Analisis Pengaruh Base Case pada Algoritma Karatsuba terhadap Waktu Eksekusi

Felix[1], Syanti Irviantina[2]
STMIK Mikroskil, Jl. Thamrin No. 112, 124, 140, Telp. (061) 4573767, Fax. (061) 4567789
Program Studi Teknik Informatika, STMIK Mikroskil, Medan
[1]felix.pandi@mikroskil.ac.id, [2]syanti@mikroskil.ac.id,

Abstrak

Algoritma Karatsuba adalah algoritma perkalian yang masih banyak diteliti oleh peneliti Ilmu Komputer dan Matematika meskipun telah berusia lebih dari setengah abad. Algoritma ini merupakan algoritma yang menerapkan konsep Divide & Conquer. Oleh karena itu terdapat nilai base case (BC) yang dapat diganti. Hipotesisnya adalah nilai BC ketika ditingkatkan akan mengurangi waktu eksekusi sampai mencapai suatu nilai x. Setelah melewati nilai x, waktu eksekusi akan bertambah. Pada penelitian ini dilakukan dengan 36 percobaan dengan kombinasi dari 3 pilihan digit, 3 kasus, dan 4 BC. Pilihan digitnya adalah 2000, 4000, dan 6000 digit. Kasus yang digunakan adalah angka acak yang dikalikan dengan angka acak itu sendiri. BC yang digunakan adalah $10^{10^{\wedge}n}$ dengan nilai n = {0, 1, 2, 3}.Penelitian ini menghasilkan fakta yang berlawanan dengan hipotesis. Semakin besar BC membutuhkan waktu eksekusi yang lebih singkat. Hal ini diduga karena di dalam Python sendiri sudah menerapkan algoritma Karatsuba secara implisit.

Kata kunci*— base case, Karatsuba, waktu eksekusi*

Abstract

Karatsuba algorithm is a multiplication algorithm which is still frequently researched by Computer Science and Mathematics researcher although it is half a century old. This algorithm is an algorithm which implements Divide & Conquer concept. Therefore it exists a base case (BC) value that can be replaced. The hypothesis is that the increment of BC value will reduce execution time until it reaches value x. After it surpasses value x, the execution time will increase. This research is conducted with 36 experiments using the combination of 3 digit choices, 3 cases, and 4 BC. The digit choices are 2000, 4000, and 6000 digit. The case used is random number multiplies with the random number itself. The BC used is $10^{10^{\wedge}n}$ with the value of n = {0, 1, 2, 3}. The research produces fact which controverts the hypothesis. As BC grows, less execution time is needed. This is presumably because inside the Python itself has implemented Karatsuba algorithm implicitly.

Keywords*— base case, execution time, Karatsuba*

1. PENDAHULUAN

Algoritma Karatsuba adalah algoritma yang dirancang oleh Anatoly Karatsuba pada 1960. Algoritma ini dikenal sebagai algoritma perkalian tertua yang lebih cepat dibandingkan dengan algoritma perkalian tingkat SD. Hingga kini memang telah banyak algoritma perkalian yang dirancang oleh ilmuwan-ilmuwan yang mampu mengalahkan kecepatan algoritma Karatsuba untuk kategori bilangan yang sangat besar, namun untuk kategori bilangan yang sedang masih tetap digunakan algoritma Karatsuba.

Algoritma Karatsuba masih terus menerus digunakan dalam berbagai riset yang dilakukan. Hal ini karena metode yang digunakan di algoritma ini terus menerus menginspirasi dan memberikan ide-ide baru dalam pengembangan beragam ilmu pengetahuan. Berikut ini beberapa publikasi terkini yang menjadi algoritma Karatsuba menjadi pusat atau bagian dari objek penelitian para pakar:

- *Flattening Karatsuba's Recursion Tree into a Single Summation* [1]

- *Speeding up the Karatsuba algorithm* [2]
- *Searching for Best Karatsuba Recurrences* [3]
- *On the Complexity of Hybrid n-Term Karatsuba Multiplier for Trinomials* [4]
- *Decoupling for moment manifolds associated to Arkhipov–Chubarikov–Karatsuba systems* [5]

Perhatian utama pada penelitian ini adalah pada bagian *base case* dari algoritma Karatsuba. *Base case* adalah tingkatan paling rendah pada proses *divide and conquer* dan *recursive call*. Penelitian ini berfokus pada pencarian nilai *base case* yang dapat menghasilkan waktu eksekusi terendah dengan melakukan beragam perkalian bilangan bulat yang berbeda untuk mengetahui jika terdapat konvergensi.

