

# **TUGAS 1 DESAIN DAN ANALISIS ALGORITME**

## **Perbandingan Kompleksitas Algoritma Pengurutan**

**(Selection Sort dan Quick Sort)**



Disusun oleh :

Cahya Aprilia Putranti (19102080)

S1 IF 07 MM4

**PRODI S1 TEKNIK INFORMATIKA**

**FAKULTAS INFORMATIKA**

**INSTITUT TEKNOLOGI TELKOM PURWOKERTO**

**2022**

## Dasar teori

Dalam beberapa konteks pemrograman, algoritma adalah spesifikasi dari urutan langkah-langkah untuk melakukan tugas tertentu. Untuk menyelesaikan suatu masalah, tidak cukup hanya dengan menemukan algoritma yang hasil pemecahan masalahnya benar. Ini berarti bahwa algoritma mengembalikan output yang diinginkan dari sejumlah input yang diberikan. Sebagus apapun algoritmanya, jika memberikan hasil yang salah, sudah pasti itu bukan algoritma yang bagus. Oleh karena itu, algoritma yang baik menggunakan algoritma yang efektif, efisien, terarah, dan terstruktur. Pilih algoritme yang kesesuaiannya dapat diukur dari segi waktu eksekusi algoritme dan jejak memori.

Selection sort adalah suatu metode pengurutan yang membandingkan elemen yang sekarang dengan elemen berikutnya sampai ke elemen yang terakhir. Jika ditemukan elemen lain yang lebih kecil dari elemen sekarang maka dicatat posisinya dan langsung ditukar. Dalam arti lain Selection Sort adalah pengurutan hard split/easy join dengan cara membagi larik menjadi dua buah upalarik yang tidak sama ukurannya

Metode selection sort adalah melakukan pemilihan dari suatu nilai yang terkecil dan kemudian menukarnya dengan elemen paling awal, lalu membandingkan dengan elemen yang sekarang dengan elemen berikutnya sampai dengan elemen terakhir, perbandingan dilakukan terus sampai tidak ada lagi pertukaran data

Quick Sort adalah sebuah algoritma sorting dari model Divide and Conquer yaitu dengan cara mereduksi tahap demi tahap sehingga menjadi 2 bagian yang lebih kecil. Algoritma pengurutan Quicksort dapat juga di sebut algoritma pengurutan yang terkenal dan tercepat (sesuai namanya).

Metode Quick Sort merupakan algoritma yang sangat cepat dibandingkan dengan algirtma sorting lainnya, karena algoritma quick sort ini melakukan sorting dengan membagi masalah menjadi sub masalah dan sub masalah dibagi lagi menjadi sub-sub masalah sehingga sorting tersebut menjadi lebih cepat walaupun memakan ruang memori yang besar.

## Implementasi

Cara kerja program :

- Selection sort
  - a. Pseudocode

Program Selection\_Sort

Deklarasi:

```
// pendeklarasian program menggunakan beberapa functions
// serta function yang bertipe data interger
randomNumber, selectionArr: Integer[]
nSelection, startTime, endTime: Integer
```

SubProgram carIndeksTerkecil

Parameter:

```
// variabel yang bertipe data interger
array: Integer[]
i, j: Integer
```

Deklarasi:

k: Integer

```
// algoritma yang digunakan menggunakan selection sort
```

Algoritma:

```
// disini merupakan algoritma yang digunakan pada program yang dibuat
```

```
if i == j then
```

```
    output i
```

```
endif
```

```
k <- carIndeksTerkecil(array, i + i, j)
```

```
if array[i] < array[k] then
```

```
    output i
```

```
else
```

```
    output k
```

```
endif
```

### SubProgram selectionSort

#### Parameter:

```
// pendeklarasian variable menggunakan tipe data integer  
array: Integer[]  
n_data, index: Integer
```

#### Deklarasi:

```
k, temp: Integer  
index <- 0
```

```
// disini merupakan pendeklarasian algoritma untuk program yang dibuat
```

#### Algoritma:

```
if index == n_data then  
    output -1  
endif
```

```
// mencari indeks diambil dari parameter diatas
```

```
k <- cariIndeksTerkecil(array, index, n_data)  
if k != index then  
    temp <- array[index]  
    array[index] <- array[k]  
    array[k] <- temp  
endif
```

```
selectionSort(array, n_data, index + 1)
```

