Pencarian Nilai Eigen dengan Menggunakan Algoritma *Divide and Conquer*

Elvina Riama K. Situmorang – 13514045[1]
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Infromatika
Institut Teknologi Bandung, Jalan Ganesha no. 10 Bandung 40132, Indonesia
[1]13514045@std.stei.itb.ac.id

***Abstract* — Metode algoritmik adalah penyelesaian masalah menggunakan langkah-langkah yang terstruktur sehinggga solusi dapat ditemukan. Pada dasarnya, dalam ilmu komputer konteks divide and conquer tidak berbeda dengan maknanya dalam bidang politik maupun militer, yaitu memecah suatu masalah menjadi masalah-masalah kecil yang akan memudahkan pencarian solusi dari masalah tersebut. Pada makalah ini akan dibahas cara menyelesaikan masalah perhitungan nilai eigen dengan pendekatan algoritma divide and conquer.**

Keywords—Algoritma; divide and conquer; nilai eigen.

I. PENDAHULUAN

Dalam kehidupan ini, terdapat banyak masalah. Masalah yang ada dapat diselesaikan dengan berbagai cara. Dalam penyelesaian masalah menggunakan komputasi, terdapat tiga kategori/cara, yaitu heuristik, algoritmik, dan gabungan keduanya..

Metode heuristik yaitu cara berpikir divergen (menuju beberapa target tujuan sekaligus). Metode heuristik cocok digunakan untuk masalah yang mengandung arti ganda dan penafsiran. Contoh permasalahan yang dapat diselesaikan dengan metode heuristik dalam bidang matematika adalah menemukan pola, membuat gambar, memodifikasi masalah, dan lain-lain.

Metode algoritmik digunakan untuk mendapatkan solusi dari permasalahan menggunakan langkah-langkah formal dan terstruktur atau sistematis untuk setiap tahap. Metode algoritmik berkaitan erat dengan algoritma. Algoritma sering dikorelasikan dengan penulisan kode dalam proses pembuatan perangkat lunak komputer. Akan tetapi, algoritma tidak hanya dapat digunakan untuk membuat sebuah perangkat lunak komputer saja. Algoritma dapat pula menyelesaikan persoalan dalam berbagai lingkup permasalahan.

Program yang baik tidak hanya dapat menyelesaikan masalah. Program yang baik adalah program yang dapat menyelesaikan masalah dengan algoritma yang baik dan efektif. Salah satu algoritma yang cukup sering digunakan adalah algoritma *Divide and Conquer*. Algoritma ini memecah suatu masalah menjadi beberapa masalah yang lebih kecil dan memungkinkan untuk diselesaikan. Setelah solusi-solusi dari permasalahan kecil ditemukan, kumpulan solusi tersebut digabungkan kembali untuk menjadi solusi dari masalah semula.

Dalam bidang ilmu matematika, terdapat sebuah kajian mengenai aljabar linear yaitu transformasi linear yang direpresentasikan oleh matriks. Sebuah matriks yang berukuran $n \times n$ memiliki nilai eigen (*eigenvalue)* dan vektor eigen (*eigenvector*).

Nilai eigen berperan sangat penting dalam studi persamaan diferensial. Selain itu, nilai eigen juga digunakan secara luas dalam berbagai aplikasi ilmu fisika. Karena nilai eigen dapat diaplikasikan dalam cakupan yang cukup luas, maka dibutuhkan perhitungan yang cepat dan tepat. Salah satu pendekatan yang dapat dilakukan untuk menemukan solusinya adalah melalui algoritma *divide and conquer*. Dalam makalah ini, akan dibahas bagaimana penerapan algoritma *divide and conquer* pada perhitungan nilai eigen suatu matriks bujur sangkar.

