Dikarenakan, pada proses sebelumnya, didapatkan:

maka

Oleh karena itu, merupakan *O()* dengan *C =* 1 dan *k =* 1.

Jika kita buat semua proses di atas dalam suatu ilustrasi grafik.

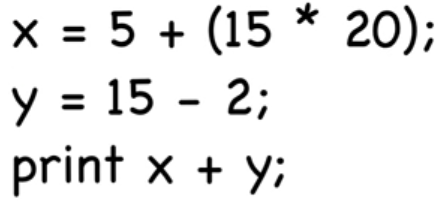


1. **Aturan Umum**

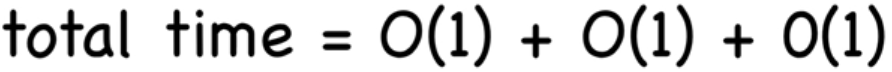
Terdapat beberapa aturan umum pada notasi *Big-O.* Pertama, notasi *Big-O* tidak memperdulikan konstanta. Sebagai contoh, program yang dijalankan dengan fungsi 5*n*, dikatakan merupakan *O*(*n*)*.* Hal ini bisa terjadi, karena saat ukuran *n* bertambah besar, maka konstanta 5 tidak lagi banyak berpengaruh. Saat ukuran *n* bertambah besar, maka beberapa fungsi akan mendominasi fungsi lainnya. Ditunjukan list sebagai berikut,



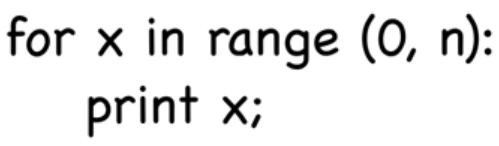
Selanjutnya, kita ambil contoh fungsi untuk waktu yang konstan. Suatu fungsi *x =* 5 + (15 \* 20), jika kita lihat, nilai *x* tidak berpengaruh terhadap ukuran dari input. Kita bisa bilang, bahwa *x* merupakan *O*(1*)*. Lalu, misalkan terdapat *source code* sebagai berikut,



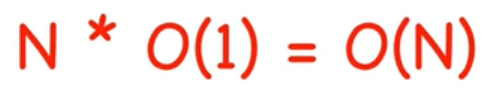
bisa kita lihat bahwa fungsi pada setiap barisnya merupakan fungsi dengan waktu yang konstan, maka untuk mendapatkan total waktu,



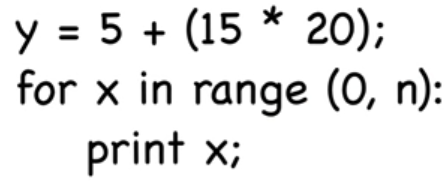
didapatkan 3\**O*(1), namun karena kita tidak memperdulikan konstanta, maka kita bisa bilang *source code* di atas merupakan *O*(1). Selanjutnya, kita ambil contoh fungsi untuk waktu yang liner. Misalkan diambil *source code* sebagai berikut,



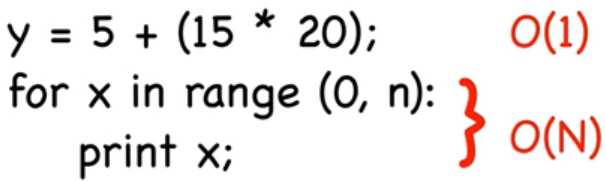
dari *source code* di atas, kita bisa lihat pada baris kedua, fungsi *print* tidak terpengaruh oleh ukuran input *n*, maka merupakan *O*(1). Pada fungsi *for* *loop* di baris pertama, terlihat bahwa fungsi terpengaruh oleh ukuran input *n.* Semakin besar *n,* semakin banyak iterasi *x.* Maka fungsi *print* menjadi*,*



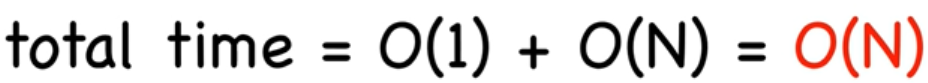
Lalu, kita ambil contoh lagi untuk *source code* sebagai berikut,



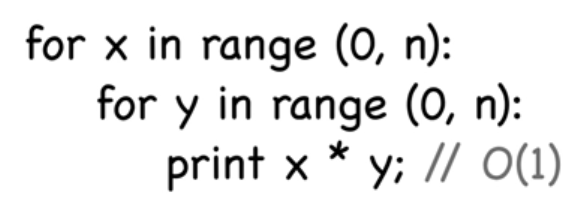
di baris pertama, kita tahu merupakan *O*(1) dan pada fungsi *loop* merupakan *O*(*n*),



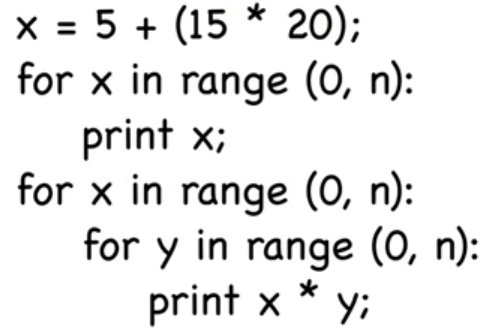
jika kita hitung total waktunya adalah *O*(1) + *O*(*n*), tetapi karena terdapat fungsi yang mendominasi, maka kita tidak hitung fungsi yang terdominasi. Maka totalnya sebagai berikut,



