Aplikasi *Divide and Conquer* pada Perkalian *Large Integer* untuk Menghitung Jumlah Rute TSP *Brute Force*

Martin Lutta Putra - 13515121
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
13515121@std.stei.itb.ac.id

***Abstract* — Seiring berkembangnya teknologi dan ilmu pengetahuan, perhitungan-perhitungan yang harus dilakukan semakin melibatkan formula-formula yang kompleks serta bilangan-bilangan yang bernilai sangat besar. Sebagai contoh, jumlah rute pada persoalan TSP meningkat secara signifikan seiring bertambahnya jumlah input dan proses perhitungan jumlahnya melibatkan bilangan yang sangat besar. Tetapi, proses-proses perhitungan dengan pendekatan *brute force* sering kali dianggap membutuhkan waktu terlalu lama. Diperlukan pendekatan lain yang dapat melakukan operasi yang sama namun dalam waktu yang lebih singkat. Makalah ini akan membahas aplikasi strategi *Divide and Conquer* pada operasi perkalian bilangan-bilangan yang bernilai sangat besar, serta menerapkannya perhitungan jumlah rute pada persoalan TSP.**

Keywords— large integer, brute force, divide and conquer, TSP

I. PENDAHULUAN

Integer, atau bilangan bulat, merupakan salah satu himpunan bilangan yang paling dikenal di dunia. Integer digunakan secara luas setiap harinya oleh berbagai kalangan, dan penggunaannya bukan hanya di bidang akademik saja, integer bahkan digunakan pula dalam aktifitas sehari-hari seperti bermain, bertransaksi, bahkan mengobrol santai sekalipun. Karakteristik integer yang merepresentasikan bilangan bulat membuatnya sangat mudah dibayangkan dan diaplikasikan di dalam kehidupan, sebab secara intuitif manusia menganggap objek-objek yang ia lihat sebagai sesuatu yang utuh dan menyatakan jumlah objek-objek tersebut dalam nilai yang bulat.

Terdapat sangat banyak operasi matematika yang dapat dilakukan pada himpunan bilangan bulat/integer, misalnya penjumlahan, pengurangan, perkalian, pembagian, perpangkatan, akar, faktorial, dsb. Operasi-operasi ini memudahkan manusia dalam mengaitkan nilai objek yang sebenenarnya hanya merupakan konsep abstrak dengan fakta yang dapat mereka lihat dalam kehidupan sehari-hari.

Dengan kemajuan teknologi serta meningkatnya kompleksitas aktivitas manusia, operasi-operasi di atas mulai dilaksanakan dengan bantuan komputer agar dapat diselesaikan dengan efektif dan efisien. Program-program komputer mulai ditulis untuk menyelesaikan permasalahan yang spesifik, seperti perhitungan data hasil eksperiman, jumlah penduduk, nilai pajak, dan nilai transaksi dagang. Perhitungan-perhitungan ini terlalu kompleks untuk diselesaikan dengan cara perhitungan konvensional. Operasi terhadap bilangan bulat menggunakan kompter membantu manusia dengan untuk menyelesaikan dengan cepat dan tepat berbagai perhitungan yang dahulu harus diselesaikan menggunakan alat tulis serta sangat rentan terhadap kesalahan perhitungan.

Namun, seiring berjalannya waktu, ilmu pengetahuan semakin berkembang dan perhitungan mulai melibatkan angka-angka yang jumlah digitnya terlalu besar untuk ditangani secara efisien oleh komputer. Misalnya, perhitungan matematika dalam eksperiman di bidang fisika kuantum memerlukan ketelitian sangat tinggi, sehingga jumlah digit di belakang koma yang harus diperhitungkan pun sangat besar. Bahkan, hasil dari operasi permutasi seratus objek 20! pun memiliki ordo 10^{22}, padahal sering kali di kehidupan nyata nilai permutasi yang harus dihitung lebih besar dari nilai tersebut. Angka-angka yang sangat besar ini membuat komputer berkecepatan tinggi pun membutuhkan waktu lama untuk menyelesaikan perhitungan, sehingga komputer kehilangan keunggulannya, yakni menyelesaikan perhitungan kompleks dengan cepat dan tepat. Oleh sebab itu, diperlukan teknik untuk memaksimalkan performansi komputer dalam melakukan perhitungan yang melibatkan angka-angka yang besar, khususnya bilangan bulat yang besar atau *large integer*.