II. DASAR TEORI

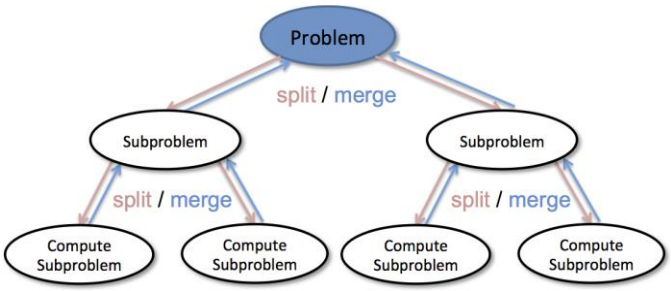
A. Algoritma Divide and Conquer

Pada jaman penjajahan Belanda terdapat sebuah strategi militer bernama *divide ut imperes* atau yang dikenal juga dengan sebutan *divide et impera* (pecah belah lalu kuasai). Awal mula dari strategi ini adalah ketika terdapat seorang jenderal perang mengamati bahwa lebih mudah mengalahkan satu pleton yang berkekuatan 50.000 orang, lalu mengalahkan satu pleton lainnya yang berkekuatan 50.000 orang juga dibandingkan langsung mengalahkan satu pleton yang beranggotakan 100.000 orang tentara. Sehingga Jenderal tersebut melakukan penyerangan dengan cara memecah tentara musuh menjadi dua bagian lalu menghancurkan musuhnya satu per satu. Sekarang strategi tersebut menjadi strategi fundamental di dalam ilmu komputer dengan nama *divide and conquer.*

Pada ilmu komputer konteks *divide and conquer* memiliki makna yang tidak jauh beda dengan bidang militer. Perbedaan yang ada yaitu pada bidang militer kekuatan musuh perlu dipecah untuk mempermudah memenangkan peperangan, sementara dalam ilmu komputer kekuatan musuh yang dimaksud adalah masalah yang dihadapi.

Algoritma *divide and conquer* terdiri dari beberapa tahapan, yaitu *divide, conquer (solve), dan combine*. *Divide* adalah

membagi persoalan menjadi beberapa bagian yang memiliki kemiripan dengan persoalan semula namun berukuran lebih kecil (idealnya berukuran hampir sama). *Conquer (solve)* merupakan tahapan untuk menyelesaikan persoalan-persoalan kecil tersebut, dengan menggunakan cara rekursif. *Combine* merupakan tahapan untuk menggabungkan masing-masing solusi sehingga membentuk solusi untuk permasalahan semula. Pada Gambar 1 diberikan ilustrasi dari algoritma *divide and conquer*.



Gambar 1 Pohon algoritma *divide and conquer*
[Sumber:http://bigdata.ices.utexas.edu/wp-content/uploads/2014/03/divide-and-conquer1.png]

Skema umum algoritma *divide and conquer* adalah sebagai berikut dapat dilihat dalam bentuk kode-semu (*pseudocode*) pada Prosedur 1.

```
Procedure DIVIDE_and_CONQUER(input n : integer)
{ Menyelesaikan masalah dengan algoritma DandC.
  Masukan : masukan yang berukuran n
  Keluaran: solusi dari masalah semula
}
Deklaasi
       r, k : integer
Algoritma
  if n ≤ n0 then {ukuran sudah cukup kecil}
       SOLVE upa-masalah yang berukuran n ini
  else
       Bagi jadi r upa-masalah, berukuran n/k
       for masing-masing dari r upa-masalah do
            DIVIDE_and_CONQUER(n/k)
       endfor
       COMBINE {solusi dari r upa-masalah jd
                solusi masalah semula}
  endif
```

Prosedur 1 Pseudocode algoritma *divide and conquer*

Kompleksitas waktu dari algoritma d*ivide and conquer* adalah,

$$T(n) = \begin{cases} g(n) & , n \le n_0 \\ 2T(n/2) + f(n) & , n \le n_0 \end{cases}$$

dengan,

$T(n) =$ waktu komputasi *divide and conquer* dengan masukan berukuran n,
$g(n) =$ waktu komputasi untuk menyelesaikan sub-masalah yang kecil (waktu komputasi fungsi SOLVE),
$f(n) =$ waktu komputasi untuk menggabungkan solusi masing masing sub-masalah (waktu komputasi fungsi COMBINE).

