LAPORAN PRAKTIKUM ALGORITMA & STRUKTUR DATA MODUL 5



SORTING

Oleh:

Indra Suryadilaga NIM. 2410817310014

PROGRAM STUDI TEKNOLOGI INFORMASI FAKULTAS TEKNIK UNIVERSITAS LAMBUNG MANGKURAT MEI 2025

LEMBAR PENGESAHAN

LAPORAN PRAKTIKUM ALGORITMA & STRUKTUR DATA MODUL 5

Laporan Praktikum Algoritma & Struktur Data Modul 5: Sorting ini disusun sebagai syarat lulus mata kuliah Praktikum Algoritma & Struktur Data. Laporan Prakitkum ini dikerjakan oleh:

Nama Praktikan : Indra Suryadilaga

NIM : 2410817310014

Menyetujui, Mengetahui,

Asisten Praktikum Dosen Penanggung Jawab Praktikum

Zulfa Auliya Akbar Andreyan Rizky Baskara, S.Kom., M.Kom

NIM. 2210817210010 NIP. 199307032019031011

DAFTAR ISI

LEMBAR PENGESAHAN			
	.R ISI		
DAFTA	R GAMBAR	4	
SOAL 1		5	
A.	Source Code	7	
B.	Output Program	16	
C.	Pembahasan	18	
GITHU	B	25	

DAFTAR GAMBAR

Gambar 1. UI Program Sorting	<i>.</i>
Gambar 2. Output Insertion Sort	16
Gambar 3. Output Merge Sort	16
Gambar 4. Output Shell Sort	16
Gambar 5. Output Bubble Sort	17
Gambar 6. Output Quick Sort	17
Gambar 7. Output Selection Sort	17

DAFTAR GAMBAR

Table	1. Source	Code Algoritma	Sorting	7
-------	-----------	----------------	---------	---

SOAL 1

Buat Program Sederhana Menggunakan Nama dan Angka NIM Masing-masing:

- Insertion Sort (Nama)
- Merge Sort (Nama)
- Shell Sort (Nama)
- Quick Sort (NIM)
- Bubble Sort (NIM)
- Selection Sort (NIM)

```
1. Insertion Sort
2. Merge Sort
3. Shell Sort
4. Quick Sort
5. Bubble Sort
6. Selection Sort
7. Exit

Masukkan Pilihan:
```