Makalah ini akan membahas dasar-dasar teori terkait *large integer*, serta operasi yang umum dilakukan pada *large integer*. Dikarenakan beberapa operasi matematika dapat diturunkan dari maupun menurunkan operasi matematika lain, dalam makalah ini hanya akan dibahas operasi perkalian (*multiplication*) untuk mewakili beberapa operasi matematika yang ada. Selain itu, akan dibahas pula salah satu teknik yang dapat dilakukan untuk mengoptimalisasi perhitungan-perhitungan yang melibatkan *large integer*, yakni teknik *Divide and Conquer*. Kemudian, makalah ini akan melakukan perbandingan terhadap perhitungan *large integer* menggunakan teknik *Brute Force* dengan perhitungan menggunakan *Divide and Conquer* pada salah satu contoh persoalan yang melibatkan *large integer*, yakni penghitungan jumlah rute pada *Travelling Salesman Problem* (TSP).

II. TEORI DASAR

A. *Integer*

Integer (bilangan bulat) adalah anggota himpunan bilangan rasional yang juga melingkupi himpunan bilangan asli. Secara intuitif, bilangan bulat adalah himpunan bilangan dari (-∞, ∞) yang dapat dinyatakan tanpa pecahan maupun angka di belakang koma.

Dalam komputer, integer direpresentasikan menggunakan bilangan biner/bilangan basis dua. Sebagai contoh, bilangan 12, 15, dan 3 secara berturut-turut direpresentasikan oleh

1100

1111

11

Bilangan integer digunakan secara luas dalam bahasa pemrogaman. Umumnya, sebuah variable yang dimaksudkan untuk menyimpan nilai integer akan dideklarasikan menggunakan tipe *int* atau *int64.* Tipe variable *int* memiliki nilai maksimal 2.147.483.647 dan nilai minimal –2.147.483.648, sedangkan tipe variable *int64* memiliki nilai maksimal 9.223.372.036.854.775.807 dan nilai minimal -9.223.372.036.854.775.808.

B. *Multiplication*

Multiplication / perkalian bilangan merupakan salah satu operasi matematika yang didasarkan pada operasi penjumlahan. Perkalian

$$a \; x \; b$$

didefinisikan sebagai :

$$\underbrace{a + a + a + a + a + a + \cdots}_{b \; kali}$$

Perkalian memiliki sifat sebagai berikut :

- Komutatif,: $a \; x \; b = b \; x \; a$
- Asosiatif : $a \; x \; (b \; x \; c) = (a \; x \; b) \; x \; c$
- Distributif : $a \; x \; (b + c) = (a \; x \; b) + (a \; x \; c)$

C. *Kompleksitas*

Kompleksitas terbagi menjadi dua, yakni kompleksitas waktu dan kompleksitas ruang. Kompleksitas waktu adalah besaran yang digunakan untuk menyatakan besarnya waktu yang diperlukan oleh suatu algoritma untuk menyelesaikan persoalan, sedangkan kompleksitas ruang adalah besaran yang menyatakan besarnya memori yang diperlukan oleh algoritma.

Kompleksitas waktu umumnya dinyatakan dengan notasi Big-Oh ($O(f(n))$), dengan $f(n)$ merupakan fungsi matematika yang tidak menyertakan koefisien fungsi. Selain itu, notasi $O(f(n))$ menyatakan batas atas dari waktu yang mungkin diperlukan oleh sebuah algoritma, sehingga nilainya akan mengikuti ordo polynomial terbesar yang ada dalam fungsi.. Sebagai contoh, untuk $f(n) = 2n^2 + 2n$, nilai $O(f(n)) = n^2$, dan untuk $f(n) = 3n^3 + 10n + 7$, maka nilai $O(f(n)) = n^3$.