Nilai *base case* pada algoritma Karatsuba dapat dikonfigurasi untuk menghasilkan waktu eksekusi yang lebih cepat namun belum diketahui dengan pasti berapa nilai *base case* yang sesuai dan apakah konvergen pada setiap kasus. Hipotesis penelitian ini adalah terdapat suatu nilai *base case* yang dapat mempercepat waktu eksekusi tanpa harus melakukan *recursive call* hingga mencapai nilai yang lebih kecil dari sepuluh.

2. METODE PENELITIAN

Algoritma Karatsuba merupakan algoritma pertama yang menerapkan konsep *Divide and Conquer*. Algoritma Karatsuba digunakan oleh Intel dan perusahaan lainnya untuk menghasilkan perkalian yang lebih cepat karena membutuhkan jumlah langkah yang lebih sedikit [5]. Selain itu, algoritma ini bisa diimplementasikan secara rekursif maupun iteratif [6]. Adapun salah satu versi algoritma Karatsuba dapat dilihat pada Gambar 1 berikut ini.

```
 1: procedure karatsuba(num1, num2)
 2:   if (num1 < 10) or (num2 < 10)
 3:     return num1*num2
 4:   m = max(size_base10(num1), size_base10(num2))
 5:   m2 = m/2
 6:   high1, low1 = split_at(num1, m2)
 7:   high2, low2 = split_at(num2, m2)
 8:   z0 = karatsuba(low1,low2)
 9:   z1 = karatsuba((low1+high1),(low2+high2))
10:   z2 = karatsuba(high1,high2)
11:   return (z2*10^(2*m2))+((z1-z2-z0)*10^(m2))+(z0)
```

Gambar 1 Algoritma Perkalian Karatsuba

Dari Gambar 1 dapat dilihat *pseudocode* untuk Algoritma Perkalian Karatsuba. Baris pertama akan menerima input dua buah bilangan. Bilangan yang dapat diterima adalah bilangan bulat positif dengan tipe data *integer, long integer, big inte*ger dan sejenisnya. Bilangan negatif, pecahan, berkoma tidak diterima.

Selanjutnya pada baris kedua dan ketiga, kedua bilangan tersebut akan diperiksa apakah salah satunya sudah lebih kecil dari 10. Bila sudah, maka akan dilakukan perkalian antara bilangan pertama dan bilangan kedua dengan cara yang standar. Bila belum, maka bari keempat hingga baris kesepuluh akan dijalankan.

Baris keempat berfungsi untuk menentukan jumlah digit dari bilangan pertama dan kedua berdasarkan basis bilangan desimal. Jumlah digit terbesar yang diambil. Setelah itu dibagi dua dan disimpan di variabel m_2 sesuai dengan baris kelima dari algoritma tersebut.

Baris keenam dan ketujuh memiliki kemiripan, bedanya adalah pada baris keenam, operasi pemisahan dilakukan pada bilangan pertama sedangkan pada baris ketujuh dilakukan pada bilangan kedua. Misalnya bilangan pertama adalah 9753 yang ditentukan secara acak, maka bilangan tersebut akan dipisahkan menjadi 97 dan 53. Nilai 97 disimpan oleh variabel $high_1$ dan nilai 53 akan disimpan oleh variabel low_1. Hal yang sama dilakukan juga pada bilangan kedua.

Pada baris kedelapan adalah z_0 = karatsuba(low_1, low_2). Nilai low_1 dan low_2 yang diperoleh akan dimasukkan kembali pada *procedure* karatsuba(num_1, num_2) pada baris pertama secara rekursif.

Hasilnya akan disimpan di variabel z_0. *Pseudocode* baris kesembilan dan kesepuluh juga menerapkan cara yang sama dengan baris kedelapan. Bedanya adalah z_1 = karatsuba((low_1 + $high_1$), (low_2 + $high_2$)) dan z_2 = karatsuba($high_1$, $high_2$). Perhitungan z_1 dan z_2 juga menerapkan cara rekursif dengan memanggil *procedure* baris pertama.

