

**LAPORAN PRAKTIKUM
ALGORITMA & STRUKTUR DATA
MODUL 5**



Sorting

Oleh:

Rizki Adhitiya Maulana

NIM. 2410817110014

**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 : Rizki Adhitiya Maulana
NIM : 2410817110014

Menyetujui,
Asisten Praktikum

Mengetahui,
Dosen Penanggung Jawab Praktikum

Muhammad Fauzan Ahsani
NIM. 2310817310009

Muti'a Maulida, S.Kom., M.TI.
NIP. 198810272019032013

DAFTAR ISI

LEMBAR PENGESAHAN	i
DAFTAR ISI.....	ii
DAFTAR TABEL.....	iii
DAFTAR GAMBAR	iv
SOAL 1	1
A Output Program.....	2
B Output Program.....	13
C Pembahasan.....	20
TAUTAN GITHUB	32

DAFTAR TABEL

DAFTAR GAMBAR

Gambar 1, Tampilan Contoh Soal Menu Sorting.....	1
Gambar 2 Tampilan Menu Sorting	13
Gambar 3 Pilihan 1 Pada Menu Sorting.....	13
Gambar 4 Masukkan Nama Pada Pilihan 1.....	13
Gambar 5 Tampilan Hasil Dari Insertion Sort	14
Gambar 6 Pilihan 2 Pada Menu Sorting.....	14
Gambar 7 Masukkan Nama Pada Pilihan 2.....	14
Gambar 8 Tampilan Hasil Dari Merge Sort.....	15
Gambar 9 Pilihan 3 Pada Menu Sorting.....	15
Gambar 10 Masukkan Nama Pada Pilihan 3.....	15
Gambar 11 Tampilan Hasil Dari Shell Sort	16
Gambar 12 Pilihan 4 Pada Menu Sorting.....	16
Gambar 13 Masukkan ID Pada Pilihan 4.....	16
Gambar 14 Tampilan Hasil Dari Bubble Sort.....	17
Gambar 15 Pilihan 5 Pada Menu Sorting.....	17
Gambar 16 Masukkan ID Pada Pilihan 5	17
Gambar 17 Tampilan Hasil Dari Quick Sort.....	18
Gambar 18 Pilihan 6 Pada Menu Sorting.....	18
Gambar 19 Masukkan ID Pada Pilihan 6.....	18
Gambar 20 Tampilan Hasil Dari Selection Sort	19
Gambar 21 Pilihan 7 Pada Menu Sorting.....	19
Gambar 22 Tampilan Program Selesai	19
Gambar 23 Ilustrasi Insertion Sort	21
Gambar 24 Ilustrasi Merge Sort.....	23
Gambar 25 Ilustrasi Shell Sort	25
Gambar 26 Ilustrasi Bubblu Sort.....	27
Gambar 27 Ilustrasi Quick Sort.....	29
Gambar 28 Ilustrasi Selection Sort	30

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, Tampilan Contoh Soal Menu Sorting

A Output Program

```

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 string name, id;
11
12 void timeSort(const function<void()>& sortFunc,
13              const string& sortName)
14 {
15     auto start =
16     chrono::high_resolution_clock::now();
17     sortFunc();
18     auto end =
19     chrono::high_resolution_clock::now();
20     chrono::duration<double> duration = end -
21     start;
22
23     cout << fixed << setprecision(10);
24     cout << sortName << " took " <<
25     duration.count() << " seconds\n";
26 }
27
28 void insertionSort(string &str)
29 {

```

```
25     for (int i = 1; i < str.size(); i++)
26     {
27         char key = str[i];
28         int j = i - 1;
29
30         while (j >= 0 && str[j] > key)
31         {
32             str[j + 1] = str[j];
33             j--;
34         }
35
36         str[j + 1] = key;
37     }
38 }
39
40 void merge(string &str, int left, int mid, int
right)
41 {
42     int n1 = mid - left + 1;
43     int n2 = right - mid;
44
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
+ i];
49     for (int j = 0; j < n2; j++) tempR[j] = str[mid
+ 1 + j];
50
51     int i = 0, j = 0, k = left;
```



```
52
53     while (i < n1 && j < n2)
54     {
55         if (tempL[i] <= tempR[j])
56         {
57             str[k] = tempL[i];
58             i++;
59         }
60         else
61         {
62             str[k] = tempR[j];
63             j++;
64         }
65         k++;
66     }
67
68     while (i < n1)
69     {
70         str[k] = tempL[i];
71         i++;
72         k++;
73     }
74
75     while (j < n2)
76     {
77         str[k] = tempR[j];
78         j++;
79         k++;
80     }
81
```

```
82     delete[] tempL;
83     delete[] tempR;
84 }
85
86 void mergeSort(string &str, int left, int right)
87 {
88     if (left < right)
89     {
90         int mid = left + (right - left) / 2;
91         mergeSort(str, left, mid);
92         mergeSort(str, mid + 1, right);
93         merge(str, left, mid, right);
94     }
95 }
96
97 void shellSort(string &str, int n)
98 {
99     for (int gap = n/2; gap > 0; gap /= 2)
100    {
101        for (int i = gap; i < n; i++)
102        {
103            int temp = str[i];
104
105            int j;
106            for (j = i; j >= gap && str[j - gap]
107 > temp; j -= gap) str[j] = str[j - gap];
108
109            str[j] = temp;
110        }
111    }
```

```
111 }
112
113 void bubbleSort(string &str)
114 {
115     for (int i = 0; i < str.size() - 1; i++)
116     {
117         bool swapped = false;
118
119         for (int j = 0; j < str.size() - i - 1;
120 j++)
121         {
122             if (str[j] > str[j + 1])
123             {
124                 swap(str[j], str[j + 1]);
125                 swapped = true;
126             }
127
128             if (!swapped) break;
129         }
130     }
131
132 int partition(string &str, int low, int high)
133 {
134     int pivot = str[high];
135     int i = (low - 1);
136
137     for (int j = low; j <= high - 1; j++)
138     {
139         if (str[j] <= pivot)
```

```
140         {
141             i++;
142             swap(str[i], str[j]);
143         }
144     }
145
146     swap(str[i + 1], str[high]);
147
148     return (i + 1);
149 }
150
151 void quickSort(string &str, int low, int high)
152 {
153     if (low < high)
154     {
155         int p_idx = partition(str, low, high);
156         quickSort(str, low, p_idx - 1);
157         quickSort(str, p_idx + 1, high);
158     }
159 }
160
161 void selectionSort(string &str)
162 {
163     for (int i = 0; i < str.size() - 1; i++)
164     {
165         int minIndex = i;
166
167         for (int j = i + 1; j < str.size(); j++)
168         {
169             if (str[j] < str[minIndex])
```

```

170         {
171             minIndex = j;
172         }
173     }
174
175     swap(str[i], str[minIndex]);
176 }
177 }
178
179 int main()
180 {
181     int ch;
182     string temp;
183
184     do
185     {
186         cout << "+=====+" << endl;
187         cout << "|      Sorting      |" << endl;
188         cout << "+=====+" << endl;
189         cout << "| 1. Insertion Sort |" << endl;
190         cout << "| 2. Merge Sort    |" << endl;
191         cout << "| 3. Shell Sort    |" << endl;
192         cout << "| 4. Bubble Sort   |" << endl;
193         cout << "| 5. Quick Sort    |" << endl;
194         cout << "| 6. Selection Sort|" << endl;
195         cout << "| 7. Exit          |" << endl;
196         cout << "+=====+" << endl;
197         cout << "Masukkan Pilihan: "; cin >> ch;
198
199         system("cls");