B. Definisi Nilai Eigen

Misalkan terdapat matriks **T** berukuran $n \times n$. Ketika terdapat sebuah persamaan $\mathbf{T}x = \lambda x$ merupakan persamaan yang *valid* dan vektor x tak nol, maka dapat dikatakan bahwa λ adalah *eigenvalue* dari **T** sedangkan x adalah *eigenvector*, hal ini akan ekuivalen dengan persamaan linear $(\mathbf{T} - \lambda\mathbf{I})x = 0$ dengan **I** adalah matriks identitas. $\mathbf{T}x = \lambda x$ dan $(\mathbf{T} - \lambda\mathbf{I})x = 0$ ekuivalen. Untuk x tak nol, persamaan ini hanya akan memiliki solosi jika $\det(\boldsymbol{T} - \lambda\mathbf{I}) = 0$.

Misalkan terdapat $\mathbf{T} = \begin{bmatrix} 3 & 2 \\ 7 & -2 \end{bmatrix}$, polinom karakteristik dari matriks tersebut adalah,

$$\begin{aligned} p(\lambda) = \det(\boldsymbol{T} - \lambda\boldsymbol{I}) &= \det\begin{bmatrix} 3-\lambda & 2 \\ 7 & -2-\lambda \end{bmatrix} \\ &= (3-\lambda)(-2-\lambda) - 14 \\ &= \lambda^2 - \lambda - 20 \\ &= (\lambda - 5)(\lambda + 4) \qquad = 0 \end{aligned}$$

Dari persamaan di atas di dapatkan bahwa nilai eigen matriks **T** adalah 5 dan -4.

C. Matriks Tridiagonal

Terdapat berbagai macam matriks, salah satunya adalah matriks tridiagonal. Matriks tridiagonal mirip dengan matriks diagonal. Matriks diagonal adalah matriks yang semua elemennya bernilai tidak nol hanya pada diagonal utamanya, sedangkan matriks tridiagonal adalah matriks yang elemennya tidak nol pada diagonal utama, diagonal pertama di bawah diagonal utama dan diagonal pertama di atas diagonal utama. Contoh matriks tridiagonal adalah

$$\begin{bmatrix} 1 & 4 & 0 & 0 \\ 3 & 4 & 1 & 0 \\ 0 & 2 & 3 & 4 \\ 0 & 0 & 1 & 3 \end{bmatrix}$$

D. LAPACK

Linear Algebra Package (LAPACK) adalah suatu *library* perangkat lunak yang disediakan untuk membantu menyelesaikan persoalan atau perhitungan aljabar linear numerik. *Library* ini menyediakan beberapa fungsi bawaan untuk memecahkan sistem persamaan linear, menemukan nilai eigen, dekomposisi nilai *singular*, dan lain-lain. Awalnya LAPACK ditulis dalam bahasa Fortran 77, namun kini LAPACK ditulis menggunakan bahasa Fortran 90.

LAPACK dapat dilihat sebagai kelanjutan dari *linear equation package*. LAPACK bergantung pada *Basic Linear Algebra Subprogram* (BLAS) yang secara efektif mengeksploitasi *cache* pada arsitektur komputer modern yang berbasis *cache*. LAPACK berada di bawah lisensi *three-clause* BSD, yaitu sebuah lisensi perangkat lunak gratis dengan beberapa batasan.

III. Penggunaan Algoritma Divide and Conquer pada Pencarian Nilai Eigen

A. Pencarian Nilai Eigen secara Matematis

Misalkan terdapat matriks **T** berukuran $n \times n$. Sebelum mencari nilai eigen dengan algoritma *divide and conquer*, matriks **T** harus direduksi menjadi matriks tridiagonal. Hal ini diperlukan untuk tahap *divide*. Untuk mencari nilai eigen, persoalan tersebut (matriks) dapat dibagi menjadi beberapa blok matriks diagonal, yaitu $\mathbf{T} = \begin{bmatrix} T_1 & 0 \\ 0 & T_2 \end{bmatrix}$.