Metode Penelitian yang dilakukan meliputi 3 tahap yaitu:

1. Tahap Pemahaman

Memahami karakteristik dan proses kerja dari algoritma Karatsuba

2. Tahap Pengujian

Melakukan pengujian dan mencatat waktu eksekusi untuk perkalian 2 bilangan bulat dengan jumlah digit yang bertambah dan *base case* yang berbeda

3. Tahap Penarikan Kesimpulan

Menarik kesimpulan berdasarkan data yang diperoleh pada tahap pengujian

3. HASIL DAN PEMBAHASAN

Bilangan yang digunakan pada penelitian ini adalah bilangan bulat positif.
Pengujian akan dilakukan dengan 3 kasus untuk masing-masing:

1. 2000 Digit: 2000 digit x 2000 digit
2. 4000 Digit: 4000 digit x 4000 digit
3. 6000 Digit: 6000 digit x 6000 digit

Kasus yang dimaksudkan adalah:

1. Kasus A: 8752377125458782697212... x 8752377125458782697212...
2. Kasus B: 6659226073176068766697... x 6659226073176068766697...
3. Kasus C: 1593750112503027419434... x 1593750112503027419434...

Nilai batas *base case* yang digunakan adalah:

1. 10^(10^0) = 10
2. 10^(10^1) = 10.000.000.000
3. 10^(10^2)
4. 10^(10^3)

Total ada 36 percobaan yaitu:

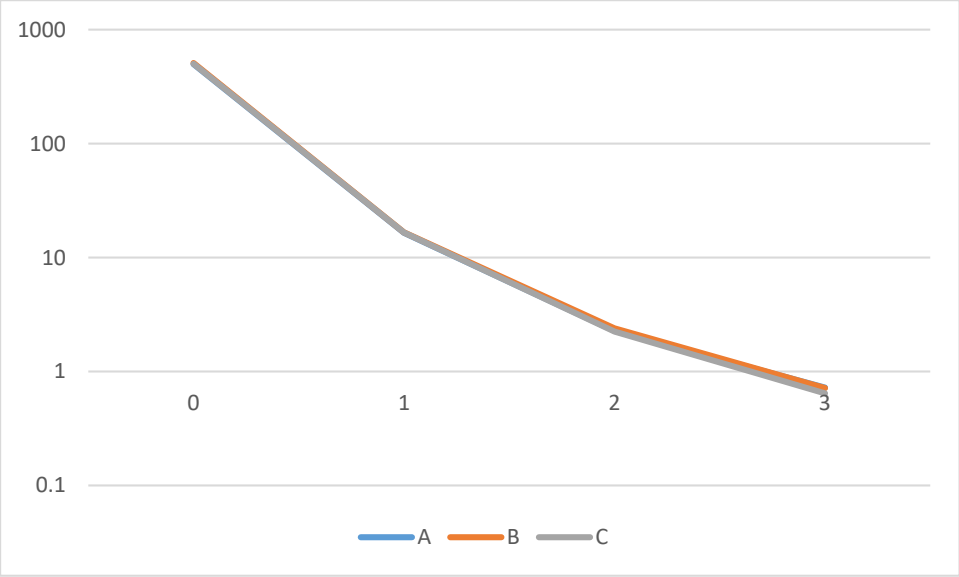
Tabel 1 Tabel Percobaan

No	Kode Eksperimen	Digit (d * 1000)	Kasus	Base (10^(10^b))	Durasi (detik)
1	2A0	2	A	0	0,4980337
2	2B0	2	B	0	0,5095401
3	2C0	2	C	0	0,5013165
4	2A1	2	A	1	0,0164685
5	2B1	2	B	1	0,0166417
6	2C1	2	C	1	0,0165820
7	2A2	2	A	2	0,0022699
8	2B2	2	B	2	0,0023842
9	2C2	2	C	2	0,0022391
10	2A3	2	A	3	0,0007219
11	2B3	2	B	3	0,0007147
12	2C3	2	C	3	0,0006417
13	4A0	4	A	0	1,5011940
14	4B0	4	B	0	1,5570072
15	4C0	4	C	0	1,4948452
16	4A1	4	A	1	0,0561234

