Analisis Kompleksitas Algoritma Brute Force Sederhana dengan Penggunaan Hashing SHA-1

Shafiq Irvansyah - 13522003[1]
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
[1]13522003@std.stei.itb.ac.id

***Abstract*—Makalah ini membahas analisis kompleksitas algoritma *brute force* sederhana yang diterapkan pada enkripsi hashing SHA-1. Metode brute force digunakan untuk menguji sejumlah kemungkinan secara berurutan dengan tujuan menemukan input yang menghasilkan hash yang sesuai. Penerapan teknik hashing SHA-1 bertujuan meningkatkan keamanan dan integritas data pada algoritma brute force. Analisis melibatkan evaluasi waktu eksekusi dengan fokus pada dampak penggunaan fungsi hash terhadap kompleksitas algoritma.**

***Keywords*—Brute force, analisis kompleksitas, algoritma, hashing SHA-1**

I. PENDAHULUAN

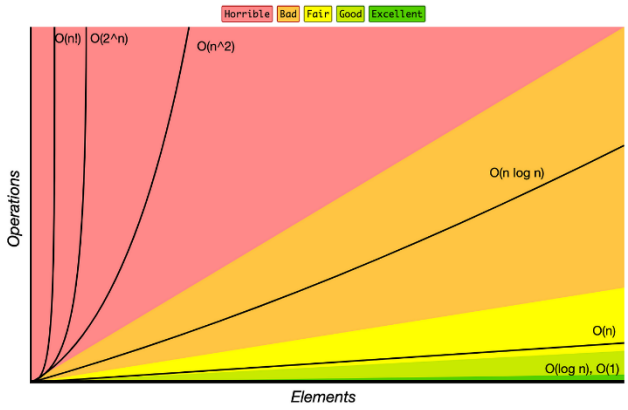
Dalam dunia komputasi, algoritma merupakan suatu langkah-langkah atau prosedur yang dijalankan oleh komputer untuk menyelesaikan suatu masalah atau tugas tertentu. Namun, tidak semua algoritma memiliki efisiensi yang sama. Beberapa algoritma mungkin memerlukan lebih banyak sumber daya atau waktu untuk menyelesaikan tugas dibandingkan dengan yang lain. Salah satu metode untuk mengevaluasi kinerja suatu algoritma adalah dengan menganalisis kompleksitasnya.

Salah satu algoritma yang umum digunakan untuk menyelesaikan suatu masalah adalah algoritma Brute Force. Meskipun sederhana, algoritma ini dapat memerlukan waktu yang signifikan terutama ketika menangani masalah dengan ukuran yang besar. Dalam konteks ini, kita akan membahas analisis kompleksitas algoritma *Brute Force* dengan penggunaan fungsi *hash* SHA-1.

