Optimasi Algoritme Perkalian Karatsuba dengan Menggunakan Metode Nikhilam II

(*Karatsuba Multiplication Algorithm Optimization by Using Nikhilam II Method*)

Felix[1]

***Abstract*—Multiplication is an essential operation in informatics engineering, for example in cryptography, cryptanalysis, and image processing fields. Researches about multiplication algorithm have been conducted and improved by experts in numerous field, ranging from mathematics, informatics engineering, to electrical engineering. The most popular multiplication algorithm is created by Anatoly Karatsuba in 1960 in Soviet Union. Although it is old and many new multiplication algorithms arise, still this algorithm is chosen for middle to large size number category. Divide and conquer technique is implemented in this algorithm to speed up the multiplication process. The weakness of Karatsuba algorithm is the excessive recursive process, causing a longer execution time. Nikhilam II method is an algorithm founded in India and is included in Vedic Mathematics. Usually, Nikhilam II method is used by common people in India to ease daily multiplication calculation. This method can replace some of the multiplication operations with addition, therefore it can be more optimum. In this paper, Nikhilam II method is implemented in the base case part of Karatsuba algorithm to reduce the recursive call. Hence, Karatsuba algortihm can be optimized from time execution point of view. As the result, this new algorithm can optimize time execution up to thrice faster than the original algorithm.**

***Intisari*—Perkalian adalah operasi penting di dalam teknik informatika, misalnya pada bidang kriptografi, kriptanalisis, dan pengolahan citra. Riset-riset mengenai algoritme perkalian terus-menerus dilakukan dan dikembangkan oleh berbagai ahli dari bidang ilmu matematika, teknik informatika, hingga teknik elektro. Algoritme perkalian yang paling populer dikembangkan oleh Anatoly Karatsuba pada tahun 1960 di Uni Soviet. Meskipun telah lama dan telah banyak algoritme perkalian baru yang bermunculan, tetapi algoritme ini masih tetap menjadi pilihan untuk kategori bilangan yang berukuran sedang hingga besar. Teknik *divide and conquer* diterapkan pada algoritme ini untuk mempercepat proses perkalian. Kelemahan dari algoritme Karatsuba adalah proses rekursif yang terlalu banyak yang menyebabkan waktu eksekusi lebih lama. Metode Nikhilam II adalah algoritme yang dikembangkan di India dan termasuk ke dalam Matematika Weda. Umumnya, Metode Nikhilam II ini digunakan oleh orang awam di India untuk memudahkan perhitungan perkalian harian. Metode ini mampu menggantikan sebagian operasi perkalian menjadi penjumlahan, sehingga bisa lebih optimal. Dalam makalah ini, metode Nikhilam II diterapkan pada bagian *base case* dari algoritme Karatsuba untuk mengurangi jumlah pemanggilan rekursif. Dengan demikian, algoritme Karatsuba dapat dioptimasi dari segi waktu eksekusi. Hasilnya, algoritme baru ini mampu mengoptimasi waktu eksekusi hingga tiga kali lebih cepat dibandingkan algoritme aslinya.**

***Kata Kunci*—Algoritme Perkalian, *Divide and Conquer*, Karatsuba, Metode Nikhilam II, Optimasi, *Recursive Call*.**

[1] *Program Studi Teknik Informatika, STMIK Mikroskil, Jl. Thamrin No. 140 Medan 20212 INDONESIA (tlp: 061-4573767; fax: 061-4567789; e-mail: felix.pandi@mikroskil.ac.id)*

I. PENDAHULUAN

Meskipun algoritme perkalian Karatsuba telah lama muncul dan banyak algoritme lainnya yang bermunculan, tetapi tetap saja algoritme Karatsuba mampu bersaing dengan algoritme terbaru lainnya untuk jumlah digit yang berukuran sedang [1]. Salah satu pertanyaan penelitian yang belum terpecahkan dalam bidang teknik informatika adalah "Apa algoritme tercepat untuk perkalian?" [2].