No	Kode Eksperimen	Digit (d * 1000)	Kasus	Base (10^(10^b))	Durasi (detik)
17	4B1	4	B	1	0,0506074
18	4C1	4	C	1	0,0529315
19	4A2	4	A	2	0,0073024
20	4B2	4	B	2	0,0076638
21	4C2	4	C	2	0,0076992
22	4A3	4	A	3	0,0031765
23	4B3	4	B	3	0,0032943
24	4C3	4	C	3	0,0026598
25	6A0	6	A	0	2,8982437
26	6B0	6	B	0	2,9401181
27	6C0	6	C	0	2,9251600
28	6A1	6	A	1	0,1389996
29	6B1	6	B	1	0,1383281
30	6C1	6	C	1	0,1367912
31	6A2	6	A	2	0,0119087
32	6B2	6	B	2	0,0139891
33	6C2	6	C	2	0,0119232
34	6A3	6	A	3	0,0066507
35	6B3	6	B	3	0,0067900
36	6C3	6	C	3	0,0066150

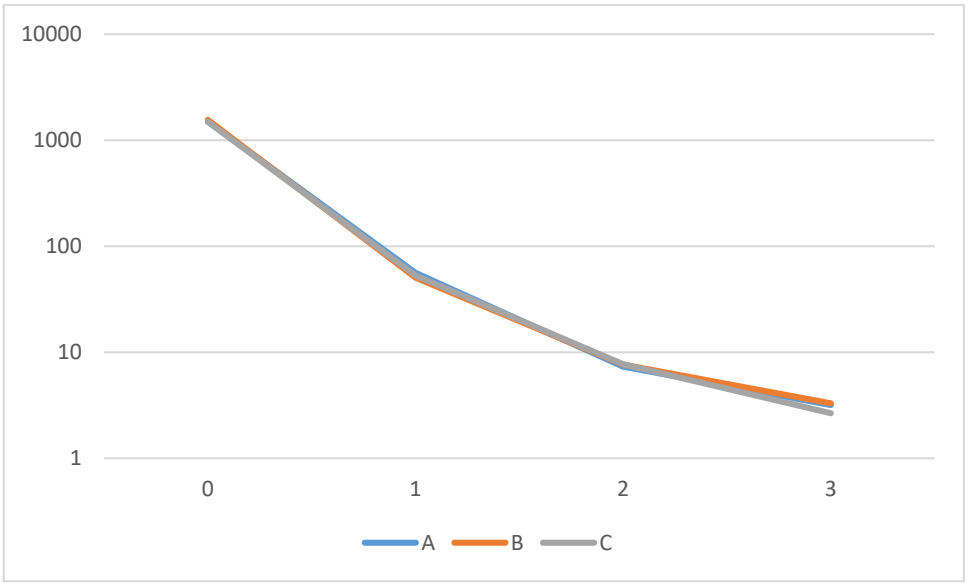
Pada Tabel 1 terdapat 6 kolom. Pada kolom 2 ada variabel d yang mewakili nilai digit. Ketika d = 2 artinya jumlah digit dari integer adalah 2000 digit dikalikan dengan 2000 digit. Pola yang sama diterapkan juga ketika d = 4 dan d = 6.

Pada kolom 5 terdapat variabel b yang mewakili nilai *base case*. Ketika b = 2 artinya nilai *base case* = 10^(10^2). Simbol ^ artinya perpangkatan. Nilai b pada percobaan yaitu = {0, 1, 2, 3}.