Kompleksitas ruang terkait dengan struktur data yang digunakan untuk menyelesaikan algoritma dan seberapa banyak jumlah *instance* struktur data tersebut yang dihasilkan selama proses penyelesaian persoalan. Kompleksitas waktu dinyatakan sebagai $S(f(n))$, dengan $f(n)$ adalah fungsi matematika yang tidak menyertakan koefisien fungsi. Namun, seiring dengan berkembangnya teknologi, kapasitas memori yang dapat digunakan oleh komputer semakin tinggi sehingga penggunaan ruang penyimpanan yang besar untuk mencapai kompleksitas waktu yang lebih rendah dipandang sebagai biaya yang 'murah' dan menjadi pilihan utama dalam menerapkan algoritma.

D. *Brute Force*

Brute Force merupakan strategi penyelesaian masalah yang didasarkan kepada deskripsi persoalan ataupun konsep yang langsung terkait dengan persoalan. *Brute Force* disebut juga *naïve approach*, sebab strategi ini menggunakan cara penyelesaian paling umum yang biasanya langsung muncul di dalam pikiran ketika memikirkan solusi dari suatu persoalan.

Umumnya, penyelesaian dengan *Brute Force* membutuhkan waktu penulisan kode yang paling sedikit serta algoritma yang paling sederhana. Namun, konsekuensinya adalah kompleksitas algoritma *Brute Force* umumnya lebih tinggi dibanding algoritma-algoritma lainnya.

E. *Divide and Conquer*

Divide and Conquer merupakan salah satu strategi algoritma yang umum digunakan. Ide dari *Divide and Conquer* adalah menyelesaikan suatu masalah dengan memecahnya menjadi upamasalah-upamasalah yang lebih kecil (yang idealnya berukuran sama), menyelesaikan upamasalah-upamasalah tersebut satu per satu, kemudian menggabungkan hasil penyelesaiannya sehingga diperoleh solusi yang utuh atas persoalan tersebut. Obyek persoalan yang dipecah ukurannya dapat berupa input maupun *instances* persoalan seperti tabel, matriks, eksponen, dsb. Tiap-tiap upamasalah hasil pembagian merupakan miniatur dari persoalan yang semula dan memiliki karakteristik permasalahan yang sama, sehingga strategi *Divide and Conquer* secara natural diungkapkan dengan skema rekursif.

Algoritma *Divide and Conquer* secara umum memiliki skema sebagai berikut :

```
procedure DnC (input N : integer)
{ Menyelesaikan suatu persoalan dengan algoritma Divide and Conquer }
Kamus
    a, b : integer
Algoritma
    if (N < k) {ukuran upa-masalah sudah cukup kecil}
      SOLVE upa-masalah
    else
      Bagi masalah menjadi x upa-masalah, masing-masing berukuran N/y
```

for masing-masing upamasalah do

DnC(x)

endfor

COMBINE solusi dari x upa-masalah

Tabel 1 Skema umum algoritma Divide and Conquer

F. Teorema Master

Teorema Master merupakan teorema yang digunakan untuk menghitung kompleksitas sebuah algoritma yang memanfaatkan rekursif. Misalkan $T(n)$ merupakan fungsi menaik yang memenuhi relasi rekurens :

$$T(n) = aT\left(\frac{n}{b}\right) + cn^d,$$

dengan $n = b^k$, k = 1,2,3, …, a >= 1, b >= 2, serta c dan d merupakan bilangan riil yang bernilai >= 0. Maka, menurut Teorema Master, fungsi menaik $T(n)$ adalah fungsi dengan kompleksitas

$$T(n) = \begin{cases} O(n^d), \text{ jika } a < b^d \\ O(n^d \log n), \text{ jika } a = b^d \\ O\left(n^{\log_b a}\right), \text{ jika } a > b^d \end{cases}$$

