# Analisis Kompleksitas Siklomatik Suatu Algoritma dengan Memanfaatkan Teori Graf

Jova Andres Riski Sirait - 13520072
*Program Studi Teknik Informatika*
*Sekolah Teknik Elektro dan Informatika*
*Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia*
*13520072@std.stei.itb.ac.id*

***Abstract*—Kode yang kompleks akan cenderung membuat program yang kita buat sulit dipelihara karena langkah yang terlalu banyak maupun keterbacaan yang rendah. Untuk menghindari hal ini, kita selalu berusaha untuk menghasilkan kode yang paling sederhana, elegan, mudah dibaca, dan jumlah langkah yang sedikit mungkin. Akan tetapi, kompleksitas kode tidak bisa diukur dengan melihat sekilas jumlah baris atau file yang banyak. Salah satu metrik yang bisa menjadi indikator yang baik dalam mengukur kompleksitas kode adalah kompleksitas siklomatik (*cylomatic complexity)* yang memanfaatkan graf aliran kontrol (*control flow graph)*, yaitu suatu model yang digunkan untuk mengindentifikasi jalur dasar dalam sistem prosedural. Model yang digunakan adalah graf berarah yang mereprensentasikan alur kerja dari sebuah algoritma.**

***Keywords*—graf, graf aliran kontrol, graf berarah, kompleksitas siklomatik**

## I. PENDAHULUAN

Ketika memulai petualangan menulis kode, mungkin kita pernah berpikir bahwa kode yang kompleks adalah pertanda kita semakin mahir dalam menuliskan kode. Semakin sulit masalah yang kita pecahkan, semakin kompleks pula kode yang kita buat.

Pada kenyataanya, kode yang kompleks merupakan hal yang ingin dihindari di dalam sebuah aplikasi yang cukup besar. Kode yang kompleks rentan menghasilkan masalah khususnya pada sisi pemeliharaan kode. Hal-hal yang rentan membuat kode menjadi kompleks dan sulit dipelihara adalah percabangan yang terlalu banyak, potongan kode yang tidak terjangkau atau tidak akan pernah dieksekusi, dan perulangan bersarang yang tidak dioptimisasi. Kode yang kompleks, sulit dibaca, dan terlalu banyak langkah akan membuat pengembangan menjadi terhambat dan harga yang dikeluarkan juga semakin besar. Untuk menghindari dampak negatif dari kode yang kompleks, dibutuhkan suatu indikator untuk mengukur kompleksitas kode yang kita buat sebelum masuk ke tahap produksi.

Salah satu indikator yang cukup baik dalam mengukur kompleksitas sebuah algoritma adalah kompleksitas siklomatik (*cyclomatic complexity*). Teknik ini dikembangkan oleh Thomas J. McCabe pada tahun 1976 dan didasarkan pada representasi aliran kontrol dari program. Aliran kontrol ini dimodelkan dengan graf berarah dan menggambarkan alur kerja program sebagai sebuah simpul yang mendefinisikan kode yang dieksekusi dan sisi berarah yang mendefiniskan jalur yang mungkin dilalui setelah suatu simpul selesai dieksekusi.

Kompleksitas siklomatik mengukur jumlah jalur linier pada potongan kode, yaitu percabangan, perulangan dan lompatan yang terdapat pada kode tesebut. Semakin besar tingkat kompleksitas yang didapatkan melalui metrik ini, semakin rentan pula potongan kode tersebut membuat program menjadi sulit dipelihara.

Berdasarkan metrik McCabe yang telah dikostumisasi, berikut adalah tingkat kompleksitas berdasarkan skala yang didapatkan: 1 sampai 10 – tidak kompleks, 11 – 20 – sedikit kompleks, 21 – 50 – sangat kompleks, dan lebih dari 50 – terlalu kompleks.

Dengan menganalisa kode yang telah dibuat dengan kompleksitas siklomatik, kita bisa menghasilkan kode yang lebih sederhana dan terhindari dari berbagai masalah yang mungkin terjadi ketika melakukan pengembangan program yang dibuat.

