Analisis Kompleksitas Algoritma dalam Perkalian Big Integer

Enrique Yanuar- 13522077
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
[1]13522077@std.stei.itb.ac.id

***Abstract*— Makalah ini membahas mengenai kompleksitas algoritma perkalian big integer dengan membandingkan Algoritma Tradisional, Karatsuba, dan Toom-Cook-3-way. Penelitian ini menunjukkan bahwa Algoritma Tradisional memiliki efisiensi paling rendah dengan kompleksitas waktu O(N^2). Algoritma Karatsuba dan Toom-Cook-3-way, yang menerapkan metode "divide and conquer", menunjukkan efisiensi yang lebih tinggi dengan kompleksitas waktu masing-masing $O(N^{1.585})$ dan $O(N^{1.465})$. Penelitian ini menggunakan metode analisis literatur dan eksperimen menggunakan bahasa pemrograman Python, menunjukkan Algoritma Toom-Cook-3-way adalah yang paling efektif untuk perkalian big integer, memberikan wawasan berharga bagi komunitas informatika dan membuka peluang penelitian lebih lanjut.**

***Keywords*—Perkalian Dasar, Kompleksitas Algoritma, Algoritma Karatsuba, Algoritma Toom-Cook**

I. PENDAHULUAN

Perkembangan teknologi informasi dan komputasi telah membawa kita mencapai era di mana pemrosesan data dalam jumlah besar sering dilakukan dan diharapkan dapat diproses secara cepat. Salah satu pemrosesan data yang sering dilakukan adalah operasi perkalian angka besar atau yang sering disebut sebagai big integer. Hal ini menimbulkan tantangan baru untuk meningkatkan efisiensi dan mengoptimalkan algoritma perkalian. Sejak munculnya algoritma perkalian dasar dalam aritmetika komputer, telah terjadi evolusi signifikan dalam meningkatkan efisiensi dari algoritma perkalian. Contohnya adalah Algoritma Karatsuba yang diperkenalkan pada tahun 1960, memberikan terobosan besar karena mengurangi waktu perkalian dari orde kuadratik menjadi subkuadratik[1]. Kemudian, algoritma seperti Toom-Cook dan Schönhage-Strassen mengembangkan konsep ini lebih lanjut.

Penelitian ini berfokus pada analisis dan perbandingan kompleksitas Algoritma Tradisional, Toom-Cook, dan Karatsuba dalam konteks perkalian big integer.

Tujuan Penelitian:

1. Mengidentifikasi tingkat kompleksitas algoritma dalam perkalian big integer.
2. Menentukan jenis algoritma yang paling efektif untuk perkalian big integer.

Manfaat Penelitian:

1. Memudahkan para programmer dan akademisi di bidang informatika untuk memahami kompleksitas algoritma serta metode optimasi dalam perkalian big integer.
2. Menyediakan referensi untuk penelitian berikutnya yang berhubungan dengan kompleksitas algoritma dan perkalian big integer.

Metode Penelitian:

Penelitian ini menggunakan metode studi literatur sebagai pendekatan utama dalam mengumpulkan dan menganalisis informasi. Studi ini melibatkan penelitian dari berbagai sumber seperti jurnal ilmiah, situs web, dan dokumen terkait lainnya. Pengumpulan data primer juga dilakukan melalui eksperimen yang menggunakan algoritma perkalian big integer dalam bahasa pemrograman Python.

II. DASAR TEORI

A. Kompleksitas Algoritma

Algoritma yang baik adalah algoritma yang mangkus (efficient). Kemangkusan algoritma diukur dari waktu (time) eksekusi algoritma dan kebutuhan ruang (space) memori. Algoritma yang mangkus ialah algoritma yang meminimumkan kebutuhan waktu dan ruang. Kebutuhan waktu dan ruang suatu algoritma bergantung pada ukuran masukan (n), yang menyatakan jumlah data yang diproses. Kemangkusan algoritma dapat digunakan untuk menilai algoritma yang baik dari sejumlah algoritma penyelesaian masalah. Besaran yang dipakai untuk menerangkan model abstrak pengukuran waktu/ruang ini adalah kompleksitas algoritma. Ada dua macam kompleksitas algoritma, yaitu kompleksitas waktu dan kompleksitas ruang. Kompleksitas waktu, T(n), diukur dari jumlah tahapan komputasi yang dibutuhkan untuk menjalankan algoritma sebagai fungsi dari ukuran masukan.