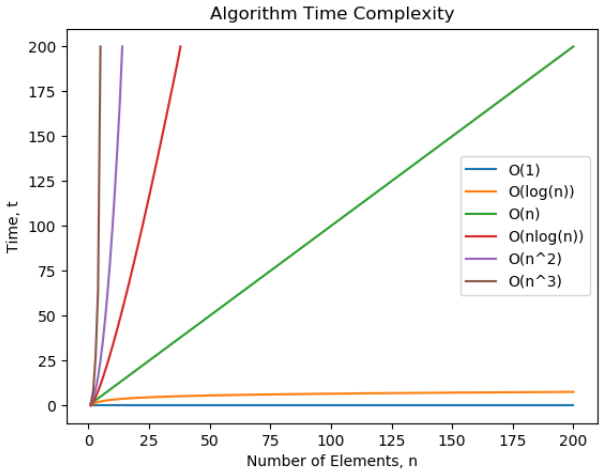
II. LANDASAN TEORI

A. Kompleksitas Algoritma

Dalam merancang suatu program, tidak cukup hanya memastikan kebenarannya. Efisiensi program juga harus dipertimbangkan untuk memastikan pemrosesan data yang optimal dalam hal waktu dan penggunaan memori. Tingkat efisiensi program sangat bergantung pada algoritma yang diterapkan. Waktu dan memori yang dibutuhkan oleh suatu program akan bergantung pada ukuran masukan (n), yang mencerminkan volume data yang akan diolah oleh algoritma. Evaluasi efisiensi dan efektivitas suatu algoritma merupakan hal yang digunakan untuk menilai keunggulan suatu solusi dibandingkan dengan algoritma lain dalam penyelesaian permasalahan yang sama. Sebab suatu permasalahan memiliki beragam pendekatan pemecahan, contohnya algoritma pengurutan (*sorting algorithm*), terdapat beberapa algoritma seperti *selection sort*, *bubble sort*, *insertion sort*, dan sebagainya.



Gambar 2.1 Diagram pertumbuhan jumlah operasi notasi Big-O.
Sumber: https://medium.com/the-legend/big-o-notation-d7def34eaed8



Gambar 2.2 Diagram pertumbuhan waktu notasi Big-O.

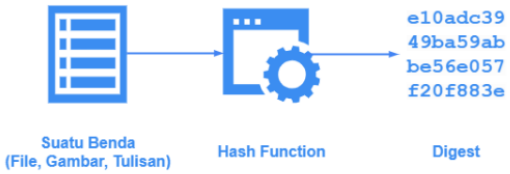
Sumber:[*https://mul14.wordpress.com/2020/07/10/mengenal-notasi-big-o/*](https://mul14.wordpress.com/2020/07/10/mengenal-notasi-big-o/)

Algoritma yang memiliki kompleksitas waktu dengan orde yang lebih kecil cenderung lebih efisien. Algoritma yang paling efisien akan memiliki kompleksitas waktu asimptotik $O(1)$, yang berarti waktu eksekusi algoritma tidak dipengaruhi oleh ukuran masukan.

B. *Brute Force*

Brute Force merupakan pendekatan yang lempang (*straightforward*) dalam menyelesaikan suatu permasalahan, di mana metodenya melibatkan percobaan semua kemungkinan secara sistematis hingga menemukan solusi yang tepat. Dalam konteks keamanan informasi, *brute force* dilakukan dengan mencoba semua kombinasi kunci atau *password* secara berurutan untuk mendapatkan akses ke sistem. Pendekatan ini terfokus pada eksplorasi semua opsi yang memungkinkan.

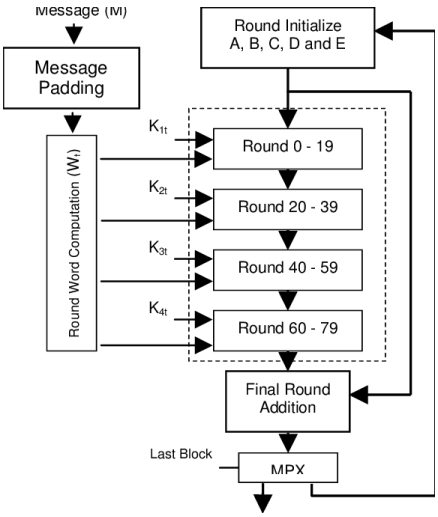
C. *Hashing*



Gambar 2.3 Diagram proses Hashing
Sumber: [*https://www.researchgate.net/figure/Secure-Hash-Algorithm-SHA-1_fig2_238180532*](https://www.researchgate.net/figure/Secure-Hash-Algorithm-SHA-1_fig2_238180532)

Hashing merupakan suatu proses yang mengubah data menjadi bentuk lain dengan ukuran tertentu. Dalam proses ini, terdapat tiga komponen krusial, yaitu data masukan, fungsi *hash*, dan nilai *hash* atau *digest*. Data masukan bisa bervariasi, mulai dari video, teks, hingga data desimal, selama masukan tersebut berbentuk data biner. Fungsi *hash* adalah suatu mekanisme yang mampu mengubah data menjadi bentuk lain dengan ukuran yang telah ditentukan. Fungsi *hash* ini bersifat satu arah, sehingga tidak memungkinkan untuk mengetahui data asli dari hasil keluarannya. Berbeda dengan enkripsi, hasil enkripsi dapat di-dekripsi dengan menggunakan kunci tertentu. *Digest*, merupakan representasi hasil dari fungsi *hash*. Pada umumnya, nilai *digest* memiliki ukuran yang tetap.