Riset-riset mengenai algoritme perkalian terus menerus dilakukan sejak pertengahan abad ke-20 hingga sekarang (abad ke-21) [3]-[9]. Riset yang dilakukan untuk mengkaji lebih dalam pengembangan algoritme Karatsuba juga banyak dilakukan oleh peneliti di berbagai belahan dunia, terutama peneliti dari benua Eropa, Amerika, dan Asia [1], [9]-[14]. Riset mengenai metode Nikhilam juga menjadi fokus penelitian oleh pakar-pakar matematika dan ilmu komputer, terutama peneliti dari negara India [9], [15]-[17].

Masalah pada algoritme Karatsuba adalah terjadinya proses *divide and conquer* yang terlalu panjang. Hal tersebut memicu pemanggilan fungsi rekursif yang terlalu banyak. Pada penelitian sebelumnya telah diterapkan metode Nikhilam I untuk mengurangi fungsi rekursif tersebut sehingga dapat mengurangi waktu eksekusinya [9]. Pada makalah ini, diuji kemampuan metode Nikhilam II dalam mengoptimasi algoritme Karatsuba.

Sistematika penulisan makalah ini terdiri atas bagian Pendahuluan; Metode: Algoritme Karatsuba Klasik (KK), Metode Nikhilam I (N1), Sintesis Karatsuba dan Metode Nikhilam I (KN1), Metode Nikhilam II (N2), Sintesis Karatsuba dan Metode Nikhilam II (KN2); Hasil dan Pembahasan; dan Kesimpulan.

II. METODOLOGI

Metodologi yang dilakukan meliputi empat tahap sebagai berikut.

1. Pemahaman algoritme Karatsuba dan metode Nikhilam II.
2. Sintesis algoritme Karatsuba dan metode Nikhilam II.
3. Pengujian dan pencatatan waktu eksekusi untuk perkalian dua bilangan bulat dengan jumlah digit yang bertambah.

4. Penarikan kesimpulan berdasarkan data yang diperoleh pada tahap pengujian.

A. Algoritme Karatsuba Klasik (KK)

Karatsuba Klasik (KK) adalah algoritme perkalian yang mengimplementasikan metode *divide and conquer* di dalamnya. Algoritme ini digunakan oleh perusahaan Intel untuk memproduksi hasil perkalian dalam durasi yang lebih singkat [16]. Algoritme ini juga dapat diterapkan dengan model iteratif selain dengan model rekursif yang telah sering digunakan [15]. *Pseudocode* untuk menjabarkan algoritme Karatsuba adalah sebagai berikut.

```
1  procedure Karatsuba(num1, num2)
2      /* Metode perkalian biasa pada base case */
3      if (num1 < 10) or (num2 < 10)
4          return num1 x num2
5
6      /* Hitung jumlah digit dari bilangan */
7      m = min(size_base10(num1), size_base10(num2))
8      m2 = floor(m / 2)
9
10     /* Membelah urutan angka menjadi 2 bagian */
11     high1, low1 = split_at(num1, m2)
12     high2, low2 = split_at(num2, m2)
13
14     /* Tiga pemanggilan rekursif */
15     z0 = Karatsuba(low1, low2)
16     z1 = Karatsuba((low1 + high1), (low2 + high2))
17     z2 = Karatsuba(high1, high2)
18
19     return (z2 x 10 ^ (m2 x 2)) + ((z1 – z2 – z0) x
       10 ^ m2) + z0
```

Pertama-tama algoritme ini akan meminta pengguna untuk menyediakan dua buah bilangan sebagai masukan, dengan persyaratan bilangan tersebut wajib merupakan bilangan cacah (0, 1, 2, ...). Tidak diizinkan untuk memasukkan bilangan pecahan, bilangan negatif, maupun bilangan yang mengandung koma. Lalu, dilakukan pengecekan terhadap bilangan yang dimasukkan. Jika ternyata ada bilangan yang lebih kecil dari 10, maka akan dihitung dengan perkalian umum. Namun, apabila bilangan masih lebih besar dari 10, maka program akan dilanjutkan ke baris berikutnya.

