TUGAS BESAR ANALISIS SORTING

DALAM ALGORITMA PEMROGRAMAN

Oleh:

Muhammad Daffa Regenta S

M. Syamsul Aqiel

NIM: 1301184291, 1301183499

e-mail: daffaregenta@student.telkomuniversity.ac.id

syamsulaqiel@student.telkomuniversity.ac.id

ABSTRAK

Algoritma dalam pemrograman perangkat lunak sangat penting. Sehingga sangat perlu memahami konsep algoritma dalam pemrograman**.** Banyak kasus yang umum maupun khusus dengan pemrograman karna dinilai sangat efisien dan memudahkan pekerjaan. Di dalam algortima pemrograman terdapat beberapa kondisi salah satu nya kondisi sorting yaitu pengurutan suatu karakter, angka atau sejenisnya. *Sorting* sendiri ada dua jenis yaitu *Ascending* (pengurutan dari kecil ke besar) dan *Descending* (pengurutan dari besar ke kecil). Dari dua tipe tersebut ada macam-macam jenis *sorting* salah satu nya yaitu *buble sort, insertion sort, selection sort, merge sort,* dan *quick sort*. Nilai algortima pengurutan untuk menyortir pengolahan menggunakan tipe data integer. efektif algortima yang dapat meminimalkan kebutuhan ruang dan waktu. Namun membutuhkan waktu dan ruang suatu algortima bergantung pada jumlah data yang di olah dan algortima yang digunakan.

Kata kunci: Kompleksitas waktu, Algortima, Pemrograman, Sorting, Data, Buble sort, Selection sort, Insertion sort, Merge sort, Quick sort.

1. PENDAHULUAN

Pengurutan (*sorting*). Merupakan salah satu algortima yang terpenting dalam dunia pemrograman. *Sorting* adalah pengurutan daftar-daftar tertentu dari urutan yang diberikan dimana unsur-unsur yang di urut tersusun secara menaik atau secara menurun. Algortima *sorting* yang populer antara lain, *buble sort, insertion sort, selection sort, merge sort, quick sort.*

Data yang sudah terurut memiliki beberapa keuntungan. Selain mempercepat waktu pencarian, dari data yang terurut dapat langsung diperoleh nilai maksimum dan nilai minimum. Misalnya untuk data numerik yang terurut menurun, nilai maksimum adalah elemen pertama array, dan nilai minimum adalah elemen terakhir array.

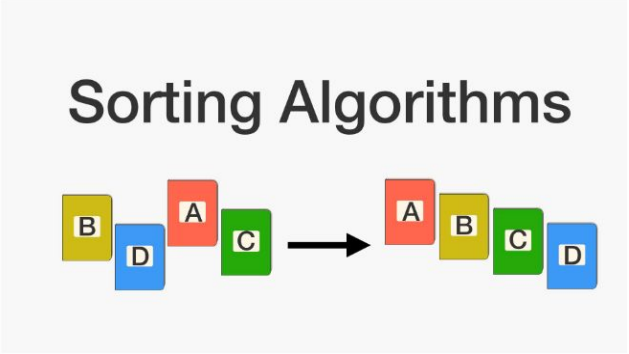
Banyaknya algoritma pengurutan menunjukan bahwa suatu persoalan kaya akan solusi dengan algoritmik. Sehingga seorang programmer harus menentukan mana algortima pengurutan yang membutuhkan waktu algoritma paling sedikit atau efisien. Sehingga diperlukan kriteria formal yang digunakan untuk menilai suatu algoritma yang tebaik. Kriteria tersebut disebut dengan kompleksitas waktu.

Kompleksitas waktu terdiri dari *best case, worst case, dan average case* merupakan hal penting untuk mengukur efisiensi suatu algoritma. Kompleksitas waktu diekspresikan sebagai jumlah tahapan komputasi yang dibutuhkan untuk menjalankan algoritma sebagai fungsi dari ukuran masukan n. Dengan menggunakan kompleksitas waktu, laju peningkatan waktu yang

diperlukan algoritma dapat ditentukan sesuai dengan meningkatnya ukuran masukan n.

pada penelitian ini hanya akan membahas 5 metode *sorting* yaitu *Bubble Sort, Insertion Sort, Selection Sort, Merge Sort, dan Quick Sort.* Dengan elemen dataset yang sudah disediakan {500,1000,10000,20000,100000,200000,1000000,2000000} berupa bilangan *integer* dalam list atau array secara acak (*random*). dari kelima lagortima tersebut pasti memiliki kecepatan waktu yang berbeda beda, ada yang lebih cepat dan ada yang lebih lama dalam melakukan pengurutan. Maka dengan itu untuk mengetahui kecepatan metode pengurutan di butuhkan bahasa pemrograman seabgai pendukung untuk melakukan perhitungan kecepatan algortima.

Bahasa pemrograman yang di gunakan adalah *Python* 3.9.1 dan *Tool Editor* yang digunakan adalah Google Colab dengan alasan karna *flexibilitas* yang sederhana tanpa meng *install* dahulu *library* yang ada di bahasa pemrograman *Python* .