1) Tahap Divide

Pada algoritma *divide and conquer*, matriks akan dibagi menjadi beberapa bagian yaitu menjadi matriks tridiagonal (hampir memblok diagonal).

$$T = \begin{bmatrix} T_1 & \beta \\ \beta & T_2 \end{bmatrix}$$

Untuk *submatrix* T_1 berukuran $m \times m$, maka T_2 akan berukuran $(n-m) \times (n-m)$ dengan $m \approx n/2$. Jika ditulis ulang, matriks **T** yang sudah dibentuk menjadi blok matriks diagonal ditambang dengan koreksi rank-1, maka matrks **T** akan menjadi,

$$T = \begin{bmatrix} \hat{T}_1 & 0 \\ 0 & \hat{T}_2 \end{bmatrix} + \begin{bmatrix} \beta & \beta \\ \beta & \beta \end{bmatrix}$$

Gambar 2 Ilustrasi perubahan T_1 dan T_2

Elemen pada T_1 bagian kanan bawah berbeda dengan $\hat{T}_1$. Elemen kanan bawah pada $\hat{T}_1 = t_{m,m} - \beta$, dengan $t_{m,m}$ merupakan elemen pada baris ke-*m* dan kolom ke-*m* pada matriks **T**. Hal ini juga berlaku pada $\hat{T}_2$, yaitu elemen kiri atas dari $\hat{T}_2 = t_{m+1,m+1} - \beta$. Untuk lebih jelas akan diilustrasikan dengan Gambar 3,

$$T = \left[\begin{array}{cc|cc} a_1 & b_1 & & \\ b_1 & \ddots & \ddots & \\ & \ddots & a_{m-1} & b_{m-1} \\ & & b_{m-1} & a_m \\ \hline & & b_m & \end{array}\right.$$

$$\left.\begin{array}{cc} b_m & \\ a_{m+1} & b_{m+1} \\ b_{m+1} & \ddots \\ & \ddots \\ & b_{n-1} \\ b_{n-1} & a_n \end{array}\right]$$

$$= \left[\begin{array}{cc|cc} a_1 & b_1 & & \\ b_1 & \ddots & \ddots & \\ & \ddots & a_{m-1} & b_{m-1} \\ & & b_{m-1} & a_m - b_m \\ \hline & & a_{m+1} - b_m & b_{m+1} \\ & & b_{m+1} & \ddots \\ & & & \ddots \\ & & & b_{n-1} \\ & & b_{n-1} & a_n \end{array}\right]$$

$$+ \left[\begin{array}{cc|cc} & & & \\ & & & \\ & & b_m & b_m \\ \hline & & b_m & b_m \\ & & & \\ & & & \end{array}\right]$$

$$= \begin{bmatrix} T_1 & 0 \\ 0 & T_2 \end{bmatrix} + b_m \cdot \begin{bmatrix} 0 \\ \vdots \\ 0 \\ 1 \\ 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix} [0, \ldots, 0, 1, 1, 0, \ldots, 0] \equiv \begin{bmatrix} T_1 & 0 \\ 0 & T_2 \end{bmatrix} + b_m v v^T.$$

Gambar 3 Langkah dari konversi dari T_1 dan T_2 matriks T

Mencari nilai eigen dari setiap sub-masalah merupakan tahap terakhir dalam tahap *divide*. Penyelesaiannya yaitu, jika terdapat matriks awal, yaitu matriks **A**, maka pencarian nilai eigen setiap permasalahan kecil dapat dirumuskan dengan,

$\hat{T}_1 = Q_1 A_1 Q_1^T$ dan $\hat{T}_2 = Q_2 A_2 Q_2^T$ (Implicit Q Theorem)

2) Tahap Conquer

Pada tahap *conquer* adalah tahap untuk menyatukan kembali submatriks yang ada menjadi matriks semula, melalui

$$\begin{aligned} T &= \begin{bmatrix} T_1 & 0 \\ 0 & T_2 \end{bmatrix} + b_m v v^T \\ &= \begin{bmatrix} Q_1 \Lambda_1 Q_1^T & 0 \\ 0 & Q_2 \Lambda_2 Q_2^T \end{bmatrix} + b_m v v^T \\ &= \begin{bmatrix} Q_1 & 0 \\ 0 & Q_2 \end{bmatrix} \left(\begin{bmatrix} \Lambda_1 & \\ & \Lambda_2 \end{bmatrix} + b_m u u^T \right) \begin{bmatrix} Q_1^T & 0 \\ 0 & Q_2^T \end{bmatrix}, \end{aligned}$$

dengan $u = \begin{bmatrix} Q_1^T & 0 \\ 0 & Q_2^T \end{bmatrix}$ dan $v = \begin{bmatrix} \text{kolom terakhir } Q_1^T \\ \text{kolom pertama } Q_2^T \end{bmatrix}$