Kompleksitas waktu mengacu pada jumlah waktu yang diperlukan algoritma untuk menyelesaikan masalah. Kompleksitas waktu, T(n), diukur dari jumlah tahapan komputasi yang dilakukan di dalam algoritma sebagai fungsi dari ukuran masukan n [2]. Misalnya, kompleksitas waktu algoritma bisa diungkapkan dalam notasi Big O, seperti O(n), $O(n^2)$, atau O(log n), di mana 'n' adalah ukuran input. Dalam prakteknya, operasi yang dihitung hanya operasi khas yang mendasari suatu algoritma. Misalnya, operasi khas di dalam

algoritma pencarian di dalam larik adalah perbandingan elemen larik. Operasi khas algoritma pengurutan adalah perbandingan dan pertukaran elemen.
Kompleksitas ruang mengacu pada jumlah memori yang diperlukan algoritma untuk menyelesaikan masalah. Kompleksitas ruang, S(n), diukur dari memori yang digunakan oleh struktur data yang terdapat di dalam algoritma sebagai fungsi dari ukuran masukan n[2].

B. Big Integer

Big Integer merupakan penanganan bilangan yang ukurannya melebihi batasan standar tipe data integer yang diawarrkan oleh suatu bahasa pemrograman. Pada umumnya suatu bahasa pemrograman menawarkan tipe data integer yang memiliki range minus 2,147,483,648 sampai 2,147,483,647 atau long yaitu dari minus 9,223,372,036,854,775,808 sampai 9,223,372,036,854,775,807. Beberapa bahasa pemrograman menyediakan tipe data big integer sebagai bagian dari perpustakaan standar atau modul tambahan. Tipe data ini memungkinkan operasi aritmetika pada bilangan bulat yang sangat besar, yang mungkin terlalu besar untuk diwakili dengan tipe data integer biasa.

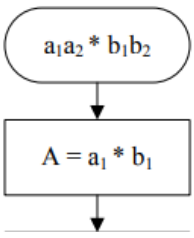
C. Algoritma Perkalian Tradisional

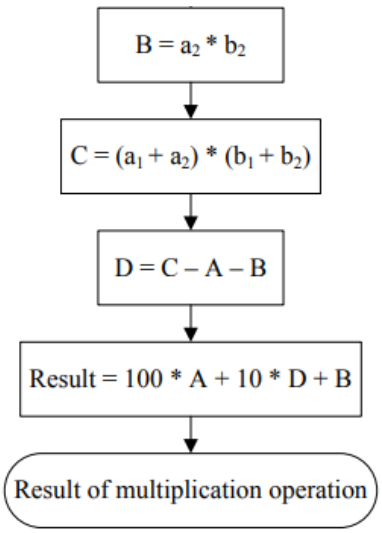
Algoritma perkalian tradisional dikenal sebagai metode perkalian yang diajarkan di sekolah dasar, merupakan salah satu metode yang paling mendasar dalam aritmatika. Proses ini melibatkan perkalian setiap digit dari kedua bilangan, kemudian penjumlahan dari semua hasil perkalian tersebut. Untuk dua bilangan dengan panjang N x M digit, proses ini akan melakukan perkalian sebanyak N X M. Ini menjadikan kompleksitas waktu algoritma menjadi $O(N^2)$.

D. Algortima Karatsuba

Algoritma Karatsuba merupakan sebuah terobosan dalam operasai perkalian big integer dengan kompleksitas waktu sebesar $O(N^{1.585})$. Algoritma ini menerapkan konsep “divide and conquer” untuk memecah perkalian bilangan besar menjadi bagian yang lebih kecil. Algoritma ini secara signifikan mengurangi jumlah perkalian digit yang diperlukan dibandingkan dengan metode tradisional.