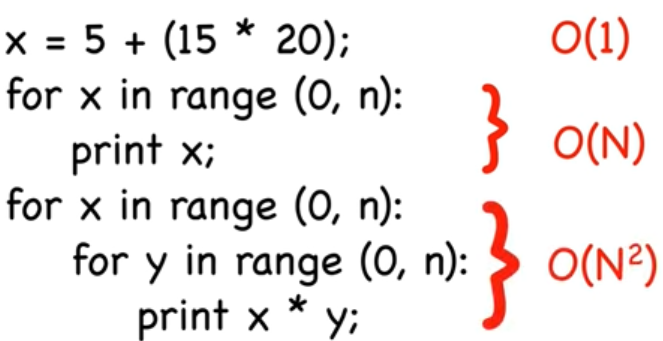
saat ukuran input *n* semakin besar, maka waktu yang dibutuhkan untuk mencari nilai *y* tidak berpengaruh secara signifikan dibandingkan waktu untuk fungsi *for loop*. Selanjutnya, kita ambil contoh untuk fungsi dengan waktu *quadratic,*



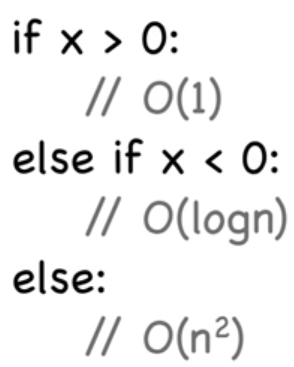
bisa dilihat bahwa pada fungsi *print* merupakan *O*(1) dan terdapat dua fungsi *for loop,* sehingga bisa kita hitung total waktunya adalah . Berikut contoh *source code* untuk merangkum yang telah kita pelajari,



jika kita lihat pada masing-masing fungsi,



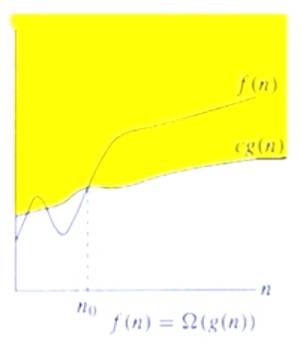
maka fungsi yang paling dominan adalah pada *double for loop*, karena semakin besar ukuran input *n*, maka waktu yang dibutuhkan pada fungsi lain tidak terlalu signifikan dibandingkan fungsi *double for loop*. Contoh selanjutnya, adalah *souce code* sebagai berikut,



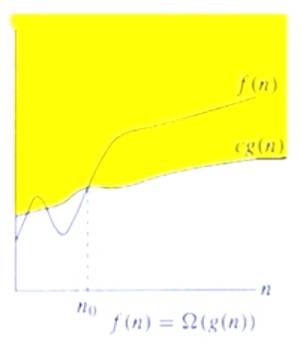
saat kita membicarakan tentang *Big-O* biasanya kita melihat *worst case,* maka pada *source code* di atas kita akan melihat *runtime* yang paling dominan, yaitu . Dalam penerapan kehidupan sehari-hari, saat anda membuat suatu program, penting untuk memperhatikan bahwa konstanta adalah hal yang penting. Dengan input ukuran kecil, maka konstanta akan cukup berpengaruh. Dan dalam penerapan algoritma pada program anda, penting untuk mengetahui tentang *best case* dan *average case* dari algoritma anda.

1. ***Big-Omega, Big-Theta,* dan *Big-O***

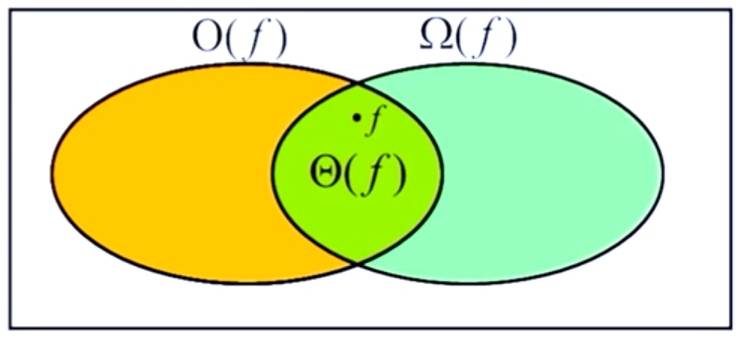
Secara umum, pemaknaan *Big-O* dan *Big-Omega* kurang lebih hampir mirip, yang membedakan adalah pemaknaan terdominasi dan mendominasi. Pada *Big-Omega*, misalnya terdapat relasi *f(x)* adalah Ω(*g(x)*), maka dapat dimaknai bahwa fungsi *f* secara asimtot mendominasi atau sama dengan fungsi *g,* dan *g* adalah batas bawah dari *f.* Secara ilustrasi grafik,

**

sementara untuk *Big-Theta* mempunyai pemaknaan *equality*. Misalnya didapatkan suatu relasi *f(x)* merupakan θ(*g*(*x*)), maka dapat dimaknai bahwa fungsi *f* secara asimtot sama dengan fungsi *g*. Secara ilustrasi grafik,



Jika kita lihat relasi dari ketiga notasi asimtot di atas, dapat dilihat pada ilustrasi sebagai berikut,



1. **Referensi**

* <https://www.youtube.com/watch?v=CpHKhXLhbwM&t=393s>
* <https://www.youtube.com/watch?v=s0glC0Ym20Q>
* <https://www.youtube.com/watch?v=__vX2sjlpXU&t=251s>
* <https://www.youtube.com/watch?v=D6xkbGLQesk>