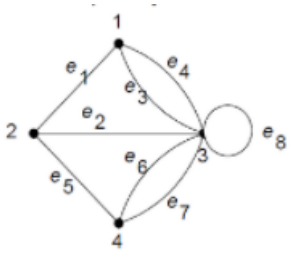
## II. TEORI DASAR

### A. Graf

Graf adalah himpunan dari simpul-simpul (*vertices*) dan sisi (*edges*) yang digunakan untuk merepresentasikan objek-objek diskrit dan hubungan antara objek-objek tersebut. Secara matematis, dapat ditulis

$$G = (V, E)$$

dengan:
V = himpunan tidak-kosong dari simpul-simpul (vertices)
= {v1, v2, ..., vn}
E = himpunan sisi (edges) yang menghubungkan simpul
= {e1, e2, ..., en}



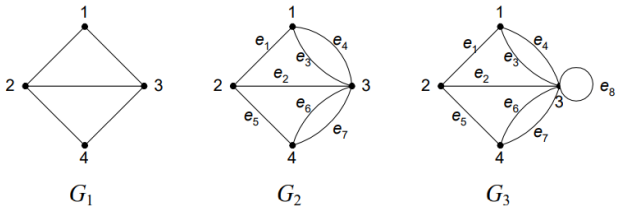

**Gambar 2.1** Contoh Graf
Sumber: https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2020-2021/Graf-2020-Bagian1.pdf

Graf dapat dikelompokkan menjadi beberapa jenis, berdasarkan ada tidaknya sisi gelang atau sisi ganda, serta berdasarkan orientasi arah pada sisinya.

Berdasarkan ada tidaknya sisi gelang atau sisi ganda, graf digolongkan menjadi dua jenis:

1. Graf Sederhana (*simple graph*)
   Graf sederhana adalah graf yang tidak mengandung sisi gelang maupun sisi ganda.

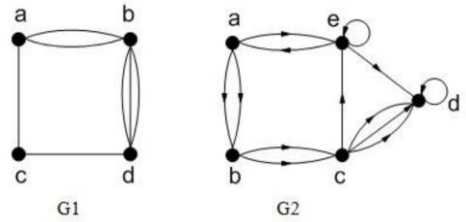
2. Graf Tak-Sederhana (*unsimple graph*)
   Graf tak-sederhana adalah graf yang mengandung sisi gelang atau sisi ganda. Graf tak-sederhana dibedakan lagi menjadi dua jenis, yaitu graf ganda (*multi-graph*), yang mengandung sisi ganda, dan graf semu (*pseudo-graph*) yang mengandung sisi gelang.




**Gambar 2.2** (a) graf sederhana, (b) graf ganda, (c) graf semu
Sumber: https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2020-2021/Graf-2020-Bagian1.pdf

Berdasarkan orientasi arah pada sisi, graf digolongkan menjadi dua jenis:

1. Graf tak-berarah (*undirected graph*)
   Graf tak-berarah adalah graf yang sisinya tidak mempunyai orientasi arah.
2. Graf berarah (*directed graph*)
   Graf berarah adalah graf yang setiap sisinya diberikan orientasi arah.




**Gambar 2.3** (a) graf tak-berarah, (b) graf berarah
Sumber: https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2020-2021/Graf-2020-Bagian1.pdf

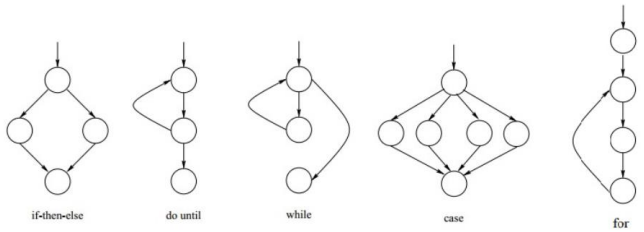
Pada teori graf, terdapat beberapa terminologi yakni sebagai berikut:

1. Ketetanggaan (*Adjacent*), dua buah simpul dikatakan bertetangga bila keduanya terhubung langsung.
2. Bersisian (*Incidency*), untuk sembarang sisi e = ($v_j$, $v_k$) dikatakan e bersisian dengan simpul $v_j$, atau e bersisian dengan simpul $v_k$.
3. Simpul Terpencil (*Isolated Vertex*), ialah simpul yang tidak mempunyai sisi yang bersisian dengannya.
4. Graf Kosong (*null graph* atau *empty graph*), yaitu graf yang himpunan sisinya merupakan himpunan kosong ($N_n$).
5. Derajat (*Degree*), derajat suatu simpul adalah jumlah sisi yang bersisian dengan simpul tersebut.
6. Lintasan (*Path*), lintasan yang panjangnya n dari simpul awal $v_0$ ke simpul tujuan $v_n$ di dalam graf G ialah barisan berselang-seling simpul-simpul dan sisi-sisi yang berbentuk $v_0$, $e_1$, $v_1$, $e_2$, $v_2$, ..., $v_{n-1}$, $e_n$, $v_n$ sedemikian sehingga $e_1$ = ($v_0$, $v_1$), $e_2$ = ($v_1$, $v_2$), ..., en = ($v_{n-1}$, $v_n$) adalah sisi-sisi dari graf G.
7. Siklus (*Cycle*) atau Sirkuit (*Circuit*), adalah lintasan yang berawal dan berakhir pada simpul yang sama.
8. Kerterhubungan (*Connected*), dua buah simpul $v_1$ dan simpul $v_2$ disebut terhubung jika terdapat lintasan dari $v_1$ ke $v_2$.
9. Upagraf (*Subgraph*) dan Komplemen Upagraf, misalkan G = (V, E) adalah sebuah graf. G1 = (V1, E1) adalah upagraf (subgraph) dari G jika V1 $\subseteq$ V dan E1 $\subseteq$ E. Komplemen dari upagraf G1 terhadap graf G adalah graf G2 = (V2, E2) sedemikian sehingga E2 = E - E1 dan V2 adalah himpunan simpul yang anggota-anggota E2 bersisian dengannya.
10. Upagraf Merentang (*Spanning Subgraph*), upagraf G1 = (V1, E1) dari G = (V, E) dikatakan upagraf rentang jika V1 =V (yaitu G1 mengandung semua simpul dari G).
11. *Cut-Set*, *cut-set* dari graf terhubung G adalah himpunan sisi yang bila dibuang dari G menyebabkan G tidak terhubung. Jadi, *cut-set* selalu menghasilkan dua buah komponen.
12. Graf Berbobot (*Weighted Graph*), adalah graf yang setiap sisinya diberi sebuah harga (bobot).

