LAPORAN TUGAS 1

DESAIN ANALISIS ALGORITMA

IMPLEMENTASI METODE *DIVIDE AND CONQUER* PADA ALGORITMA *SELECTION SORT* DAN *MERGE SORT*

Oleh:

Ahmad Riau Ardi 19102241

Kelas: S1IF-07-MM2

PROGRAM STUDI S1 TEKNIK INFORMATIKA

FAKULTAS TEKNOLOGI INDUSTRI DAN INFORMATIKA

INSTITUT TEKNOLOGI TELKOM PURWOKERTO

2022

A. Dasar Teori

Divide and Conquer merupakan salah satu metode pada pemrograman di mana suatu permasalahan dibagi menjadi beberapa bagian dengan syarat masalah yang dibagi memiliki kesamaan. Pada algoritma ini alur pemecahan masalah dibagi menjadi tiga tahap, yaitu *divide* atau membagi masalah menjadi beberapa sub-masalah dengan ukuran lebih kecil, *conquer* atau penyelesaian masalah(umumnya secara rekursif) dan *combine* atau menggabungkan solusi-solusi sub-masalah sehingga membentuk solusi bagi masalah utama [1]. Metode *Divide and Conquer* dapat diimplementasikan dalam beberapa algoritma, salah satunya pada algoritma pengurutan. Algoritma pengurutan yang bisa diimplemtasikan metode ini ialah algoritma pengurutan *merge sort* dan *selection sort.*

Selecton sort merupakan bentuk perbaikan dari algoritma *sorting bubble sort*. Algoritma ini bekerja dengan mengurangi jumlah pembanding. Algoritma ini akan memilih salah satu nilai dari himpunan nilai kemudian membandingkannya dengan himpunan nilai yang belum diurutkan. Kemudian menukar nilai tersebut dengan salah satu nilai yang lebih kecil atau lebih besar [2].

Sedangkan *Merge sort* merupakan pengembangan dari *quick sort.* Algoritma *sorting* ini secara eksplisit menggunakan metode *divide and conquer* dalam penyelesaiannya [3]. Algoritma ini ditemukan pada tahun 1945 oleh John von Neumann dengan tujuan mempermudah pengurutan rangkaian data yang tidak memungkinkan untuk ditampung oleh memori komputer kala itu. Algoritma ini bekerja dengan terus membagi rangkaian data menjadi dua bagian hingga tidak bisa dibagi. Kemudian setiap bagian yang telah dibagi dibandingkan satu sama lain hingga menjadi teurut [4].

B. Implementasi

Berikut merupakan bentuk *pseudocode* dari program yang dibuat.

1. *Pseudocode Selection Sort*

Pseudocode berikut berguna agar program dapat menentukan index elemen array dengan nilai terkecil dari kumpulan array yang belum terurut.

```
Procedure index_terkecil(input data: Array, input/output index: integer, input n: integer)
{Mencari index elemen array dengan nilai terkecil diantara kelompok array yang belum diurutkan, kemudian men-return nilai index sebgai index elemen yang akan dibandingkan dengan elemen array terkecil di kelompok array terurut.
Masukan: array data[0...n-1] yang belum terurut
         index adalah index elemen yang sedang dibandingkan
         n adalah jumlah elemen array data
```

```
Luaran: index elemen bernilai minimun dari bagian array data yang belum terurut.
}

Deklarasi:
        // key sebagai index dari elemen array terkecil di kumpulan array tak teurut
        key : integer

Algoritma:
        // jika nilai index sama dengan n, maka fungsi mengembalikan nilai index
        // hal ini berarti pengurutan sudah mencapai membandingkan elemen ke n-1
        // dengan elemen ke n
        If index == n then
                Return index
        Endif

        // nilai key diisi dengan return fungsi index_terkecil(rekursif)
        key ← index_terkecil(data[], index, n-1)

        // jika nilai data[index] (elemen ke n+1)  lebih kecil dari
        // data[key] (elemen terkecil dari array tak terurut)
        If data[index] < data[key] then
                // jika true, maka return index ke n+1
                Return index
        Else
                // jika false, maka return index elemen terkecil dari array tak teurut
                Return key
        Endif
```

Pseudocode di atas berguna untuk mengurutkan elemen-elemen yang ada di array dengan algoritma *selection sort*.

```
Procedure selectionSort(input/output data : Array, input n: integer, input index:
integer)
{ Mengurutkan array data[0...n-1] menggunakan algoritma selection sort, dengan n
adalah panjang array data dan index sebagai counter index array.
Masukan: array data[0...n-1] yang elemen-elemennya belum terurut
            n adalah jumlah elemen array data
            index adalah counter index elemen yang dibandingkan
Luaran: array data[0...n-1] yang sudah terurut
}
Deklarasi:
        // key sebagai nilai index yang akan dibandingkan dengan index ke n
        key : integer
```