Sehingga, nilai eigen dari matriks **T** memiliki nilai yang sama dengan matriks $\mathbf{D} + \beta u u^T$ dengan $\mathbf{D} = \begin{bmatrix} A_1 & 0 \\ 0 & A_2 \end{bmatrix}$ adalah

matriks diagonal dan $\beta = b_m$ adalah skalar dan u adalah sebuah vektor. Asumsikan $\mathbf{D} - \lambda\mathbf{I}$ tidak singular, kemudian menghitung polinom karakteristik dari persamaan,

$$\det(\mathbf{D} + \beta uu^T - \lambda\mathbf{I}) = \det((\mathbf{D} - \lambda\mathbf{I})(\mathbf{I} + \beta(\mathbf{D} - \lambda)^{-1}uu^T))$$

Karena $\mathbf{D} - \lambda\mathbf{I}$ tidak singular, maka

$\det\big((\mathbf{D} - \lambda\mathbf{I})(\mathbf{I} + \beta(\mathbf{D} - \lambda)^{-1}uu^T)\big) = 0$ dan $(\mathbf{I} + \beta(\mathbf{D} - \lambda)^{-1}uu^T)$ adalah identitas dari rank-1 sebuah matriks dan mencari determinan menjadi lebih mudah karena,

$$det(\mathbf{I} + \beta(\mathbf{D} - \lambda)^{-1}uu^T) = 1 + \beta u^T(\mathbf{D} - \lambda)^{-1}u$$

$$= 1 + \beta\sum_{i=1}^{n}\frac{u_i^2}{d_1 - \lambda} \equiv f(\lambda) = 0$$

B. Kode untuk Mencari Nilai Eigen Menggunakan Algoritma *Divide and Conquer*

Algoritma *divide and conquer* diimplementasikan dalam kode yang menggunakan LAPACK *serial routines.* Berikut ini potongan dari kode program,

```
SUBROUTINE SSTEDC (COMPZ, N, D, E, Z, LDZ, WORK,
LWORK, IWORK, LIWORK INFO)
* SSTEDC adalah fungsi untuk mencari nilai eigen
* dan/atau vektor eigen dari symmetric
* tridiagonal matrix.
* COMPZ adalah CHARACTER*1.
* Jika = 'N' : menghitung nilai eigen saja
* Jika = 'I' : menghitung vektor eigen juga
* Jika = 'V' : menghitung vektor eigen dari
       matriks orginial. Dan Z akan bernilai
       orthogonal matriks yang digunakan untuk
       merduksi original matriks ke bentuk
       matriks tridiagonal
* N adalah ukuran dari matriks. Prekondisi: N>=0
* D adalah array bil. real dengan ukuran (N)
* E adalah array bil. Real berukuran (N-1)
* Z adalah array bil. real berukuran (LDZ,N)

* LDZ adalah leading dimension dari array Z,
* Prekondisi: LDZ>=1
* Jika akan mencari vektor eigen juga, maka
* LDZ >= max(1,N)

* WORK adalah array bil. real berdimensi
* MAX(1,LWORKD)
* Jika INFO = 0, WORK(1) mengembalikan nilai
* optimal dari LWORK

* LWORK adalah ukuran dari array WORK
* Jika COMPZ = 'N' atau N <= 1 maka LWORK
* setidaknya 1
* Jika COMPZ = 'V' dan N > 1 maka LWORK harus
* lebih besar dari (1 + 3*N  + 2*N*log(N) +
* 4*N^2)
* Jika COMPZ = 'I' atau 'V; maka jika
* N <= ukuran minimum pemecahan (25) maka
* LWORK adalah max(1,2*(N-1))
* Jika LWORK = -1 maka procedure akan mehitung
* optimal size dari array yang ada pada WORK.
* IWORK adalah array integer berukuran
* (MAX(1,LIWORK)
* Jika INFO = 0, IWORK (1) mengembalikan LIWORK
* optimal

* LIWORK ukuran dari IWORK
* Jika COMPZ = 'N' atau N <= 1 maka LIWORK
* setidaknya 1
* Jika COMPZ = 'V' dan N > 1 mmaka LIWORK
* minimal (6 + 6*N + 5*N*log(N))
* Jika COMPZ = 'I' atau 'V', maka jika N
* kurang dari atau sama dengan ukuran minimu
* pembagian, biasanya 25, maka LIWORK akan 1
* Jika LIWORK = -1, dapat diasumsikan bahwa
* LIWORK akan bernilai nilai pertama dari
* elemen pertama array IWORK

* INFO bernilai 0 jika berhasil keluar
* <0 jika INFO = -i, dan argumen tersebut
* merupakan nilai ilegal
* >0 ketika algoritma tidak dapat menemukan
* nilai eigen dengan submatriks dengan baris
* dan columns yaitu INFO/(N+1) mod (INFO, N+1)
```