## *B. Graf Aliran Kontrol*

Graf aliran kontrol (*control flow graph*) dalam Ilmu Informatika adalah sebuah representasi alur kontrol program, menggunakan notasi graf, yang terdiri dari simpul yang menunjukkan blok dasar, dan sisi yang menunjukkan jalur yang mungkin dilalui antar blok pada sebuah program. Blok dasar yang dimaksud adalah potongan kode yang tidak memiliki lompatan ke potongan kode yang lain, sedangkan lompatan yang dimaksud adalah percabangan maupun perulangan pada program.

Representasi graf yang dihasilkan berbeda pada tiap statement dan perulangan, seperti pada gambar berikut:




**Gambar 2.4** representasi graf aliran kontrol (a) if-else-then, (b) do-while, (c) while, (d) switch-case, (e) for
Sumber: https://dzone.com/articles/how-draw-control-flow-graph

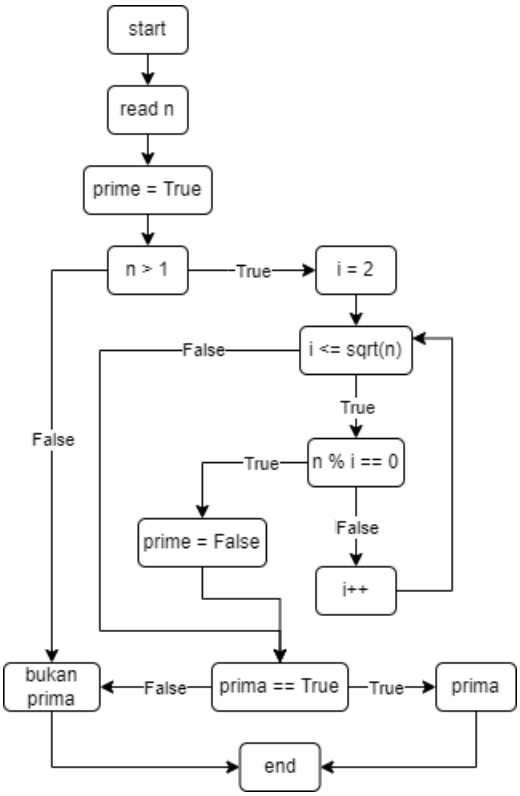
Graf aliran kontrol dari sebuah program diawali dengan membuat blok masuk (*entry block*) sebagai simpul awal dan blok keluar (*exit block*) sebagai simpul akhir program. Di antara blok masuk dan blok keluar merupakan kode utama dari program yang berisi blok-blok dasar yang berupa operasi:

baca/tulis, aritmetika (+, -, *, /, %), pengisian (*assignment*), akses larik, percabangan, perulangan, dan tipe operasi lain yang bisa beragam tergantung bahasa pemrograman yang digunakan.

Contoh sebuah program sederhana untuk menentukan apakah sebuah bilangan adalah prima atau bukan dalam bahasa python.

```
n = int (input ())
prime = True
if n > 1:
    for i in range (2, int (n ** 0.5) + 1):
        if n % i == 0:
            prime = False
            break
    if prime:
        print (n, "adalah bilangan prima.")
    else:
        print (n, "bukan bilangan prima.")
else:
  print (n, "bukan bilangan prima.")
```

Berdasarkan kode di atas, terdapat banyak operasi yang dapat dibagi menjadi blok-blok dasar yang memiliki alur keberjalanan. Graf yang dihasilkan dari blok-blok tersebut adalah seperti di bawah ini.