Algoritma:

// jika nilai index sama dengan n, selection sort selesai
If index == n
// nilai -1 dapat diartikan sebagai proses berhenti
Return -1
Endif

// nilai key diisi nilai dari index dengan elemen terkecil dari array tak teurut
key ← index_terkecil(data, index, n-1)

// jika nilai key dan index sama, berarti elemen ke-n sudah yang paling kecil
// jika nilai key dan index tidak sama
If key != index **then**
// maka nilai data[key] (elemen dari array tak terurut)
// dan data[index] (elemen yang dibandingkan saat ini)
Swap(data[key], data[index])
Endif

// memanggil kembali fungsi selection sort untuk mengecek elemen di index
// selanjutnya
selectionSort(data, n, index+1)

2. *Pseudocode Merge Sort*

Berikut ini merupakan *pseudocode* untuk *merge sort* yang digunakan untuk mengurutkan data pada projek ini.

Procedure *mergeSort*(***input/output*** data*:* Array)
{ Algoritma untuk mengurutkan data dalam array utama dengan membagi array utama menjadi 2 bagian secara terus menerus hingga tidak dapat dibagi lagi. Kemudian array-array yang telah dibagi, elemen-elemennya saling dibandingkan lalu digabungkan kembali ke array utama.
Masukan: array data[] berupa array yang elemen-elemennya belum terurut
Luaran: array data[] yang elemen-elemennya telah diurutkan
}
Deklarasi:
// array untuk menampung elemen-elemen yang terbagi
L_side, R_side : Array
// variable untuk counter index array
L_count, R_count, key : integer
// variable untuk menampung nilai tengah
indexTengah : integer

Algoritma:

```
// jika jumlah elemen array > 1, maka
If len(data) > 1 then
    // maka bagi jumlah elemen menjadi 2 dibulatkan ke nilai terdekat
    indexTengah ← len(data) div 2

    // deklarasi isi array L_side dan R_side
    // L_side berisi elemen dari 0 hingga indexTengah array data
    L_side ← data[0...indexTengah]
    // R_side berisi elemen dari indexTengah+1 hingga akhir array data
    R_side ← data[indexTengah+1 … len(data]

    // proses membagi sisi kiri dan kanan array hingga tidak bisa dibagi
    mergeSort(L_side)
    mergeSort(R_side)

    // deklarasi nilai L_count dan R_count
    // untuk indexing array L_side dan R_side
    L_count ← 0
    R_count ← 0

    // deklarasi nilai key sebagai index array utama
    Key ← 0

    // dilakukan perulangan untuk menggabungkan elemen yang ada di
    // array sisi kiri dan kanan ke dalam array utama
    While L_count < len(L_side) and R_count < len(R_side) do

        // jika di array isi kiri > array di sisi kanan
        If L_Side[L_count] < R_side[R_count] then
            // maka elemen di sisi kiri dipindahkan ke array utama
            data[key] ← L_side[L_count]
            // L_count ditambah 1 untuk geser ke elemen selanjutnya
            L_count ← L_count + 1
        Else
            // sebalikanya, elemen di sisi kanan yang dipindahkan
            data[key] ← R_side[R_count]
            // L_count ditambah 1 untuk geser ke elemen selanjutnya
            R_count ← R_count + 1
        Endif
        // nilai key ditambah 1 untuk bergeser ke index selanjutnya
        key ← key + 1
    EndWhile
```

```
            // terkadang jumlah elemen di sisi kiri dan kanan tidak sama,
            // perulangan di bawah ini untuk memindahkan sisa elemen

            // apabila masih ada elemen di array kiri
            While L_count < len(L_side) do
                // elemen tersebut dipindahkan ke array utama
                data[key] ← L_side[L_count]
                // index array sisi kiri digeser ke index selanjutnya
                L_count ← L_count + 1
                // index array utama digeser ke index selanjutnya
                key ← key + 1
            EndWhile

            // apabila masih ada elemen di array kanan
            While R_count < len(R_side) do
                // elemen tersebut dipindahkan ke array utama
                data[key] ← R_side[R_count]
                // index array sisi kanan digeser ke index selanjutnya
                R_count ← R_count + 1
                // index array utama digeser ke index selanjutnya
                key ← key + 1
            EndWhile
        Endif
```