Pada dasarnya, algoritma ini membagi setiap bilangan menjadi dua bagian yang lebih kecil. Contohnya, jika kita memiliki dua bilangan, masing-masing dapat dibagi menjadi bagian atas dan bawah. Daripada mengalikan dua bilangan secara langsung yang membutuhkan $O(N^2)$ operasi perkalian, Karatsuba melakukan beberapa perkalian dengan bilangan yang lebih kecil dan kemudian menggabungkan hasilnya untuk mendapatkan produk akhir. Proses ini mengurangi jumlah operasi perkalian secara signifikan, terutama untuk bilangan dengan banyak digit[3].

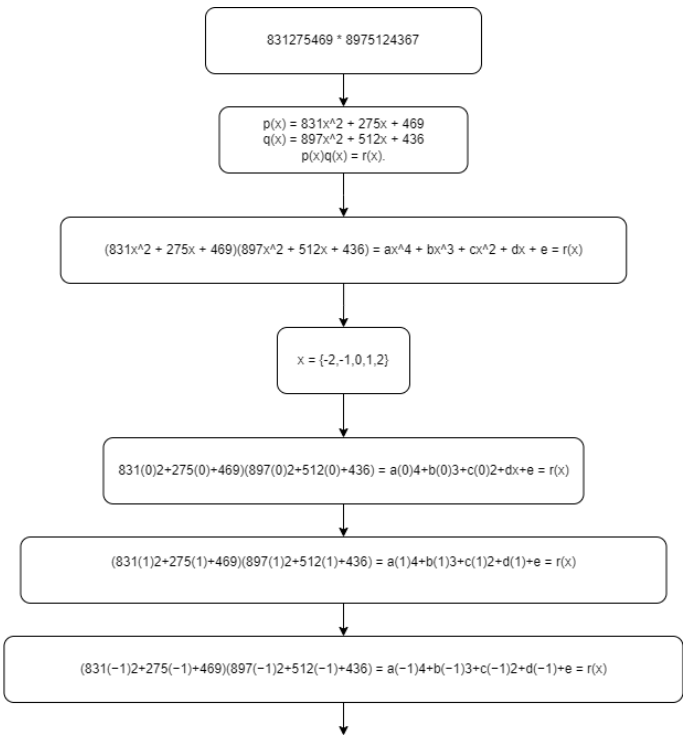


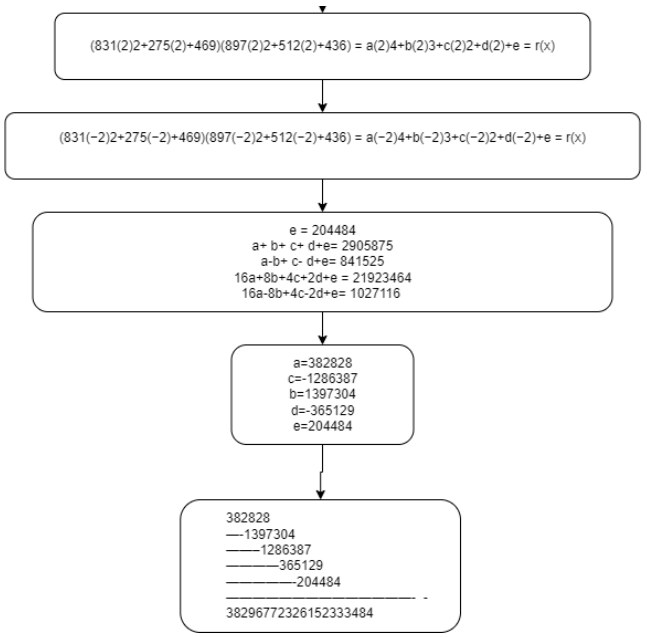


Gambar 2.1 flowchart perkalian 2 digit dari algoritma karatsuba
Sumber: Performance Analysis of Karatsuba Multiplication Algorithm for Different Bit Lengths