G. Karatsuba's Fast Multiplication

Teknik perkalian cepat Karatsuba / *Karatsuba's Fast Multiplication* adalah algoritma untuk perkalian bilangan yang memanfaatkan strategi *Divide and Conquer*. Misalkan terdapat dua buah bilangan x1 dan x2, masing-masing berjumlah n digit. Ide dasar dari teknik perkalian Karatsuba adalah membagi baik x1 maupun x2 menjadi dua bagian, misalnya $x1_{kiri}$, $x1_{kanan}$, $x2_{kiri}$, dan $x2_{kanan}$, masing-masing berjumlah $\frac{n}{2}$ digit, kemudian mengurangi jumlah perkalian yang dibutuhkan dengan memanfaatkan sifat distributif perkalian

$x1_{kiri}.x2_{kanan} + x1_{kanan}.x2_{kiri} = (x1_{kiri} + x1_{kanan})(x2_{kiri} + x2_{kanan}) - (x1_{kiri}.x2_{kiri}) - (x1_{kanan}.x2_{kanan})$.

Sehingga, perkalian x1 dan x2 dimanipulasi menjadi :

$x1 \,.\, x2 = x1_{kiri}\, x2_{kiri} \,.\, 10^n + ((x1_{kiri} + x1_{kanan})(x2_{kiri} + x2_{kanan}) - (x1_{kiri}.x2_{kiri}) - (x1_{kanan}.x2_{kanan})) \,.\, 10^{n/2} + x1_{kanan}.\, x2_{kanan}$

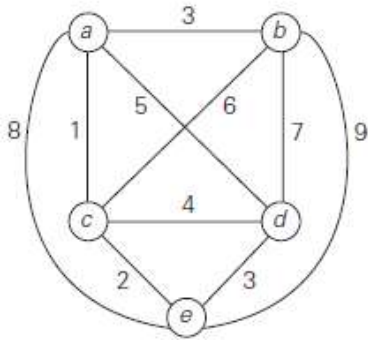
Dengan memanfaatkan Teorema Master, diperoleh bahwa kompleksitas teknik perkalian cepat Karatsuba untuk dua buah bilangan berjumlah n digit adalah

$$T(n) = O(n^{1.585})$$

H. Travelling Salesman Problem (TSP)

TSP merupakan salah satu persoalan yang sangat dikenal di dalam dunia komputasi. TSP tergolong jenis persoalan NP-Complete yang hingga saat ini belum ditemukan penyelesaian mangkusnya.

Deskripsi persoalan TSP adalah sebagia berikut. Diberikan *integer* bernilai n yang mewakili jumlah kota serta sebuah graf lengkap G yang mewakili jarak dari tiap kota ke kota lainnya (Graf dapat direpresentasikan menggunakan matriks ketetanggaan, matriks bersisian, maupun gambar graf secara langsung), carilah sebuah rute terpendek sedemikian sehingga setiap kota dikunjungi maksimal sekali kemudian kembali ke kota asal.



Gambar 1 Contoh Graf TSP
Sumber : Introduction to Analysis and Design of Algorithms, 3rd ed

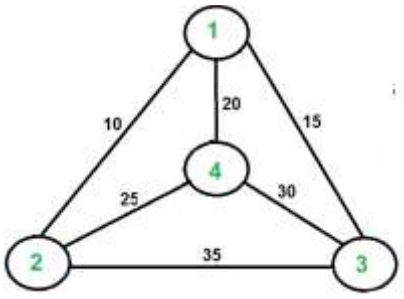
Terdapat banyak pendekatan yang dapat digunakan untuk menyelesaikan persoalan TSP. Beberapa pendekatan yang umum digunakan diantaranya adalah strategi *Brute Force* dan *Branch and Bound*.