3. ***Script Program***

Selection Sort

```
# 19102241 - Ahmad Riau Ardi - S1IF07MM2

import json  # library untuk membaca file json
import time  # library untuk menampilkan format waktu

# deklarasi variable untuk menhitung waktu program
execute_time = time.time()

def index_terkecil(data, index, n):  # fungsi Return index terkecil
    # data berupa array
    # index untuk nilai index awal
    # n berupa int jumlah elemen array - 1,

    # jika nilai index sama dengan n (elemen terakhir di array)
    if index == n:
        # maka return nilai index
        return index
```

```
    # rekursif mencari elemen terkecil dari sisa array yang belum disort
    key = index_terkecil(data, index + 1, n)

    # Return nilai index elemen terkecil jika nilai data[index] < data[key]
    # jika tidak lebih kecil, maka return nilai key
    return (index if data[index] < data[key] else key)

def selectionSort(data, n, index=0):  # fungsi selection sort
    # data berupa array
    # n berupa int jumlah elemen array,
    # index untuk nilai index awal

    # ketika index dan ukuran array sama
    if index == n:
        # return nilai -1
        return -1

    # Memanggil fungsi index_terkecil untuk mencari index elemen
    # yang lebih kecil dari index elemen yang terakhir disort
    key = index_terkecil(data, index, n-1)

    # jika nilai key tidak sama dengan index
    if key != index:
        # menukar elemen data[index] dengan dengan data[key]
        data[key], data[index] = data[index], data[key]

    # memanggil selection sort lagi hingga nilai index == n
    # dengan nilai index + 1 atau index selanjutnya
    selectionSort(data, n, index + 1)

# fungsi main program
if __name__ == '__main__':
    # memanggil berkas berisi data
    berkas = open("./data/sauce100.json")
    # isi dari berkas dimuat ke json_data
    json_data = json.load(berkas)
    # variable untuk menampung isi json_data
    data = []

    # perulangan untuk memasukan isi json ke dalam array 'data'
    for i in json_data['nama']:
        # memasukan nilai 'i' ke dalam array data
        data.append(i)
```

```
print("\n=========================================")

# mengetahui jumlah data pada list
n = len(data)
print("Jumlah data : ", n)

print("\n=========================================")

# uji coba sorting sebanyak 10 kali untuk melihat seberapa cepat algoritma
for i in range(10):
    # penampung sementara data
    data_temp = data
    # sorting array
    selectionSort(data_temp, n)
    # deklarasi variable berisi lama waktu eksekusi program
    hasil_execute = time.time() - execute_time
    # menampilkan waktu eksekusi program
    print("Waktu eksekusi ke-", i, ":  %s detik" % (round(hasil_execute, 5)))

print("=========================================")
# mencetak output proses selection sort
print("Data setelah diurutkan : \n")
# perulangan untuk menampilkan setiap item(nilai) pada array data
for item in data_temp:
    print(item, end=", ")

print("\n=========================================")
# untuk menjeda program
input("\nTekan enter")
```