Gambar 1. UI Program Sorting

A. Source Code

Table 1. Source Code Algoritma Sorting

```
#include <iostream>
1
2
    #include <functional>
    #include <chrono>
3
    #include <string>
4
    #include <iomanip>
5
    #include <conio.h>
6
7
    using namespace std;
8
9
    // Ganti dengan nama dan NIM Anda
10
    11
    Anda
    12
    Anda
13
    // Function untuk mengukur waktu eksekusi
14
    void timeSort(const function<void()>& sortFunc, const
15
    string& sortName) {
        auto start =
16
    chrono::high resolution clock::now();
       sortFunc();
17
       auto end = chrono::high resolution clock::now();
18
        chrono::duration<double> duration = end - start;
19
20
       cout << fixed << setprecision(10);</pre>
21
       cout << sortName << " took " << duration.count()</pre>
22
    << " seconds\n";
23
```

```
24
25
     // 1. INSERTION SORT untuk karakter dalam string
     void insertionSort(string &str) {
26
27
         for (int i = 1; i < str.size(); i++) {
28
             char key = str[i];
             int j = i - 1;
29
30
             while (j \ge 0 \&\& str[j] > key) {
31
                 str[j + 1] = str[j];
32
                 j--;
33
34
             }
35
             str[j + 1] = key;
36
         }
37
38
     }
39
     // 2. MERGE SORT untuk karakter dalam string
40
     void merge(string &str, int left, int mid, int right)
41
         int n1 = mid - left + 1;
42
43
         int n2 = right - mid;
44
         char *tempL = new char[n1];
45
         char *tempR = new char[n2];
46
47
         for (int i = 0; i < n1; i++) tempL[i] = str[left
48
    + i];
         for (int j = 0; j < n2; j++) tempR[j] = str[mid +
49
     1 + j];
50
```

```
int i = 0, j = 0, k = left;
51
52
         while (i < n1 \&\& j < n2) {
53
              if (tempL[i] <= tempR[j]) {</pre>
54
                 str[k] = tempL[i];
55
                 i++;
56
              } else {
57
                  str[k] = tempR[j];
58
                  j++;
59
              }
60
             k++;
61
         }
62
63
         while (i < n1) {
64
              str[k] = tempL[i];
65
              i++;
66
67
              k++;
68
         }
69
         while (j < n2) {
70
71
             str[k] = tempR[j];
              j++;
72
             k++;
73
         }
74
75
76
         delete[] tempL;
77
                          delete[] tempR;
     }
78
79
```

```
void mergeSort(string &str, int left, int right) {
80
         if (left < right) {</pre>
81
             int mid = left + (right - left) / 2;
82
83
             mergeSort(str, left, mid);
84
             mergeSort(str, mid + 1, right);
             merge(str, left, mid, right);
85
         }
86
87
88
     // 3. SHELL SORT untuk karakter dalam string
89
     void shellSort(string &str, int n) {
90
         for (int gap = n/2; gap > 0; gap /= 2) {
91
             for (int i = gap; i < n; i++) {
92
                  char temp = str[i];
93
94
                 int j;
95
                  for (j = i; j \ge gap \&\& str[j - gap] >
96
     temp; j -= gap) {
97
                      str[j] = str[j - gap];
                  }
98
99
                 str[j] = temp;
100
101
             }
102
         }
103
104
     // 4. BUBBLE SORT untuk karakter dalam string
105
     void bubbleSort(string &str) {
106
107
         for (int i = 0; i < str.size() - 1; i++) {
             bool swapped = false;
108
```

```
109
             for (int j = 0; j < str.size() - i - 1; j++)
110
     {
111
                  if (str[j] > str[j + 1]) {
                      swap(str[j], str[j + 1]);
112
                      swapped = true;
113
                  }
114
115
             }
116
             if (!swapped) break;
117
118
         }
119
120
     // 5. QUICK SORT untuk karakter dalam string
121
     int partition(string &str, int low, int high) {
122
123
         char pivot = str[high];
124
         int i = (low - 1);
125
         for (int j = low; j \le high - 1; j++) {
126
127
             if (str[j] <= pivot) {</pre>
128
                  i++;
                  swap(str[i], str[j]);
129
130
             }
         }
131
132
         swap(str[i + 1], str[high]);
133
         return (i + 1);
134
135
136
    void quickSort(string &str, int low, int high) {
137
```

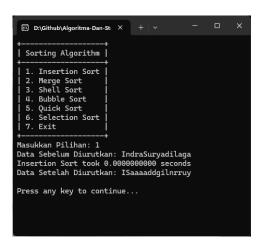
```
if (low < high) {
138
139
             int p idx = partition(str, low, high);
             quickSort(str, low, p idx - 1);
140
141
             quickSort(str, p idx + 1, high);
142
         }
143
144
     // 6. SELECTION SORT untuk karakter dalam string
145
     void selectionSort(string &str) {
146
         for (int i = 0; i < str.size() - 1; i++) {
147
             int minIndex = i;
148
149
             for (int j = i + 1; j < str.size(); j++) {
150
                 if (str[j] < str[minIndex]) {</pre>
151
                     minIndex = j;
152
153
                 }
154
             }
155
156
             swap(str[i], str[minIndex]);
         }
157
158
159
    int main() {
160
         int ch;
161
162
         string temp;
163
         do {
164
             cout << "+========+" << endl;
165
             cout << "| Sorting Algorithm |" << endl;</pre>
166
```

```
cout << "+========+" << endl;</pre>
167
              cout << "| 1. Insertion Sort |" << endl;</pre>
168
              cout << "| 2. Merge Sort | " << endl;</pre>
169
              cout << "| 3. Shell Sort | " << endl;</pre>
170
              cout << "| 4. Bubble Sort | " << endl;</pre>
171
              cout << "| 5. Quick Sort |" << endl;</pre>
172
              cout << "| 6. Selection Sort |" << endl;</pre>
173
             cout << "| 7. Exit
                                              |" << endl;
174
             cout << "+========+" << endl;
175
             cout << "Masukkan Pilihan: ";</pre>
176
             cin >> ch;
177
178
             switch(ch) {
179
180
                  case 1:
                      temp = name;
181
                       cout << "Data Sebelum Diurutkan: " <<</pre>
182
     temp << endl;</pre>
                       timeSort([&]() {insertionSort(temp);
183
     }, "Insertion Sort");
                      cout << "Data Setelah Diurutkan: " <<</pre>
184
     temp << endl;</pre>
                      break:
185
                  case 2:
186
                      temp = name;
187
                      cout << "Data Sebelum Diurutkan: " <<</pre>
188
     temp << endl;</pre>
                      timeSort([&]() {mergeSort(temp, 0,
189
     temp.size() - 1); }, "Merge Sort");
                      cout << "Data Setelah Diurutkan: " <<</pre>
190
     temp << endl;</pre>
```