D. SHA-1 (*Secure Hash Algorithm* 1):



Gambar 2.4 *Diagram sederhana proses hashing SHA-1*
Sumber: [*https://www.researchgate.net/figure/Secure-Hash-Algorithm-SHA-1_fig2_238180532*](https://www.researchgate.net/figure/Secure-Hash-Algorithm-SHA-1_fig2_238180532)

Dalam sistem keamanan, *password* tidak disimpan dalam bentuk *string password* aslinya. Sebaliknya, password akan dienkripsi terlebih dahulu sebelum disimpan. Salah satu algoritma enkripsi yang umum digunakan pada pertengahan tahun 2000-an adalah SHA-1. Algoritma ini dirancang dan dipublikasikan oleh NSA (National Security Agency). Hasil dari proses enkripsi SHA-1 selalu terdiri dari 40 digit heksadesimal, tanpa dipengaruhi nilai dan panjang dari *input* yang diberikan. Pada saat ini, enkripsi SHA-1 dianggap tidak aman, sehingga penggunaannya sudah jarang terjadi.

III. IMPLEMENTASI

Program dikembangkan dalam Bahasa python. Pada program ini, beberapa penyederhanaan dan batasan diterapkan untuk mempermudah implementasi dan pemahaman. Metode *brute force* dari SHA-1 yang diterapkan hanya dapat mengenkripsi hash dari bilangan bulat positif dengan panjang maksimum 10 digit (maksimum = 9999999999). Artinya program ini hanya bisa mendekripsi suatu *hash* yang memiliki nilai dekripsinya antara 0 hingga 9999999999.

A. Definisi Main

Pada definisi *main*, *user* akan diminta untuk memasukkan suatu *hash* SHA-1 yang ingin didekripsi. Program dirancang sehingga akan selalu menerima input *hash* Kembali setiap telah melakukan dekripsi *hash* sebelumnya. Setelah proses dekripsi, akan ditampilkan *string* hasil dekripsi dan lama waktu untuk mendekripsi *hash* tersebut.

```python
def main():
    display()
    while True:
        target_hash = input("Masukkan nilai Hash yang akan di-decrypt (max 10 digit): \n")
        # Melakukan brute force untuk mencari input yang menghasilkan target hash
        start = time.time()
        result = brute_force(target_hash)
        end = time.time()

        if result:
            print(f"Hash ditemukan: {result}")
        else:
            print("Tidak ditemukan tebakan yang cocok. (Di luar batas pencarian))")
        print(f"Total waktu yang diperlukan: {end - start} detik\n")
```

Gambar 3.1 Fungsi Main Program
Sumber: Dokumen penulis

```python
def display():
    print("
    print("
    print("
```

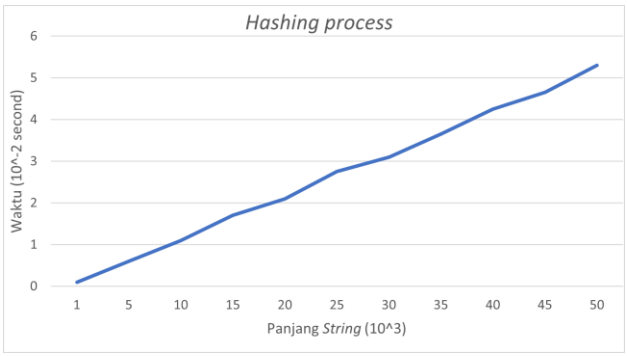
Gambar 3.2 Definisi Display Inisialisasi
Sumber: Dokumen penulis