E. Algortima Toom-Cook-3-way

Algoritma Toom-Cook-3-way dikembangkan oleh Andrei Toom pada tahun 1963 kemudian diperluas oleh Stephen Cook. Algoritma ini membawa kemajuan yang signifikan dalam teknik perkalian big integer. Algoritma ini merupakan generalisasi dari Algoritma Karatsuba, Toom-Cook-3-way memecah bilangan yang akan dioperasikan menjadi tiga bagian, algoritma ini juga menerapkan konsep “divide and conquer” namun lebih kompleks. Toom-Cook-3-way mengurangi jumlah perkalian yang diperlukan dan meningkatkan efisiensi, terutama dalam konteks bilangan dengan ukuran sangat besar. Algoritma ini memiliki kompleksitas waktu sebesar $O(N^{1.465})$[4].





Gambar 2.2 flowchart perkalian 6 digit dari Algoritma Toom-Cook-3-way
Sumber: Dokumentasi pribadi.

III. IMPLEMENTASI DAN PEMBAHASAN

A. Algoritma Perkalian Tradisional

```python
import time
import random

CodiumAI: Options | Test this function
def multiply_digit_arrays(a, b):
    result = [0] * (2 * D)
    for i in range(D - 1, -1, -1):
        for j in range(D - 1, -1, -1):
            result[i + j + 1] += a[i] * b[j]
            result[i + j] += result[i + j + 1] // 10
            result[i + j + 1] %= 10
    while len(result) > 1 and result[0] == 0:
        result.pop(0)
    return result

res = []
for i in range(2):
    D = 3**(i+1)
    # Generating random digit arrays for a and b
    a = [random.randrange(10) for _ in range(D)]
    b = [random.randrange(10) for _ in range(D)]
    start = time.time()
    print(a)
    print(b)
    product = multiply_digit_arrays(a, b)
    print(product)
    end = time.time()
    print("The time of execution of above program is :",
        (end-start) * 10**3, "ms")
    res.append((end-start)*10**3)
print(res)
```

Sumber: Dokumentasi Pribadi

Perhitungan Kompleksitas waktu dari Algoritma Tradisional:

Kompleksitas waktu dalam inisiasi:

$T(n)=2n$

Kompleksitas waktu dalam nested ‘for’ loop:

$T(n)=n^2$

Kompleksitas waktu dalam ‘while’:

$T(n)=n^2$

Maka $T(n) = 2n + n^2 + n$

$T(n) = O(n^2)$

Program ini merupakan implementasi dari algoritma perkalian tradisional yang memiliki kompleksitas waktu $O(N^2)$ menggunakan bahasa pemrograman Python. Program ini dirancang untuk menghasilkan dua buah bilangan dengan dengan jumlah digit D, kemudian dikalikan dengan metode perkalian tradisional. Pada program ini suatu bilangan direpresentasikan dalam bentuk list dari sebuah digit sehingga dapat menangani kasus bilangan dengan jumlah digit di atas 4300.

Proses ini dilakukan dengan mengalikan setiap digit pada bilangan dengan cara diiterasi. Hasil iterasi tersebut akan tersimpan ke dalam variabel result serta dapat dipastikan bahwa setiap element dalam variabel result berkisar antara nol sampai sampai dengan 9 ($0 < x < 9$). Program ini mudah dipahami secara logika. Namun, program ini tidak efisien secara kompleksitas waktu dikarenakan memiliki kompleksitas waktu sebesar $O(N^2)$.