III. PENYELESAIAN PERSOALAN

A. Penyelesaian TSP dengan Brute Force

TSP mengharuskan setiap kota untuk dikunjungi sebanyak tepat satu kali, kemudian kembali ke kota awal. Rute harus dipilih sedemikian rupa sehingga jarak total yang ditempuh adalah jarak terpendek yang mungkin berdasarkan data pada representasi graf yang diberikan. Mempertimbangkan hal tersebut, cara apakah yang harus ditempuh untuk memperoleh solusi persoalan ?

Secara intuitif, cara paling sederhana adalah dengan menghitung semua kemungkinan rute beserta jaraknya, kemudian membandingkan seluruhnya dan mengambil rute yang terpendek. Misalkan diberikan sebuah *instance* persoalan TSP dengan graf sebagai berikut :



Gambar 2 Instance persoalan TSP,
Sumber : http://www.geeksforgeeks.org/branch-bound-set-5-traveling-salesman-problem/

Maka, langkah-langkah yang harus dilakukan untuk menemukan rute terpendeknya adalah :

a. Memilih sebuah simpul sebagai representasi kota asal, misal simpul 1.

b. Menentukan semua kemungkinan rute yang dimulai dari simpul 1 hingga kembali ke simpul 1, yakni 1-2-3-4-1, 1-3-2-4-1, 1-4-2-3-1, 1-3-4-2-1, dst.

c. Menghitung jarak yang ditempuh untuk setiap rute. Jarak antara kota a dan b adalah nilai pada busur yang menghubungkan simpul a dan b, sehingga, total jarak yang ditempuh untuk suatu rute adalah total nilai pada semua busur yang dilalui oleh rute tersebut. Sebagai contoh, rute 1-2-3-4-1 memiliki total jarak 95.

d. Membandingkan semua total jarak setiap rute dan memilih yang terkecil.

Namun, ada kalanya kita dihadapkan dengan persoalan TSP namun tidak memiliki struktur data graf yang siap digunakan. Untuk mengatasi hal tersebut, graf dapat dikonversi menjadi sebuah matriks yang menyatakan informasi yang ekivalin. Misalkan dipilih matriks ketetanggaan (*adjacency matrix*) untuk merepresentasikan graf. Maka, untuk *instance* persoalan di atas, matriks ketetanggaannya adalah

$$M = \begin{bmatrix} \infty & 10 & 15 & 20 \\ 10 & \infty & 35 & 25 \\ 15 & 35 & \infty & 30 \\ 20 & 25 & 30 & \infty \end{bmatrix}$$

Gambar 3Matriks ketetanggaan untuk instance persoalan

dengan M[i][j] menyatakan jarak dari kota i ke kota j.

Rute yang mungkin dilalui akan direpresentasikan sebagai sebuah string berbentuk kota1-kota2-kota3-…-kotaN-kota1. Untuk setiap rute, akan disimpan jarak tempuhnya. Berikut merupakan pseudocode untuk menyelesaikan persoalan di atas.

```
function SolveTSPV (input M : Matriks, final_route : string)
{ Menyelesaikan persoalan TSP secara intuitif}
Deklarasi
    route  : string {rute perjalanan, dimulai dan diakhiri dengan simpul 1}
    min_dist, temp_dist, i : integer
Algoritma
    min_dist ← 99999 {Inisialisasi dengan nilai besar}
    while (masih ada kemungkinan permutasi pada route) do
        temp_dist ← 0; i ← 0
        foreach (huruf pada route) do
            temp_dist ← temp_dist + M[route[i]][route[i+1]]
        endforc
        if (temp_dist < min_dist) then
            min_dist ← temp_dist
            final_route ← route
    endwhile
```

Tabel 2 Pseudocode penyelesaian TSP dengan Brute Force

Sesuai dengan langkah-langkah yang dituliskan di atas, untuk mendapatkan rute terpendek, hal selanjutnya yang harus dilakukan adalah menghitung semua kemungkinan rute serta jaraknya, kemudian memilih nilai terkecil. Namun, apabila diperhatikan, rute 1-2-3-4-1 sesungguhnya merupakan jalur yang sama dengan rute 1-4-3-2-1. Demikian pula rute 1-3-2-4-1 adalah jalur yang sama dengan rute 1-4-2-3-1. Oleh sebab itu, hanya diperlukan evaluasi terhadap setengah dari seluruh jumlah rute yang mungkin untuk menghitung jarak yang ditempuh oleh semua rute.