B. *Hashing* SHA-1

```python
def sha1(message):
    # Fungsi-fungsi bantuan
    def left_rotate(n, b):
        return ((n << b) | (n >> (32 - b))) & 0xFFFFFFFF

    def process_chunk(chunk, h0, h1, h2, h3, h4):
        w = [0] * 80

        # Inisialisasi pesan pada blok 512-bit
        for i in range(16):
            w[i] = struct.unpack('>I', chunk[i * 4:i * 4 + 4])[0]

        for i in range(16, 80):
            w[i] = left_rotate(w[i - 3] ^ w[i - 8] ^ w[i - 14] ^ w[i - 16], 1)

        # Inisialisasi variabel hash
        a, b, c, d, e = h0, h1, h2, h3, h4

        # Proses ekspansi blok pesan
        for i in range(80):
            if 0 <= i <= 19:
                f = (b & c) | ((~b) & d)
                k = 0x5A827999
            elif 20 <= i <= 39:
                f = b ^ c ^ d
                k = 0x6ED9EBA1
            elif 40 <= i <= 59:
                f = (b & c) | (b & d) | (c & d)
                k = 0x8F1BBCDC
            else:
                f = b ^ c ^ d
                k = 0xCA62C1D6

            temp = (left_rotate(a, 5) + f + e + k + w[i]) & 0xFFFFFFFF
            e, d, c, b, a = d, c, left_rotate(b, 30), a, temp

        # Update variabel hash
        h0 = (h0 + a) & 0xFFFFFFFF
        h1 = (h1 + b) & 0xFFFFFFFF
        h2 = (h2 + c) & 0xFFFFFFFF
        h3 = (h3 + d) & 0xFFFFFFFF
        h4 = (h4 + e) & 0xFFFFFFFF

        return h0, h1, h2, h3, h4

    # Inisialisasi variabel hash awal
    h0 = 0x67452301
    h1 = 0xEFCDAB89
    h2 = 0x98BADCFE
    h3 = 0x10325476
    h4 = 0xC3D2E1F0

    # Padding pesan
    original_length = len(message) * 8
    message += b'\x80'
    while (len(message) * 8) % 512 != 448:
        message += b'\x00'
    message += struct.pack('>Q', original_length)

    # Proses setiap blok pesan
    for i in range(0, len(message), 64):
        h0, h1, h2, h3, h4 = process_chunk(message[i:i+64], h0, h1, h2, h3, h4)

    # Hasil akhir hash
    hash_result = '{:08x}{:08x}{:08x}{:08x}{:08x}'.format(h0, h1, h2, h3, h4)
    return hash_result
```

Gambar 3.3 Fungsi hashing SHA-1
Sumber: Dokumen penulis

Secara singkat, tahapan hashing SHA-1 dapat diringkas dalam beberapa langkah penting. Pertama, *string* masukkan diubah sesuai kriteria dengan langkah padding. Kemudian, tahap inisialisasi melibatkan pengaturan lima variabel 32-bit (A, B, C, D, dan E) menggunakan "*magic constants*". Proses perhitungan nilai hash terdiri dari ekspansi pesan, kompresi, dan pembaruan nilai hash di setiap blok 512-bit. Akhirnya, nilai hash 160-bit terbentuk dengan menggabungkan nilai-nilai A, B, C, D, dan E setelah memproses semua blok *string*.