Gambar 1. Gambaran *Sorting*, Referensi: Sorting Algorithms | Brilliant Math & Science Wiki

2. METODE SORTING

Metode yang digunakan dalam analisa terhadap penelitian Algortima pemrograman sorting ini, yaitu:

1. Memahami persoalan yang ada.
2. Membuat code dengan bahasa Python 3.9.1 dari Algoritma *Bubble Sort, Insertion Sort, Selection Sort, Merge Sort, dan Quick Sort.*
3. *Running* program dengan Google Colab.
4. Menganalisis kompleksitas waktu asimptotik T(n) dari setiap algoritma (*Bubble Sort, Insertion Sort, Selection Sort, Merge Sort, dan Quick Sort).*
   3.1. Menghitung banyaknya operasi yang dilakukan algoritma sampai diperoleh T(n)
   3.2. Mendapatkan Notasi O-Besar (worst case) O(f(n)).
   3.3. Jika T(n) yang memenuhi Persamaan maka Notasi O-Besar (worst case).
5. Mencatat waktu dan membandingkan dengan Algoritma *sorting Bubble Sort, Insertion Sort, Selection Sort, Merge Sort, dan Quick Sort* beserta dengan dataset.
6. Membandingkan dari kelima algortima tersebut untuk menentukan algortima yang paling efisien.

2.1 *Bubble Sort*

Bubble sort adalah algoritma pengurutan yang paling sederhana untuk di implementasikan. Algortima ini juga cukup mudah untuk dimengerti. Algoritma ini bekerja dengan cara membandingkan nilai tiap elemen dalam tabel dengan elemen setelahnya, dan menukar nilainya jika sesuai dengan kondisi yang diperlukan. Proses ini akan terus berulang hingga seluruh elemen dalam tabel telah diproses dan elemen dalam tabel telah terurut. Pengurutan gelembung ini adalah algoritma yang paling lamban dan tidak mangkus dibandingkan dengan algoritma pengurutan yang lain dalam penggunaan secara umum.

a) Konsep *Bubble Sort* bekerja adalah sebagai berikut:

1. Bandingkan A [1] dengan A [2] dan susun sehingga A [1] < A [2] .
2. Bandingkan A [2] dengan A [3] dan susun sehingga A [2] < A [3]
3. Bandingkan A [n-1] dengan A [n] dan susun sehingga A [n-1] < A [n] setelah (n-1) kali perbandingan, A [n] akan merupakan elemen terbesar peratama terurut. Langkah ke-2
4. Ulangi step 2 sampai kita telah mebandingkan dan kemungkinan menyusun A [n- 2], A [n-1]. Setelah (n-2) perbandingan, (n-1) akan merupakan elemen terbesar kedua.
5. Dan seterusnya. Langkah ke(n-1)
6. Bandingkan A [1] dengan A [2] dan susun sehingga A [1] < A[2].
7. Sesudah (n-1) langkah, array akan tersususn dalam urutan naik.

b.) Algoritma dan *Pseudocode Bubble Sort* Algoritma *Bubble Sort* adalah sebagai berikut :

```
def bubbleSort()
  n = len(E500)
   for i in range(n-1):
      for j in range(0, n-i-1):
         if E500[j] > E500[j+1] :
            E500[j], E500[j+1] = E500[j+1], E500[j]
```

c) Kompleksitas *Bubble Sort*
Kompleksitas Algoritma Bubble Sort dapat dilihat dari beberapa jenis kasus, yaitu worst-case, average-case, dan best-case. Kondisi best-case data yang akan disorting telah terurut sebelumnya, sehingga proses perbandingan hanya dilakukan sebanyak (n-1) kali, dengan satu kali pass. Proses perbandingan pada bubble sort ini hanya dilakukan sebanyak (n-1) kali. Persamaan Big-O yang diperoleh dari proses ini adalah O(n). Dengan kata lain, pada kondisi Best-Case algoritma Bubble Sort termasuk pada algoritma lanjar. Kondisi worst-case, data terkecil berada pada ujung array. setiap kali melakukan satu pass, data terkecil akan bergeser ke arah awal sebanyak satu step. Dengan kata lain, untuk menggeser data terkecil dari urutan keempat menuju urutan pertama, dibutuhkan pass sebanyak tiga kali, ditambah satu kali pass untuk memverifikasi. Sehingga jumlah proses pada kondisi best case dapat dirumuskan sebagai berikut.

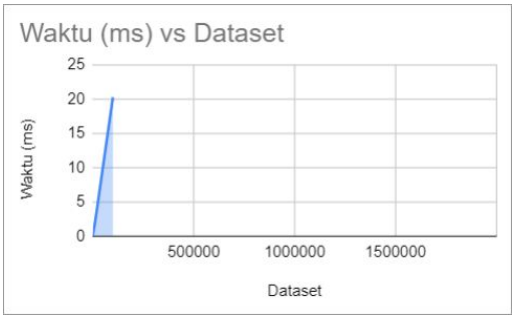
Jumlah proses = n2 + n

Pada kondisi average-case, jumlah pass ditentukan dari elemen mana yang mengalami penggeseran ke kiri paling banyak. Dengan kata lain, jumlah proses perbandingan dapat dihitung sebagai berikut.

Jumlah proses = x2+x.