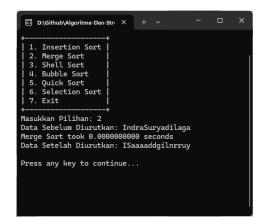
```
191
                       break;
                   case 3:
192
193
                       temp = name;
194
                       cout << "Data Sebelum Diurutkan: " <<</pre>
     temp << endl;</pre>
195
                      timeSort([&]() {shellSort(temp,
     temp.size()); }, "Shell Sort");
                       cout << "Data Setelah Diurutkan: " <<</pre>
196
     temp << endl;</pre>
                       break;
197
                   case 4:
198
199
                       temp = id;
                       cout << "Data Sebelum Diurutkan: " <<</pre>
200
     temp << endl;</pre>
201
                       timeSort([&]() {bubbleSort(temp); },
     "Bubble Sort");
                      cout << "Data Setelah Diurutkan: " <<</pre>
202
     temp << endl;</pre>
203
                       break;
                   case 5:
204
205
                       temp = id;
                       cout << "Data Sebelum Diurutkan: " <<</pre>
206
     temp << endl;</pre>
                       timeSort([&]() {quickSort(temp, 0,
207
     temp.size() - 1); }, "Quick Sort");
                       cout << "Data Setelah Diurutkan: " <<</pre>
208
     temp << endl;</pre>
                       break;
209
                   case 6:
210
                       temp = id;
211
```

```
cout << "Data Sebelum Diurutkan: " <<</pre>
212
     temp << endl;</pre>
                       timeSort([&]() {selectionSort(temp);
213
     }, "Selection Sort");
                        cout << "Data Setelah Diurutkan: " <<</pre>
214
     temp << endl;</pre>
215
                       break;
                   case 7:
216
                        cout << "Terima Kasih" << endl;</pre>
217
                       cout << "This Program Was Made by</pre>
218
     [Indra Suryadilaga] (2410817310014)" << endl;</pre>
                       break;
219
                   default:
220
                       cout << "Opsi Tidak Valid. Silahkan</pre>
221
     Coba Lagi." << endl;</pre>
222
              }
223
              cout << "\nPress any key to continue..." <<</pre>
     endl;
224
              getch();
              system("cls");
225
          \} while (ch != 7);
226
227
228
          return 0;
229
```

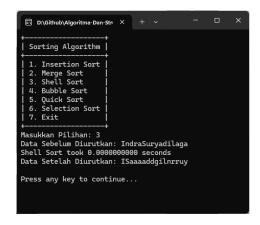
B. Output Program



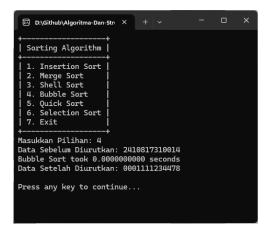
Gambar 2. Output Insertion Sort



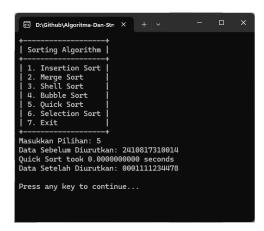
Gambar 3. Output Merge Sort



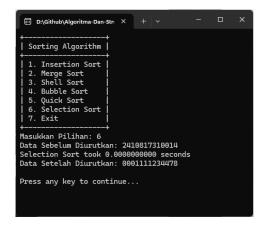
Gambar 4. Output Shell Sort



Gambar 5. Output Bubble Sort



Gambar 6. Output Quick Sort



Gambar 7. Output Selection Sort

C. Pembahasan

Alur Program Sorting

Ketika program dijalankan, sistem akan menampilkan menu utama dengan 7 pilihan sorting algorithm yang tersedia. Program menggunakan struktur do-while loop yang akan terus berjalan hingga user memilih opsi 7 (Exit). Setiap kali user memilih opsi 1-6, program akan melakukan proses sorting sesuai dengan algoritma yang dipilih.