Merge Sort

```
# 19102241 - Ahmad Riau Ardi - S1IF07MM2

import json  # library untuk membaca file json
import time  # library untuk menampilkan format waktu

# deklarasi variable untuk menhitung waktu program
execute_time = time.time()

def mergeSort(data):  # fungsi untuk sorting, parameter data berupa array
    # sorting hanya akan dijalankan jika panjang array > 1
    if len(data) > 1:
        # mengambil index tengah
```

```
    indexTengah = len(data)//2
    # L_side berisi array sebanyak 'indexTengah' diambil dari setengah pertama array data
    L_side = data[:indexTengah]
    # R_side berisi array sebanyak 'indexTengah' diambil dari setengah terakhir array data
    R_side = data[indexTengah:]

    # recursive untuk membagi array menjadi 2 hingga tidak bisa dibagi lagi
    mergeSort(L_side)
    mergeSort(R_side)

    # variabel untuk indexing sisi kiri dan kanan array
    L_count = 0
    R_count = 0

    # untuk index array
    key = 0

    # proses penggabungan data
    while L_count < len(L_side) and R_count < len(R_side):
        # jika jumlah array kiri lebih sedikit dari jumlah array kanan
        if L_side[L_count] < R_side[R_count]:
            # elemen di array L_side dimasukkan ke data index 'key'
            data[key] = L_side[L_count]
            # naikan nilai 'L_count'
            L_count += 1

        # jika jumlah array kanan lebih sedikit dari jumlah array kiri
        else:
            # elemen di array R-side dimasukkan ke data index 'key'
            data[key] = R_side[R_count]
            # naikan nilai 'R_count'
            R_count += 1

        # tambah nilai 'key' dengan 1
        key += 1

    # mengecek jumlah elemen yang tersisa di array kanan dan kiri
    while L_count < len(L_side):
        # data index 'key' diisi elemen L_side index ke 'L_count'
        data[key] = L_side[L_count]
        # tambah nilai L_count dan key sebanyak 1
        L_count += 1
        key += 1
```

```
        while R_count < len(R_side):
            # data index 'key' diisi elemen R_side index ke 'R_count'
            data[key] = R_side[R_count]
            # tambah nilai L_count dan key sebanyak 1
            R_count += 1
            key += 1

# untuk menampilkan elemen elemen array data
def printList(data):
    # perulangan sebanyak panjang array data untuk menampilkan elemen array
    for i in range(len(data)):
        print(data[i], end=", ")

# function main
if __name__ == '__main__':
    # memanggil berkas berisi elemen array
    berkas = open("./data/sauce100.json")
    # isi dari berkas dimuat ke json_data
    json_data = json.load(berkas)
    # variable untuk menampung isi json_data
    data = []

    # perulangan untuk memasukan isi json ke dalam array 'data'
    for i in json_data['nama']:
        # memasukan nilai 'i' ke dalam array data
        data.append(i)

    print("=======================================")
    # menampilkan jumlah data
    print("Jumlah data: ", len(data))
    print("=======================================\n")

    # uji coba sorting sebanyak 10 kali untuk melihat seberapa cepat algoritma
    for i in range(10):
        # penampung sementara data
        data_temp = data
        # sorting array
        mergeSort(data_temp)
        # deklarasi variable berisi lama waktu eksekusi program
        hasil_execute = time.time() - execute_time
        # menampilkan waktu eksekusi program
        print("Waktu eksekusi ke-",i," :  %s detik" % (round(hasil_execute, 5)))
```

```
print("=========================================\n")

# menampilkan elemen array data sesudah disort
print("Menampilkan array dengan sorting ", end="\n")

# memanggil fungsi printList untuk menampilkan isi array data
print("=========================================")
printList(data_temp)

print("\n=========================================")
# untuk menjeda program
input("\nTekan enter")
```

4. Hasil *Running*

Dalam poin ini, digunakan sebuah array berisi 100 elemen string yang mana proses pengurutan dijalankan sebanyak sepuluh kali untuk masing-masing algoritma.

a. Hasil *running selection sort*

```
PS D:\Kuliah\Tingkat 3\Desain Analisis Algoritma\Tugas 1> & C:/Users/riolo/AppData/Local/Programs/Python/Python36/python.exe "d:/Kuliah/Tingkat 3/Desain Analisis Algoritma/Tugas 1/src/selection.py"

=====================================
Jumlah data :  100

=====================================
Waktu eksekusi ke- 1 :  0.00596 detik
Waktu eksekusi ke- 2 :  0.00596 detik
Waktu eksekusi ke- 3 :  0.00596 detik
Waktu eksekusi ke- 4 :  0.01424 detik
Waktu eksekusi ke- 5 :  0.01666 detik
Waktu eksekusi ke- 6 :  0.01965 detik
Waktu eksekusi ke- 7 :  0.02236 detik
Waktu eksekusi ke- 8 :  0.0245 detik
Waktu eksekusi ke- 9 :  0.02603 detik
Waktu eksekusi ke- 10 :  0.02907 detik
=====================================
Data setelah diurutkan :

Abdul, Achmad, Ade, Adi, Aditya, Agung, Agus, Agustina, Ahmad, Amalia, Amelia, Andi, Angga, Anggraini, Anita, Ari, Arief, Arif, Astuti, Aulia, Ayu, Bagus, Bayu, Budi, Citra, Devi, Dewi, Diah, Dian, Diana, Dina, Dwi, Dyah, Eka, Eko, Endah, Eva, Fajar, Fitri, Fitria, Hadi, Handayani, Hendra, Hidayat, Ika, Ilham, Indah, Indra, Intan, Irma, Kartika, Kurnia, Kurniawan, Kusuma, Lestari, Made, Maria, Maya, Muhamad, Muhammad, Munaroh, Ni, Novi, Novita, Nugroho, Nur, Nurul, Pratiwi, Prima, Puji, Puspita, Putra, Putri, Putu, Raden, Rahayu, Rahmawati,
 Ratih, Ratna, Retno, Ria, Rina, Rini, Rizki, Rizky, Saputra, Sari, Setiawan, Siti, Sri, Surya, Tri, Utami, Wahyu, Wahyuni, Widya, Wulan, Wulandari, Yulia, Yunita,
=====================================

Tekan enter
PS D:\Kuliah\Tingkat 3\Desain Analisis Algoritma\Tugas 1>
```