Bubble Sort	
Dataset	Waktu (ms)
500	0,135
1000	0,255
10000	6,814
20000	24,884
100000	539,186
200000	-
1000000	-
2000000	-

Tabel 1 *Bubble Sort.*



Grafik 1 *Bubble Sort.*

2.2 *Insertion Sort*

dengan penyisipan bekerja dengan cara menyisipkan masing-masing nilai di tempat yang sesuai (di antara elemen yang lebih kecil atau sama dengan nilai tersebut. Untuk menghemat memori, implementasinya menggunakan pengurutan di tempat yang membandingkan elemen saat itu dengan elemen sebelumnya yang sudah diurut, lalu menukarnya terus sampai posisinya tepat. Hal ini terus dilakukan sampai tidak ada elemen tersisa di input.

a) Konsep *Insertion Sort* bekerja adalah sebagai berikut:

1. Elemen awal di masukkan sembarang, lalu elemen berikutnya dimasukkan dibagian paling akhir.
2. Elemen tersebut dibandingkan dengan elemen ke (x-1). Bila belum terurut posisi elemen sebelumnya digeser sekali ke kanan terus sampai elemen yang sedang diproses. menemukan posisi yang tepat atau sampai elemen pertama.
3. Setiap pergeseran akan mengganti nilai elemen berikutnya, namun hal ini tidak menjadi persoalan sebab elemen berikutnya sudah diproses lebih dahulu.

b) Algoritma dan *Pseudocode Insertion Sort* Algoritma *Insertion Sort* adalah sebagai berikut :

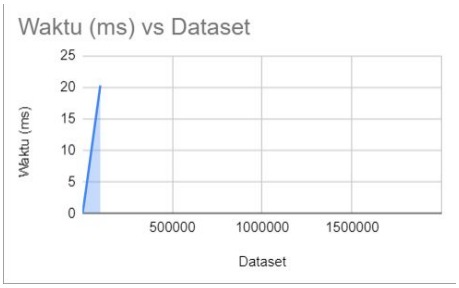
```
def insertionSort()

  for i in range(1, len(E500)):
     key = E500[i]
     j = i-1
     while j >=0 and key < E500[j] :
           E500[j+1] = E500[j]
           j -= 1
     E500[j+1] = key
```

c) Kompleksitas Insertion Sort
Kondisi terbaik (*best case*) tercapai jika data telah terurut. Hanya satu perbandingan dilakukan untuk setiap posisi i,

sehingga terdapat n – 1 perbandingan, atau O (n). Kondisi terburuk (*worst case*) tercapai jika data telah urut namun dengan urutan yang terbalik. Pada kasus ini, untuk setiap i, elemen data[i] lebih kecil dari elemen data[0], …, data[i-1], masing-masing dari elemen dipindahkan satu posisi. Untuk setiap iterasi i pada kalang for terluar, selalu ada perbandingan i.

Insertion Sort	
Dataset	Waktu (ms)
500	0,04
1000	0,212
10000	1,775
20000	3,841
100000	20,294
200000	-
1000000	-
2000000	-

Tabel 2 *Insertion Sort.*



Grafik 2 *Insertion Sort.*

2.3 *Selection Sort*

Secara efisien kita membagi list menjadi dua bagian yaitu bagian yang sudah diurutkan, yang didapat dengan membangun dari kiri ke kanan dan dilakukan pada saat awal, dan bagian list yang elemennya akan diurutkan.

a) Konsep *Selection Sort* bekerja adalah sebagai berikut:

1. Mencari nilai minimum (jika ascending) atau maksimum (jika descending) dalam sebuah list.
2. Menukarkan nilai ini dengan elemen pertama list
3. Mengulangi langkah di atas untuk sisa list dengan dimulai pada posisi kedua.

b) Algoritma dan *Pseudocode Selection Sort* Algoritma *Selection Sort* adalah sebagai berikut :

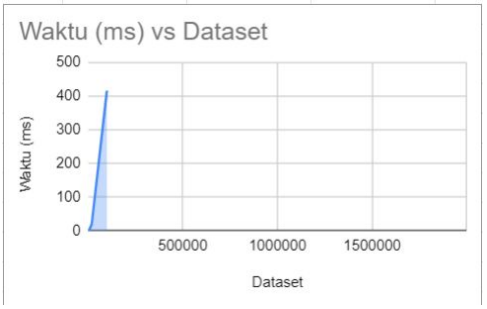
```
def selectionSort()
        for i in range(len(E500)):
            min_idx = i
            for j in range(i+1, len(E500)):
                if E500[min_idx] > E500[j]:
                    min_idx = j
            E500[i], E500[min_idx] = E500[min_idx],
        E500[i]
```

c) Kompleksitas *Selection Sort*

Algoritma di dalam Selection Sort terdiri dari kalang bersarang. Dimana kalang tingkat pertama (disebut pass) berlangsung N-1 kali. Di dalam kalang kedua, dicari elemen dengan nilai terkecil. Jika didapat, indeks yang didapat ditimpakan ke variabel min. Lalu dilakukan proses penukaran. Begitu seterusnya untuk setiap Pass. Pass sendiri makin berkurang hingga nilainya menjadi semakin kecil.