#### Algoritma:

```
randomNumber <- input(array)  
selectionArr <- randomNumber
```

```
startTime <- input(time)  
selectionSort(selectionArr)  
endTime <- input(time)
```

```
output(selectionArr)

output(startTime)

output(endTime)

output(endTime - startTime)
```

## b. Screenshoot Program

```
1 #NAMA : CARVA APRILIA PUTRANTI
2 #NILAS : 51 IF 07 PM
3 #NIM : 19102000
4
5 # Import Library
6
7 # library time process time digunakan untuk mengukur kecepatan eksekusi dari algoritma
8 from time import process_time
9 # library numpy digunakan untuk memproduksi array number secara random
10 import numpy as np
11 # library sys digunakan untuk mengatur limit batasan aksi rekursif
12 import sys
13 # merubah batasan rekursif menjadi 10000 aksi rekursif
14 sys.setrecursionlimit(10000)
15
16 """a Produksi Array Number secara Random Sebanyak 1000 buah"""
17
18 # menciptakan random array menggunakan numpy (angka awal, angka akhir, jumlah data)
19 randomNumber = np.random.randint(1, 1000, 1000)
20
21 # memasukkan random array ke variabel variabel dituju
22 selectionArr = randomNumber
23 quickArr = randomNumber
24
25 # menghitung jumlah data yang ada dalam array
26 nSelection = len(selectionArr)
27 nQuick = len(quickArr)
28
29 """# Export data"""
30
31 # membuka file sample-data.txt apabila belum tersedia maka membuat file
32 file = open("sample-data.txt", "w")
33
34 # memasukkan tiap item array kedalam file
35 for i in randomNumber:
36     file.write(str(i) + "\n")
37
38 # menutup atau menyimpan file dengan perubahan terbaru
39 file.close()
40
41 """a Selection Sort"""
42
43 # fungsi minIndex digunakan untuk mencari nilai terkecil diantara array
44 def cariIndexTerkecil(array, i, j):
45     # apabila pencarian sudah di ujung array maka kembalikan index i
46     if i == j:
47         return i
48     # melakukan pencarian index nilai terkecil menggunakan metode rekursif
49     k = cariIndexTerkecil(array, i + 1, j)
50
51     if(array[i] < array[k]):
52         # kembalikan nilai i apabila nilai array[i] kurang dari array[k]
53         return i
54     else:
55         # kembalikan nilai k apabila nilai array[i] lebih besar dari array[k]
56         return k
57
58
59 def selectionSort(array, n_data, index = 0):
60     # kalo index sama dengan jumlah data di array maka berhenti
61     if index == n_data:
62         return -1
63
64     # mencari index dengan nilai terkecil
65     k = cariIndexTerkecil(array, index, n_data-1)
66
67     # apabila index k tidak sama dengan nilai variable index
68     if k != index:
69         # tukar array[k] menjadi array[index]
```

```

64 # mencari index dengan nilai terkecil
65 k = cariindekstertekcil(array, index, n_data-1)
66
67 # apabila indeks k tidak sama dengan nilai variable index
68 if k != index:
69     # tukar array[k] menjadi array[index]
70     # tukar array[index] menjadi array[k]
71     temp = array[index]
72     array[index] = array[k]
73     array[k] = temp
74
75 # melanjutkan sorting menggunakan metode rekursif
76 selectionSort(array, n_data, index + 1)
77
78 """* Running Algoritma"""
79
80 start_time_selection_sort = process_time()
81 selectionSort(selectionArr, nselection)
82 end_time_selection_sort = process_time()
83
84 """* Hasil Running"""
85
86 print(f'Sorted array: {selectionArr}')
87 print('start time: ', start_time_selection_sort)
88 print('end time: ', end_time_selection_sort)
89 print('executing time: ', (end_time_selection_sort - start_time_selection_sort), 'seconds')
90
91 """* Reset Rekursif limit"""
92
93 sys.setrecursionlimit(1000)

```

- Quick sort

- a. Pseudocode

Program quick\_Sort

Deklarasi:

// pendeklarasian program menggunakan beberapa functions

// serta semua function yang bertipe data integer

randomNumber, quickArr: Integer[]

nQuick, startTime, endTime: Integer

// variabel yang bertipe data integer

SubProgram partition

Parameter:

array: Integer[]

low, high: Integer

// pendeklarasian variable berupa I, pivot, temp, yang bertipe data integer

Deklarasi:

i, pivot, temp: Integer

// disini menggunakan algoritma quick sort yang digunakan pada program ini

Algoritma:

pivot <- array[high]

i <- low - 1

// dan perulangan yang digunakan untuk program ini adalah menggunakan perulangan for

```

for j in low to high do
    if array[j] <= pivot then
        i <- i + 1
        temp <- array[i]
        array[i] <- array[j]
        array[j] <- temp
    endif
endfor

temp <- array[i + 1]
array[i + 1] <- array[high]
array[high] <- temp

output i + 1

```

#### SubProgram quicksort

// parameter variable semuanya bertipe integer

Parameter:

array: Integer[]

low, high: Integer

// pendeklrasian variable pi bertipe data integer

Deklarasi:

pi: Integer

// dan ini merupakan algoritma yang variabelnya diambil dari pendeklrasian diatas yang menggunakan quick sort

Algoritma:

if low < high then

pi <- partition(array, low, high)

quick\_sort(array, low, pi - 1)

quick\_sort(array, pi + 1, high)

endif

Algoritma:

```
randomNumber <- input(array)
```

```
quickArr <- randomNumber
```

```
startTime <- input(time)
```

```
quickSort(quickArr)
```

```
endTime <- input(time)
```

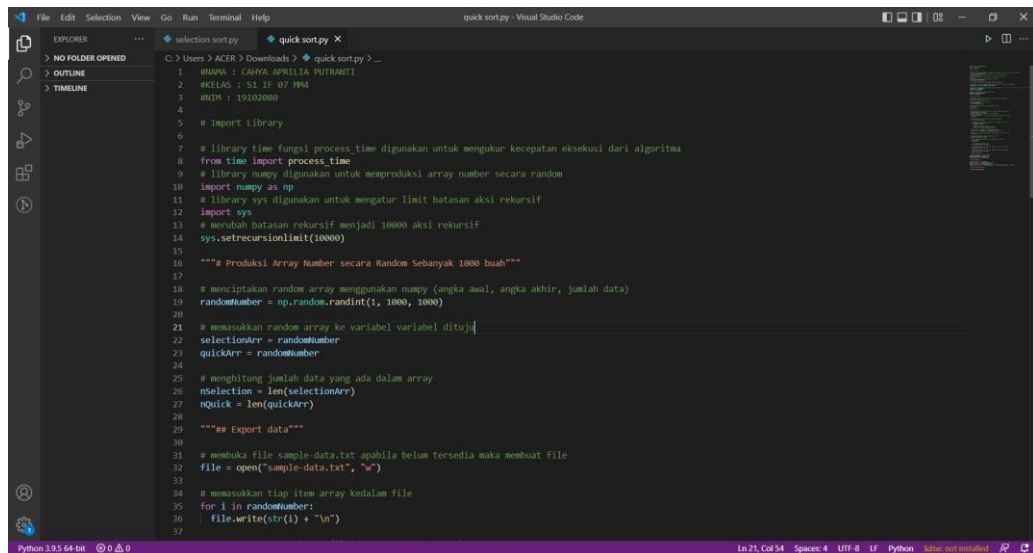
```
output(quickArr)
```

```
output(startTime)
```

```
output(endTime)
```

```
output(endTime - startTime)
```

