Hubungan Kompleksitas Algoritma dengan Cara Belajar

Ryan Ignatius Hadiwijaya / 13511070
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
13511070@std.stei.itb.ac.id

***Abstrak*—Makalah ini membahas salah satu hubungan kompleksitas algoritma dengan cara belajar. Terdapat banyak jenis cara belajar. Dalam makalah ini, cara-cara belajar tersebut akan dianalogikan sebagai suatu paradigma penyelesaian masalah (*problem solving paradigm*) yang memiliki kompleksitas waktu dan efektivitas yang berbeda-beda. Akan dibahas pula kelebihan dan kekurangan masing-masing cara belajar dalam analogi tersebut.**

***Kata Kunci* — algoritma, cara belajar, kompleksitas, paradigma penyelesaian masalah (problem solving paradigm).**

I. PENDAHULUAN

Belajar merupakan hal yang umum dilakukan oleh seseorang. Bahkan, proses belajar dapat dialami seseorang sepanjang hidupnya. Tentunya, banyak cara-cara belajar yang dilakukan oleh setiap orang, yang mungkin saja berbeda-beda. Setiap cara belajar tersebut mungkin ada yang efektif dalam waktu singkat, atau memerlukan waktu yang lama.

Siswa ataupun mahasiswa, dalam proses pembelajarannya, sering dihadapkan pada kuis ataupun ujian. Apalagi mendekati akhir semester, berbagai ujian pun menghadang di waktu yang hampir bersamaan. Di saat demikian, cara belajar yang efektif sangat dibutuhkan agar dapat memperoleh hasil yang memuaskan di semua ujian.

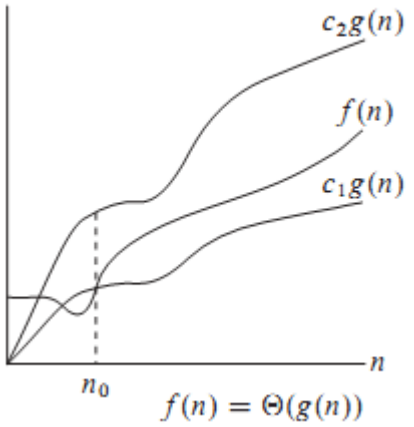
Cara-cara belajar tersebut dapat dianalogikan sebagai algoritma dalam menyelesaikan masalah (problem solving). Setiap algoritma memiliki karakteristiknya masing-masing, terutama dalam kebutuhan waktu dan efektivitasnya.

II. TEORI DASAR

A. Kompleksitas Algoritma

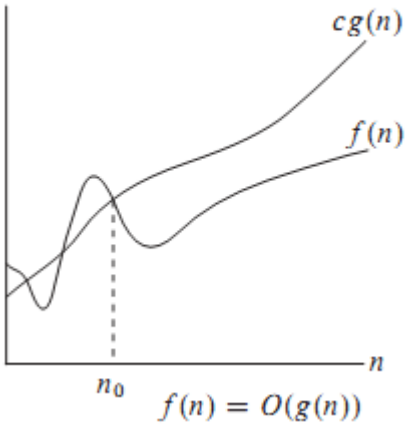
Menurut [4], kompleksitas algoritma merupakan suatu besaran yang dipakai untuk mengukur waktu / ruang yang independen dari pertimbangan mesin dan kompiler apapun. Kompleksitas suatu algoritma dapat dibagi menjadi kompleksitas waktu dan kompleksitas ruang. Kebutuhan waktu dan ruang suatu algoritma tersebut bergantung pada jumlah ukuran masukan (n) yang menyatakan jumlah data yang diproses. Kompleksitas waktu, T(n), diukur dari jumlah tahapan komputasi yang dibutuhkan untuk menjalankan algoritma sebagai fungsi dari ukuran masukan n. Sedangkan, kompleksitas ruang, S(n), diukur dari memori yang digunakan oleh struktur data yang terdapat di dalam algoritma sebagai fungsi dari ukuran masukan n. Dalam praktiknya, kita lebih memerhatikan pertumbuhan kebutuhan waktu tersebut saat n bertambah besar, daripada menghitung dengan tepat T(n)-nya. Maka, dirumuskanlah suatu notasi kompleksitas waktu asimptotik yang terdiri dari notasi-Θ, notasi-O, dan notasi-Ω. (Menurut [2], terdapat pula notasi-o dan notasi-ω yang tidak akan dibahas pada makalah ini).