**Gambar 2.5** graf aliran kontrol dari algoritma validasi bilangan prima
Sumber: Arsip Penulis

Pada implementasinya, graf aliran kontrol dapat membantu pemrogram untuk menganalisis program, memastikan semua blok dapat dicapai, mendeteksi anomali, dan menganalisis kompleksitas siklomatik program.

### *C. Kompleksitas Siklomatik*

Kompleksitas siklomatik (cyclomatic complexity) adalah suatu metrik perangkat lunak yang digunakan untuk mengukur kompleksitas sebuah potongan kode atau algoritma dari sebuah program. Kompleksitas siklomatik dapat diukur dengan memanfaatkan graf aliran kontrol dari suatu fungsi, module, metode maupun kelas yang terdapat pada suatu program.

Kompleksitas siklomatik diukur berdasarkan jumlah jalur independen linear pada kode. Jika pada kode tidak terdapat jalur keputusan seperti percabangan atau perulangan, maka kompleksitasnya adalah 1, sedangkan jika terdapat dua jalur keputusan yang tergantung pada satu blok simpul, maka kompleksitasnya adalah 2.

Untuk memudahkan perhitungan kompleksitas siklomatik pada graf yang cukup kompleks, dapat digunakan beberapa persamaan yaitu:

$$V(G) = E - N + 2$$

dengan:
E = jumlah sisi
N = jumlah simpul

$$V(G) = P + 1$$

dengan:
P = jumlah node yang mengandung jalur keputusan

Metode lain yang bisa dimanfaatkan untuk menghitung kompleksitas siklomatik adalah dengan menggunakan langkah sebagai berikut:

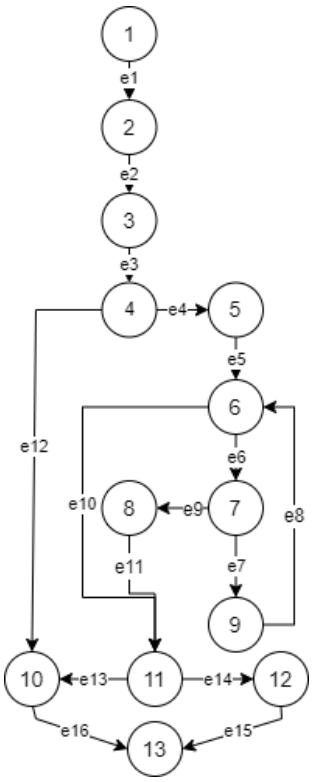
1. Hubungkan simpul terakhir dengan simpul awal dengan sebuah sisi berarah.
2. Kompleksitas siklomatik adalah jumlah semua ruang yang terdapat di dalam graf.

Arti dari nilai V(G) yang didapat memiliki makna yang disajikan pada tabel di bawah ini (rentang nilai bisa berbeda tergantung sumber yang diambil).

| V(G) / Nilai Kompleksitas | Makna |
|---|---|
| 1-10 | Kode terstruktur dan ditulis dengan baik, mudah diuji, biaya pengembangan lebih sedikit. |
| 11-20 | Kode cukup kompleks, relatif lebih sulit untuk diuji, biaya pengembangan lebih tinggi. |
| 21-50 | Kode sangat kompleks, sulit untuk diuji, biaya pengembangan tinggi. |
| >50 | Kode terlalu kompleks, sangat sulit bahkan tidak dapat diuji, biaya pengembangan sangat tinggi. |

**Table 2.1** Nilai kompleksitas siklomatik dan maknanya

Dengan mengambil contoh pada gambar 2.5, kita bisa menghitung kompleksitas siklomatik dengan menyederhanakan graf yang dibuat sebelumnya, seperti pada gambar di bawah.




**Gambar 2.6** graf aliran kontrol dari algoritma validasi bilangan prima disederhanakan
Sumber: Arsip Penulis

Berdasarkan gambar diatas, jumlah simpul adalah 13 dan jumlah sisi adalah 16, sehingga kompleksitas siklomatik algoritma tersebut adalah:

$$V(G) = E - N + 2$$
$$V(G) = 16 - 13 + 2$$
$$V(G) = 5$$

Nilai kompleksitas 5, berarti algoritma sudah terstruktur dan ditulis dengan baik. Dengan mengetahui nilai kompleksitas seperti ini, pemrogram ataupun timnya dapat mengambil keputusan yang paling tepat terhadap program yang dibuatnya. Kompleksitas yang lebih rendah berarti kode yang ditulis semakin efisien. Akan tetapi, kompleksitas yang rendah ini tidak selalu menjamin kode yang ditulis mudah untuk dibaca dan dipahami.