B. Algoritma Karatsuba

```python
import random
import sys
import traceback
import time

CodiumAI: Options | Test this class
class MultiplyKaratsuba:

    CodiumAI: Options | Test this method
    def __init__(self, a):
        self.D_MAX = a  # (power of 2)
        self.D = a  # (<= D_MAX)
        # create random number with specific length
        self.a = [random.randrange(10) for _ in range(self.D)]
        self.b = [random.randrange(10) for _ in range(self.D)]

    CodiumAI: Options | Test this method
    def compute(self):
        """ Computation of multiplication """
        for i in range(self.D_MAX - len(self.a)):
            self.a.append(0)
        for i in range(self.D_MAX - len(self.b)):
            self.b.append(0)
        z = self.__multiply_karatsuba(self.a, self.b)
        z = self.__do_carry(z)
        # self.__display(self.a, self.b, z)


    CodiumAI: Options | Test this method
    def __multiply_normal(self, a, b):
        # Normal multiplication
        a_len, b_len = len(a), len(b)
        z = [0 for _ in range(a_len + b_len)]
        for j in range(b_len):
            for i in range(a_len):
                z[j + i] += a[i] * b[j]
        return z

    CodiumAI: Options | Test this method
    def __multiply_karatsuba(self, a, b):
        # Karatsuba multiplication
        try:
            t_len = len(a)
            if t_len <= 4:
                return self.__multiply_normal(a, b)
            a_0 = a[:(t_len // 2)]
            a_1 = a[(t_len // 2):]
            b_0 = b[:(t_len // 2)]
            b_1 = b[(t_len // 2):]
            v, w = [], []
            for i in range(t_len // 2):
                v.append(a_1[i] + a_0[i])
                w.append(b_1[i] + b_0[i])
            x_1 = self.__multiply_karatsuba(a_0, b_0)
            x_2 = self.__multiply_karatsuba(a_1, b_1)
            x_3 = self.__multiply_karatsuba(v,  w)
            for i in range(t_len):
                x_3[i] -= x_1[i] + x_2[i]
            z = x_1 + x_2
            for i in range(t_len):
                z[i + t_len//2] += x_3[i]
            return z
        except Exception as e:
            raise


    CodiumAI: Options | Test this method
    def __do_carry(self, a):
        # Process of carrying
        cr = 0
        for i in range(len(a)):
            a[i] += cr
            cr = a[i] // 10
            a[i] -= cr * 10
        if cr != 0:
            print("[ OVERFLOW!! ] ", cr)
        return a
```

```python
if __name__ == '__main__':
    res = []
    for i in range (30):
        print(2**(i+1))
        obj = MultiplyKaratsuba(2**(i+1))
        start = time.time()
        obj.compute()
        end = time.time()
        print("The time of execution of above program is :",
        (end-start) * 10**3, "ms")
        res.append((end-start)*10**3)
    print(res)
```

Sumber: Dokumentasi pribadi.

Perhitungan Kompleksitas waktu dari Algoritma Karatsuba:

Kompleksitas waktu dalam persamaan rekurens:

$$T(n)=3T(n/2)+O(n)$$

Melalui persamaan rekurens bisa diperoleh:

$$T(n)=O(n^{\log_2 3})\approx O(n^{1.585})$$

Program ini merupakan implementasi dari perkalian menggunakan Algoritma Karatsuba yang memiliki kompleksitas waktu $O(N^{1.585})$ dengan menggunakan bahasa pemrograman Python. Program ini dirancang untuk menghasilkan dua bilangan dengan jumlah digit D, kemudian dikalikan menggunakan Algoritma Karatsuba. Dalam program ini, sebuah bilangan direpresentasikan dalam bentuk daftar digit sehingga dapat menangani kasus bilangan dengan jumlah digit yang lebih dari 4300.

Proses ini dilakukan secara rekursif, yaitu dengan membagi dua bilangan menjadi bagian-bagian yang lebih kecil sampai nilai kedua bilangan kurang dari 10^4, kemudian bilangan tersebut dikalikan dengan metode tradisional. Hasil dari perkalian tersebut akan digabungkan untuk membentuk nilai akhir dalam bentuk list dari sebuah digit. Algoritma ini cukup sederhana dibandingkan dengan Algoritma Toom-Cook-3-way dan cukup baik dalam menangani perkalian bilangan yang sangat besar karena kompleksitas waktu dari algoritma ini lebih baik daripada algoritma perkalian tradisional, yaitu $O(N^{1.585})$.