Prosedur 2 Potongan kode program

C. Hasil Pengujian

Pengujian dilakukan dengan memasukan matriks yang akan dicari nili eigennya. Sebagai default, matriks selalu disebut sebagai matriks **A**. Masukan matriks diawali dengan "[" dan diakhir dengan "]". Cara penulisan masukan yaitu, ditulis elemen dari sebuah baris. Pergantian dari baris ke $2 \ldots n-1$ adalah karakter titik koma ";". Berikut ini merupakan data beberapa pengujian.

Input:

```
A = [1,4,0,0;3,4,1,0;0,2,3,4;0,0,1,3]
```

Gambar 4 Input matriks percobaan pertama

Output matriks :

```
A =

     1     4     0     0
     3     4     1     0
     0     2     3     4
     0     0     1     3

e =

   -1.4535
    1.0000
    6.7982
    4.6553
```

Gambar 5 Output matriks percobaan pertama dan nilai eigen matriks percobaan pertama

Input:

```
A = [10,1,0,0,0;5,20,2,0,0;0,6,30,3,0;
     0,0,7,40,4;0,0,0,8,50]
```

Gambar 6 Input matriks percobaan kedua

Output matriks :

```
A =

    10     1     0     0     0
     5    20     2     0     0
     0     6    30     3     0
     0     0     7    40     4
     0     0     0     8    50
e =

    9.4949
   19.2931
   29.1115
   39.3834
   52.7171
```

*Gambar 7 Output matriks percobaan kedua dan nilai eigen matriksp*ercobaan kedua

Input:

```
A = [3,1,0,0,0,0,0;1,2,1,0,0,0,0;
     0,1,1,1,0,0,0;0,0,1,1,1,0,0;
     0,0,0,1,1,1,0;0,0,0,0,1,2,1;
     0,0,0,0,0,1,3]
```

Gambar 8 Input matriks percobaan ketiga

Output matriks :

```
A =

     3     1     0     0     0     0     0
     1     2     1     0     0     0     0
     0     1     1     1     0     0     0
     0     0     1     1     1     0     0
     0     0     0     1     1     1     0
     0     0     0     0     1     2     1
     0     0     0     0     0     1     3
e =

   -0.6412
    0.2679
    1.2763
    2.0000
    2.5892
    3.7321
    3.7757
```

Gambar 9 Output matriks percobaan ketiga dan nilai eigen matriks percobaan ketiga