Untuk opsi 1-3 (Insertion Sort, Merge Sort, dan Shell Sort), program akan menggunakan string name yang berisi "IndraSuryadilaga" sebagai data input. Sedangkan untuk opsi 4-6 (Bubble Sort, Quick Sort, dan Selection Sort), program menggunakan string id yang berisi "2410817310014" sebagai data input.

Dalam setiap proses sorting, program akan menampilkan data sebelum diurutkan, kemudian memanggil fungsi timeSort() yang menggunakan lambda function untuk mengukur waktu eksekusi algoritma sorting. Function timeSort() menggunakan chrono::high_resolution_clock untuk mendapatkan waktu yang presisi hingga 10 digit decimal. Setelah proses sorting selesai, program menampilkan data yang telah diurutkan dan waktu eksekusi yang dibutuhkan.

Pada akhir setiap operasi sorting, program akan menampilkan pesan "Press any key to continue..." dan menunggu input dari user menggunakan getch(). Setelah user menekan tombol apapun, layar akan dibersihkan menggunakan system("cls") dan menu utama akan ditampilkan kembali. Loop ini akan terus berlanjut hingga user memilih opsi 7 untuk keluar dari program.

Analisis Algoritma Insertion Sort

Algoritma Insertion Sort bekerja dengan cara mengambil elemen satu per satu dari bagian yang belum terurut, kemudian menyisipkannya ke posisi yang tepat di bagian yang sudah terurut. Dalam implementasi ini, algoritma dimulai dari indeks ke-1 (i=1) dan menggunakan variabel key untuk menyimpan karakter yang akan disisipkan.

Proses sorting dimulai dengan mengambil karakter pada posisi i sebagai key, kemudian membandingkannya dengan karakter-karakter sebelumnya (j = i-1). Jika karakter sebelumnya lebih besar dari key, maka karakter tersebut akan digeser ke kanan. Proses pergeseran ini berlanjut hingga ditemukan posisi yang tepat untuk key atau hingga mencapai awal array.

Kelebihan dari Insertion Sort adalah algoritma ini sangat efisien untuk data berukuran kecil dan memiliki data sudah hampir terurut. Algoritma ini juga bersifat stable, mempertahankan urutan relatif elemen yang memiliki nilai sama. Selain itu, Insertion Sort merupakan in-place sorting algorithm yang hanya membutuhkan O(1) extra memory space.

Kekurangan utama Insertion Sort terletak pada time complexity yang kurang efisien untuk data berukuran besar. Pada worst case (data terurut terbalik), algoritma ini memiliki time complexity $O(n^2)$ karena setiap elemen harus dibandingkan dengan semua elemen sebelumnya. Average case juga memiliki time complexity $O(n^2)$, sedangkan best case (data sudah terurut) memiliki time complexity O(n).

Analisis Algoritma Merge Sort

Algoritma Merge Sort menggunakan pendekatan membagi array menjadi dua bagian secara rekursif hingga mencapai elemen tunggal, kemudian menggabungkan kembali dengan cara yang terurut. Implementasi ini menggunakan dua fungsi utama: mergeSort() untuk membagi array dan merge() untuk menggabungkan sub-array.

Proses sorting dimulai dengan menentukan titik tengah array menggunakan formula mid = left + (right - left) / 2. Kemudian fungsi mergeSort() dipanggil secara rekursif untuk bagian kiri dan kanan. Setelah kedua bagian telah terurut, fungsi merge() akan menggabungkan kedua sub-array dengan membandingkan elemen-elemen dari kedua bagian dan menyusunnya secara ascending.

Dalam fungsi merge(), program menggunakan dynamic memory allocation untuk membuat temporary array tempL dan tempR yang menyimpan data dari subarray kiri dan kanan. Proses merging dilakukan dengan membandingkan elemen terdepan dari kedua temporary array, kemudian memilih yang lebih kecil untuk dimasukkan ke array utama. Setelah salah satu array habis, sisa elemen dari array lainnya akan disalin ke array utama.

Kelebihan utama Merge Sort adalah time complexity yang stabil O(n log n) pada semua kasus (best, average, dan worst case). Algoritma ini juga bersifat stable dan memiliki performa yang predictable. Merge Sort sangat cocok untuk data berukuran besar dan dapat diparalelkan dengan mudah karena sifat divide and conquernya.