B. Jumlah Rute dengan Brute Force

Setiap rute yang mungkin dilalui dalam persoalan TSP diwakili oleh sebuah permutasi string. Terdapat n! kemungkinan permutasi dari sebuah string dengan panjang n. Oleh sebab itu, banyaknya rute yang harus dievaluasi untuk menghasilkan solusi juga adalah sebanyak n!.

Apabila kita mengasumsikan bahwa perkalian dua buah bilangan integer memiliki kompleksitas sebesar $O(1)$, maka perhitungan terhadap kompleksitas waktu yang dibutuhkan untuk mengetahui jumlah rute TSP *brute* force sangat sederhana. Definisi faktorial adalah

$$n! = n\ x\ (n-1)x\ (n-2)\ x\ \dots x\ 2\ x\ 1$$

Diperlukan satu operasi untuk perhitungan setiap dua suku dalam faktorial. Maka, untuk menghasilkan jumlah rute yang dievaluasi, komputer harus melakukan perhitungan sebanyak (n-1), sehingga

$$T(n) = O(n-1)$$
$$= O(n)$$

Tetapi, operasi perkalian antara dua buah bilangan tidak selalu dapat diasumsikan memiliki kompleksitas $O(1)$. Asumsi yang lebih tepat untuk perkalian antara dua buah integer, masing-masing sepanjang n digit adalah $O(n^2)$. Hal ini disebabkan dalam perkalian dua buah bilangan, setiap digit dari bilangan pertama harus dikalikan dengan setiap digit dari bilangan kedua, sehingga operasi perkalian yang demikian memerlukan operasi sebanyak $O(n\ x\ n) = O(n^2)$. Observasi bahwa setiap rute memiliki pasangan permutasi yang melalui jalur yang sama, seperti 1-2-3-4-1 dengan 1-4-3-2-1 pun tidak mengurangi kompleksitas yang dibutuhkan untuk menghitung jumlah rute. Berdasarkan observasi tersebut, waktu yang dibutuhkan adalah $O\left(\frac{1}{2}n^2\right) = O(n^2)$.

C. Jumlah Rute dengan Divide and Conquer

Persoalan TSP dalam dunia nyata sering kali melibatkan jumlah input yang sangat besar. Untuk jumlah input yang demikian, sekedar perhitungan jumlah rute memerlukan waktu yang lama dikarenakan kompleksitas operasi perkalian $O(n^2)$. Untungnya, terdapat teknik yang dapat mengurangi waktu operasi perkalian bilangan yang tergolong *large integer* tersebut secara cukup signifikan, yakni teknik *Divide and Conquer*.

Teknik perkalian yang akan digunakan didasarkan pada algoritma perkalian cepat Karatsuba (Karatsuba's *fast multiplication*), yang pada intinya berusaha mengurangi jumlah

perkalian antar suku-suku yang terlibat dalam perkalian dengan memanfaatkan sifat distributif pada operasi perkalian.

