

**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 Praktikum ini dikerjakan oleh:

Nama Praktikan : Indra Suryadilaga
NIM : 2410817310014

Menyetujui,
Asisten Praktikum

Mengetahui,
Dosen Penanggung Jawab Praktikum

Zulfa Auliya Akbar
NIM. 2210817210010

Andreyan Rizky Baskara, S.Kom., M.Kom
NIP. 199307032019031011

DAFTAR ISI

LEMBAR PENGESAHAN	2
DAFTAR ISI.....	3
DAFTAR GAMBAR	4
SOAL 1	5
A. Source Code	7
B. Output Program.....	16
C. Pembahasan.....	18
GITHUB.....	25

DAFTAR GAMBAR

Gambar 1. UI Program Sorting	6
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)



Gambar 1. UI Program Sorting

A. Source Code

Table 1. Source Code Algoritma Sorting

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

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

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

```

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

```

```

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

```

```

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

```

```

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

```

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

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

B. Output Program

```
D:\Github\Algoritma-Dan-Str x + - □ x
+-----+
| Sorting Algorithm |
+-----+
| 1. Insertion Sort |
| 2. Merge Sort     |
| 3. Shell Sort     |
| 4. Bubble Sort    |
| 5. Quick Sort     |
| 6. Selection Sort |
| 7. Exit           |
+-----+
Masukkan Pilihan: 1
Data Sebelum Diurutkan: IndraSuryadilaga
Insertion Sort took 0.000000000 seconds
Data Setelah Diurutkan: ISaaaaddgilnrruy

Press any key to continue...
```

Gambar 2. Output Insertion Sort

```
D:\Github\Algoritma-Dan-Str x + - □ x
+-----+
| 1. Insertion Sort |
| 2. Merge Sort     |
| 3. Shell Sort     |
| 4. Bubble Sort    |
| 5. Quick Sort     |
| 6. Selection Sort |
| 7. Exit           |
+-----+
Masukkan Pilihan: 2
Data Sebelum Diurutkan: IndraSuryadilaga
Merge Sort took 0.000000000 seconds
Data Setelah Diurutkan: ISaaaaddgilnrruy

Press any key to continue...
```

Gambar 3. Output Merge Sort

```
D:\Github\Algoritma-Dan-Str x + - □ x
+-----+
| Sorting Algorithm |
+-----+
| 1. Insertion Sort |
| 2. Merge Sort     |
| 3. Shell Sort     |
| 4. Bubble Sort    |
| 5. Quick Sort     |
| 6. Selection Sort |
| 7. Exit           |
+-----+
Masukkan Pilihan: 3
Data Sebelum Diurutkan: IndraSuryadilaga
Shell Sort took 0.000000000 seconds
Data Setelah Diurutkan: ISaaaaddgilnrruy

Press any key to continue...
```

Gambar 4. Output Shell Sort


```
D:\Github\Algoritma-Dan-Stri x + - □ x
+-----+
| Sorting Algorithm |
+-----+
| 1. Insertion Sort |
| 2. Merge Sort    |
| 3. Shell Sort    |
| 4. Bubble Sort   |
| 5. Quick Sort    |
| 6. Selection Sort |
| 7. Exit          |
+-----+
Masukkan Pilihan: 4
Data Sebelum Diurutkan: 2410817310014
Bubble Sort took 0.0000000000 seconds
Data Setelah Diurutkan: 0001111234478

Press any key to continue...
```

Gambar 5. Output Bubble Sort

```
D:\Github\Algoritma-Dan-Stri x + - □ x
+-----+
| Sorting Algorithm |
+-----+
| 1. Insertion Sort |
| 2. Merge Sort    |
| 3. Shell Sort    |
| 4. Bubble Sort   |
| 5. Quick Sort    |
| 6. Selection Sort |
| 7. Exit          |
+-----+
Masukkan Pilihan: 5
Data Sebelum Diurutkan: 2410817310014
Quick Sort took 0.0000000000 seconds
Data Setelah Diurutkan: 0001111234478

Press any key to continue...
```

Gambar 6. Output Quick Sort

```
D:\Github\Algoritma-Dan-Stri x + - □ x
+-----+
| Sorting Algorithm |
+-----+
| 1. Insertion Sort |
| 2. Merge Sort    |
| 3. Shell Sort    |
| 4. Bubble Sort   |
| 5. Quick Sort    |
| 6. Selection Sort |
| 7. Exit          |
+-----+
Masukkan Pilihan: 6
Data Sebelum Diurutkan: 2410817310014
Selection Sort took 0.0000000000 seconds
Data Setelah Diurutkan: 0001111234478

Press any key to continue...
```

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 sub-array 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 conquer-nya.

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 $O(n^2)$ 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 $O(n^2)$ 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 $O(n^2)$ 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

<https://github.com/IndraSuryadilaga/Algoritma-Dan-Struktur-Data>