## III. Studi Kasus: Membandingkan Kompleksitas Siklomatik Beberapa *Quadratic Sorting Algorithms*

### *A. Quadratic Sorting Algorithm*

*Quadratic sorting algorithm* adalah jenis algoritma pengurutuan yang pada umumnya melakukan traversal dua tahap, sehingga terdapat pengulangan bersarang sehingga kompleksitasnya menjadi $O(n^2)$. Pada algoritma ini, jenis operasi yang dilakukan adalah operasi perbandingan tiap elemen dan operasi pertukaran tempat. Meskipun memiliki kompleksitas yang lebih tinggi daripada *comparison sorting* maupun *linearithmic sorting*, algoritma jenis ini masih cukup baik digunakan pada data masukan yang kecil karena tidak memerlukan tambahan memori. Beberapa algoritma *quadratic sorting* yang terkenal adalah *bubble sort, selection sort,* dan *insertion sort.*

1. *Bubble Sort*

   *Bubble sort* adalah algoritma pengurutan yang bekerja dengan membandingkan dua elemen yang bedekatan, kemudian menukar posisi elemen tersebut jika berada pada urutan yang salah. Proses ini dilakukan secara traversal dan berulang kali sampai didapat larik dengan urutan yang benar. *Average time complexity* untuk algoritma ini adalah $O(n^2)$. Berikut adalah implementasi algoritma *bubble sort* dalam bahasa python:

```
n = len(arr)
for i in range(n-1):
    for j in range (0, n-i-1):
        if arr[j] > arr [j + 1]:
            arr[j], arr [j + 1] = arr [j + 1], arr[j]
```

2. *Selection Sort*

   *Selection sort* adalah algoritma pengurutan yang bekerja dengan mencari nilai maksimum atau minimum dari suatu larik, kemudian menempatkannya pada posisi yang benar, berdasarkan tipe pengurutan yang diminta. Proses dilakukan berulang kali pada larik yang belum terurut, sampai seluruh larik selesai diproses. *Average time complexity* untuk algoritma ini adalah $O(n^2)$. Berikut adalah implementasi algoritma *selection sort* dalam bahasa python:

```
n = len(arr)
for i in range(n):
    min_idx = i
    for j in range (i+1, n):
        if arr[min_idx] > arr[j]:
            min_idx = j
    arr[i], arr[min_idx] = arr[min_idx], arr[i]
```

3. *Insertion Sort*

   *Insertion sort* adalah algoritma pengurutan yang bekerja dengan melakukan traversal dari elemen pertama dan elemen terakhir, dan pada setiap traversalnya dilakukan perbandingkan elemen yang dijadikan kunci, dengan elemen-elemen berikutnya. Elemen yang lebih besar akan dipindahkan ke kanan elemen kunci. *Average time complexity* untuk algoritma ini adalah $O(n^2)$. Berikut adalah implementasi algoritma *insertion sort* dalam bahasa python:

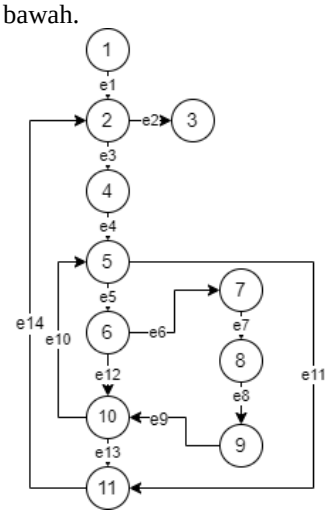
```
for i in range (1, len(arr)):
    key = arr[i]
    j = i-1
    while j >=0 and key < arr[j]:
        arr[j+1] = arr[j]
        j -= 1
    arr[j+1] = key
```

### *B. Implementasi Algoritma ke Bentuk Graf Aliran Kontrol*

1. *Bubble Sort*

   Pada algoritma *bubble* sort, terdapat perulangan bersarang, satu operasi perbandingan, dan operasi

pertukaran posisi pada perulangan bersarangnya. Graf aliran kontrol untuk algoritma bubble sort adalah seperti pada gambar di bawah.

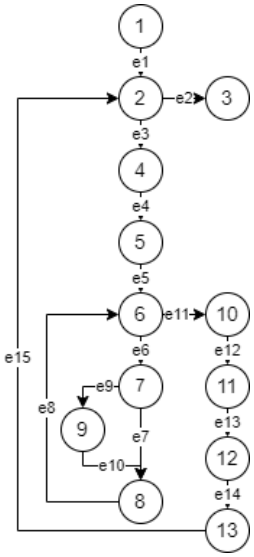



**Gambar 3.1** graf aliran kontrol dari algoritma *bubble sort*
Sumber: Arsip Penulis

Pada graf aliran kontrol di atas, simpul 2 sampai 11 merepresentasikan perulangan yang pertama, simpul 5 sampai 10 merepresentasikan perulangan yang kedua, dan simpul 7 sampai 9 merepresentasikan pertukaran elemen ketika kondisinya memenuhi.