b. Hasil *running merge sort*

```
PS D:\Kuliah\Tingkat 3\Desain Analisis Algoritma\Tugas 1> & C:/Users/riolo/AppData/Local/Programs/Python/Python36/python.exe "d:/Kuliah/Tingkat 3/Desain Analisis Algoritma/Tugas 1/src/merge.py"
=====================================
Jumlah data:  100
=====================================

Waktu eksekusi ke- 1  :  0.01501 detik
Waktu eksekusi ke- 2  :  0.01675 detik
Waktu eksekusi ke- 3  :  0.01775 detik
Waktu eksekusi ke- 5  :  0.01775 detik
Waktu eksekusi ke- 6  :  0.01775 detik
Waktu eksekusi ke- 7  :  0.02387 detik
Waktu eksekusi ke- 8  :  0.02494 detik
Waktu eksekusi ke- 9  :  0.02688 detik
Waktu eksekusi ke- 10  :  0.02884 detik
=====================================

Menampilkan array dengan sorting
=====================================
Abdul, Achmad, Ade, Adi, Aditya, Agung, Agus, Agustina, Ahmad, Amalia, Amelia, Andi, Angga, Anggraini, Anita, Ari, Arief, Arif, Astuti, Aulia, Ayu, Bagus, Bayu, Budi, Citra, Devi, Dewi, Diah, Dian, Diana, Dina, Dwi, Dyah, Eka, Eko, Endah, Eva, Fajar, Fitri, Fitria, Hadi, Handayani, Hendra, Hidayat, Ika, Ilham, Indah, Indra, Intan, Irma, Kartika, Kurnia, Kurniawan, Kusuma, Lestari, Made, Maria, Maya, Muhamad, Muhammad, Munaroh, Ni, Novi, Novita, Nugroho, Nur, Nurul, Pratiwi, Prima, Puji, Puspita, Putra, Putri, Putu, Raden, Rahayu, Rahmawati,
 Ratih, Ratna, Retno, Ria, Rina, Rini, Rizki, Rizky, Saputra, Sari, Setiawan, Siti, Sri, Surya, Tri, Utami, Wahyu, Wahyuni, Widya, Wulan, Wulandari, Yulia, Yunita,
=====================================

Tekan enter
PS D:\Kuliah\Tingkat 3\Desain Analisis Algoritma\Tugas 1>
```

C. Pengujian

Dalam pengujian algoritma *selection sort* dan *merge sort*, digunakan spesifikasi *hardware* dan *software* sebagai berikut.