Selection Sort	
Dataset	Waktu (ms)
500	0,07
1000	0,26
10000	5,735
20000	19,688
100000	415,718
200000	-
1000000	-
2000000	-

Tabel 3 *Selection Sort.*



Grafik 3 *Selection Sort.*

2.4 Merge *Sort*

Algoritma pengurutan data merge sort dilakukan dengan menggunakan cara divide and conquer yaitu dengan memecah kemudian menyelesaikan setiap bagian kemudian menggabungkannya kembali. Pertama data dipecah menjadi 2 bagian dimana bagian pertama merupakan setengah (jika data genap) atau setengah minus satu (jika data ganjil) dari seluruh data, kemudian dilakukan pemecahan kembali untuk masing-masing blok sampai hanya terdiri dari satu data tiap blok. Setelah itu

digabungkan kembali dengan membandingkan pada blok yang sama apakah data pertama lebih besar daripada data ke-tengah+1, jika ya maka data ke-tengah+1 dipindah sebagai data pertama, kemudian data ke-pertama sampai ke-tengah digeser menjadi data ke-dua sampai ke-tengah+1, demikian seterusnya sampai menjadi satu blok utuh seperti awalnya. Sehingga metode merge sort merupakan metode yang membutuhkan fungsi rekursi untuk penyelesaiannya.

a) Konsep *Merge Sort* bekerja adalah sebagai berikut:

1. Divide, Memilah elemen – elemen dari rangkaian data menjadi dua bagian.
2. Conquer, Conquer setiap bagian dengan memanggil prosedur merge sort secara rekursif
3. Kombinasi, Mengkombinasikan dua bagian tersebut secara rekursif untuk mendapatkan rangkaian data berurutan.

b) Algoritma dan *Pseudocode Merge Sort* Algoritma *Merge Sort* adalah sebagai berikut :

```
def merge(E500, l, m, r) :
    n1 = m - l + 1
    n2 = r- m

    L = [0] * (n1)
    R = [0] * (n2)

    for i in range(0 , n1):
        L[i] = E500[l + i]

    for j in range(0 , n2):
        R[j] = E500[m + 1 + j]

    i = 0
    j = 0
    k = l

    while i < n1 and j < n2 :
        if L[i] <= R[j]:
            E500[k] = L[i]
            i += 1
        else:
            E500[k] = R[j]
            j += 1
        k += 1

    while i < n1:
        E500[k] = L[i]
        i += 1
        k += 1

    while j < n2:
        E500[k] = R[j]
        j += 1
        k += 1
```

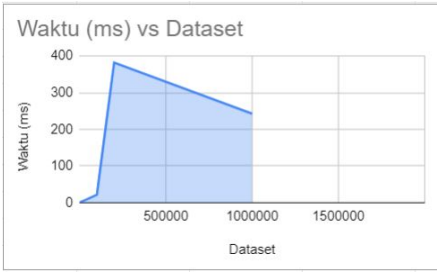
```
def mergeSort(E500,l,r):
    if l < r:
        m = (l+(r-1))//2
        mergeSort(E500, l, m)
        mergeSort(E500, m+1, r)
        merge(E500, l, m, r)
```

c) Kompleksitas *Merge Sort*

Kompleksitas algoritma merge sort adalah O(n log n). Secara umum, algoritma merge sort dapat diimplementasikan secara rekursif. Fungsi rekursif adalah sebuah fungsi yang didalam implementasinya memanggil dirinya sendiri. Pemanggilan diri sendiri ini berakhir jika kondisi tertentu terpenuhi (terminated condition is true). Pada contoh berikut ini, terminated condition dari proses rekursif mergesort akan berakhir jika data tidak dapat dibagi lagi (data tunggal telah diperoleh). Dengan kata lain, proses pembagian data dilakukan terus selama S.size > 1 (belum tunggal).

Merge Sort	
Dataset	Waktu (ms)
500	0,05
1000	0,182
10000	1,969
20000	4,249
100000	21,32
200000	381,94
1000000	242,625
2000000	-

Tabel 4 *Merge Sort.*



Grafik 4 *Merge Sort.*

2.5 *Quick Sort*

Sistem algoritma Quick Sort sendiri adalah membagi kumpulan suatu data menjadi beberapa sub bagian/partisi. Pembagian partisi ini berdasarkan letak dari suatu pivot yang dapat dipilih secara acak. Akan tetapi justru penentuan pivot inilah yang sangat mempengaruhi dalam proses kecepatan sorting. Pemilihan pivot bisa dengan berbagai cara. Bisa dari elemen pertama, elemen tengah,

elemen terakhir atau secara acak. Cara yang diangap paling bagus dan lazim adalah pemilihan pivot pada elemen tengah dari suatu tabel. Karena dengan memilih elemen tengah, tabel tersebut akan dibagi menjadi 2 partisi yang sama besar.

a) Konsep *Quick Sort* bekerja adalah sebagai berikut:

1. Pilih nilai pivot Kita ambil nilai di tengah-tengah elemen sebagai sebagai nilaidari pivot tetapi bisa nilai mana saja.
2. Partisi Atur ulang semua elemen sedemikian rupa, lalu semua elemen yang lebihrendah daripada pivot dipindahkan ke sebelah kiri dari array/list dan semuaelemen yang lebih besar dari pivot dipindahkan ke sebelah kanan dari array/list. Nilai yang sama dengan pivot dapat diletakkan di mana saja dari array. Ingat,mungkin array/list akan dibagi dalam bagian yang tidak sama.
3. Urutkan semua bagian (kiri/kanan) Aplikasikan algoritma quicksort secararekursif pada bagian sebelah kiri dan kanan.

