

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

I. 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.

II. 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
```

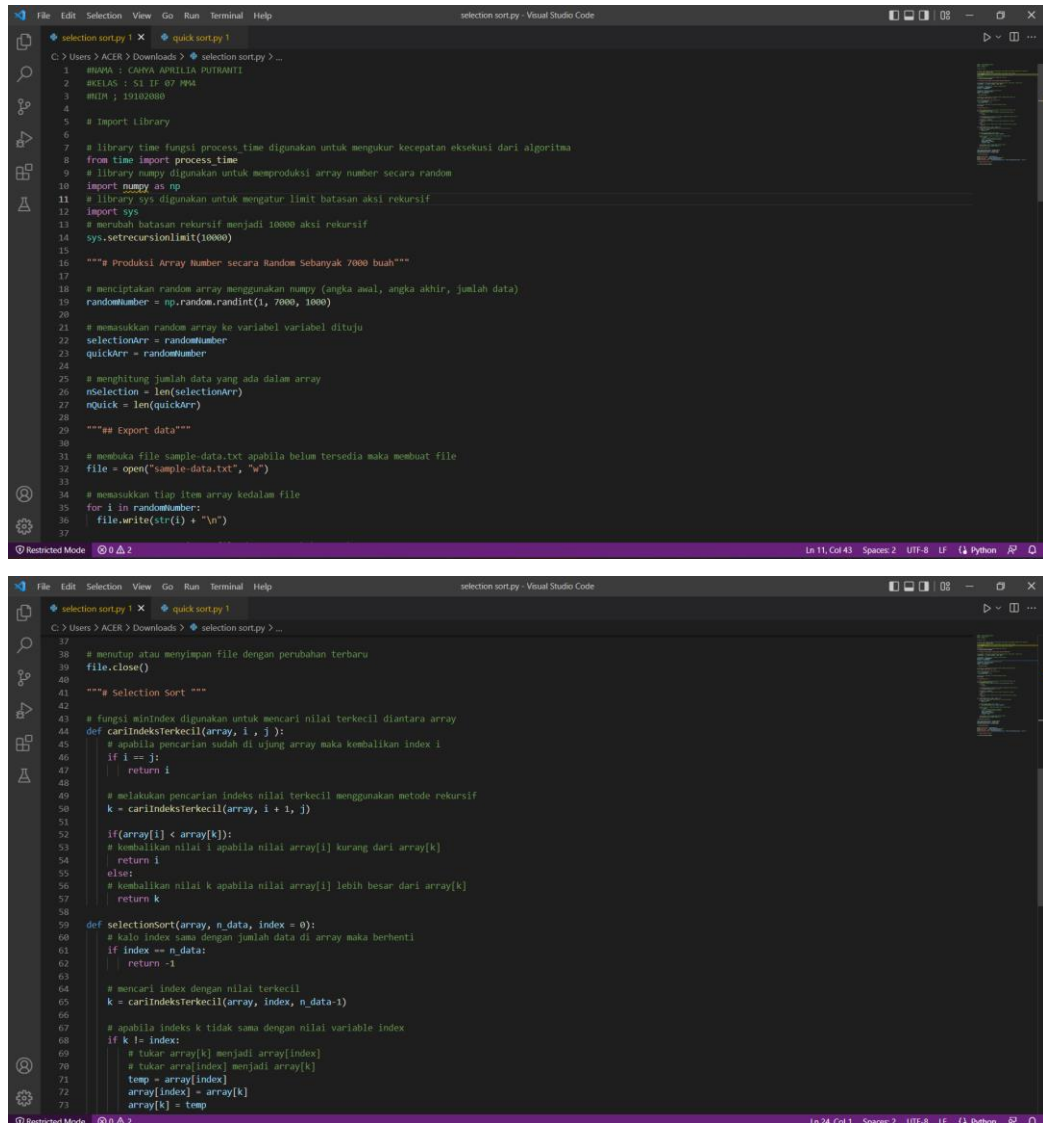
```
startTime <- input(time)
```