1. *Hardware*

   Prosessor : Intel Core i7-8750H 2.2GHz

   Jumlah memori : 8 Giga Byte

2. *Software*

   Sistem operasi : Windows 11 Home 64 bit

   IDE : Visual Studio Code versi 1.66.2

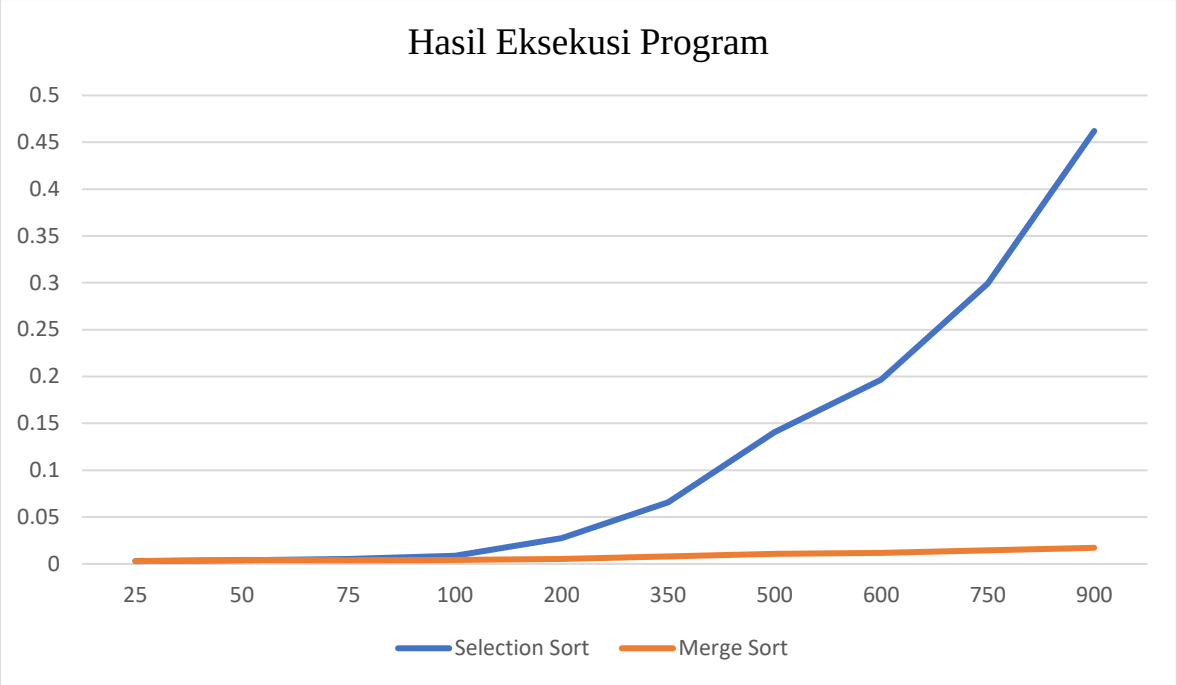
   Compiler : Python 3.6

   Bahasa pemrograman : Python

Dalam pengujian kedua algoritma, digunakan sebuah *array* berisi string dengan jumlah elemen mulai dari 25, 50, 75, 100, 500, 750 dan 1000 elemen. Elemen-elemen *array* berisi nama-nama yang biasa digunakan penduduk di Indonesia [5]. Berikut merupakan tabel berisi data rata-rata waktu eksekusi program yang dijalankan sebanyak sepuluh kali untuk sejumlah n data.

Jumlah data (n)	Rata-rata waktu eksekusi (detik)	
	Selection sort	Merge sort
25	0.00298	0.00292
50	0.00396	0.00407
75	0.00531	0.00379
100	0.00869	0.00424
200	0.02763	0.00510
350	0.06557	0.00813
500	0.14054	0.01059
600	0.19641	0.01172
750	0.29902	0.01440
900	0.46199	0.01719

Data waktu eksekusi dari *running* kedua algoritma diilustrasikan dalam grafik di halaman berikutnya.



Nilai vertikal merupakan lama waktu eksekusi, sedangkan nilai horizontal merupakan jumlah data yang diurutkan.

Berdasarkan grafik di atas, diketahui bahwa algoritma *selection sort* memerlukan waktu eksekusi yang lebih lama dibanding *merge sort*. Pada grafik diilustrasikan bahwa apabila kedua algoritma mengurutkan data dengan jumlah terbilang kecil(25 hingga 100 data) tidak terlihat perbedaan yang signifikan. Namun, ketika jumlah data mencapai lebih dari ratusan data, barulah terlihat perbedaan kedua algoritma. Peningkatan waktu eksekusi yang signifikan terjadi pada algoritma *selection sort.* Sedangkan pada *merge sort*, peningkatan waktu eksekusi terbilang lambat.

D. Analisis Hasil Pengujian

1. Komplesitas waktu algortima secara teori

a. *Selection sort*

Kompleksitas waktu

- Pemanggilan rekursif untuk mencari index elemen terkecil dengan nilai n dikurangi 1 untuk menandai elemen yang telah di cek → T(n-1).
- Pada pemanggilan rekursif fungsi *selection sort* nilai index ditambah 1. Hal ini dapat diartikan sebagai perulangan karena fungsi *selection sort* akan berhenti berulang jika nilai index sama dengan n(jumlah elemen) → n
- Diketahui **T(n) = T(n-1) + n**

$$T(n) = \begin{cases} a & ,n = 1 \\ \mathrm{T(n-1)} + \mathrm{n} & ,n > 1 \end{cases}$$

Jika masukan nilai n adalah 1 maka akan memberikan output kompleksitas a atau 1. Dan jika n lebih dari 1, maka perhitungan kompleksitas adalah T(n-1) +n.