Berikut penjelasan dari tiga notasi tersebut disertai dengan grafik (sumber grafik adalah [2] halaman 45) :



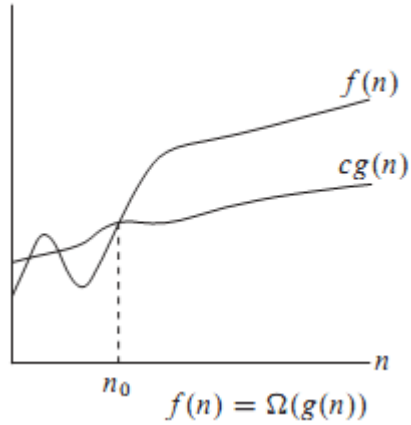
Gambar 1. Notasi-Θ

Dari gambar 1, didapat bahwa bila kompleksitas waktu suatu algoritma T(n) = f(n), maka T(n) = Θ(g(n)) jika terdapat bilangan positif konstan c1, c2, dan n_0 sehingga memenuhi persamaan : $0 \leq c1.g(n) \leq f(n) \leq c2.g(n)$, untuk semua $n \geq n_0$. Dalam hal ini, g(n) merupakan batas rapat (batas atas maupun batas bawah) asimptotik untuk fungsi f(n) (=T(n)). Notasi-Θ membatasi fungsi dari atas maupun bawah, yang berarti bahwa kompleksitas waktu tersebut sebanding dengan fungsi g(n). Notasi-Θ hanya didapat jika dan hanya jika T(n) = O(g(n)) dan T(n) = Ω(g(n)). Pada kontes pemrograman, seringkali yang dipakai adalah notasi-O daripada notasi- Θ ini. Hal ini disebabkan karena mayoritas hanya memperhitungkan kasus terburuk saja untuk mengkalkulasi waktu program.



Gambar 2. Notasi-O

Dari gambar 2, didapat bahwa bila kompleksitas waktu suatu algoritma T(n) = f(n), maka T(n) = O(g(n)) jika terdapat bilangan positif konstan c dan n_0 sehingga memenuhi persamaan : $0 \leq f(n) \leq c.g(n)$ untuk semua $n \geq n_0$. Dalam hal ini, g(n) merupakan batas atas asimptotik dari fungsi f(n) (= T(n)). Notasi-O ini hanya membatasi fungsi dari atas, yang memiliki arti bahwa fungsi f(n) tersebut tidak akan lebih besar dari g(n) tanpa mengetahui seberapa rapat g(n) dengan f(n).



Gambar 3. Notasi-Ω

Dari gambar 3, didapat bahwa bila kompleksitas waktu suatu algoritma T(n) = f(n), maka T(n) = Ω(g(n)) jika terdapat bilangan positif konstan c dan n_0 sehingga memenuhi persamaan : $0 \leq c.g(n) \leq f(n)$ untuk semua $n \geq n_0$. Dalam hal ini, g(n) merupakan batas bawah asimptotik dari fungsi f(n) (= T(n)). Notasi-Ω ini hanya membatasi fungsi dari bawah, yang memiliki arti bahwa fungsi f(n) tersebut tidak akan lebih kecil dari g(n) tanpa mengetahui seberapa rapat g(n) dengan f(n).

B. Problem Solving Paradigm

Merujuk pada [1], terdapat empat jenis paradigma penyelesaian masalah ini, yaitu *Complete Search*, *Divide and Conquer*, *Greedy*, dan *Dynamic Programming*. Berikut penjelasan singkat dari setiap paradigma tersebut beserta kelebihan dan kekurangannya :

1. *Complete Search*

Menurut [3], paradigma ini menyelesaikan masalah dengan melakukan pencarian lengkap ke seluruh sumber untuk mendapatkan jawaban. Solusi dengan *Complete Search* mudah untuk dikode. Kelebihan lainnya adalah kebenaran dari solusinya yang jarang sekali salah. Hal ini disebabkan *Complete Search* melakukan pencarian lengkap ke seluruh sumber sehingga kemungkinan besar akan menemukan jawaban yang benar. Kekurangan dalam *Complete Search* adalah kebutuhan waktunya yang sangat besar, yang seringkali melebihi batas waktu dalam kontes pemrograman.