Baris nomor tujuh digunakan untuk mengetahui jumlah karakter atau digit dari kedua bilangan yang dimasukkan pada baris pertama tadi. Nilai terbesar dari kedua jumlah karakter itulah yang akan digunakan. Kemudian, nilai tersebut dibagi dua, lalu disimpan oleh m2 sebagaimana yang tertera pada baris kelima dari *pseudocode*.

Baris ke-11 dan ke-12 digunakan untuk memisahkan bilangan menjadi dua potongan yang sama besar. Contohnya, jika bilangan bernilai 8745 maka bilangan tersebut dipisah menjadi 87 dan 45, dengan 87 disimpan pada high1, sedangkan 45 pada low1.

Baris ke-15 berisi perintah z0 = Karatsuba(low1, low2). Nilai low1 dan low2 yang ada dikirimkan lagi pada *procedure* Karatsuba(num1, num2) yang tertera di baris pertama secara rekursif. Hasilnya disimpan di z0.

Pseudocode baris ke-16 dan ke-17 juga mengimplementasikan metode yang sama dengan baris ke-15. Bedanya adalah z1 = Karatsuba((low1 + high1), (low2 + high2)) dan z2 = Karatsuba(high1, high2). Perhitungan z1 dan z2 juga menerapkan cara rekursif dengan memanggil *procedure* baris pertama.

B. Metode Nikhilam I (N1)

Metode Nikhilam I (disingkat menjadi N1) menggunakan *pseudocode* seperti berikut ini.

```
1  procedure NikhilamI(num1, num2)
2      A = nearestBase – num1
3      B = nearestBase – num2
4      C = A x B
5      D = num1 – B = num2 – A
6      return nearestBase x D + C
```

Dalam metode N1 terdapat enam proses utama sebagai berikut.

1. Memasukkan dua buah angka yang hendak dilakukan operasi perkalian, misalnya a dan b.
2. Mengalkulasi A dengan rumus A = nearestBase – num1. Variabel nearestBase adalah angka perpangkatan sepuluh yang memiliki nilai yang mendekati num1. Contohnya, jika num1 = 97, maka nilai nearestBase adalah 100.
3. Proses perhitungan B menggunakan cara yang sama dengan langkah 2.
4. Nilai A dikalikan dengan B dengan cara biasa, lalu hasilnya disimpan di C.
5. Proses num1 – B akan memiliki hasil yang ekivalen dengan num2 – A. Hasil operasi pengurangan num1 – B disimpan di D.
6. Operasi Result = nearestBase * D + C menjadi hasil akhir yang didapatkan [17].

Pseudocode N1 akan digunakan nantinya untuk menggantikan *base case* pada *pseudocode* algoritme KK, yaitu baris 3 dan 4.

C. Sintesis Karatsuba dan Metode Nikhilam I (KN1)

Berikut ini adalah *pseudocode* algoritme Karatsuba yang telah dimodifikasi di bagian *base case* dengan metode Nikhilam I (baris 3-9).

```
1  procedure KaratsubaNikhilamI(num1, num2)
2      /* Metode Nikhilam I pada base case */
3      if (num1 < 100) or (num2 < 100)
4          A = 100 – num1
5          B = 100 – num2
6          C = A x B
7          D = num1 – B
8          result = 100 x D + C
9          return result
10
11     /* Hitung jumlah digit dari bilangan */
12     m = min(size_base10(num1), size_base10(num2))
13     m2 = floor(m / 2)
14
15     /* Membelah urutan angka menjadi 2 bagian */
16     high1, low1 = split_at(num1, m2)
17     high2, low2 = split_at(num2, m2)
18
19     /* Tiga pemanggilan rekursif */
20     z0 = KaratsubaNikhilamI(low1, low2)
21     z1 = KaratsubaNikhilamI((low1 + high1), (low2 +
       high2))
22     z2 = KaratsubaNikhilamI(high1, high2)
23
24     return (z2 x 10 ^ (m2 x 2)) + ((z1 – z2 – z0) x
       10 ^ m2) + z0
```

Untuk menyingkat penyebutan, algoritme ini disebut sebagai KN1 [9]. Bagian *base case* untuk KK akan menunggu hingga terdapat salah satu bilangan (num1 atau num2) yang mencapai

nilai lebih kecil dari 10 sebelum *base case* dapat dijalankan. Sementara pada KN1, nilai *base case* akan berjalan jika salah satu bilangan (num1 atau num2) telah mencapai nilai lebih kecil dari 100.