2. *Selection Sort*

Pada algoritma *selection sort*, hampir mirip dengan *bubble sort*, terdapat perulangan bersarang, perbandingan elemen, dan juga pertukaran elemen. Yang membedakannya adalah pada algoritma *selection sort*, fungsi dari perulangan bersarangnya adalah mencari nilai ekstrem dari elemen larik, sedangkan operasi pertukaran elemen hanya terjadi setiap perulangan yang terluar. Graf aliran kontrol untuk algoritma bubble sort adalah seperti pada gambar di bawah.

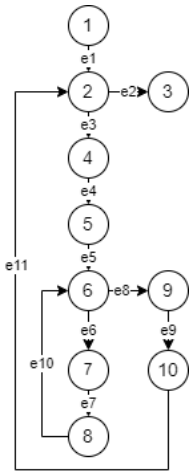



**Gambar 3.2** graf aliran kontrol dari algoritma *selection sort*
Sumber: Arsip Penulis

Pada graf aliran kontrol di atas, simpul 2 sampai 13 merepresentasikan perulangan yang pertama, simpul 6 sampai 9 merepresentasikan perulangan yang kedua untuk mencari nilai ekstrem, dan simpul 10 sampai 12 merepresentasikan pemindahan elemen yang bernilai ekstrem ke posisi yang benar.

3. *Insertion Sort*

Pada algoritma *insertion sort*¸ hampir mirip dengan dua algoritma sebelumnya, terdapat perulangan bersarang, operasi perbandingan elemen, dan juga operasi pertukaran nilai kunci. Graf aliran kontrol untuk algoritma bubble sort adalah seperti pada gambar di bawah.




**Gambar 3.3** graf aliran kontrol dari algoritma *insertion sort*
Sumber: Arsip Penulis

Pada graf aliran kontrol di atas, simpul 2 sampai 10 merepresentasikan perulangan yang pertama, simpul 6 sampai 9 merepresentasikan perulangan yang kedua untuk mencari nilai ekstrem, serta simpul 7 dan simpul 9 yang masing-masing merepresentasikan pertukaran elemen dan pergantian nilai kunci.

## *C. Hasil yang Didapatkan*

1. *Bubble Sort*

Berdasarkan graf aliran kontrol pada gambar 3.1, terdapat 11 simpul dan 14 sisi, sehingga kompleksitas siklomatiknya adalah:

$$V(G) = E - N + 2$$
$$V(G) = 14 - 11 + 2$$
$$V(G) = 5$$

Berdasarkan nilai kompleksitasnya, algoirtma *bubble sort* masih tergolong algoritma yang efisien. Namun, jika dilihat dari *time complexity* yaitu $O(n^2)$, algoritma ini kurang efisien untuk jumlah data yang besar.

2. *Selection Sort*

Berdasarkan graf aliran kontrol pada gambar 3.2, terdapat 13 simpul dan 15 sisi, sehingga kompleksitas siklomatiknya adalah:

$$V(G) = E - N + 2$$
$$V(G) = 15 - 13 + 2$$
$$V(G) = 4$$

Berdasarkan nilai kompleksitasnya, algoirtma *selection sort* masih tergolong algoritma yang efisien. Jika dibandingkan dengan *bubble sort*, algoritma ini bisa dibilang lebih efisien berdasarkan kompleksitas siklomatik. Namun, sama halnya dengan *bubble sort*, jika dilihat dari *time complexity* yaitu $O(n^2)$, algoritma ini kurang efisien untuk jumlah data yang besar.

3. *Insertion Sort*

Berdasarkan graf aliran kontrol pada gambar 3.3, terdapat 10 simpul dan 11 sisi, sehingga kompleksitas siklomatiknya adalah:

$$V(G) = E - N + 2$$
$$V(G) = 11 - 10 + 2$$
$$V(G) = 3$$

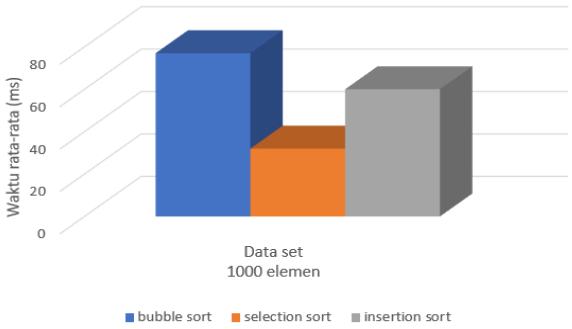
Berdasarkan nilai kompleksitasnya, algoirtma *insertion sort* masih tergolong algoritma yang efisien. Jika dibandingkan dengan algoritma *bubble sort* dan *insertion sort*, algoritma ini lebih efisien karena nilai kompleksitasnya lebih rendah. Namun, sama halnya seperti *bubble sort* dan *insertion sort*, algoritma ini masih belum cukup baik untuk data masukan yang besar.