```
selectionSort(selectionArr)
```

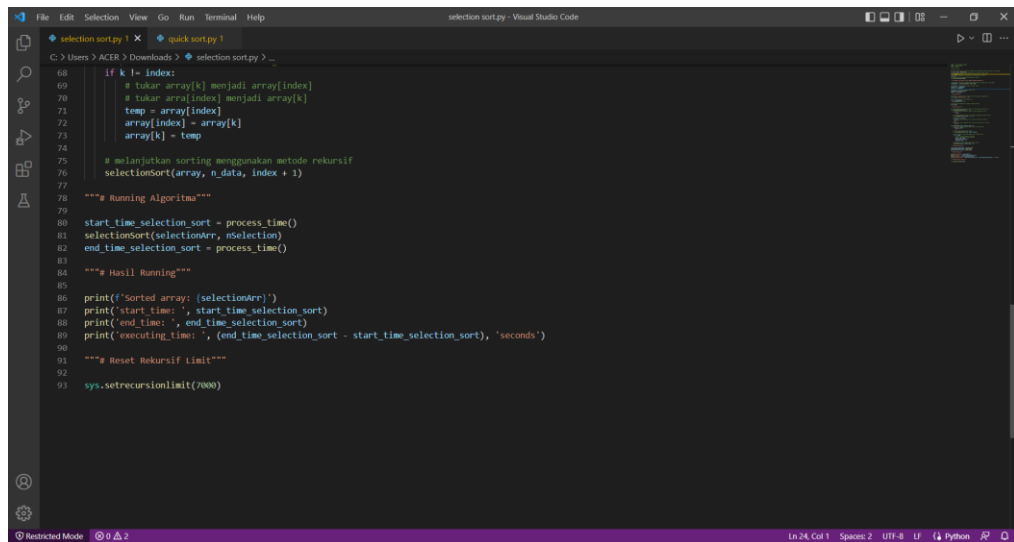
```
endTime <- input(time)
```

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

b. Screenshoot Program



```
File Edit Selection View Go Run Terminal Help
selection.sort.py - Visual Studio Code
C:\Users\ACER> Downloads > selection.sort.py > ...
1 #NAMA : CAHYA APRILIA PUTRANTI
2 #KELAS : SI IF 07 PPM
3 #NIM : 51102000
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 7000 buah"""
17
18 # menciptakan random array menggunakan numpy (angka awal, angka akhir, jumlah data)
19 randomNumber = np.random.randint(1, 7000, 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
39 # menutup atau menyimpan file dengan perubahan terbaru
40 file.close()
41
42 """# Selection Sort """
43
44 # fungsi minIndex digunakan untuk mencari nilai terkecil diantara array
45 def cariIndexTerkecil(array, i, j):
46     # apabila pencarian sudah di ujung array maka kembalikan index i
47     if i == j:
48         return i
49     # melakukan pencarian indeks nilai terkecil menggunakan metode rekursif
50     k = cariIndexTerkecil(array, i + 1, j)
51
52     if(array[i] < array[k]):
53         # kembalikan nilai i apabila nilai array[i] kurang dari array[k]
54         return i
55     else:
56         # kembalikan nilai k apabila nilai array[i] lebih besar dari array[k]
57         return k
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 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
```



```
File Edit Selection View Go Run Terminal Help
selection sort.py - Visual Studio Code

C:\Users\ACER\Downloads> selection sort.py >...