b) Algoritma dan *Pseudocode Quick Sort* Algoritma *Quick Sort* adalah sebagai berikut:

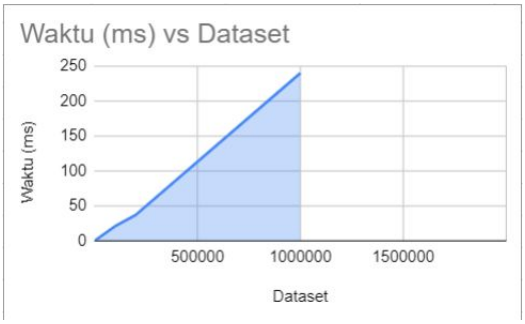
```
def partition(l, bwh, atas):
    pivot = l[bwh]
    pos_batas = bwh+1
    for j in range(bwh+1,atas):
        if l[j] < pivot:
            l[pos_batas],l[j]=l[j],l[pos_batas]
            pos_batas += 1
    l[pos_batas-1],l[bwh] = l[bwh],l[pos_batas-1]
    return pos_batas

def quicksort(l, bwh, atas):
    if atas <= bwh:
        return
    q = partition(l, bwh, atas)
    quicksort(l, bwh, q-1)
    quicksort(l, q, atas)
    return l
```

c) Kompleksitas *Quick Sort*

Efisiensi algoritma Quick Sort sangat dipengaruhi oleh pemilihan elemen pivot. Pemilihan pivot akan menentukan jumlah dan besar partisi pada setiap tahap rekursif. Kasus terbaik (best case) terjadi bila pivot berada pada elemen tengah dan n adalah 2k dimana k=konstanta, sehingga kedua tabel akan selalu berukuran sama setiap pemartisian.

Quick Sort	
Dataset	Waktu (ms)
500	0,08
1000	0,221
10000	1,729
20000	4,118
100000	20,771
200000	36,937
1000000	240,356
2000000	-

Tabel 5 *Quick Sort.*



Grafik 6 *Quick Sort.*

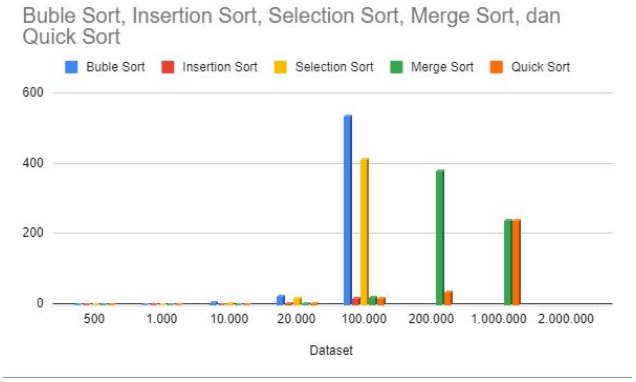
3. Hasil eksperimen/pengujian

Dari hasil pengujian didapatkan waktu dari berbagai dataset dan tipe pengurutan (*Sorting*) sebagai berikut dalam bentuk tabel dan grafik :

Dataset	**Sort**				
	Buble Sort	**Insertion Sort**	**Selection Sort**	**Merge Sort**	**Quick Sort**
500	0.135	0.04	0.07	0.05	0.08
1.000	0.255	0.212	0.26	0.182	0.221
10.000	6.814	1.775	5.735	1.969	1.729

20.000	24.884	3.841	19.688	4.249	4.118
100.000	539.186	20.294	415.718	21.32	20.771
200.000	-	-	-	381.94	36.937
1.000.000	-	-	-	242.625	240.356
2.000.000	-	-	-	-	-

Tabel 6 Hasil Pengujian.



Grafik 6 Hasil Pengujian.

Pengujian dilakukan dengan spesifikasi laptop sebagai berikut:

Device specifications

Device name	DaffaRegenta-LAPTOP
Processor	AMD Ryzen 5 3500U with Radeon Vega Mobile Gfx 2.10 GHz
Installed RAM	8,00 GB (5,95 GB usable)
Device ID	DABD105C-7045-48A6-9765-8FAF6E8A0027
Product ID	00327-35851-86550-AAOEM
System type	64-bit operating system, x64-based processor
Pen and touch	No pen or touch input is available for this display

Gambar 3 Spesifikasi Laptop.

Dari gambar hasil program diatas (Gambar 2) bisa kita lihat perbedaan waktu yang cukup jelas di antara algoritma-algoritma tersebut. Pengurutan data dengan metode quicksort lebih cepat jika dibandingkan dengan algritma yang lain. Hal ini dibuktikan dengan keadaan grafik waktu perbandingan setiap algoritma. Sedangkan untuk *Bubble Sort dan Selection Sort* lebih lama untuk pengurutan dengan jumlah dataset yang lebih banyak, bahkan untuk dataset yang lebih tinggi misalnya 1.000.000 sudah sangat lama untuk meng eksekusi .