D. Metode Nikhilam II (N2)

Pseudocode metode Nikhilam II (N2) dituliskan seperti berikut ini.

```
1  procedure NikhilamII(num1, num2)
2     A = num1 - nearestBase
3     B = num2 - nearestBase
4     C = A x B
5     D = num1 + B = num2 + A
6     return nearestBase x D + C
```

Untuk metode N2 seperti yang ditunjukkan oleh *pseudocode* di atas terdapat enam proses utama, yaitu sebagai berikut.

1. Memasukkan dua buah angka yang hendak dilakukan operasi perkalian, misalnya a dan b.
2. Mengalkulasi A dengan rumus A = num1 – nearestBase. Variabel nearestBase adalah angka perpangkatan sepuluh yang memiliki nilai yang mendekati num1. Contohnya, jika num1 = 105, maka nilai nearestBase adalah 100.
3. Proses perhitungan B menggunakan cara yang sama dengan langkah 2.
4. Nilai A dikalikan dengan B dengan cara biasa, lalu hasilnya disimpan di C.
5. Proses num1 + B akan memiliki hasil yang ekivalen dengan num2 + A. Hasil operasi penjumlahan num1 + B disimpan di D.
6. Proses Result = nearestBase * D + C akan menjadi hasil akhir yang didapatkan [17].

N2 memiliki kemiripan dengan N1, yaitu sama-sama memiliki enam langkah. Perbedaan utama antara kedua metode ini adalah saat perhitungan nilai A dan nilai B. N1 akan melakukan nearestBase – num 1, sedangkan N2 akan menghitung num1 – nearestBase. Perhitungan nilai B juga memiliki proses yang serupa dengan perhitungan nilai A.

Untuk menghasilkan nilai C, baik N1 maupun N2 tidak memiliki perbedaan apapun. Perbedaan muncul kembali ketika perhitungan D, yaitu N1 melakukan operasi pengurangan, sedangkan N2 melakukan operasi penjumlahan. Pada langkah terakhir, N1 sama dengan N2, yaitu sama-sama melakukan perkalian antara D dengan nearestBase, lalu hasilnya dijumlahkan dengan nilai yang dikandung oleh variabel C. *Pseudocode* pada N2 akan digunakan nantinya untuk menggantikan *base case* pada algoritme KK, yaitu baris 3 dan baris 4.

E. Sintesis Karatsuba dan Metode Nikhilam II (KN2)

Pseudocode berikut merupakan algoritme Karatsuba yang telah dimodifikasi di bagian *base case* dengan metode Nikhilam II (baris 3-9). Untuk menyingkat penyebutan, algoritme ini akan disebut sebagai KN2.

```
1  procedure KaratsubaNikhilamII(num1, num2)
2     /* Metode Nikhilam II pada base case */
3     if (num1 < 100) or (num2 < 100)
4        A = num1 - 100
5        B = num2 - 100
6        C = A x B
7        D = num1 + B
8        result = 100 x D + C
9        return result
10
11    /* Hitung jumlah digit dari bilangan */
12    m = min(size_base10(num1), size_base10(num2))
13    m2 = floor(m / 2)
14
15    /* Membelah urutan angka menjadi 2 bagian */
16    high1, low1 = split_at(num1, m2)
17    high2, low2 = split_at(num2, m2)
18
19    /* Tiga pemanggilan rekursif */
20    z0 = KaratsubaNikhilamII(low1, low2)
21    z1 = KaratsubaNikhilamII((low1 + high1), (low2
      + high2))
22    z2 = KaratsubaNikhilamII(high1, high2)
23
24    return (z2 x 10 ^ (m2 x 2)) + ((z1 – z2 – z0) x
      10 ^ m2) + z0
```

III. HASIL DAN PEMBAHASAN

Hasil dan pembahasan secara rinci meliputi hal-hal sebagai berikut.