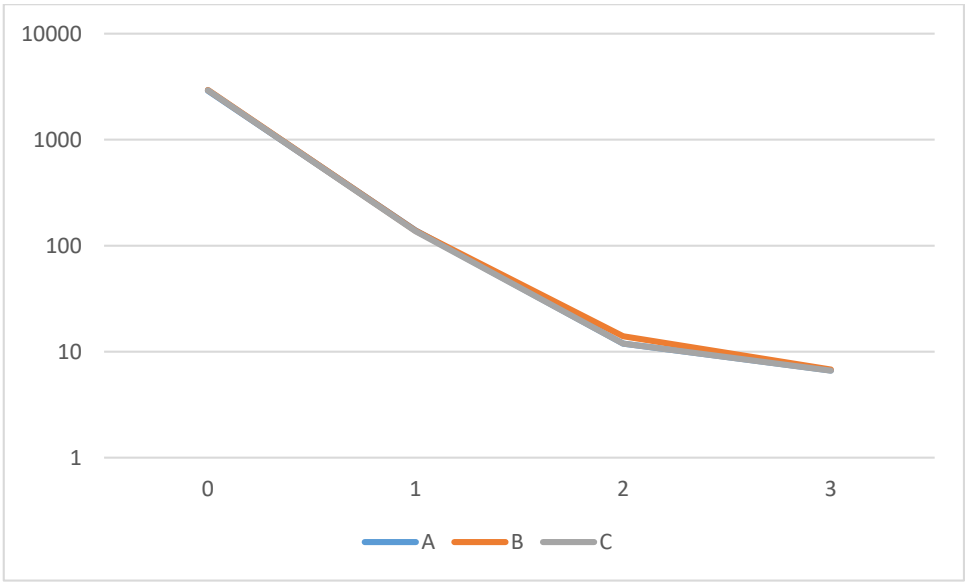
Gambar 2 Grafik d = 2

Gambar 2 menunjukkan grafik berskala logaritmik untuk d = 2. Terlihat pada grafik ketika nilai b semakin besar, grafik akan semakin menurun. Perbedaan kasus antara A, B, dan C tidak signifikan terhadap waktu eksekusi.



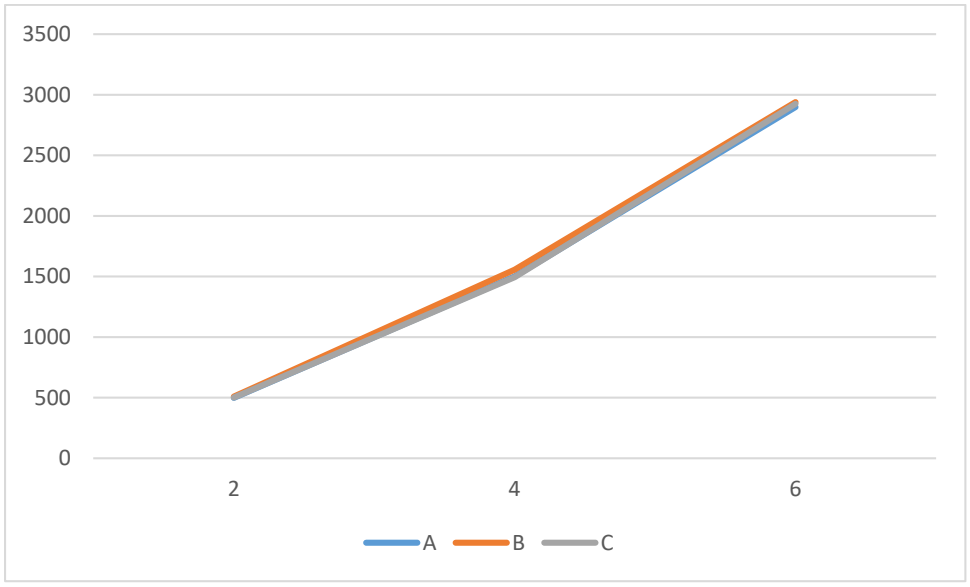
Gambar 1 Grafik d = 4

Gambar 3 menunjukkan grafik berskala logaritmik untuk d = 4. Terlihat pada grafik ketika nilai b semakin besar, grafik juga akan semakin menurun. Perbedaan kasus juga tidak berarti pada grafik ini.