Gambar 3.4 Grafik pertumbuhan waktu hashing SHA-1
Sumber: Dokumen penulis

Pada proses *hashing*, terdapat iterasi yang bergantung pada panjang string. Oleh karena itu, waktu eksekusi proses hashing menjadi tergantung pada panjang string masukan. Dengan demikian, notasi Big-O dari proses hashing adalah *O(n)*, di mana *n* adalah panjang string masukan.

$$T(n) = c * 80 * \frac{n}{512} * k$$

- *c* adalah konstanta yang mencakup faktor-faktor seperti operasi bitwise dan aritmatika per iterasi.
- *80* adalah jumlah iterasi per blok pesan.
- $\frac{n}{512}$ adalah jumlah blok pesan yang diolah (asumsi setiap blok berukuran 512 bit).
- *k* adalah jumlah operasi bitwise dan aritmatika per-iterasi.

dapat diserhanakan menjadi:

$$T(n) = n$$

```
Masukkan String Yang Ingin di-hash: Aku Suka Matdis
SHA-1 Hash: 26fb990317be2b52a0097666e252d789aa831b4c
Panjang Teks: 15
Total waktu yang diperlukan: 0.0 detik
```

Gambar 3.5 Contoh hasil hashing SHA-1
Sumber: Dokumen penulis

C. Dekripsi *Hash* SHA-1

```python
def brute_force(target_hash):
    # Angka-angka yang akan diuji (contoh: 1 sampai 9999999999)
    for i in range(1, 9999999999):
        guess = str(i)
        hashed_guess = sha1_hash(guess)

        # Jika hash yang dihasilkan sama dengan target_hash, maka tebakan ditemukan
        if hashed_guess == target_hash:
            return guess

    # Jika tidak ditemukan tebakan yang cocok
    return None
```

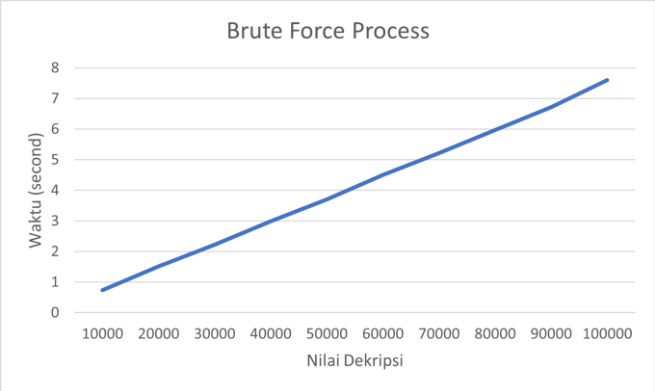
Gambar 3.6 Fungsi Brute Force
Sumber: Dokumen penulis

Proses dekripsi menggunakan algoritma SHA-1 melibatkan pendekatan *brute force*, di mana algoritma akan mengiterasi secara sekuensial dimulai dari nilai terkecil (dalam implementasi ini digunakan 0 sebagai nilai minimum) hingga menemukan hasil *hash* yang sesuai dengan nilai *digest* yang dicari. Jika dekripsi tidak ditemukan maka akan ditampilkan *output* yang sesuai.

```
SHA-1 DECRYPTOR
Masukkan nilai Hash yang akan di-decrypt (max 10 digit):
8a12a315082a345f1a9d3ad14b214cd36d310cf8
Hash ditemukan: 10000
Total waktu yang diperlukan: 0.7452073097229004 detik

Masukkan nilai Hash yang akan di-decrypt (max 10 digit):
352bc7d47decfa6b5052a0dd871ef73d6a91c7de
Hash ditemukan: 20000
Total waktu yang diperlukan: 1.4793601036071777 detik

Masukkan nilai Hash yang akan di-decrypt (max 10 digit):
a5f4fde1e3afaa49ea70ad81fe864fd20af93f8c
Hash ditemukan: 30000
Total waktu yang diperlukan: 2.2393155097961426 detik

Masukkan nilai Hash yang akan di-decrypt (max 10 digit):
437c6788c6ce0b957d61ef61f21a9ecbe5052d6a
Hash ditemukan: 40000
Total waktu yang diperlukan: 2.9684343338012695 detik

Masukkan nilai Hash yang akan di-decrypt (max 10 digit):
c2d4c5452f59cff5973dd9d08df95f8b54cad995
Hash ditemukan: 50000
Total waktu yang diperlukan: 3.7673158645629883 detik

Masukkan nilai Hash yang akan di-decrypt (max 10 digit):
a0849622401ebc624406648bc39eca144427cd90
Hash ditemukan: 60000
Total waktu yang diperlukan: 4.535325527191162 detik

Masukkan nilai Hash yang akan di-decrypt (max 10 digit):
f41f840e61e4acaed373a21c8d286614698f90ce
Hash ditemukan: 70000
Total waktu yang diperlukan: 5.272014617919922 detik

Masukkan nilai Hash yang akan di-decrypt (max 10 digit):
00bd44d0994ee0a71020743cee02292ed5c49ccb
Hash ditemukan: 80000
Total waktu yang diperlukan: 6.026076316833496 detik

Masukkan nilai Hash yang akan di-decrypt (max 10 digit):
e5530378a91b8c0a9d418c2503fece1dfbd25535
Hash ditemukan: 90000
Total waktu yang diperlukan: 6.831802845001221 detik

Masukkan nilai Hash yang akan di-decrypt (max 10 digit):
409e9519c66216726447bd4a07d6aed0475338cc
Hash ditemukan: 100000
Total waktu yang diperlukan: 7.551753520965576 detik
```