## b. Screenshoot Program



```
1 #NAMA : CAHYA APRILIA PUTRANTI
2 #KELAS : SI IF 07 PM4
3 #IDN : 19103000
4
5 # Import library
6
7 # library time fungsi process.time digunakan untuk mengukur kecepatan eksekusi dari algoritma
8 from time import process.time
9 # library numpy digunakan untuk memproduksi array number secara random
10 import numpy as np
11 # library sys digunakan untuk mengatur limit batasan aksi rekursif
12 import sys
13 # merubah batasan rekursif menjadi 10000 aksi rekursif
14 sys.setrecursionlimit(10000)
15
16 """% Produksi Array Number secara Random Sebanyak 1000 buah"""
17
18 # menciptakan random array menggunakan numpy (angka awal, angka akhir, jumlah data)
19 randomNumber = np.random.randint(1, 1000, 1000)
20
21 # memasukkan random array ke variabel variabel dituju
22 selectionArr = randomNumber
23 quickArr = randomNumber
24
25 # menghitung jumlah data yang ada dalam array
26 nselection = len(selectionArr)
27 nQuick = len(quickArr)
28
29 """% Export data"""
30
31 # membuka file sample-data.txt apabila belum tersedia maka membuat file
32 file = open("sample-data.txt", "w")
33
34 # memasukkan tiap item array kedalam file
35 for i in randomNumber:
36     file.write(str(i) + "\n")
37
```



```
File Edit Selection View Go Run Terminal Help quicksort.py - Visual Studio Code
EXPLORER
> NO FOLDER OPENED
> OUTLINE
> TIMELINE
selection sort.py quicksort.py X
C:\Users\ACER> Downloads > quicksort.py > ...
34 # memasukkan tiap item array kedalam file
35 for i in randomnumber:
36     file.write(str(i) + "\n")
37
38 # menutup atau menyimpan file dengan perubahan terbaru
39 file.close()
40
41 *** Quick Sort ***
42
43 # fungsi partition berfungsi untuk mencari indeks tengah dari suatu array
44 def partition(array, low, high):
45     # pivot adalah array pembanding dari array terakhir
46     pivot = array[high]
47     # i dimulai dari dari -1 supaya dapat membandingkan indeks 0 dengan pivot
48     i = low - 1
49
50     # perulangan dari nilai variable low hingga variable high
51     for j in range(low, high):
52
53         # apabila nilai array indeks j kurang dari sama dengan nilai pivot
54         if array[j] <= pivot: # 0 <= 0
55             # tambahkan i dengan 1
56             i = i + 1
57             # tukar nilai array[i] dengan array[j]
58             (array[i], array[j]) = (array[j], array[i])
59
60     # tukar array[i + 1] dengan array[high] atau pivot
61     (array[i + 1], array[high]) = (array[high], array[i + 1])
62
63     # mengembalikan indeks tengah sebagai pembagi antara sisi kiri dan kanan
64     return i + 1
65
66 # fungsi untuk melakukan quick sort dengan rekursif
67 def quick_sort(array, low, high):
68
69     # untuk membandingkan apakah index low kurang dari index high
70     if low < high:
```

```
File Edit Selection View Go Run Terminal Help quicksort.py - Visual Studio Code
EXPLORER
> NO FOLDER OPENED
> OUTLINE
> TIMELINE
selection sort.py quicksort.py X
C:\Users\ACER> Downloads > quicksort.py > partition
66 # fungsi untuk melakukan quick sort dengan rekursif
67 def quick_sort(array, low, high):
68
69     # untuk membandingkan apakah index low kurang dari index high
70     if low < high:
71         # apabila ya
72
73         # cari indeks tengah dari array
74         pi = partition(array, low, high)
75
76         # melakukan pengurutan rekursif pada bagian kiri dari indeks tengah
77         quick_sort(array, low, pi - 1)
78
79         # melakukan pengurutan rekursif pada bagian kanan dari indeks tengah
80         quick_sort(array, pi + 1, high)
81
82 *** Running Algorithm ***
83
84 start_time_quick_sort = process_time()
85 quick_sort(quickdér, 0, nquick - 1)
86 end_time_quick_sort = process_time()
87
88 *** Hasil Running ***
89
90 print('Sorted array: (quickdér)')
91 print('start time: ', start_time_quick_sort)
92 print('end time: ', end_time_quick_sort)
93 print('executing time: ', (end_time_quick_sort - start_time_quick_sort), 'seconds')
94
95 *** Reset Rekursif Limit ***
96
97 sys.setrecursionlimit(10000)
```

## Pengujian

Dalam pengujian ini, pengungi menggunakan algoritma selection sort dan quick sort sebagai perbandingannya. Data yang digunakan untuk pengujian yaitu data dummy atau random data yang kemudian dimasukan dalam array. Untuk pengujian yang pertama menggunakan inputan data 100 kemudian di running dan di dapatkan waktu hasil eksekusi sebesar 0.003 detik pada selection sort dan 0.005 detik pada quick sort, kemudian pada pengujian kedua menggunakan inputan data 500 dan di dapatkan waktu hasil eksekusi sebesar 0,5 detik pada selection sort dan 0,6 detik pada quick sort, dan yang terakhir yaitu dengan menginputkan data sebesar 1000 kemudian di running dan di dapatkan hasil eksekusi sebesar 0.2 detik pada selection sort dan 0.26 detik pada quick sort.