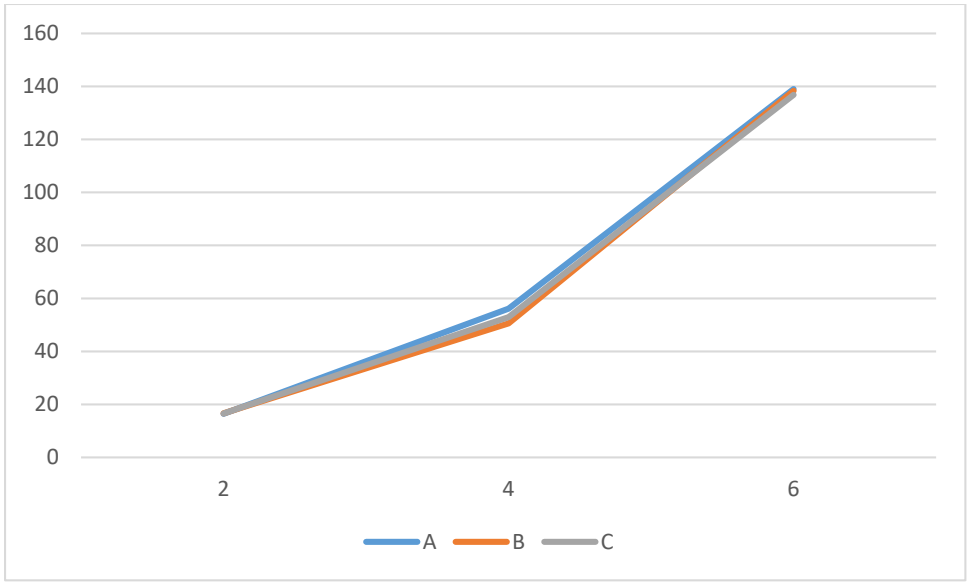
Gambar 2 Grafik d = 6

Gambar 4 adalah grafik dengan d = 6. Hasilnya serupa dengan d = 2 dan d = 4 yang terlihat pada Gambar 2 dan Gambar 3.



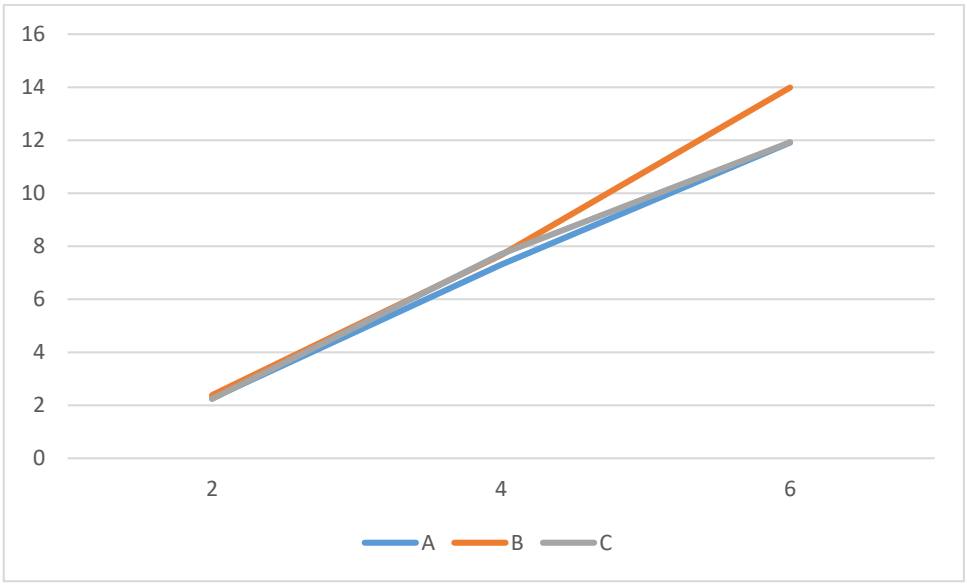
Gambar 3 Grafik b = 0

Pada Gambar 5 adalah Grafik dengan nilai b = 0. Terlihat grafik mendekati pola linier untuk nilai d = 2, 4, dan 6. Hal ini menunjukkan semakin besar nilai d maka dibutuhkan waktu eksekusi yang semakin besar.