Gambar 3.7 Pengujian Dekripsi Hashing SHA-1
Sumber: Dokumen penulis

Dalam pengujian, dipilih hash dari angka ribuan karena tidak terlalu kecil sehingga perbedaan waktu antar nilai uji masih cukup signifikan, dan tidak terlalu besar sehingga tidak memerlukan waktu yang lama untuk didekripsi.



Gambar 3.8 Grafik Waktu Dekripsi *Hashing* SHA-1
Sumber: Dokumen penulis

Pada fungsi *brute force*, dapat diamati bahwa terjadi iterasi sebanyak *N* kali proses *hashing*, dengan *N* dapat diketahui secara implisit merupakan hasil dari dekripsi suatu *digest* (*hash* yang dicari). Sebelumnya, telah diidentifikasi bahwa notasi Big-O untuk tahap hashing SHA-1 adalah *O(n)*, di mana *n* adalah panjang *string* masukan. Namun, karena terdapat pembatasan pada jumlah digit dalam implementasi ini, nilai maksimum n hanya mencapai 10.

Dengan pembatasan tersebut, tiap satu kali proses *hashing* memiliki notasi Big-O sebesar O(1), yang mengindikasikan kompleksitas waktu tidak bergantung pada panjang *string* masukkan. Sebagai hasilnya, notasi Big-O dari proses dekripsi *hash* SHA-1 dengan pendekatan *brute force* dapat dianggap sebagai O(1). Sehingga notasi Big-O dari program dekripsi *hash* SHA-1 adalah O(n) dengan n adalah nilai dekripsi dari *digest*.

$$\boldsymbol{T(n) = 1 * n}$$

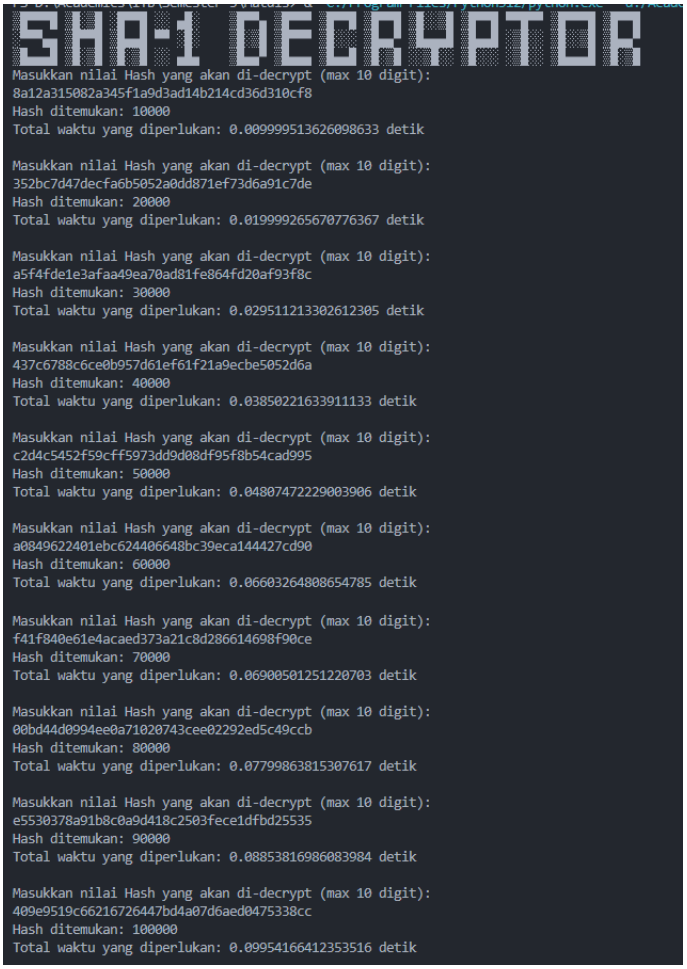
dapat disederhanakan menjadi:

$$\boldsymbol{T(n) = n}$$

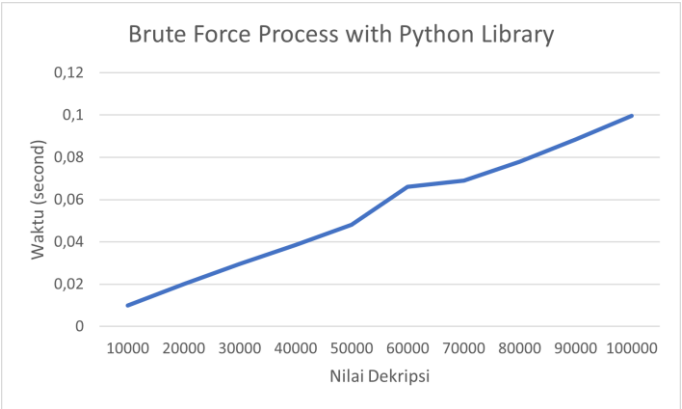
D. Dekripsi *Hash* SHA-1 dengan *Library* Python

```python
import hashlib

def sha1_hash(input_string):
    sha1 = hashlib.sha1()
    sha1.update(input_string.encode('utf-8'))
    return sha1.hexdigest()
```

Gambar 3.9 Fungsi *Hashing* SHA-1 Python *Library*
Sumber: Dokumen penulis

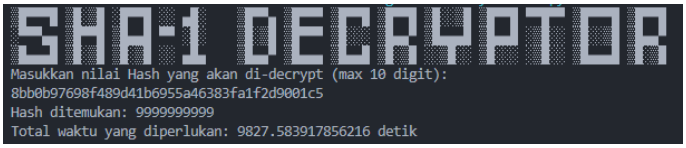


Gambar 3.9 Pengujian Dekripsi *Hashing* SHA-1 dengan *Library* Python
Sumber: Dokumen penulis



Gambar 3.10 Grafik Waktu Dekripsi dengan Library Python
Sumber: Dokumen penulis

Dilakukan juga pengujian dengan menggunaakan *hashing library* bawaan Python untuk dekripsi hashing SHA-1. Berdasarkan dari data waktu pemrosesan yang didapatkan saat pengujian, ditunjukkan bahwa kompleksitas waktu tetap sejalan dengan notasi Big-O *O(n).* Akan tetapi, terdapat perbedaan waktu yang cukup signifikan dalam prosesnya jika dibandingkan dengan fungsi hashing yang telah diimplementasikan secara manual.