1. KK: waktu eksekusi algoritme perkalian Karatsuba Klasik,
2. KN1: waktu eksekusi algoritme sintesis Karatsuba dan metode Nikhilam I, dan
3. KN2: waktu eksekusi algoritme sintesis Karatsuba dan Nikhilam II.

Bilangan bulat positif digunakan pada penelitian ini. Pengujian dilakukan dengan tiga kasus untuk masing-masing:

1. 2.000 Digit: 2.000 digit x 2.000 digit,
2. 4.000 Digit: 4.000 digit x 4.000 digit, dan
3. 6.000 Digit: 6.000 digit x 6.000 digit

dengan kasus yang dimaksudkan adalah sebagai berikut.

1. Kasus A: 111...111 x 111...111.
2. Kasus B: 999...999 x 999...999.
3. Kasus C: 123456789123... x 123456789123...
4. Kasus D: 665922607317... x 665922607317...

Total pengujian dilakukan sebanyak 36 kali dengan rincian sesuai Tabel I. Semua pengujian yang ada di Tabel I diterapkan ke dalam program komputer dengan menggunakan Python 3.7 sebagai bahasa pemrograman pilihan dengan aplikasi PyScripter. Komputer yang digunakan adalah laptop Asus A456U dengan sistem operasi Microsoft Windows 10. Setiap algoritme diuji sebanyak dua belas kali dan dicatat durasinya dalam detik pada tabel dan disajikan dalam bentuk grafik batang.

Pada penelitian ini dilakukan dua belas pengujian khusus untuk algoritme KK. Durasi yang diperoleh disajikan pada Tabel II. Berdasarkan data pada Tabel II, terlihat bahwa durasi waktu yang dibutuhkan untuk mengalikan dua bilangan berdigit 2.000 adalah 0,505–0,723 detik. Nilai ini bertambah kurang lebih tiga kali lipat ketika jumlah digit bertambah dua kali lipat. Ketika jumlah digit bertambah tiga kali lipat, nilai durasi bertambah kurang lebih enam kali lipat.

Gbr. 1 menunjukkan hasil pencatatan yang diperoleh dari Tabel II. Dapat terlihat bahwa kasus B membutuhkan waktu eksekusi yang lebih lama dibandingkan kasus A, C, dan D untuk jumlah digit mana pun.

TABEL I
PENGUJIAN

No	Algoritme	Digit	Kasus
1	KK	2.000	A
2	KK	2.000	B
3	KK	2.000	C
4	KK	2.000	D
5	KK	4.000	A
6	KK	4.000	B
7	KK	4.000	C
8	KK	4.000	D
9	KK	6.000	A
10	KK	6.000	B
11	KK	6.000	C
12	KK	6.000	D
13	KN1	2.000	A
14	KN1	2.000	B
15	KN1	2.000	C
16	KN1	2.000	D
17	KN1	4.000	A
18	KN1	4.000	B
19	KN1	4.000	C
20	KN1	4.000	D
21	KN1	6.000	A
22	KN1	6.000	B
23	KN1	6.000	C
24	KN1	6.000	D
25	KN2	2.000	A
26	KN2	2.000	B
27	KN2	2.000	C
28	KN2	2.000	D
29	KN2	4.000	A
30	KN2	4.000	B
31	KN2	4.000	C
32	KN2	4.000	D
33	KN2	6.000	A
34	KN2	6.000	B
35	KN2	6.000	C
36	KN2	6.000	D

Setelah selesai percobaan dengan algoritme KK, berikutnya dilakukan percobaan dengan algoritme KN1 dengan jumlah percobaan juga sebanyak dua belas kali, seperti yang ditunjukkan pada Tabel III.