C. Algoritma Toom-Cook-3-way

```python
import random
import sys
import traceback
import time

CodiumAI: Options | Test this class
class MultiplyToomCook3:

    CodiumAI: Options | Test this method
    def __init__(self, a):
        self.D_MAX = a  # (power of 3)
        self.D = a  # (<= D_MAX)
        # create random number with specific length
        self.a = [random.randrange(10) for _ in range(self.D)]
        self.b = [random.randrange(10) for _ in range(self.D)]

    CodiumAI: Options | Test this method
    def compute(self):
        # Computation of multiplication

        for i in range(self.D_MAX - len(self.a)):
            self.a.append(0)
        for i in range(self.D_MAX - len(self.b)):
            self.b.append(0)
        z = self.__multiply_toom_cook_3(self.a, self.b)
        z = self.__do_carry(z)
        # self.__display(self.a, self.b, z)

    CodiumAI: Options | Test this method
    def __multiply_normal(self, a, b):
        # Normal multiplication
        a_len, b_len = len(a), len(b)
        z = [0 for _ in range(a_len + b_len)]
        for j in range(b_len):
            for i in range(a_len):
                z[j + i] += a[i] * b[j]
        return z

    CodiumAI: Options | Test this method
    def __multiply_toom_cook_3(self, a, b):
        # Toom-Cook 3-way multiplication
        a_m1, a_m2, a_0, a_1, a_inf = [], [], [], [], []
        b_m1, b_m2, b_0, b_1, b_inf = [], [], [], [], []
        c_m1, c_m2, c_0, c_1, c_inf = [], [], [], [], []
        c0, c1, c2, c3, c4          = [], [], [], [], []
        t_len = len(a)
        if t_len <= 9:
            return self.__multiply_normal(a, b)
        a0 = a[:(t_len // 3)]
        a1 = a[(t_len // 3):(t_len * 2 // 3)]
        a2 = a[(t_len * 2 // 3):]
        b0 = b[:(t_len // 3)]
        b1 = b[(t_len // 3):(t_len * 2 // 3)]
        b2 = b[(t_len * 2 // 3):]
        for i in range(t_len // 3):
            a_m2.append((a2[i] << 2) - (a1[i] << 1) + a0[i])
            b_m2.append((b2[i] << 2) - (b1[i] << 1) + b0[i])
        for i in range(t_len // 3):
            a_m1.append(a2[i] - a1[i] + a0[i])
            b_m1.append(b2[i] - b1[i] + b0[i])
        a_0, b_0 = a0, b0
        for i in range(t_len // 3):
            a_1.append(a2[i] + a1[i] + a0[i])
            b_1.append(b2[i] + b1[i] + b0[i])
        a_inf, b_inf= a2, b2
        c_m2  = self.__multiply_toom_cook_3(a_m2, b_m2)
        c_m1  = self.__multiply_toom_cook_3(a_m1, b_m1)
        c_0   = self.__multiply_toom_cook_3(a_0, b_0)
        c_1   = self.__multiply_toom_cook_3(a_1, b_1)
        c_inf = self.__multiply_toom_cook_3(a_inf, b_inf)
        c4 = c_inf
        for i in range((t_len // 3) * 2):
            c  = -c_m2[i]
            c += (c_m1[i] << 1) + c_m1[i]
            c -= (c_0[i] << 1) + c_0[i]
            c += c_1[i]
            c += (c_inf[i] << 3) + (c_inf[i] << 2)
            c  = c // 6
            c3.append(c)
        for i in range((t_len // 3) * 2):
            c  = (c_m1[i] << 1) + c_m1[i]
            c -= (c_0[i] << 2) + (c_0[i] << 1)
            c += (c_1[i] << 1) + c_1[i]
            c -= (c_inf[i] << 2) + (c_inf[i] << 1)
            c  = c // 6
            c2.append(c)
        for i in range((t_len // 3) * 2):
            c  = c_m2[i]
            c -= (c_m1[i] << 2) + (c_m1[i] << 1)
            c += (c_0[i] << 1) + c_0[i]
            c += (c_1[i] << 1)
            c -= (c_inf[i] << 3) + (c_inf[i] << 2)
            c  = c // 6
            c1.append(c)
        c0 = c_0
        z = c0 + c2 + c4
        for i in range((t_len // 3) * 2):
            z[i + t_len // 3] += c1[i]
        for i in range((t_len // 3) * 2):
            z[i + t_len] += c3[i]
        return z

    CodiumAI: Options | Test this method
    def __do_carry(self, a):
        # Process of carrying
        cr = 0
        for i in range(len(a)):
            a[i] += cr
            cr = a[i] // 10
            a[i] -= cr * 10
        if cr != 0:
            print("[ OVERFLOW!! ] ", cr)
        return a
```

```python
if __name__ == '__main__':
    res = []
    for i in range (30):
        print(3**(i+1))
        obj = MultiplyToomCook3(3**(i+1))
        start = time.time()
        obj.compute()
        end = time.time()
        print("The time of execution of above program is :",
        (end-start) * 10**3, "ms")
        res.append((end-start)*10**3)
    print(res)
```