Gambar 3.10 Pengujian batas Maksimum Dekripsi *Hashing* SHA-1 dengan *Library* Python
Sumber: Dokumen penulis

Dari *plotting* gambar 3.10, dapat dicari regresi linearnya ,yaitu

$$y = 9{,}925(10^{-7})x + 0{,}0001351$$

Dengan *y* adalah waktu proses dalam *second* dan *x* adalah dari *hash* yang dicari. Jika dihitung hasil hash dari 9999999999, maka akan didapatkan.

$$y = 9{,}925(10^{-7})10.000.000.000 + 0{,}0001351$$
$$y = 9925.0001351$$

Perhitungan ini menunjukkan bahwa hasil regresi dan uji coba memberikan estimasi yang sebanding, terimplikasi bahwa notasi O(n) terbukti relevan hingga nilai maksimal n yang diuji.

IV. KESIMPULAN

Kompleksitas algoritma memiliki peran yang sangat penting dalam proses perancangan program. Pentingnya mengevaluasi efisiensi dan efektivitas suatu algoritma terletak pada kemampuannya untuk menyelesaikan permasalahan dengan lebih baik dibandingkan algoritma lain yang mungkin ada untuk tugas yang sama. Visualisasi pertumbuhan jumlah operasi dan waktu menggunakan notasi Big-O memberikan gambaran yang jelas tentang seberapa efisien suatu algoritma.

Implementasi dekripsi sederhana hash SHA-1 menunjukkan bahwa kompleksitas algoritma tersebut dapat direpresentasikan dengan notasi Big-O O(n). Hal ini diperoleh melalui perhitungan jumlah operasi serta analisis pertumbuhan grafiknya terhadap waktu. Dalam mengekripsi integer kecil, program ini terbukti efektif dan efisien. Namun, ketika diperlukan dekripsi untuk hasil hashing integer besar atau string acak, implementasi ini menjadi tidak efektif karena memerlukan waktu yang sangat lama.

Dengan demikian, pemilihan algoritma dalam merancang program tidak hanya berkaitan dengan kebenaran solusi, melainkan juga sangat perlu memperhatikan efisiensi algoritma. Hal ini diperlukan untuk memastikan bahwa pemrosesan data dapat dilakukan dengan optimal, baik dari segi waktu maupun penggunaan memori.

V. UCAPAN TERIMA KASIH

Puji syukur kepada Tuhan Yang Maha Esa atas rahmat, karunia, taufik, dan hidayah-Nya, yang memungkinkan penulis menyelesaikan makalah berjudul "Analisis Kompleksitas Algoritma Brute Force dengan Penggunaan Hashing SHA-1"

sebagai pemenuhan tugas pada mata kuliah Matematika Diskrit IF2120.

Terima kasih kepada Ibu Dr. Nur Ulfa Maulidevi, S.T., M.Sc., sebagai dosen dalam mata kuliah Matematika Diskrit IF2120 Kelas K1, yang dengan penuh dedikasi telah mendidik dan mengajari kami, para mahasiswa, selama satu semester ini. Penghargaan juga disampaikan kepada semua pihak yang turut berkontribusi, baik secara langsung maupun tidak langsung, terhadap kelancaran penulisan makalah ini, meskipun namanya tidak dapat saya sebutkan satu per satu.

Penulis juga ingin menyampaikan permohonan maaf apabila terdapat kesalahan dalam penulisan makalah ini.

REFERENSI

[1] https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2023-2024/24-Kompleksitas-Algoritma-Bagian1-2023.pdf (Diakses pada 9 Desember 2023)

[2] How Does SHA-1 Work - Intro to Cryptographic Hash Functions and SHA-1 (youtube.com) (Diakses pada 9 Desember 2023)

[3] https://kutu.dev/artikel/mengenal-hashing (Diakses pada 9 Desember 2023)

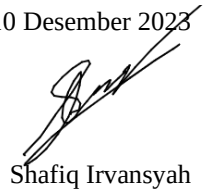
[4] https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2023-2024/25-Kompleksitas-Algoritma-Bagian2-2023.pdf. (Diakses pada 9 Desember 2023)

[5] https://experience.dropbox.com/id-id/resources/what-is-encryption (Diakses pada 9 Desember 2023)

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, 10 Desember 2023



Shafiq Irvansyah
13522003