Berdasarkan data pada Tabel III, terlihat bahwa durasi waktu yang dibutuhkan untuk mengalikan dua bilangan berdigit 2.000 adalah 0,187–0,280 detik. Nilai ini bertambah kurang lebih tiga kali lipat ketika jumlah digit bertambah dua kali lipat. Ketika jumlah digit bertambah tiga kali lipat, nilai durasi bertambah kurang lebih enam kali lipat.

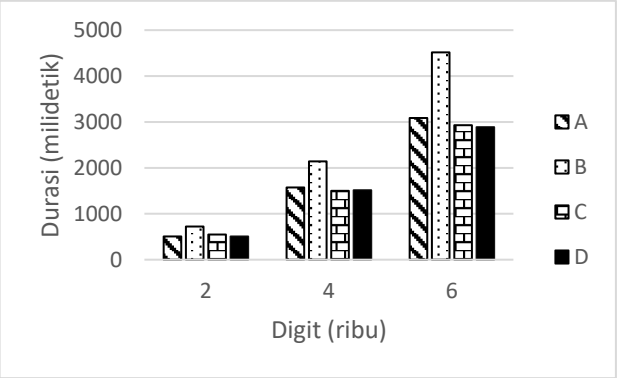
Gbr. 2 menunjukkan hasil pencatatan yang diperoleh dari Tabel III. Terlihat bahwa kasus B membutuhkan waktu eksekusi yang lebih lama dibandingkan kasus A, C, dan D

TABEL II
HASIL KK

No	Algoritme	Digit	Kasus	Durasi (detik)
1	KK	2.000	A	0,505
2	KK	2.000	B	0,723
3	KK	2.000	C	0,548
4	KK	2.000	D	0,506
5	KK	4.000	A	1,572
6	KK	4.000	B	2,140
7	KK	4.000	C	1,499
8	KK	4.000	D	1,514
9	KK	6.000	A	3,087
10	KK	6.000	B	4,516
11	KK	6.000	C	2,930
12	KK	6.000	D	2,889

TABEL III
HASIL KN1

No	Algoritme	Digit	Kasus	Durasi (detik)
1	KN1	2.000	A	0,187
2	KN1	2.000	B	0,280
3	KN1	2.000	C	0,192
4	KN1	2.000	D	0,198
5	KN1	4.000	A	0,569
6	KN1	4.000	B	0,700
7	KN1	4.000	C	0,565
8	KN1	4.000	D	0,565
9	KN1	6.000	A	1,072
10	KN1	6.000	B	1,700
11	KN1	6.000	C	1,095
12	KN1	6.000	D	1,184



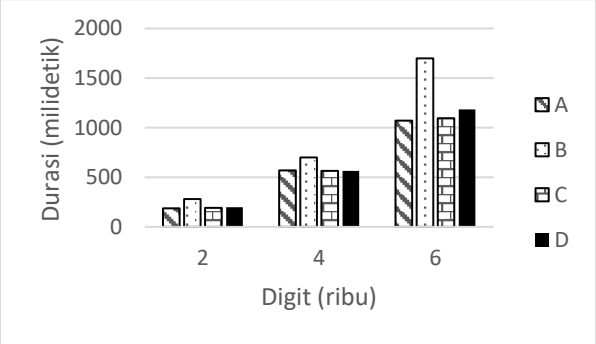
Gbr. 1 Grafik KK.

untuk jumlah digit mana pun. Hal ini juga serupa dengan grafik KK pada Gbr. 1.