IV. ANALISIS

A. Analisis Kompleksitas

Dengan menggunakan teorema master, *running time* dari program ini dapat dianalisis. Karena $m \approx n/2$, dapat dituliskan hubungan dari pengulangan yang terjadi sebagai berikut,

$$T(n) = 2 \times T\left(\frac{n}{2}\right) + \theta(n^2)$$

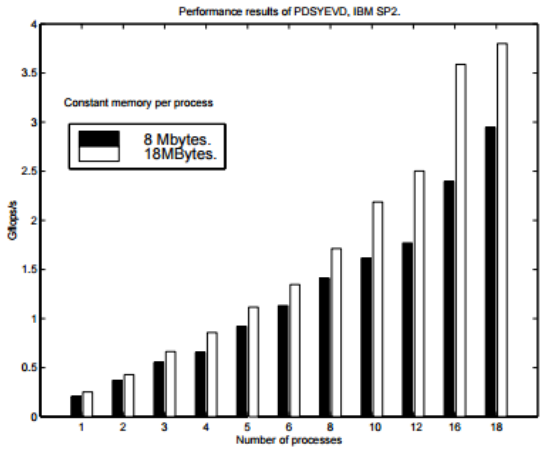
Dalam notasi Teorema Master, $a = b = 2$ sehingga $\log_b a = 1$. Sehingga didapat $\theta(n^2) = \Omega(n)$ dan $T(n) = \theta(n^2)$. Jadi, kompleksitas waktu dari pencarian semua nilai eigen sebuah matriks dapat ditentukan dengan

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

B. Perbandingan Dua komputer Berbeda

Dengan menerapkan algoritma *divide and conquer* pada perhitungan nilai eigen, didapat hasil yang cenderung variatif. Hal ini juga tergantuk oleh spesifikasi komputer yang

digunakan. Berikut ilustrasi hasil perhitngan dalam bentuk grafik dua buah komputer



Gambar 10 Perhitungan nilai eigen pada dua buah komputer[2]

Grafik di atas menunjukan penggunaan algoritma *divide and conquer* pada proses perhitungan nilai eigen memiliki perbedaan kecepatan antara komputer dengan memori proses 8 Mbytes dan 18 Mbytes. Walaupun pada komputer yang memiliki memori proses 8 Mbytes lebih lambat, namun proses ini masih dapat dikategorikan dalam waktu yang relatif cepat.

V. KESIMPULAN

Penerapan algoritma *divide and conquer* sangat baik untuk menyelesaikan permasalahan pencarian nilai eigen. Karena dengan algoritma *divide and conquer*, pencarian nilai eigen dari matriks yang berukuran cukup besar dapat ditemukan dalam waktu yang singkat. Algoritma *divide and conquer* saat ini dikategorikan sebagai salah satu algoritma tercepat untuk menemukan semua nilai eigen suatu matriks.

VI. UCAPAN TERIMA KASIH

Penulis mengucapkan terima kasih kepada Tuhan Yang Maha Esa atas segala berkat yang telah diberikannya kepada penulis sehingga makalah ini dapat diselesaikan dengan baik.

Selanjunya, penulis ingin mengucapkan terima kasih kepada pihak-pihak yang telah membantu serta memberikan masukan dan bimbingan dalam penyelesaian makalah ini :

1. Bapak Dr.Ir. Pak Rinaldi Munir, M.T. dan Ibu Dr. Nur Ulfa Maulidevi, S.T., M.Sc. selaku pengajar mata kuliah IF2211 Strategi Algoritma atas segala bimbingan serta ilmu yang telah diberikan kepada penulis.
2. Teman-teman yang telah memberikan ide dan inspirasi tentang bagaimana peluang adanya kesamaan sidik jari individu.
3. Pihak-pihak lain yang telah membantu.

REFERENSI

[1] Munir, Rinaldi. 2009. *Diktat Kuliah Strategi Algoritmik IF2251 Strategi Algoritmik.* Departemen Teknik Informatika ITB.

[2] W. Cheney dan D. Kincaid. 2008. *Numerical Mathematics and Computing 6th Ed.*. Thomson Brooks/Cole, USA.

[3] Demmel, James. 1997. *Applied Nuberical Linear Algebra.* Philadelphia.

[4] http://www.netlib.org/utk/people/JackDongarra/PAPERS/104_1999_a-parallel-divide-and-conquer-algorithm.pdf, 5 Mei 2016

[5] https://ristikhoirunnisa.wordpress.com/2014/01/03/pendekatan-heuristik-untuk-meningkatkan-kemampuan-berpikir-kritis-dalam-materi-dalil-pythagoras/, 8 Mei 2016

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, 7 Mei 2015

Elvina Riama K. Situmorang
13514045