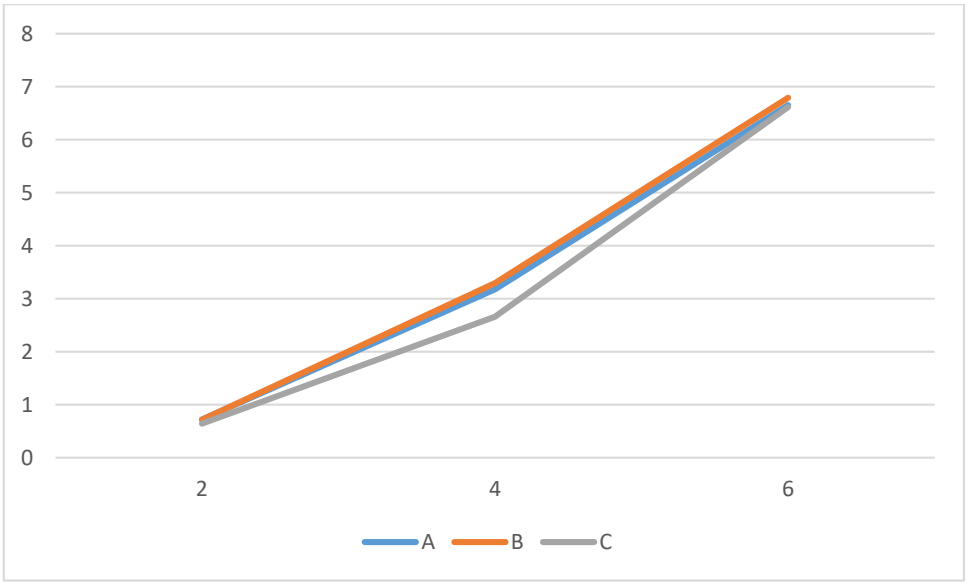
Gambar 4 Grafik b = 1

Gambar 6 adalah grafik dengan nilai b = 1. Polanya serupa dengan Gambar 5 di mana semakin besar nilai d, grafik semakin ke atas.



Gambar 5 Grafik b = 2

Gambar 7 adalah grafik dengan nilai b = 2. Untuk kasus B pada d = 6, nilainya terlihat lebih besar dibandingkan ketika d = 2 dan d = 4.



Gambar 6 Grafik b = 3

Gambar 8 adalah grafik dengan nilai b = 3. Untuk kasus C ketika nilai d = 4 terlihat lebih rendah dibandingkan kasus A dan B.

4. KESIMPULAN

Berdasarkan hasil penelitian dan pembahasan pada subbab sebelumnya, diperoleh fakta sebagai berikut:

1. Jumlah digit yang lebih besar akan membutuhkan waktu eksekusi yang lebih lama dengan syarat nilai *base*-nya sama
2. Perbedaan kasus tidak menghasilkan perbedaan yang signifikan
3. Semakin besar nilai base maka waktu eksekusi yang dibutuhkan menjadi lebih singkat

Hal tersebut ternyata tidak sesuai dengan dugaan dari peneliti dimana seharusnya nilai base case yang sangat besar menyebabkan waktu eksekusi yang dibutuhkan menjadi lebih lama. Ada dugaan bahwa hasil eksperimen ini tidak sesuai dengan dugaan awal dikarenakan pada Bahasa Pemrograman Python sendiri telah menerapkan algoritma Karatsuba untuk proses perkalian. Hal tersebut tidak dapat dibuktikan karena peneliti tidak menemukan fakta bahwa di dalam Python sendiri telah menerapkan Karatsuba. Namun, terdapat pembahasan seperti itu pada forum di internet.

5. SARAN

Untuk dapat mengatasi masalah tersebut, maka penelitian perlu dilakukan dengan menggunakan bahasa pemrograman yang tidak mengimplementasi Karatsuba secara implisit ke dalamnya.

DAFTAR PUSTAKA

[1] Baruchel, T. 2019. Flattening Karatsuba's Recursion Tree into a Single Summation. SN Computer Science (2020) 1:48. Springer
[2] Ramakrishna, S. 2019. Speeding up the Karatsuba algorithm. arXiv:1905.07455v3 [cs.DS] 23 Oct 2019.
[3] Çalik, Ç. dkk. 2019. Searching for Best Karatsuba Recurrences. Springer: Analysis of Experimental Algorithms Special Event, SEA2 2019, Kalamata, Greece, June 24–29, 2019 Revised Selected Papers
[4] Li, Y. dkk. 2019. On the Complexity of Hybrid n-Term Karatsuba Multiplier for Trinomials. IEEE Transactions on Circuits and Systems–I: Regular Papers. IEEE
[5] Guo, S. dan Zorin-Kranich, P. 2020. Decoupling for moment manifolds associated to Arkhipov–Chubarikov–Karatsuba systems, Advances in Mathematics 360 (2020) 106889. Elsevier
[6] Dwivedi, S. P. 2013. An efficient multiplication algorithm using Nikhilam method. Fifth International Conference on Advances in Recent Technologies in Communication and Computing (ARTCom), 20-21 Sept, Bangalore, IET, ISBN: 978-1-84919-842-4, 223-228.
[7] Eyupoglu, C. 2015. Performance Analysis of Karatsuba Multiplication Algorithm for Different Bit Lengths. World Conference on Technology, Innovation and Entrepreneurship. Elsevier Procedia - Social and Behavioral Sciences Journal, Volume 195. pp 1860-1864.