| Algoritma | Jumlah simpul | Jumlah sisi | *Time complexity/ Cyclomatic Complexity* |
|---|---|---|---|
| *Bubble Sort* | 11 | 14 | $O(n^2)$ / 5 |
| *Selection Sort* | 13 | 15 | $O(n^2)$ / 4 |
| *Insertion Sort* | 10 | 11 | $O(n^2)$ / 3 |

**Table 3.1** Perbandingan tiga algoritma *quadratic sorting* (*bubble sort, selection sort, insertion sort*)

Berdasarkan data yang telah diperoleh, meskipun ketiga algoritma tersebut memiliki *time complexity* yang sama, jika dibandingkan lagi berdasarkan kompleksitas siklomatik, algoritma yang paling efisien dengan nilai yang paling rendah adalah *insertion sort*, yang kedua adalah *selection sort*, dan yang ketiga adalah *bubble sort*.

Jika hanya menilai dari kompleksitasnya, dari ketiga algoritma tersebut, kita akan memilih *selection sort*, karena dirasa paling efisien. Untuk perbandingan yang lebih jelas, berikut adalah visualisasi waktu yang dibutuhkan tiap algoritma, pada data masukan berupa array dengan 1000 elemen (algoritma dibandingan dalam bahasa python pada kasus terburuk).




**Gambar 3.4** perbandingan waktu pengurutan 1000 elemen algoritma *bubble sort, selection sort,* dan *insertion sort*
Sumber: Arsip Penulis

Sedikit berbeda dengan hasil analisis berdasarkan kompleksitas siklomatik, algoritma yang secara waktu lebih cepat adalah *selection sort* dan yang paling lambat dari ketiganya tetap *bubble sort.* Berdasarkan kedua hasil tersebut, ketika perbedaan nilai kompleksitasnya tidak terlalu jauh, nilai kompleksitas yang lebih kecil tidak selalu berarti bahwa algoritma tersebut adalah yang paling efisien secara waktu, tetapi pada perbedaan yang lebih besar, pengaruh kompleksitas siklomatik lebih terlihat dengan jelas.

## IV. KETERBATASAN KOMPLEKSITAS SIKLOMATIK

Nilai kompleksitas siklomatik yang rendah pada suatu potongan kode program atau algoritma tertentu tidak selalu mengindikasikan bahwa kode tersebut adalah yang paling baik dan mudah untuk dikembangkan di kemudian hari. Faktor lain yang tidak kalah penting adalah keterbacaan program, yakni kemudahan algoritma dalam program tersebut untuk dibaca dan dipahami oleh pemrogram yang bertugas mengembangkannya.

Berikut adalah perbandingan dua buah program dengan tujuan yang sama, tetapi dengan algoritma yang berbeda.

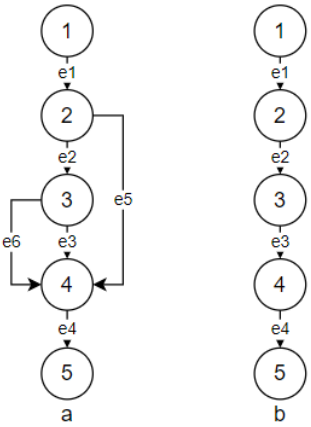
**Program 1**

```
public boolean checkWithdrawal (Account acc, int amount)
{
    if (!acc.isLocked()) && acc.getBalance() >= amount)
    {
        return true;
    }
    return false;
}
```

**Program 2**

```
public boolean checkWithdrawal (Account acc, int amount)
{
    boolean result = true;
    result &= account.getBalance()>= amount;
    result &= !account.isLocked();
    return return;
}
```

Graf aliran kontrol yang dihasilkan kedua program di atas adalah seperti pada gambar berikut.




**Gambar 4.1** graf aliran kontrol untuk (a) program 1, (b) program 2
Sumber: Arsip Penulis

Jika kompleksitas siklomatik dari kedua program tersebut dihitung, maka hasil yang didapatkan yaitu kompleksitas siklomatik program 1 adalah 3, sedangkan untuk program 2 adalah 1.

Jika hanya dilihat berdasarkan angka, program 2 lebih efisien secara penulisan kode. Akan tetapi, program 1 lebih mudah dibaca dan dipahami dengan adanya percabangan dengan tujuan yang jelas.