2. *Divide and Conquer*

Menurut [2], paradigma ini melakukan tiga hal utama berikut, yaitu membagi (divide) masalah ke dalam beberapa sub-masalah yang lebih kecil dari masalah utama. Menyelesaikan (conquer) semua sub-masalah yang terbentuk. Dan terakhir menggabungkan (combine) solusi dari setiap sub-masalah (bila diperlukan). Kelebihan dari *Divide and Conquer* ini adalah mudah dalam menyelesaikan masalah yang besar, karena masalah tersebut dibagi-bagi dahulu menjadi masalah yang lebih kecil dan penyelesaian sub masalah pun relative lebih mudah. Kekurangan *Divide and Conquer* adalah tidak dapat mendeteksi adanya sub-masalah yang sama, sehingga akan tetap dianggap sebagai sub-masalah yang berbeda.

3. *Greedy*

Menurut [3], paradigma ini menyelesaikan masalah dengan membuat pilihan-pilihan. *Greedy* akan mengambil pilihan yang pada saat pilihan itu dipilih merupakan pilihan yang terbaik. Kelebihan dari *Greedy* adalah pengkodeannya yang relative cepat dan mudah. Selain itu, kebutuhan waktu dan ruangnya pun sedikit. Kekurangan dari *Greedy* adalah kebenaran dari solusinya. Hal ini disebabkan *Greedy* membuat pilihan yang terbaik pada saat itu tanpa mempertimbangkan adanya pilihan-pilihan lain yang nantinya akan menghasilkan solusi terbaik.

4. *Dynamic Programming*

Menurut [1], paradigma ini adalah suatu bentuk pengembangan dari paradigma *Divide and Conquer*, yang sama-sama membagi masalah menjadi sub-masalah yang lebih kecil. Perbedaannya adalah *Dynamic Programming* dapat mengetahui sub-masalah yang sejenis sehingga menghemat waktu karena setiap sub-masalah yang berbeda hanya akan dikerjakan sekali saja. Paradigma ini juga mirip dengan *Greedy*, yang membuat pilihan-pilihan unutk menyelesaikan masalah. Perbedaannya adalah *Dynamic Programming* membuat pilihan setelah pilihan dari sub-masalahnya tersedia. Kelebihan dari *Dynamic Programming* adalah dapat mengingat atau mencatat hasil dari setiap sub-masalah sehingga jika terdapat suatu sub-masalah baru yang telah dikerjakan, solusinya dapat langsung dihasilkan dengan melihat ke tabel pencatatnya tersebut. Kekurangannya adalah membutuhkan ruang yang lebih besar untuk menyimpan hasil-hasil tersebut.

III. PEMBAHASAN

A. Analogi Cara Belajar dengan Paradigma Penyelesaian Masalah

1. *Complete Search*

Cara belajar seperti *Complete Search* adalah belajar dengan cara membaca semua sumber yang dibutuhkan dimulai dari awal hingga akhir tanpa ada yang terlewat satu bagian pun. Bila jumlah bab yang dipelajari sebesar n bab dengan masing-masing bab sebesar b sub-bab, dengan masing-masing sub-bab sebanyak h halaman, maka kompleksitas waktunya adalah O(n.b.h).

Dengan metode ini, waktu yang dibutuhkan untuk belajar sangatlah besar, terutama bila sumbernya pun banyak. Namun, hasil yang didapatkan dari *Complete Search* ini seharusnya sudah menjadi hasil yang baik dan memuaskan (tanpa mempertimbangkan factor acak).

Pseudocode untuk *Complete Search* ini adalah :

```
For i =1 to MAXBAB do
    For j = 1 to MAXSUBBAB do
        For k = 1 to MAXHLMN do
            "Pelajari halaman ini"
```

Sehingga "Pelajari halaman ini" akan dieksekusi sebanyak MAXBAB x MAXSUBBAB x MAXHLMN (atau = n.b.h).