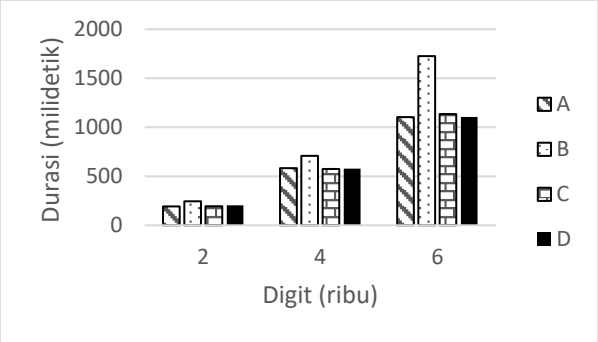
Berdasarkan data pada Tabel IV, tampak bahwa durasi waktu yang dibutuhkan untuk mengalikan dua bilangan berdigit 2.000 adalah 0,192–0,245 detik. Nilai ini bertambah kurang lebih tiga kali lipat ketika jumlah digit bertambah dua kali lipat. Ketika jumlah digit bertambah tiga kali lipat, nilai durasi bertambah kurang lebih enam kali lipat.

TABEL IV
HASIL KN2

No	Algoritme	Digit	Kasus	Durasi (detik)
1	KN2	2.000	A	0,192
2	KN2	2.000	B	0,245
3	KN2	2.000	C	0,194
4	KN2	2.000	D	0,204
5	KN2	4.000	A	0,583
6	KN2	4.000	B	0,708
7	KN2	4.000	C	0,574
8	KN2	4.000	D	0,578
9	KN2	6.000	A	1,104
10	KN2	6.000	B	1,725
11	KN2	6.000	C	1,135
12	KN2	6.000	D	1,105



Gbr. 2 Grafik KN1.



Gbr. 3 Grafik KN2.

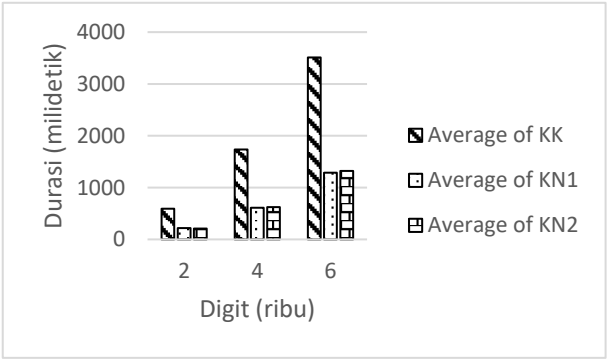
Sesuai hasil yang diperoleh pada Tabel IV, dibuat grafik yang diperlihatkan pada Gbr. 3. Sumbu x grafik ini adalah digit angka yang dikalikan dalam ribuan, sedangkan sumbu y adalah durasi dalam milidetik yang dibutuhkan untuk mengalkulasi perkalian kedua bilangan bulat tersebut.

Pada Tabel V terlihat bahwa untuk kategori 2.000 digit, rata-rata terbaik diperoleh algoritme KN2 dengan selisih 0,005 detik. Untuk kategori 4.000 dan 6.000 digit, KN1 mengungguli KN2 dengan selisih tipis, yaitu sebesar 0,011 detik dan 0,004 detik. Algoritme KK tetap merupakan algoritme dengan durasi terlama pada setiap kategori.

Sesuai hasil yang diperoleh pada Tabel V, dibuat grafik yang ditunjukkan pada Gbr. 4. Sumbu x merupakan digit angka yang dikalikan dalam ribuan, sedangkan sumbu y adalah durasi dalam milidetik yang dibutuhkan untuk mengalkulasi perkalian kedua bilangan bulat tersebut.

TABEL V
PERBANDINGAN TIGA ALGORITME

Digit	Rata-Rata Durasi (detik)		
	KK	KN1	KN2
2.000	0,570	0,214	0,209
4.000	1,681	0,600	0,611
6.000	3,356	1,263	1,267



Gbr. 4 Grafik perbandingan ketiga algoritme.

IV. KESIMPULAN

Dari hasil penelitian dan pembahasan yang telah dilakukan, dapat disimpulkan bahwa KN2 memiliki keunggulan dibandingkan KK. Algoritme KN2 yang baru ini mampu mengoptimasi waktu eksekusi hingga tiga kali lebih cepat dibandingkan algoritme aslinya (KK).