Perkalian cepat Karatsuba dapat diterapkan karena ketika menghitung jumlah rute yang mungkin dari TSP, yang sesungguhnya merupakan operasi penghitungan faktorial, penghitungan dapat dipandang sebagai perkalian bilangan pertama dengan bilangan kedua, kemudian dikali bilangan ketiga, lalu dikali bilangan keempat, dan seterusnya sehingga operasi faktorial merupakan perkalian dari hasil perkalian sebelumnya dengan satu angka selanjutnya,

$$20! = \underbrace{20 \; x \; 19}_{} \; x \; 18 \; x \; 17 \; x \; \ldots x \; 5 \; x \; 4 \; x 3 \; x \; 2 \; x \; 1$$

Gambar 4 Ilustrasi cara pandang pertama terhadap operasi faktorial

atau dipandang sebagai perkalian antar dua bilangan hasil perkalian sebelumnya. Pandangan pertama condong kepada strategi *brute force*, sedangkan pandangan kedua lebih condong kepada strategi *Divide and Conquer*.

$$20! = \underbrace{20 \; x \; 19}_{} \; x \; \underbrace{18 \; x \; 17}_{} \; x \; \ldots x \; \underbrace{5 \; x \; 4}_{} \; x 3 \; x \; \underbrace{2 \; x \; 1}_{}$$

Gambar 5 Ilustrasi cara pandang kedua terhadap operasi faktorial

Cara pandang kedua dapat diterapkan dengan menyimpan bilangan dalam sebuah larik integer, kemudian secara rekursif membagi larik menjadi dua upa-larik hingga setiap elemen upa-larik berjumlah <=2 elemen, lalu melakukan perkalian untuk kedua elemen tersebut dan menggabungkannya secara rekursif pula (mirip dengan ide algoritma *Merge Sort*). Berikut merupakan pseudocode dari cara penghitungan diatas.

```
function HitungJumlahRute (input N : integer) → integer
{ Menghitung jumlah rute pada TSP dengan metode Divide and Conquer}

Deklarasi
    arr : ArrayOfInteger
    i, j, k, sum : integer

Algoritma
    for i ← 1 to N do
        arr[i] ← i
    endfor
    sum ← PerkalianDnC(arr, 1, N)
    → sum
```

```
function PerkalianDnC(input Arr :ArrayOfInteger, input i,j:integer) → integer
{ Melakukan operasi perkalian Divide and Conquer pada array of integer}

Deklarasi
    k, hasilkiri, hasilkanan : integer

Algoritma
    if (j-i > 1) then
        k ← (i+j) div 2
        hasilkiri ← PerkalianDnC(Arr, i, k)
        hasilkanan ← PerkalianDnC(Arr, k+1, j)
        → KaratsubaMultiplication(hasilkiri, hasilkanan)
    else
        → KaratsubaMultiplication(Arr[i], Arr[j])
```

Tabel 3Pseudocode penghitungan jumlah rute dengan DnC

D. Perbandingan Kompleksitas

Kompleksitas perkalian dengan menggunakan algoritma Karatsuba adalah $O(n^{1.585})$, secara signifikan lebih cepat dibandingkan perhitungan operasi perkalian dengan menggunakan *brute force*. Namun, apakah proses perkalian dengan menggunakan algoritma *Divide and Conquer* seperti yang telah dijabarkan di atas lebih optimal daripada proses perkalian faktorial menggunakan pendekatan *Brute Force* ?

Proses perkalian di atas membagi larik yang berisi seluruh bilangan dari 1 s.d. n menjadi dua upa-larik yang sama besar jika n genap, atau upa-larik berukuran $\lfloor n/2 \rfloor$ dan $\lfloor n/2 \rfloor$+1 bila n berjumlah ganjil. Pembagian dilakukan terus menerus hingga upa-larik berukuran <= 2 elemen, kemudian dilakukan operasi perkalian. Hasil perkaliannya dikalikan dengan hasil perkalian upa-larik lain hingga diperoleh hasil akhir. Ketika bilangan yang terlibat perkalian setiap upa-larik masih tergolong kecil, teknik perkalian Karatsuba tidak memberikan hasil yang signifikan. Namun, seiring bertambahnya ukuran upa-larik yang telah dikalikan, bilangan yang terlibat pun akan semakin besar dan teknik perkalian Karatsuba mulai menunjukkan perbedaan waktu operasi yang berarti.