2. *Divide and Conquer*

Cara belajar seperti *Divide and Conquer* adalah membagi-bagi sub-bab yang akan dipelajari ke dalam kategori sudah dikuasai atau belum. Kemudian, belajar hanya difokuskan kepada sub-bab yang belum terkuasai sehingga akan menghemat waktu. Bila jumlah bab yang dipelajari sebesar n bab dengan masing-masing bab yang belum dikuasai sebesar d sub-bab, dengan masing-masing sub-bab sebanyak h halaman, maka kompleksitas waktunya adalah O(n.d.h). Kompleksitas waktu ini lebih kecil dari *Complete Search* karena $d \leq b$.

Dengan metode ini, waktu yang dibutuhkan untuk belajar relatif lebih rendah daripada metode *Complete Search* (berbeda-beda bergantung jumlah sub-bab yang belum dikuasai). Hasil yang didapat pun diharapkan memuaskan karena sub-bab yang belum dikuasai sudah dipelajari sehingga semua bab pun dikuasai.

Cara belajar lain yang sesuai dengan metode *Divide and Conquer* ini adalah dengan mencicil belajar untuk ujian dari jauh-jauh hari sebelumnya. Dengan cara ini, orang tersebut tidak akan kesulitan belajar dalam menghadapi ujian karena sudah dikuasai sebelumnya.

Pseudocode untuk *Divide and Conquer* ini adalah :

```
For i =1 to MAXBAB do
    For j = 1 to MAXSUBBAB do
        If (belum mengerti) then
            For k = 1 to MAXHLMN do
                "Pelajari halaman ini"
        Else "lanjut ke sub-bab selanjutnya"
```

For ketiga hanya dieksekusi saat sub-bab belum dimengerti sehingga "Pelajari halaman ini" akan dieksekusi sebanyak MAXBAB x jumlah sub-bab yang belum mengerti x MAXHLMN (atau = n.d.h).

3. *Greedy*

Cara belajar seperti *Greedy* adalah dengan belajar hanya dari kisi-kisi saja atau hanya dari contoh-contoh soal sebelumnya. Atau dapat pula hanya memelajari sub-bab - sub-bab yang kemungkinan besar akan muncul pada soal. Selain itu, proses belajarnya pun tidak mengiterasi setiap halamannya, namun hanya berfokus pada rumus-rumus penting dan hal-hal penting lainnya. Bila jumlah bab yang dipelajari sebesar n bab dengan masing-masing bab yang dianggap penting sebesar p sub-bab, dengan masing-masing sub-bab sebanyak r halaman yang berisi rumus atau hal penting lainnya, maka kompleksitas waktunya adalah O(n.p.r). Kompleksitas waktu ini lebih kecil dari *Complete Search* maupun *Divide and Conquer* karena $p \leq d \leq b$ dan r jauh lebih kecil dari h.

Dengan metode ini, waktu yang diperlukan untuk belajar menjadi sangat sedikit karena hanya memelajari hal-hal yang dianggap penting saja. Hasil yang didapat pun dapat saja memuaskan jika memang soal-soal yang muncul mirip dengan soal-soal sebelumnya. Hasil memuaskan juga didapat bila perkiraannya tentang soal-soal yang akan muncul tepat. Namun, jika yang terjadi bukanlah hal-hal seperti di atas, maka hasil yang didapat tentunya akan tidak memuaskan.

Pseudocode untuk *Greedy* ini adalah :

```
For i =1 to MAXBAB do
    For j = 1 to MAXSUBBAB do
        If (penting) then
            For k = 1 to MAXHLMN do
                If (penting) then
                    "Pelajari halaman ini"
                Else "lanjut ke halaman selanujutnya"
        Else "lanjut ke sub-bab selanjutnya"
```

For ketiga hanya akan dieksekusi saat sub-bab tersebut dianggap penting. Di dalam for ketiga itu juga, halaman tersebut hanya akan dipelajari bila terdeapat hal/rumus yang pentinga saja, sehingga "Pelajari halaman ini" akan dieksekusi sebanyak MAXBAB x jumlah sub-bab yang penting x jumlah halaman yang memuat rumus/hal penting lainnya (atau = n.p.r).