UCAPAN TERIMA KASIH

Ucapan terima kasih disampaikan kepada STMIK Mikroskil yang telah memberikan bantuan hibah pada penelitian ini melalui skema Hibah Kompetisi Internal Kreativitas dan Inovasi Dosen Semester Ganjil TA. 2019/2020 STMIK - STIE Mikroskil.

REFERENSI

[1] T. Baruchel, "Flattening Karatsuba's Recursion Tree into a Single Summation," *SN Computer Science*, Vol. 1, No. 48, hal 1–9, Nov. 2020.

[2] Ç.K. Koç, *Open Problems in Mathematics and Computational Science*, 1st ed., Cham, Switzerland: Springer International Publishing, 2014.

[3] A. Karatsuba dan Y. Ofman, "Multiplication of Multidigit Numbers on Automata," *English Translation in Soviet Physics Doklady*, Vol. 7, hal. 595–596, Jan. 1963.

[4] S.A. Cook dan S.O. Aanderaa, "On the Minimum Computation Time of Functions," *Transactions of the American Mathematical Society*, Vol. 142, hal. 291–314, Agu. 1969.

[5] A. Schönhage dan V. Strassen, "Schnelle Multiplikation großer Zahlen," *Computing*, Vol. 7, No. 3-4, hal. 281–292, Sep. 1971.

[6] M. Fürer, "Faster Integer Multiplication," *SIAM Journal on Computing*, Vol. 39, No. 3, hal. 979–1005, Sep. 2009.

[7] D. Harvey, J.V.D. Hoeven, dan G. Lecerf, "Even Faster Integer Multiplication," *Journal of Complexity*, Vol. 36, hal. 1–30, Okt. 2016.

[8] S. Covanov dan E. Thomé, "Fast Arithmetic for Faster Integer Multiplication," *HAL-Inria preprints*, hal. 1-8, Jan. 2015.

[9] Felix, "Analisis Waktu Eksekusi Algoritma Perkalian Karatsuba dan Nikhilam," *JSM (Jurnal SIFO Mikroskil)*, Vol. 17, No. 2, hal. 153–162, Okt. 2016.

[10] T. Jebelean, "Using the Parallel Karatsuba Algorithm for Long Integer Multiplication and Division," *Euro-Par '97: Proceedings of the Third International Euro-Par Conference on Parallel Processing*, 1997, hal. 1169–1172.

[11] F.O. Ehtiba and A. Samsudin, "Multiplication and Exponentiation of Big Integers with Hybrid Montgomery and Distributed Karatsuba Algorithm," *Proceedings. 2004 International Conference on Information and Communication Technologies: From Theory to Applications*, 2004, hal. 421–422.

[12] F. Tsang, "A Comparison of Traditional, Karatsuba and Fourier Big Integer Multiplication," B.Sc. dissertation, University of Bath, Somerset, United Kingdom, Mei 2004.

[13] J.B. Lima, D. Panario, dan Q. Wang, "A Karatsuba-Based Algorithm for Polynomial Multiplication in Chebyshev Form," *IEEE Transactions on Computers*, Vol. 59, No. 6, hal. 835–841, Jun. 2010.

[14] M. Bodrato dan A. Zanoni, "Karatsuba and Toom-Cook Methods for Multivariate Polynomials," *Acta Universitatis Apulensis Special Issue ICTAMI 2011*, hal. 11–60, Jul. 2011.

[15] B.K. Tirthaji, *Vedic Mathematics*, Delhi, India: Motilal Banarsidass Publishers, 1965.

[16] S.P. Dwivedi, "An Efficient Multiplication Algorithm using Nikhilam Method," *Fifth International Conference on Advances in Recent Technologies in Communication and Computing (ARTCom 2013)*, 2013, hal. 223–228.

[17] C. Eyupoglu, "Investigation of the Performance of Nikhilam Multiplication Algorithm," *Procedia - Social and Behavioral Sciences*, Vol. 195, hal. 1959–1965, Jul. 2015.