Sumber: Dokumentasi pribadi.

Perhitungan Kompleksitas waktu dari Algoritma Toom-Cook-3-way:

Kompleksitas waktu dalam persamaan rekurens:

$$T(n)=5T(n/3)+O(n)$$

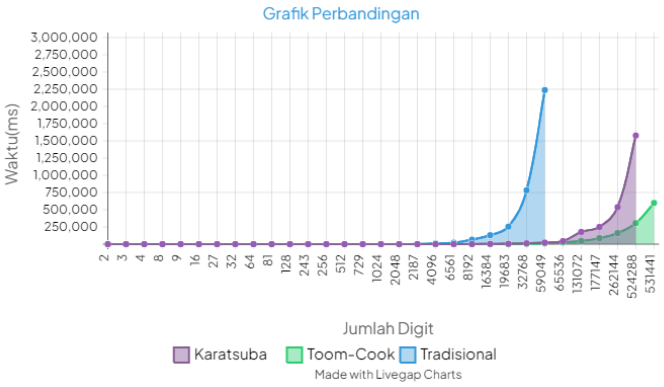
Melalui persamaan rekurens bisa diperoleh:

$$T(n)=O(n^{\log_3 5})\approx O(n^{1.465})$$

Program ini merupakan implementasi dari perkalian menggunakan Algoritma Toom-Cook-3-way yang memiliki kompleksitas waktu $O(N^{1.465})$ menggunakan bahasa pemrograman Python. Program ini dirancang untuk menghasilkan dua buah bilangan dengan dengan jumlah digit D, kemudian dikalikan dengan menggunakan Algoritma Toom-Cook-3-way. Pada program ini suatu bilangan direpresentasikan dalam bentuk list dari sebuah digit sehingga dapat menangani kasus bilangan dengan jumlah digit di atas 4300.

Proses ini dilakukan dengan cara rekursif, dasar dari algoritma ini merupakan generalisasi dari Algoritma Karatsuba namun yang membedakan adalah penggunakan konsep "divide and conquer" dalam algoritma ini lebih kompleks. Suatu bilangan disimpan dalam bentuk list dari sebuah digit kemudian list tersebut dibagi menjadi 3 bagian dan dioperasikan kedalam bentuk polinomial. Setelah mendapatkan hasil dari operasi polinomial, seluruh variabel di hitung kembali dan akan diperoleh hasil akhir berupa suatu bilangan yang direpresentasikan dalam bentuk list dari sebuah digit. Implementasi program ini merupakan yang paling kompleks jika dibandingkan dengan kedua algoritma. Dalam penanganan kasus perkalian bilangan yang sangat besar algoritma ini bekerja dengan jauh lebih efisien dibandingkan dengan Algoritma Tradisonal maupun Karatsuba.

D. Perbandingan Efisiensi Waktu



Sumber: Dokumentasi pribadi.