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('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(7000)
```

- Quick sort
 - a. Pseudocode

Program quick_Sort

Deklarasi:

// pendeklarasian program menggunakan beberapa functions

// serta semua function yang bertipe data interger

randomNumber, quickArr: Integer[]

nQuick, startTime, endTime: Integer

// variabel yang bertipe data interger

SubProgram partition

Parameter:

array: Integer[]

low, high: Integer

// pendeklrasian 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

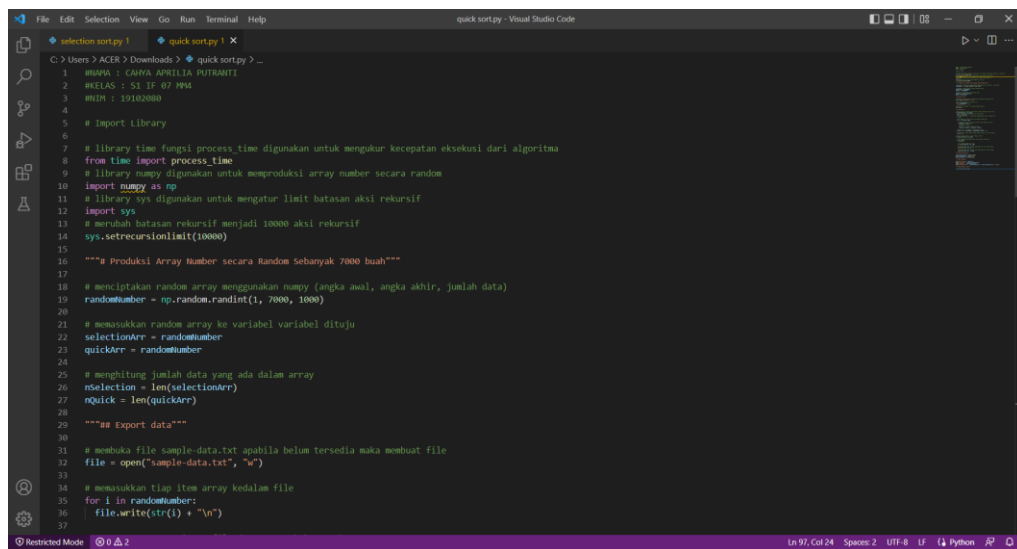
```

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



```
File Edit Selection View Go Run Terminal Help
quick sortpy - Visual Studio Code
selection sortpy 1 quick sortpy 1 X
C:\Users\ACER>Downloads> quick sortpy > ...
1 #NAMA : CANVA APHELIA PUTRANITI
2 #KELAS : SI IF 07 PM
3 #UID : 1910000
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 7000 buah"""
17
18 # menciptakan random array menggunakan numpy (angka awal, angka akhir, jumlah data)
19 randomNumber = np.random.randint(1, 7000, 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 Restricted Mode 0 2 Ln 97, Col 24 Spaces: 2 UTF-8 LF Python RP
```



```
File Edit Selection View Go Run Terminal Help quicksort.py - Visual Studio Code
selectionsort.py ! quicksort.py ! X
C:\Users\ACER> Downloads > quicksort.py > ...
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:
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 Algorithms"""
83
84 start_time_quick_sort = process_time()
85 quick_sort(quickArr, 0, nQuick - 1)
86 end_time_quick_sort = process_time()
87
88 """* Hasil Running"""
89
90 print(f'Sorted array: {quickArr}')
91 print(f'start time: ', start_time_quick_sort)
92 print(f'end time: ', end_time_quick_sort)
93 print(f'executing time: ', (end_time_quick_sort - start_time_quick_sort), 'seconds')
94
95 """* Reset Rekursif Limit"""
96
97 sys.setrecursionlimit(7000)
```

```
File Edit Selection View Go Run Terminal Help quicksort.py - Visual Studio Code
selectionsort.py ! quicksort.py ! X
C:\Users\ACER> Downloads > quicksort.py > partition
69
70 # untuk membandingkan apakah index low kurang dari index high
71 if low < high:
72     # apabila ya
73
74     # cari indeks tengah dari array
75     pi = partition(array, low, high)
76
77     # melakukan pengurutan rekursif pada bagian kiri dari indeks tengah
78     quick_sort(array, low, pi - 1)
79
80     # melakukan pengurutan rekursif pada bagian kanan dari indeks tengah
81     quick_sort(array, pi + 1, high)
82
83 """* Running Algorithms"""
84
85 start_time_quick_sort = process_time()
86 quick_sort(quickArr, 0, nQuick - 1)
87 end_time_quick_sort = process_time()
88
89 """* Hasil Running"""
90
91 print(f'Sorted array: {quickArr}')
92 print(f'start time: ', start_time_quick_sort)
93 print(f'end time: ', end_time_quick_sort)
94 print(f'executing time: ', (end_time_quick_sort - start_time_quick_sort), 'seconds')
95
96 """* Reset Rekursif Limit"""
97
98 sys.setrecursionlimit(7000)
```