Notasi Big-O

- Dalam *selection sort*, notasi Big-O dapat dicari dengan metode iteratif sebagai berikut.

$$
\begin{aligned}
T(n) &= n + T(n-1) \\
&= n + \{(n-1) + T(n-2)\} \\
&= n + (n-1) + \{(n-2) + T(n-3)\} \\
&\dots \\
&= n + (n-1) + (n-2) + (n-3) + \dots + 2 + T(1) \\
&= \{(n-1)(n+2)/2\} + a \\
&= \{(n-1)(n+2)/2\} + 1 \\
&= \frac{n^2}{2} + \frac{n}{2} + a
\end{aligned}
$$

- Didapat notasi Big-O adalah $O\left(\frac{n^2}{2} + \frac{n}{2} + a\right) \rightarrow$ **O(n^2)**

b. ***Merge sort***

Kompleksitas waktu

- 2 buah pemanggilan rekursif saat membagi array menjadi 2 bagian → T(n/2) + T(n/2) = 2T(n/2)
- 3 perulangan, 1 untuk melakukan perbandingan elemen dan 2 untuk memasukan elemen yang tersisa di sisi kiri atau kanan → 3n
- Diketahui **T(n) = 2T(n/2) + 3n**

$$T(n) = \begin{cases} a & ,n = 1 \\ 2\mathrm{T}(n/2) + 3n & ,n > 1 \end{cases}$$

Jika nilai n adalah 1, maka akan menghasilkan kompleksitas a atau 2 berdasarkan teorema master. Namun jika n > 1 maka kompleksitasnya adalah 2T(n/2) + 3n.

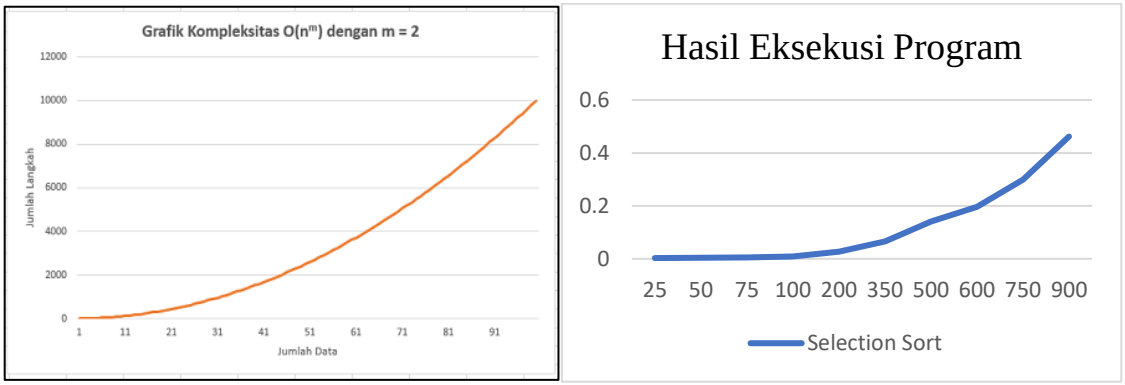
Notasi Big-O

- Diketahui T(n) sama dengan 2T(n/2) + 3n, maka berdasarkan teorema master $aT\left(\frac{n}{b}\right) + n^c$ diketahui nilai a = 2, b = 2, dan c = 1. Karena nilai a dan b sama didapat nilai notasi Big-O = **O(n log n)** .

2. Perbandingan hasil eksperimen dengan perhitungan teoritis

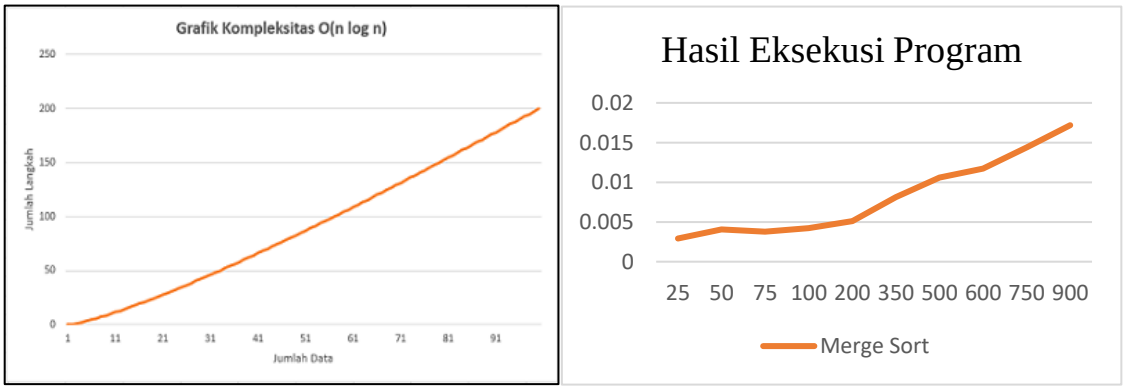
Diketahui pada algoritma *selection sort*, kompleksitas waktu terbutuknya atau Big-O adalah O(n^2) atau berupa algoritma kuadratik. Pada algoritma kuadratik, peningkatan waktu penyelesaian algoritma jauh lebih besar dari pada jumlah data [6]. Misalkan nilai n_0 adalah 12, maka waktu penyelesaiannya adalah 144, dan jika n_1 adalah 13, maka

waktu penyelesaiannya adalah 169. Jika dibandingkan dengan hasil uji eksperimen, peningkatan ini hampir serupa pada saat program mengurutkan data berjumlah 500 dan 600 data.