Untuk spesifikasi hardware saya menggunakan device laptop acer aspire dengan processor Intel(R) Core(TM) i3-1005G1 CPU @ 1.20GHz 1.19 GHz, RAM (4,00 GB), dan system type (64-bit operating system, x64-based processor) dan untuk software yang saya gunakan yaitu windows 11, menggunakan Bahasa pemrograman python, compiler python dan IDE google collab.

n	Waktu eksekusi algoritma Selection	Waktu eksekusi algoritma Quick
100	0.0037 seconds	0.0027 seconds
500	0.0807 seconds	0.0741 seconds
1000	0.2046 seconds	0.2698 seconds

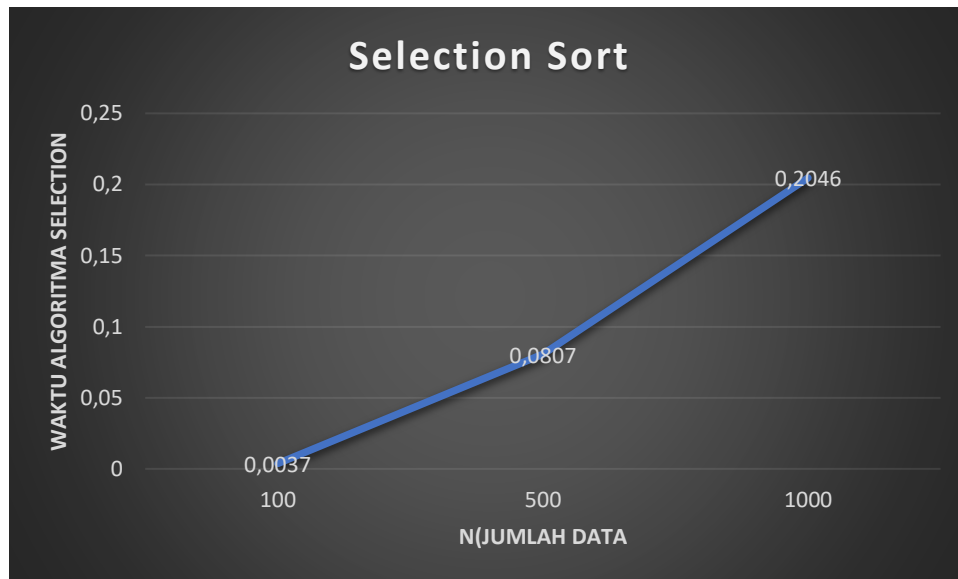
Tabel eksekusi waktu masing masing algoritma

Keterangan:

n = jumlah inputan

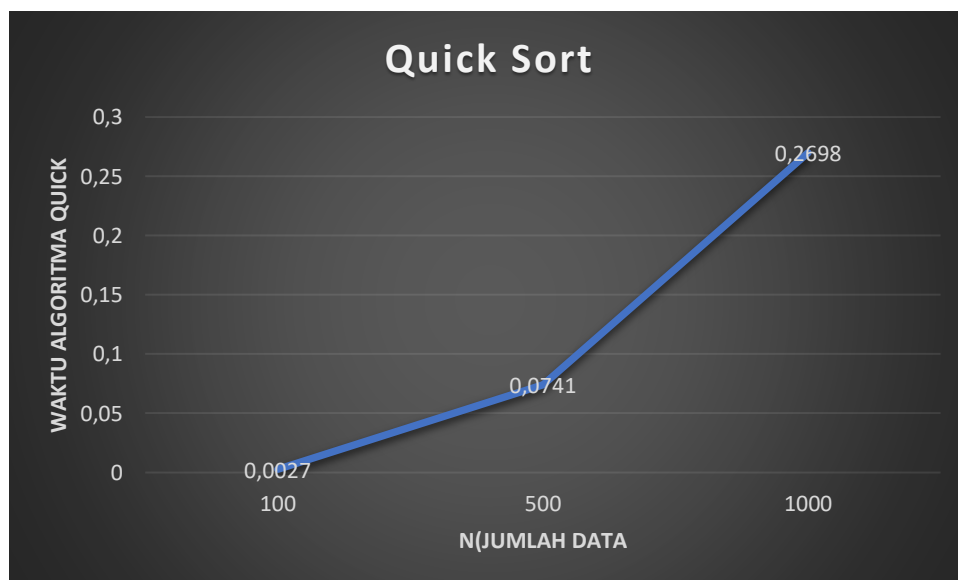
waktu eksekusi dalam detik

- Algoritma selection sort



Grafik selection sort

- Algoritma quick sort



Grafik quick sort

## Analisis Hasil Pengujian

- Selection sort

Algoritma ini mempunyai dua for loop, satu loop di dalam loop yang lainnya. Banyaknya perbandingan yang harus dilakukan untuk siklus pertama adalah  $n$ , perbandingan yang harus dilakukan untuk siklus yang kedua  $n-1$ , dan seterusnya. Sehingga jumlah keseluruhan perbandingan adalah  $n(n+1)/2-1$  perbandingan.

Kompleksitas Waktu nya  $O(n^2)$  karena ada dua loop bersarang. Hal yang baik tentang sortir seleksi adalah tidak pernah membuat lebih dari  $O(n)$  swap dan dapat berguna ketika penulisan memori adalah operasi yang mahal.

- Quick sort

Kompleksitas efisiensi quicksort dari algoritma quicksort sangat di pengaruhi oleh pilihan elemen pivot. Pemilihan pivot menentukan jumlah dan ukuran partisi di setiap fase rekursif. Waktu yang dibutuhkan oleh QuickSort secara umum dapat dituliskan sebagai berikut.

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

Dua istilah pertama untuk dua panggilan rekursif, istilah terakhir untuk proses partisi.  $k$  adalah jumlah elemen yang lebih kecil dari pivot. Waktu yang dibutuhkan oleh QuickSort tergantung pada array input dan strategi partisi.

1. Kasus terburuk:

Kasus terburuk terjadi ketika proses partisi selalu memilih elemen terbesar atau terkecil sebagai pivot. Jika kita mempertimbangkan strategi partisi di atas di mana elemen terakhir selalu dipilih sebagai pivot, kasus terburuk akan terjadi ketika array sudah diurutkan dalam urutan naik atau turun. Berikut ini adalah pengulangan untuk kasus terburuk.

$$T(n) = T(0) + T(n-1) + O(n)$$

yang ekuivalen dengan

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

Solusi dari pengulangan di atas adalah  $O(n^2)$ .

2. Kasus terbaik:

Kasus terbaik terjadi ketika proses partisi selalu memilih elemen tengah sebagai pivot. Berikut ini adalah pengulangan untuk kasus terbaik.

$$T(n) = 2T(n/2) + O(n)$$

Solusi dari pengulangan di atas adalah  **$O(n \log n)$** .

3. Kasus Rata-rata:

Untuk melakukan analisis kasus rata-rata, kita perlu mempertimbangkan semua kemungkinan permutasi array dan menghitung waktu yang dibutuhkan oleh setiap permutasi yang tidak terlihat mudah. Kita dapat memperoleh gambaran kasus rata-rata dengan mempertimbangkan kasus ketika partisi menempatkan elemen  $O(n/9)$  dalam satu himpunan dan elemen  $O(9n/10)$  dalam himpunan lain. Berikut rekurensi untuk kasus ini.

$$T(n) = T(n/9) + T(9n/10) + O(n)$$

Solusi dari pengulangan di atas juga  **$O(n \log n)$**

Perbandingan hasil perhitungan kompleksitas algoritma selection dan quick sort yaitu pada algoritma selection sort ( $O(n^2)$ ) membutuhkan waktu lebih lama untuk mengurutkan data sedangkan quick sort tidak membutuhkan waktu yang lama untuk mengurutkan datanya, hal ini sesuai dengan kompleksitas algoritmanya. Dan dapat di simpulkan Algoritma Quick Sort lebih cepat dalam melakukan pengurutan data jika dibandingkan dengan Selection Sort.

## Referensi

- [1] Rahayuningsih, Panny Agustia. "Analisis Perbandingan Kompleksitas Algoritma Pengurutan Nilai (Sorting)." *EVOLUSI: Jurnal Sains dan Manajemen* 4.2 (2016).
- [2] Al Rivan, Muhammad Ezar. "Perbandingan Kecepatan Gabungan Algoritma Quick Sort dan Merge Sort dengan Insertion Sort, Bubble Sort dan Selection Sort." *Jurnal Teknik Informatika dan Sistem Informasi* 3.2 (2017).