4. KESIMPULAN

Teknik – teknik pengurutan data memang cukup beragam, namun tentunya dalam proses ada metode yang tercepat. masingmasing metode mempunyai kelebihan dan kelemahan. Apalagi dalam pembuatan program komputer tidak lepas dari algoritma, karena program yang dibuat sangat kompleks. Program dapat dibuat tanpa menggunakan algoritma, akan tetapi program tersebut memiliki akses yang lambat atau memakai banyak memori. Berdasarkan logika proses pengurutan data dengan menggunakan algoritma *Bubble Sort, Insertion Sort, Selection Sort, Merge Sort, dan Quick Sort* maka dapat di simpulkan sebagai berikut:

1. Algoritma *Quick Sort lebih* lebih cepat dalam melakukan pengurutan data jika dibandingkan *Bubble Sort, Insertion Sort, Selection Sort, Merge Sort.*
2. Dalam pengurutan data *Bubble sort* yang paling mudah di implementasikan dan paling mudah dipahami.
3. Selection Sort lebih cepat mengurutkan data dibandingkan algoritma bubble Sort. Hal ini ditunjukkan kecilnya nilai yang didapat oleh algoritma tersebut.
4. Untuk dataset yang banyak >100.000 didapatkan pengurutan dengan metode *Bubble sort* dan *Selection sort* sangat memakan waktu yang lama sehingga untuk dataset >100.000 tidak dijalankan dengan maksimal.

LAMPIRAN

a) Dataset 500

```
Dataset ke- 497
Dataset ke- 498
Dataset ke- 499
Dataset ke- 500
Sort: Bubble Sort
Lama eksekusi:  0.13561833200037654 Detik
============================
Pilih Sorting: [1] Bubble Sort
               [2] Insertion Sort
               [3] Selection Sort
               [4] Merge Sort
               [5] Quick Sort
               [0] Exit
Pilihan:
```

```
Dataset ke- 497
Dataset ke- 498
Dataset ke- 499
Dataset ke- 500
Sort: Insertion Sort
Lama eksekusi:  0.038655307999761135 Detik
============================
Pilih Sorting: [1] Bubble Sort
               [2] Insertion Sort
               [3] Selection Sort
               [4] Merge Sort
               [5] Quick Sort
               [0] Exit
Pilihan:
```

```
Dataset ke- 497
Dataset ke- 498
Dataset ke- 499
Dataset ke- 500
Sort: Merge Sort
Lama eksekusi:  0.058013179000226955 Detik
============================
Pilih Sorting: [1] Bubble Sort
               [2] Insertion Sort
               [3] Selection Sort
               [4] Merge Sort
               [5] Quick Sort
               [0] Exit
Pilihan:
```

```
Dataset ke- 498
Dataset ke- 499
Dataset ke- 500
Sort: Quick Sort
Lama eksekusi:  0.08417186499991658 Detik
============================
Pilih Sorting: [1] Bubble Sort
               [2] Insertion Sort
               [3] Selection Sort
               [4] Merge Sort
               [5] Quick Sort
               [0] Exit
Pilihan:
```

```
Dataset ke- 497
Dataset ke- 498
Dataset ke- 499
Dataset ke- 500
Sort: Selection Sort
Lama eksekusi:  0.06608439499996166 detik
============================
Pilih Sorting: [1] Bubble Sort
               [2] Insertion Sort
               [3] Selection Sort
               [4] Merge Sort
               [5] Quick Sort
               [0] Exit
Pilihan:
```

b) Dataset 1000

```
Dataset ke- 998
Dataset ke- 999
Dataset ke- 1000
Sort: Bubble Sort
Lama eksekusi:  0.2552641089999952 Detik
============================
Pilih Sorting: [1] Bubble Sort
               [2] Insertion Sort
               [3] Selection Sort
               [4] Merge Sort
               [5] Quick Sort
               [0] Exit
Pilihan:
```

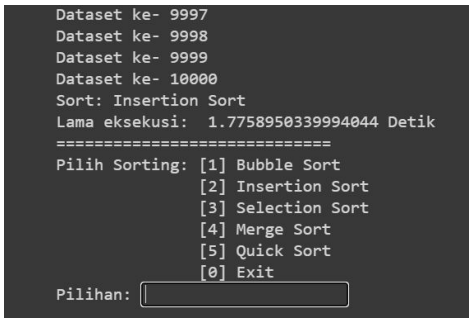
```
Dataset ke- 998
Dataset ke- 999
Dataset ke- 1000
Sort: Insertion Sort
Lama eksekusi:  0.21285381500001677 Detik
============================
Pilih Sorting: [1] Bubble Sort
               [2] Insertion Sort
               [3] Selection Sort
               [4] Merge Sort
               [5] Quick Sort
               [0] Exit
Pilihan:
```

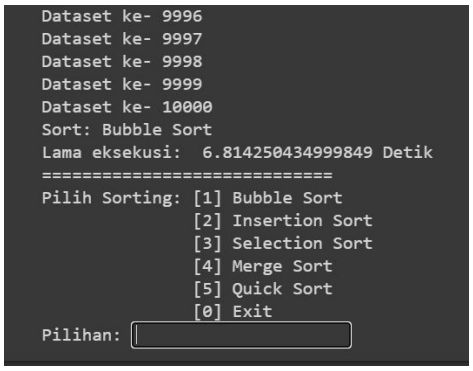
```
Dataset ke- 999
Dataset ke- 1000
Sort: Merge Sort
Lama eksekusi:  0.1825749669999368 Detik
============================
Pilih Sorting: [1] Bubble Sort
               [2] Insertion Sort
               [3] Selection Sort
               [4] Merge Sort
               [5] Quick Sort
               [0] Exit
Pilihan:
```