Pada grafik hasil secara teoritis dan hasil eksperimen di atas, terlihat peningkatan waktu penyelesaian memiliki peningkatan yang hampir serupa.

Selanjutnya, pada pada algoritma *merge sort* diketahui kompleksitas waktu terburuknya adalah O(n log n) atau berupa algoritma linier logaritmik. Pada algoritma linier logaritmik, penyelesaian masalah akan dibagi-bagi(log n) dan dijalankan sebanyak n [6]. Kompleksitas O(n log n) memiliki bentuk grafik seperti pada gambar di bawah ini.



Jika dibandingkan dengan hasil pengujian, terdapat perbedaan hasil yang cukup jelas. Namun, peningkatan waktu penyelesaian pada hasil eksperimen masih memiliki kemiripan dengan hasil secara teoritis. Perbedaan yang terjadi bisa saja disebabkan karena jenis CPU yang digunakan tergolong tinggi serta jumlah memori yang digunakan terbilang memadai.

3. Kesimpulan

Berdasarkan hasil pengujian program pengurutan data menggunakan algortima *selection sort* dan *merge sort*, diketahui hasil pengeksekusian program dengan algoritma *merge sort* memiliki hasil yang lebih cepat dibandingkan algoritma *selection sort.* Pada grafik di poin pengujian, grafik menunjukan saat kedua algoritma mengurutkan data

dengan jumlah terbilang kecil(25 hingga 100 data) tidak terlalu terlihat perbedaan yang signifikan. Namun, ketika jumlah data mencapai lebih dari ratusan data, barulah terlihat perbedaan kedua algoritma. Peningkatan waktu eksekusi yang signifikan terjadi pada algoritma *selection sort.* Sedangkan pada *merge sort,* peningkatan waktu eksekusi terbilang lambat atau sedikit.

Pada analisis kompleksitas waktu algortitma *selection sort* dan *merge sort* didapat nilai *worst case* atau Big-O kedua algoritma. Algoritma *selection sort* memiliki nilai *worst case* berupa $O(n^2)$, sedangkan algortima *merge sort* memiliki nilai *worst case* berupa O(n log n). Berdasarkan hasil analisis ini, diketahui bahwa algoritma *merge sort* memiliki komplesitas waktu yang lebih baik daripada *selection sort.* Pernyataan ini juga dikuatkan dengan waktu eksekusi *merge sort* yang jauh lebih cepat dibandingkan algoritma *selection sort*.

E. Referensi

[1] Aryo Pinandito, "Design and Analysis of AlgorithmDivide and Conquer Algorithm", DAA V – Divide and Conquer, April 2019. [Online]. Available: https://documen.site/download/daa-v-divide-and-conquer_pdf. [Accessed: 24 April 2022].

[2] Finn Christoffer K., "Algoritma selection sort di python", BINUS UNIVERSITY BANDUNG - Kampus Teknologi Kreatif, Desember 2019. [Online]. Available: https://binus.ac.id/bandung/2019/12/algoritma-selection-sort-di-python/. [Accessed: 24 April 2022].

[3] Taufik Fuadi Abidin and Irvanizam Zamanhuri, "Metode Pengurutan merge sort - unsyiah", Website Jurusan informatika Universitas Syiah, Desember 2012. [Online]. Available: https://www.informatika.unsyiah.ac.id/tfa/ds/mergesort.pdf. [Accessed: 24 April 2022].

[4] Arfian Hidayat, "Algoritma merge sort", Algoritma Merge Sort - Arfian Hidayat, Mei 2019. [Online]. Available: https://arfianhidayat.com/algoritma-merge-sort. [Accessed: 24 April 2022].

[5] Info Akurat, "100+ nama-nama Pasaran di Indonesia yang paling Banyak Digunakan", InfoAkurat.com, 10 September 2019. [Online]. Available: https://www.infoakurat.com/2019/09/nama-nama-pasaran-di-indonesia.html. [Accessed: 24 April 2022].

[6] Bertzzie, "Kompleksitas algoritma", Kompleksitas Algoritma - Dasar Analisis Algoritma, 13-Sep-2013. [Online]. Available: http://dev.bertzzie.com/knowledge/analisis-algoritma/KompleksitasAlgoritma.html. [Accessed: 29-Apr-2022].