## V. KESIMPULAN

Analisis kompleksitas memiliki peran yang sangat penting pada pengujian perangkat lunak. Kode dengan kompleksitas yang tinggi mengindikasikan bahwa program rentan menghasilkan kesalahan yang berujung pada biaya pengembangan yang tinggi. Dengan memanfaatkan graf, pemrogram dapat merepresentasikan potongan kode yang dibuatnya ke dalam graf aliran kontrol, kemudian mengukur kompleksitas siklomatik dari graf aliran kontrol tersebut. Dengan mengetahui nilai kompleksitas siklomatik, pemrogram dengan timnya dapat mengambil keputusan untuk mencari algoritma yang lebih baik jika kompleksitasnya tinggi dan melanjutkan ke tahap produksi jika kompleksitasnya dirasa sudah paling baik.

Akan tetapi, keputusan hanya berdasarkan kompleksitas siklomatik tidak selalu berarti kode yang telah dibuat sudah paling efisien, karena perhitungan kompleksitas siklomatik masih memiliki keterbatasan. kemudahan kode untuk dibaca dan dimengerti oleh pemrogram yang bertugas mengembangkan kode juga harus diperhitungkan.

Untuk kode yang cukup kompleks, dan tidak efisien untuk menggambar graf aliran kontrol program dan menghitung secara manual, kompleksitas siklomatik dapat dihitung dengan berbagai kakas perangkat lunak yang tersedia. Beberapa kakas yang dapat digunakan yaitu:

1. OCLint – Static code analyzer for C and Related Languages
2. Reflector Add In – Code metrics for .NET assemblies
3. GMetrics – Find metrics in Java related applications

Melalui metrik sederhana seperti ini, pemrogram dapat meminimaliasasi potensi kerugian yang dapat ditimbulkan di masa mendatang ketika program sudah berada pada tahap produksi dan pengembangan.

## VI. UCAPAN TERIMA KASIH

Puji syukur kepada Tuhan Yang Maha Esa, karena atas berkat dan rahmat-Nya, penulis dapat menyelesaikan makalah ini dengan baik dan tepat waktu. Penulis juga ingin mengucapkan terimakasih yang sebesar-besarnya kepada para dosen pengampu mata kuliah IF2120 Matematika Diskrit, yaitu Dr. Ir. Rinaldi Munir, M.T., Dra. Harlili S., M.Sc., dan Dr. Nur Ulfa Maulidevi, S.T, M.Sc., yang telah memberikan banyak ilmu bermanfaat selama satu semester yang dapat membantu penulis dalam menyusun makalah ini. Terakhir, tetapi tidak kalah penting, penulis ucapkan terima kasih kepada orangtua dan teman-teman saya atas cinta, pengertian, dan dukungan selama ini.

## REFERENSI

[1] Hamilton, Thomas. 2021. Mccabe's Cyclomatic Complexity: Calculate with Flow Graph (Example), https://www.guru99.com/cyclomatic-complexity.html, diakses pada 11 Desember 2021.

[2] Patel, Pankaj, 2019, Software Engineering | Control Flow Graph (CFG), https://www.geeksforgeeks.org/software-engineering-control-flow-graph-cfg/, diaskes pada 11 Desember 2021.

[3] Islam, Syariq. 2021. Code Based Analysis of Object-Oriented Systems Using Extended Control Flow Graph, https://core.ac.uk/download/pdf/53189291.pdf, diakses pada 10 Desember 2021.

[4] Dissanayake, Pubudu. 2014. How to draw a Control flow graph & Cyclometric complexity for a given procedure, https://dzone.com/articles/how-draw-control-flow-graph, diakses pada 11 Desember 2021.

[5] Cyclomatic Complexity, https://bbau.ac.in/dept/CS/TM/Cyclomatic.pdf, diakses pada 10 Desember 2021.

[6] Cyclomatic Complexity for Developers by Jeroen Resoort, https://www.youtube.com/watch?v=JwTQywqpZ5Y&t=629s&ab_channel=Devoxx, dikses pada 9 Desember 2021.

[7] What is Cyclomatic Complexity in software development?, https://www.youtube.com/watch?v=PDYmEtBSn60&t=99s&ab_channel=Smok, diakses pada 9 Desember 2021

[8] Control Flow Graph, https://www.sciencedirect.com/topics/computer-science/control-flow-graph, diakses pada 12 Desember 2021

[9] Setiawan, Reina. 2016. Comparing Sorting Algorithm Complexity Based on Control Flow Structure, http://eprints.binus.ac.id/34958/1/Comparing%20Sorting%20Algorithm%20Complexity%20Based%20on%20Control%20Flow%20Structure.pdf, diakses pada 13 Desember 2021.

## 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, 14 Desember 2021

Jova Andres Riski Sirait
13520072