Kekurangan Merge Sort terletak pada space complexity yang tinggi, yaitu O(n) karena membutuhkan additional memory untuk temporary arrays. Algoritma ini juga tidak bersifat in-place, sehingga membutuhkan memory tambahan yang signifikan. Untuk data berukuran kecil, overhead dari recursive calls dan memory allocation dapat membuat performanya lebih lambat dibanding algoritma sederhana seperti Insertion Sort.

Analisis Algoritma Shell Sort

Algoritma Shell Sort merupakan pengembangan dari Insertion Sort yang menggunakan konsep gap sequence untuk mengurangi jumlah perbandingan. Implementasi ini menggunakan gap sequence yang dimulai dari n/2 dan terus dibagi dua hingga mencapai 1. Pada setiap gap, algoritma melakukan insertion sort pada elemen-elemen yang berjarak gap.

Proses sorting dimulai dengan gap = n/2, kemudian melakukan insertion sort pada sub-array yang terdiri dari elemen-elemen dengan jarak gap. Misalnya dengan gap = 4, elemen pada indeks 0, 4, 8, 12 akan diurutkan terlebih dahulu, kemudian elemen pada indeks 1, 5, 9, 13, dan seterusnya. Setelah semua sub-array dengan gap tertentu selesai diurutkan, gap akan diperkecil menjadi gap/2 dan proses diulangi.

Ketika gap = 1, Shell Sort akan melakukan insertion sort pada seluruh array. Namun pada tahap ini, array sudah hampir terurut karena proses-proses sebelumnya, sehingga insertion sort akan berjalan dengan sangat efisien. Inilah yang membuat Shell Sort lebih cepat dari Insertion Sort biasa.

Kelebihan Shell Sort adalah time complexity yang lebih baik dari O(n²) pada kebanyakan kasus praktis. Dengan gap sequence yang tepat, Shell Sort dapat mencapai time complexity O(n^1.5) atau bahkan lebih baik. Algoritma ini juga bersifat in-place dan tidak membutuhkan extra memory yang signifikan. Shell Sort juga adaptive, performanya akan lebih baik pada data yang sudah hampir terurut.

Kekurangan Shell Sort terletak pada time complexity yang bergantung pada gap sequence yang digunakan. Pemilihan gap sequence yang tidak optimal dapat membuat performanya mendekati $O(n^2)$. Algoritma ini juga tidak stable, sehingga urutan relatif elemen dengan nilai sama tidak terjamin. Analisis time complexity Shell Sort juga lebih kompleks dibanding algoritma lainnya.

Analisis Algoritma Bubble Sort

Algoritma Bubble Sort bekerja dengan cara membandingkan pasangan elemen yang bersebelahan dan menukarnya jika urutannya salah. Proses ini diulang secara berulang hingga tidak ada lagi pertukaran yang diperlukan. Implementasi ini menggunakan optimasi dengan flag swapped untuk mendeteksi apakah masih ada pertukaran pada iterasi tertentu.

Proses sorting dimulai dengan loop luar yang berjalan sebanyak n-1 kali, di mana n adalah jumlah elemen. Pada setiap iterasi loop luar, loop dalam akan membandingkan elemen-elemen bersebelahan dari awal hingga akhir yang belum terurut. Jika elemen kiri lebih besar dari elemen kanan, kedua elemen akan ditukar dan flag swapped akan di-set menjadi true.

Optimasi yang digunakan adalah early termination, yaitu jika pada suatu iterasi tidak ada pertukaran yang terjadi (swapped = false), berarti array sudah terurut dan proses sorting dapat dihentikan. Hal ini membuat Bubble Sort memiliki best case time complexity O(n) ketika data sudah terurut, meskipun average dan worst case tetap $O(n^2)$.

Kelebihan Bubble Sort adalah kesederhanaannya yang membuatnya mudah dipahami dan diimplementasikan. Algoritma ini juga bersifat stable dan in-place, serta dapat mendeteksi apakah array sudah terurut dengan menggunakan flag optimasi. Bubble Sort juga memiliki adaptive property pada implementasi yang dioptimasi.