4. *Dynamic Programming*

Cara belajar seperti *Dynamic Programming* adalah dengan mencatat dan/atau membuat ringkasan dari setiap sub-bab. Kompleksitas waktu untuk pertama kali mencatat/meringkas sama dengan metode *Complete Search*. Namun, untuk setiap penggunaannya di kemudian hari, *Dynamic Programming* hanya membutuhkan O(1) untuk mempelajari kembali hal-hal yang telah dicatat/diringkas sebelumnya. Karena kompleksitas waktunya sama dengan *Complete Search*, maka sebaiknya proses pencatatan/peringkasan dilakukan sejak sebelumnya. Sehingga, saat dibutuhkan, dapat memakainya dengan cepat. Namun, dibutuhkan kompleksitas ruang yang lebih besar daripada yang lainnya, yaitu untuk catatan/ringkasan tersebut.

Pseudocode untuk *Dynamic Programming* ini adalah :

```
For i =1 to MAXBAB do
    For j = 1 to MAXSUBBAB do
        For k = 1 to MAXHLMN do
            If (belum tercatat) then
                "Pelajari halaman ini"
                "Catat hal penting"
            Else "lihat catatan"
```

Sehingga "Pelajari halaman ini" dan "Catat hal penting" akan dieksekusi sebanyak MAXBAB x MAXSUBBAB x MAXHLMN (atau = n.b.h) untuk pertama kali. Sedangkan untuk memelajari selanjutnya, yang dieksekusi adalah "lihat catatan" yang memiliki kompleksitas waktu O(1).

B. Pemilihan Cara Belajar yang Sesuai

Pembahasan selanjutnya mengenai pemilihan cara belajar yang sesuai. Yang dimaksud dengan sesuai adalah cara belajar yang paling efektif dan efisien yang penulis sarankan untuk kondisi-kondisi tertentu. Contoh-contoh dari kondisi tersebut, yaitu :

1. Latihan soal atau PR

Mengerjakan latihan soal atau PR merupakan salah satu cara belajar lain. Dengan mengerjakan semakin banyak soal, maka diharapkan seseorang akan lebih mengerti dan memahami tentang materi tersebut. Cara-cara mengerjakan latihan soal atau PR ini pun bermacam-macam. Jika memiliki waktu yang banyak untuk mengerjakannya, maka disarankan untuk mencari solusi (*Complete Search*) cara/rumus untuk mengerjakan latihan soal atau PR tersebut di buku pegangan yang sesuai. Dengan mencari jawabannya pada buku pegangan, maka kemungkinan besar jawaban tersebut benar karena sesuai dengan buku pegangan tersebut. Namun, akan lebih baik pula bila cara/rumus untuk mengerjakan latihan soal atau PR tersebut dicatat/diingat (metode *Dynamic Programming*) sehingga bila ada soal lain yang mirip, maka tinggal melihat kembali catatan rumusnya, tidak perlu mencari kembali di buku pegangan.

Namun, jika tidak memiliki waktu yang cukup untuk mengerjakan semuanya, maka disarankan untuk membagi-bagi (*Divide and Conquer*) soal tersebut dengan memperkirakan kesulitan soalnya dan kemampuan kita untuk menjawabnya. Untuk soal-soal yang dianggap mudah dan mampu untuk dikerjakan, maka jawablah sesuai dengan kemampuan (*Greedy*). Dan untuk soal-soal yang dianggap sulit saja, baru mencarinya di buku pegangan. Kemungkinan besar, untuk soal-soal yang kita anggap mudah dan mampu untuk dikerjakan, kita akan mendapatkan hasil yang memuaskan asalkan kita teliti dalam mengerjakannya. Sedangkan, untuk soal-soal yang dianggap sulit, hasilnya masih dapat memuaskan asalkan kita berhasil menemukan jawabannya pada buku pegangan.

2. Kuis

Kuis merupakan latihan untuk menghadapi ujian. Terkadang kuis diadakan mendadak, terkadang sudah dijadwalkan sebelumnya. Untuk kuis yang sudah dijadwalkan sebelumnya, maka cara belajar yang disarankan adalah dengan metode *Complete Search + Dynamic Programming*, yaitu dengan memelajari semua bahan kuis sambil mencatat hal-hal yang penting atau membuat ringkasannya. Dengan mengetahui jadwal dan bahan kuisnya, kita dapat menghitung waktu sebelum kuis tersebut agar semua bahan dapat selesai dipelajari. Sehingga, saat kuis tiba semua bahan sudah selesai dipelajari dan diharapkan kita dapat mengerjakan semua soal kuis tersebut. Metode lain adalah dengan metode *Divide and Conquer*, yaitu dengan menandai setiap sub-bab yang sudah yakin dikuasai. Selanjutnya, perlahan memelajari sub-bab yang belum dikuasai. Sehingga, sebelum kuis, cukup memelajari sub-bab yang belum sepenuhnya dikuasai.