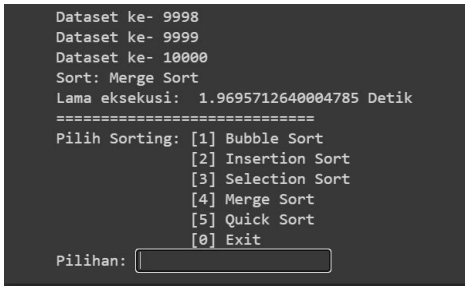
```
Dataset ke- 998
Dataset ke- 999
Dataset ke- 1000
Sort: Quick Sort
Lama eksekusi:  0.2215832599999885 Detik
============================
Pilih Sorting: [1] Bubble Sort
               [2] Insertion Sort
               [3] Selection Sort
               [4] Merge Sort
               [5] Quick Sort
               [0] Exit
Pilihan:
```

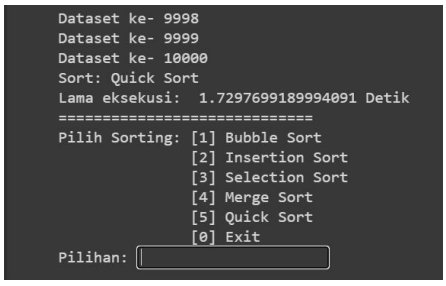
```
Dataset ke- 998
Dataset ke- 999
Dataset ke- 1000
Sort: Selection Sort
Lama eksekusi:  0.2601900490000162 detik
============================
Pilih Sorting: [1] Bubble Sort
               [2] Insertion Sort
               [3] Selection Sort
               [4] Merge Sort
               [5] Quick Sort
               [0] Exit
Pilihan:
```