Kekurangan utama Bubble Sort adalah time complexity $O(n^2)$ yang membuatnya tidak efisien untuk data berukuran besar. Algoritma ini juga melakukan banyak pertukaran yang tidak perlu, sehingga performanya lebih lambat dibanding algoritma $O(n^2)$ lainnya seperti Selection Sort. Bubble Sort juga tidak cocok untuk aplikasi yang membutuhkan performa tinggi.

Analisis Algoritma Quick Sort

Algoritma Quick Sort menggunakan strategi divide and conquer dengan memilih satu elemen sebagai pivot, kemudian mempartisi array sehingga elemen yang lebih kecil dari pivot berada di sebelah kiri dan elemen yang lebih besar berada di sebelah kanan. Implementasi ini menggunakan elemen terakhir sebagai pivot dan menggunakan Lomuto partition scheme.

Proses sorting dimulai dengan memanggil fungsi partition() yang akan menentukan posisi akhir pivot setelah partitioning. Dalam fungsi partition, variabel i digunakan untuk melacak batas antara elemen yang lebih kecil dan lebih besar dari pivot. Setiap kali ditemukan elemen yang lebih kecil atau sama dengan pivot, elemen tersebut akan ditukar ke bagian kiri array dan i akan diincrement.

Setelah partitioning selesai, pivot akan ditempatkan pada posisi yang tepat dengan menukar elemen pada posisi i+1 dengan pivot. Kemudian fungsi quickSort() akan dipanggil secara rekursif untuk sub-array kiri (low to pivot-1) dan sub-array kanan (pivot+1 to high). Proses rekursi akan berhenti ketika sub-array hanya memiliki satu elemen atau kosong.

Kelebihan utama Quick Sort adalah average case time complexity yang sangat baik yaitu O(n log n), dan biasanya lebih cepat dari algoritma O(n log n) lainnya dalam praktik. Algoritma ini juga bersifat in-place dan memiliki space complexity O(log n) karena recursive calls. Quick Sort juga memiliki good cache locality yang membuatnya efisien pada sistem modern.

Kekurangan Quick Sort terletak pada worst case time complexity O(n²) yang terjadi ketika pivot yang dipilih selalu merupakan elemen terkecil atau terbesar. Algoritma ini juga tidak stable dan performanya sangat bergantung pada pemilihan pivot. Pada implementasi rekursif, Quick Sort dapat menyebabkan stack overflow untuk data berukuran sangat besar jika tidak dioptimasi dengan tail recursion atau iterative approach.

Analisis Algoritma Selection Sort

Algoritma Selection Sort bekerja dengan cara mencari elemen terkecil dalam array, kemudian menukarnya dengan elemen pertama. Proses ini diulang untuk posisi kedua, ketiga, dan seterusnya hingga seluruh array terurut. Implementasi ini menggunakan variabel minIndex untuk melacak posisi elemen terkecil pada setiap iterasi.

Proses sorting dimulai dengan loop luar yang berjalan dari indeks 0 hingga n-2. Pada setiap iterasi, program akan mencari elemen terkecil dalam sub-array yang dimulai dari posisi i hingga akhir array. Pencarian dilakukan dengan loop dalam yang membandingkan setiap elemen dengan elemen pada minIndex dan mengupdate minIndex jika ditemukan elemen yang lebih kecil.

Setelah menemukan elemen terkecil, program akan menukarnya dengan elemen pada posisi i menggunakan fungsi swap(). Proses ini akan berlanjut hingga seluruh array terurut. Karakteristik unik dari Selection Sort adalah jumlah pertukaran yang minimal, yaitu maksimal n-1 kali pertukaran untuk n elemen.

Kelebihan Selection Sort adalah jumlah pertukaran yang minimal, membuatnya cocok untuk aplikasi di mana operasi swap memiliki cost yang tinggi. Algoritma ini juga bersifat in-place dan memiliki implementasi yang sederhana. Selection Sort juga memiliki performa yang konsisten karena selalu melakukan n(n-1)/2 perbandingan terlepas dari kondisi awal data.

Kekurangan utama Selection Sort adalah time complexity O(n²) pada semua kasus, membuatnya tidak efisien untuk data berukuran besar. Algoritma ini juga tidak stable dan tidak adaptive, artinya performanya tidak akan membaik meskipun data sudah hampir terurut. Selection Sort juga melakukan banyak perbandingan yang tidak perlu pada kasus-kasus tertentu.

GITHUB

 $\underline{https://github.com/IndraSuryadilaga/Algoritma-Dan-Struktur-Data}$