Grafik tersebut merupakan hasil dari pengolahan data eksekusi suatu bilangan acak dengan jumlah digit tertentu oleh algoritma tradisional, Toom-Cook, dan Karatsuba. Dari grafik tersebut terpampang data yang menunjukkan bahwa perkalian integer di bawah 6561 digit oleh ketiga algoritma tersebut masih belum menunjukkan perbedaan yang signifikan. Namun, jika dilihat lebih lanjut setelah digit tersebut, baru nampak perbedaan yang jelas terkait waktu eksekusi dari setiap algoritma. Algoritma tradisional menunjukkan nilai yang sangat tinggi dalam waktu eksekusinya ketika mengolah perkalian dengan 59.049 digit, yang mana hal ini menunjukkan jika efisiensi Algoritma Tradisional dalam perkalian big integer kurang baik. Kemudian, Algoritma Karatsuba dapat dilihat memiliki efisiensi waktu yang lebih baik dibandingkan dengan Algoritma tradisional. Yang terakhir, Algoritma Toom-Cook-3-way memiliki efisiensi yang paling baik jika dibandingkan dengan kedua algoritma lainnya. Waktu eksekusi dari algoritma ini masih belum mencapai angka yang cukup tinggi walaupun mengeksekusi perkalian 2 bilangan dengan digit setiap bilangannya 531.441. Hasil dari perbandingan ini menunjukkan bahwa kompleksitas waktu dari suatu algoritma perkalian memiliki dampak yang besar. Perbandingan ini juga menunjukkan keselarasan dengan teori kompleksitas algoritma (kompleksitas waktu) terkait nilai dari notasi Big O dengan prakteknya di dunia nyata yang mana menunjukkan Algoritma Tradisional yang memiliki kompleksitas waktu O(N^2) merupakan yang terlambat dan Algoritma Toom-Cook-3-way dengan kompleksitas waktu O(N^1.465) merupakan yang tercepat.

IV. KESIMPULAN

Dari analisis yang dilakukan, dapat disimpulkan bahwa kompleksitas waktu algoritma perkalian berbeda-beda. Algoritma Tradisional memiliki kompleksitas waktu $O(N^2)$, Algoritma Karatsuba memiliki kompleksitas waktu $O(N^{1.585})$, dan Algoritma Toom-Cook-3-way memiliki kompleksitas waktu $O(N^{1.465})$. Terlihat bahwa Algoritma Tradisional membutuhkan waktu paling lama dalam perkalian big integer, sedangkan Algoritma Toom-Cook-3-way paling cepat. Oleh karena itu, untuk perkalian big integer, Algoritma Toom-Cook-3-way adalah yang paling efektif di antara ketiganya.

V. UCAPAN TERIMA KASIH

Puji syukur kepada Tuhan Yang Maha Esa karena atas rahmat-Nya, makalah dengan judul "Analisis Kompleksitas Algoritma dalam Perkalian Big Integer" dapat diselesaikan dengan baik dan lancar. Penulis juga ingin berterima kasih kasih kepada dosen IF2120 - Matematika Diskrit K02, Dr. Fariska Zakhralativa Ruskanda, S.T., M.T., atas ilmu yang telah diajarkan selama perkuliahan sehingga penulis dapat menyelesaikan makalah ini dengan lancar. Selain itu penulis juga ingin berterima kasih kepada keluarga dan teman-teman penulis yang sudah memberi dukungan dan semangat dalam menyelesaikan makalah penelitian ini.

REFERENCES

[1] A. Karatsuba and Y. Ofman, *Multiplication of Multidigit Numbers on Automata*, vol. 7. Soviet Physics Doklady, 1962.

[2] R. Munir, *Kompleksitas Algoritma*, vol. 1, 2 vols. 2023.

[3] C. Eyupoglu, "Performance Analysis of Karatsuba Multiplication Algorithm for Different Bit Lengths," 2015.

[4] S. K. Yedugani, "Toom-Cook", [Online]. Available: https://cs.indstate.edu/~syedugani/ToomCook.pdf

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

Enrique Yanuar (13522077)