III. Pengujian

Dalam pengujian ini, penguni menggunakan algoritma selection sort dan quick sort sebagai perbandingannya. Data yang digunakan untuk pengujian yaitu data dummy atau random data pada bahasa pemrograman python dengan jumlah keseluruhan datanya yaitu 7000 yang kemudian dimasukan dalam array. Untuk pengujian yang pertama menggunakan inputan data 1000 kemudian untuk inputan kedua 3000 data, selanjutnya, inputan ke 3 ada 5000 data dan inputan terakhir ada 7000 data

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
1000	0.301 seconds	0.264 seconds
3000	2.747 seconds	1.994 seconds
5000	9.094 seconds	6.661seconds
7000	15.948 seconds	15.524 seconds

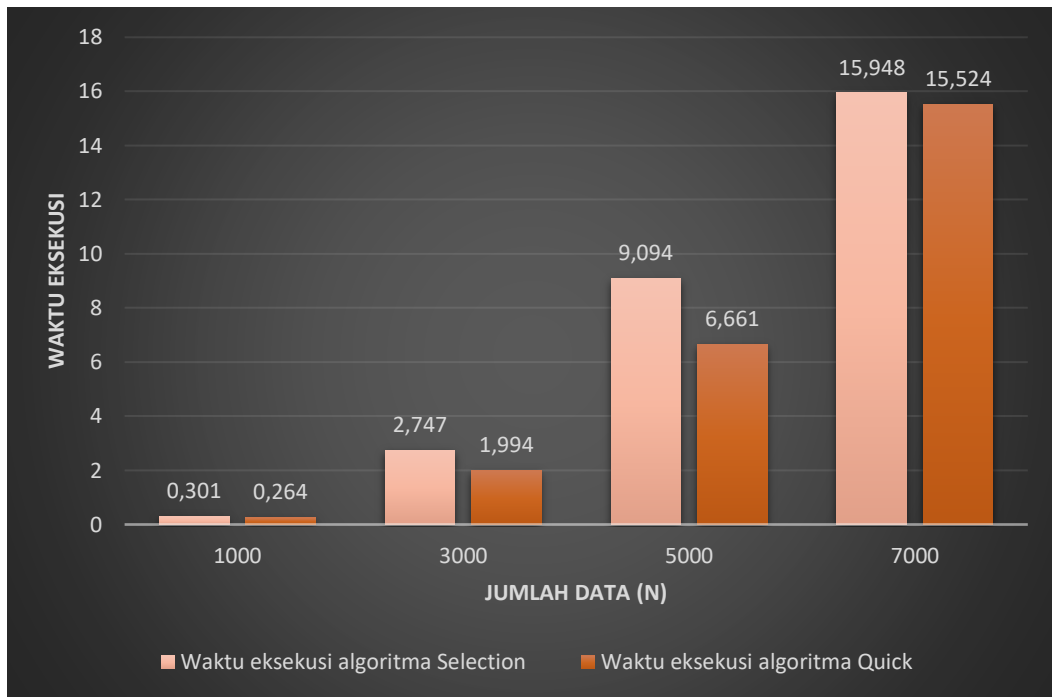
Tabel eksekusi waktu masing masing algoritma

Keterangan:

n = jumlah inputan

waktu eksekusi dalam detik

- Grafik Algoritma selection sort dan quick sort



Dapat di lihat pada grafik tersebut untuk eksekusi waktunya pada selection sort lebih lama di bandingkan dengan algoritma 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. Satu for loop melangkah melalui semua elemen dalam array dan kami menemukan indeks elemen minimum menggunakan loop for lain yang bersarang di dalam for loop luar.

$$T(n) = (n - 1) + (n - 2) + \dots + 2 + 2 = \sum_{f=1}^{n-1} n - 1 = \frac{n(n-1)}{2} = O(n^2)$$

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 rekrusif. 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 **$O(n \log n)$**

- Kesimpulan

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 disimpulkan Algoritma Quick Sort lebih cepat dalam melakukan pengurutan data jika dibandingkan dengan Selection Sort. Jadi dapat diambil kesimpulan bahwa dari dua algoritma di atas, algoritma Quick Sort dapat dikatakan sebagai algoritma yang lebih bagus

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).