Namun, untuk kuis yang dilaksanakan mendadak, maka cara belajar yang disarankan adalah dengan metode *Greedy*, yaitu dengan memilih suatu sub-bab yang kemungkinan besar akan ada dalam soal kuis tersebut, atau sub-bab yang dianggap paling sulit. Kemudian, lihat catatan atau ringkasan tentang sub-bab tersebut bila ada. Atau dapat pula dengan menanyakannya kepada teman sekitar. Hal ini dilakukan dengan harapan jika soal tersebut benar-benar muncul, maka kita dapat mengerjakannya. Ulangi langkah di atas selama masih memungkinkan. Karena semakin banyak sub-bab yang dibahas, semakin yakin pula kita mengerjakan kuis tersebut dan semakin besar pula peluang kita untuk mendapatkan hasil yang memuaskan.

3. Ujian

Cara belajar untuk ujian yang disarankan mirip dengan cara belajar untuk kuis yang sudah terjadwal, melalui metode *Complete Search + Dynamic Programming*, yaitu dengan memelajari semua bahan ujian sambil mencatat hal-hal yang penting atau membuat ringkasannya. Dengan mengetahui jadwal dan bahan ujian, kita dapat menghitung waktu sebelum ujian tersebut agar semua bahan dapat selesai dipelajari. Sehingga, saat ujian tiba semua bahan sudah selesai dipelajari dan diharapkan kita dapat mengerjakan semua soal ujian tersebut. Perbedaannya, adalah kita tidak boleh terlalu yakin bahwa kita telah menguasai materi tersebut walaupun sudah selesai dipelajari. Maksudnya, adalah dengan terus berlatih mengerjakan latihan-latihan soal. Dapat pula menggabungkan dengan metode *Greedy*, yaitu memelajari kisi-kisi jika ada dan soal-soal yang muncul sebelumnya. Jika semua bahan sudah selesai dipelajari dan sudah banyak berlatih dari latihan soal, maka diharapkan hasil yang didapat pun akan memuaskan.

4. Pelajaran yang benar-benar baru

Untuk memelajari sesuatu yang benar-benar baru, mka tidak ada pilihan lain kecuali dengan metode *Complete Search*, yaitu memelajari setiap bagian tersebut tanpa terlewat. Dapat pula digabungkan dengan metode *Dynamic Programming*, yaitu dengan mencatat hal-hal yang dianggap penting.

IV. FAKTOR ACAK

Dalam pembahasan makalah ini, penulis mengabaikan beberapa factor acak yang mungkin dapat memengaruhi. Antara lain adalah :

1. Kecepatan membaca setiap orang
2. Kemampuan mengingat setiap orang
3. Keinginan/niat belajar setiap orang

V. SARAN

Dari uraian pembahasan di atas, penulis dapat menyimpulkan saran-saran dalam pemilihan cara belajar, yaitu:

1. Gunakan metode *Complete Search* + *Dynamic Programming* dalam belajar, yaitu dengan memelajari seluruh bahan pembelajaran sambil mencatat hal-hal yang penting.
2. Lakukan metode *Divide and Conquer*, yaitu dengan memelajari sub-bab satu-persatu. Dengan memelajari satu-persatu sub-bab tersebut, diharapkan akan lebih mudah untuk dimengerti.
3. Lakukan metode *Greedy*, yaitu memelajari hanya hal-hal yang dianggap penting saja, jika waktu yang tersedia untuk belajar tidak mencukupi.

REFERENSI

[1] Steven Halim and Felix Halim, *Competitive Programming (Increasing the Lower Bound of Programming Contest)*. Bandung : diperbanyak oleh TOKI, 2010.

[2] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Cifford Stein, *Introduction to Algorithm third edition*. London : MIT Press, 2009.

[3] USACO Training Program Gateway, URL: http://ace.delos.com/usacogate. Tanggal akses 16 Desember 2012 pukul 10.40.

[4] Rinaldi Munir, *Matematika Diskrit edisi III*. Bandung : Informatika Bandung, 2007.

PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 16 Desember 2012

Ryan Ignatius Hadiwijaya / 13511070