Berdasarkan *pseudocode* di atas, relasi rekurens untuk proses perkalian menggunakan *Divide and Conquer* adalah :

$$T(n) = 2T\left(\frac{n}{2}\right) + cn^{1.585}$$

Berdasarkan teorema master, apabila relasi rekurens berbentuk

$$T(n) = aT\left(\frac{n}{b}\right) + cn^d$$

memiliki $a < b^d$, maka $T(n) = O(n^d)$. Dengan $a = 2$, $b = 2$, dan $d = 1.585$, relasi rekurens di atas memiliki $a < b^d$, sehingga proses perkalian menggunakan *Divide and Conquer* di atas memiliki kompleksitas

$$T(n) = O(n^{1.585}).$$

Dengan hasil tersebut, maka proses perkalian dengan *Divide and Conquer* yang memanfaatkan teknik perkalian cepat Karatsuba memberikan hasil yang lebih optimal dibanding proses perkalian secara *brute force*.

IV. KESIMPULAN

Proses pencarian jumlah rute TSP menggunakan pendekatan *Brute Force* memerlukan waktu yang lama, terutama ketika jumlah kota yang terlibat dalam persoalan sangat besar. Hal ini dikarenakan proses pencarian jumlah rute melibatkan operasi faktorial, yang dengan cepat akan melibatkan perkalian *large integer* ketika menerima nilai input yang besar, sedangkan b*rute force* tidak mangkus untuk diterapkan pada perkalian *large integer*. Pendekatan *Divide and Conquer* dan teknik perkalian cepat Karatsuba dapat diterapkan pada proses perkalian ini untuk menghasilkan waktu pencarian jumlah rute yang lebih optimal. Pendekatan ini memiliki kompleksitas waktu $T(n) = O(n^{1.585})$, lebih baik ketimbang pendekatan *brute force* yang memerlukan waktu $T(n) = O(n^2)$.

V. UCAPAN TERIMA KASIH

Penulis mengucapkan syukur kepada Tuhan Yang Maha Esa, yang telah menganugerahkan berkat-Nya secara berlimpah sehingga penulis dapat menyelesaikan makalah ini dengan baik. Penulis mengucapkan terima kasih kepada Dr. Masayu Leylia Khodra, S.T., M.T., Dr. Ir. Rinaldi Munir, M.T., dan Dr. Nur ulva Maulidevi, S.T., M.Sc., sebagai dosen pengampu matakuliah IF2211 – Strategi Algoritma yang telah memberikan bekal ilmu yang cukup sehingga penulis dapat menyusun makalah ini. Penulis mengucapkan terima kasih kepada orang tua yang senantiasa memberikan dukungan baik moril maupun materiil kepada penulis dalam menjalani segala kegiatan perkuliahan di ITB. Penulis mengucapkan terima kasih kepada teman-teman yang namanya tidak dapat penulis sebutkan satu per satu atas semangat, dorongan, dan dukungan yang telah diberikan, serta diskusi yang baik yang telah menambah wawasan serta pengetahuan penulis. Penulis juga mengucapkan terima kasih kepada seluruh pihak lainnya yang telah berperan dalam penyusunan makalah sehingga makalah ini dapat diselesaikan tepat pada waktunya.

VI. REFERENSI

[1] Munir, Rinaldi. 2006. *Strategi Algoritma.* Bandung : Penerbut Informatika.

[2] Levitin, Anany. 2008. *Introduction to Design and Analysis of Algorithms, 3rd edition.* New York City : Pearson Education.

[3] http://www.geeksforgeeks.org/divide-and-conquer-set-2-karatsuba-algorithm-for-fast-multiplication/, diakses 17 Mei pukul 09.00 WIB

[4] http://www.cburch.com/csbsju/cs/160/notes/31/1.html, diakses 18 Mei pukul 13.00 WIB.

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, 19 Mei 2017

Martin Lutta Putra
13515121