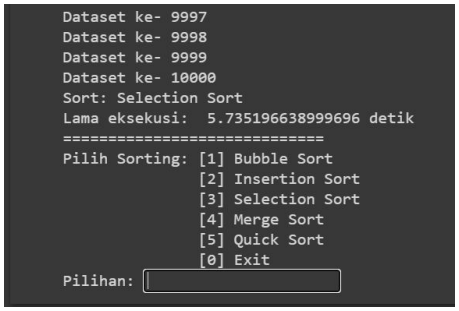
c) Dataset 10.000



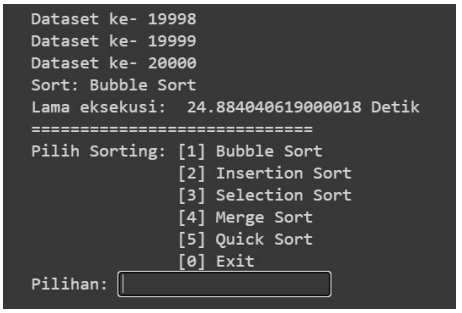


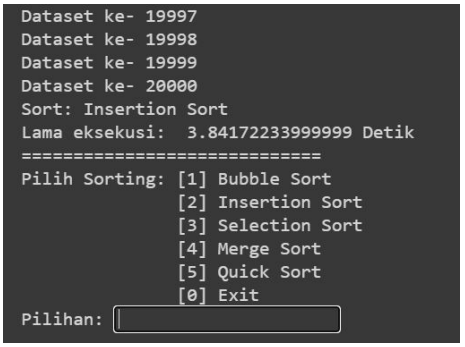


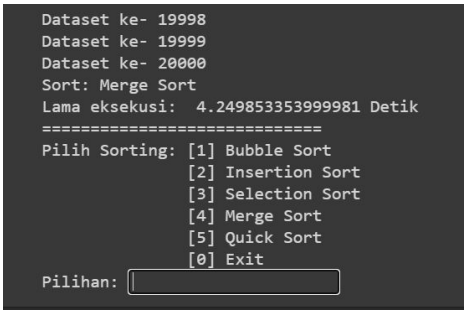


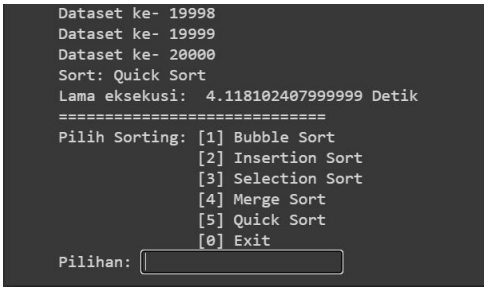


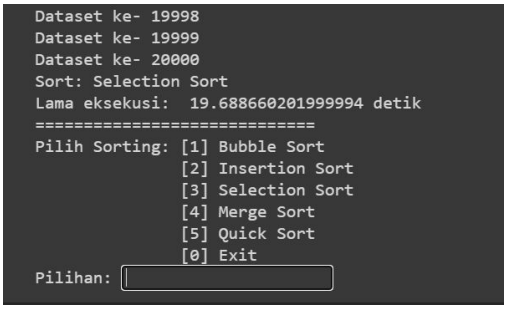
d) Dataset 20.000











e) Dataset 100.000

```
Dataset ke- 99997
Dataset ke- 99998
Dataset ke- 99999
Dataset ke- 100000
Sort: Bubble Sort
Lama eksekusi:  539.1867975570003 Detik
============================
Pilih Sorting: [1] Bubble Sort
               [2] Insertion Sort
               [3] Selection Sort
               [4] Merge Sort
               [5] Quick Sort
               [0] Exit
Pilihan:
```

```
Dataset ke- 99999
Dataset ke- 100000
Sort: Insertion Sort
Lama eksekusi:  20.294111901999713 Detik
============================
Pilih Sorting: [1] Bubble Sort
               [2] Insertion Sort
               [3] Selection Sort
               [4] Merge Sort
               [5] Quick Sort
               [0] Exit
Pilihan:
```

```
Dataset ke- 99998
Dataset ke- 99999
Dataset ke- 100000
Sort: Merge Sort
Lama eksekusi:  21.32078960600029 Detik
============================
Pilih Sorting: [1] Bubble Sort
               [2] Insertion Sort
               [3] Selection Sort
               [4] Merge Sort
               [5] Quick Sort
               [0] Exit
Pilihan:
```

```
Dataset ke- 99998
Dataset ke- 99999
Dataset ke- 100000
Sort: Quick Sort
Lama eksekusi:  20.771771314000034 Detik
============================
Pilih Sorting: [1] Bubble Sort
               [2] Insertion Sort
               [3] Selection Sort
               [4] Merge Sort
               [5] Quick Sort
               [0] Exit
Pilihan:
```

```
Dataset ke- 99998
Dataset ke- 99999
Dataset ke- 100000
Sort: Selection Sort
Lama eksekusi:  415.7183379949993 detik
============================
Pilih Sorting: [1] Bubble Sort
               [2] Insertion Sort
               [3] Selection Sort
               [4] Merge Sort
               [5] Quick Sort
               [0] Exit
Pilihan:
```

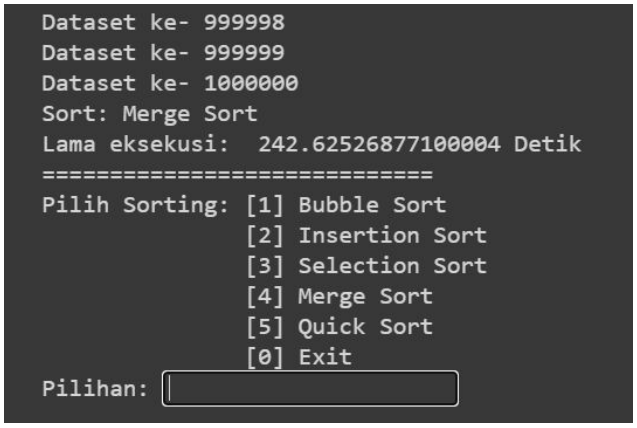
f) Dataset 200.000

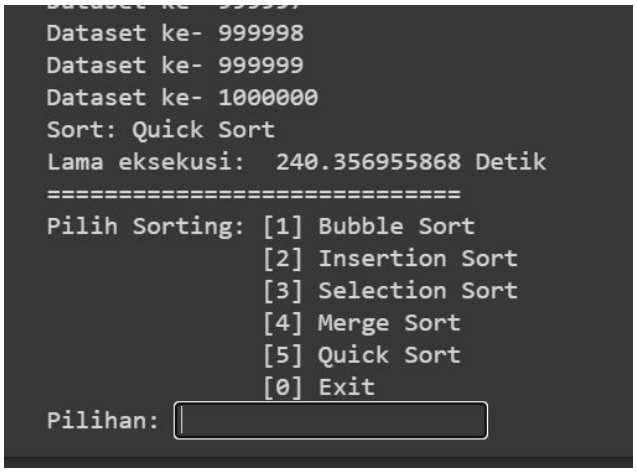
```
Dataset ke- 199998
Dataset ke- 199999
Dataset ke- 200000
Sort: Merge Sort
Lama eksekusi:  38.194338061999986 Detik
============================
Pilih Sorting: [1] Bubble Sort
               [2] Insertion Sort
               [3] Selection Sort
               [4] Merge Sort
               [5] Quick Sort
               [0] Exit
Pilihan:
```

```
Dataset ke- 199999
Dataset ke- 200000
Sort: Quick Sort
Lama eksekusi:  36.937770207 Detik
============================
Pilih Sorting: [1] Bubble Sort
               [2] Insertion Sort
               [3] Selection Sort
               [4] Merge Sort
               [5] Quick Sort
               [0] Exit
Pilihan:
```

```
Dataset ke- 199998
Dataset ke- 199999
Dataset ke- 200000
Sort: Merge Sort
Lama eksekusi:  38.194338061999986 Detik
============================
Pilih Sorting: [1] Bubble Sort
               [2] Insertion Sort
               [3] Selection Sort
               [4] Merge Sort
               [5] Quick Sort
               [0] Exit
Pilihan: 3
```

g) Dataset 1.000.000





REFERENSI

[1] **Pengertian Quick Sort dan implementasi (catatan-ati.blogspot.com)**

[2] **mergesort-edited (unsyiah.ac.id)**

[3]https://ejournal.bsi.ac.id/ejurnal/index.php/evolusi/article/download/702/577

[4]http://seminar.ilkom.unsri.ac.id/index.php/ars/article/download/1683/848