```

```

200
201         switch(ch)
202         {
203             case 1:
204                 cout << "Masukkan Nama: ";
205                 cin.ignore();
206                 getline(cin, name);
207
208                 system("cls");
209
210                 temp = name;
211                 cout << "Data Sebelum Diurutkan:
212 " << temp << endl;
213
214                 timeSort([&]()
215 {insertionSort(temp); }, "Insertion Sort");
216
217                 cout << "Data Setelah Diurutkan:
218 " << temp << endl;
219
220                 break;
221             case 2:
222                 cout << "Masukkan Nama: ";
223                 cin.ignore();
224                 getline(cin, name);
225
226                 system("cls");
227
228                 temp = name;
229                 cout << "Data Sebelum Diurutkan:
230 " << temp << endl;
231
232                 timeSort([&]() {mergeSort(temp, 0,
233 temp.size() - 1); }, "Merge Sort");

```

```
225         cout << "Data Setelah Diurutkan:
      " << temp << endl;
226         break;
227         case 3:
228             cout << "Masukkan Nama: ";
229             cin.ignore();
230             getline(cin, name);
231
232             system("cls");
233
234             temp = name;
235             cout << "Data Sebelum Diurutkan:
      " << temp << endl;
236             timeSort([&]() {shellSort(temp,
temp.size()); }, "Shell Sort");
237             cout << "Data Setelah Diurutkan:
      " << temp << endl;
238             break;
239             case 4:
240                 cout << "Masukkan ID: "; cin >>
id;
241
242                 system("cls");
243
244                 temp = id;
245                 cout << "Data Sebelum Diurutkan:
      " << temp << endl;
246                 timeSort([&]() {bubbleSort(temp);
}, "Bubble Sort");
```

```

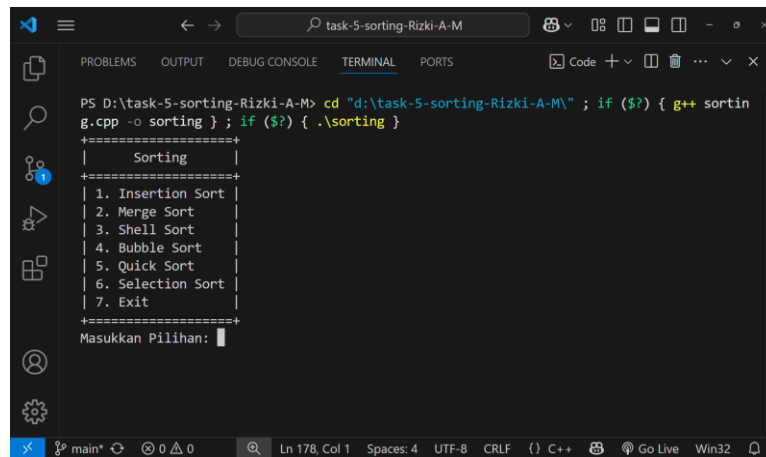
247         cout << "Data Setelah Diurutkan:
      " << temp << endl;
248         break;
249         case 5:
250             cout << "Masukkan ID: "; cin >>
id;
251
252             system("cls");
253
254             temp = id;
255             cout << "Data Sebelum Diurutkan:
      " << temp << endl;
256             timeSort([&]() {quickSort(temp, 0,
temp.size() - 1); }, "Quick Sort");
257             cout << "Data Setelah Diurutkan:
      " << temp << endl;
258             break;
259             case 6:
260                 cout << "Masukkan ID: "; cin >>
id;
261
262                 system("cls");
263
264                 temp = id;
265                 cout << "Data Sebelum Diurutkan:
      " << temp << endl;
266                                     timeSort([&]()
{selectionSort(temp); }, "Selection Sort");
267                 cout << "Data Setelah Diurutkan:
      " << temp << endl;

```



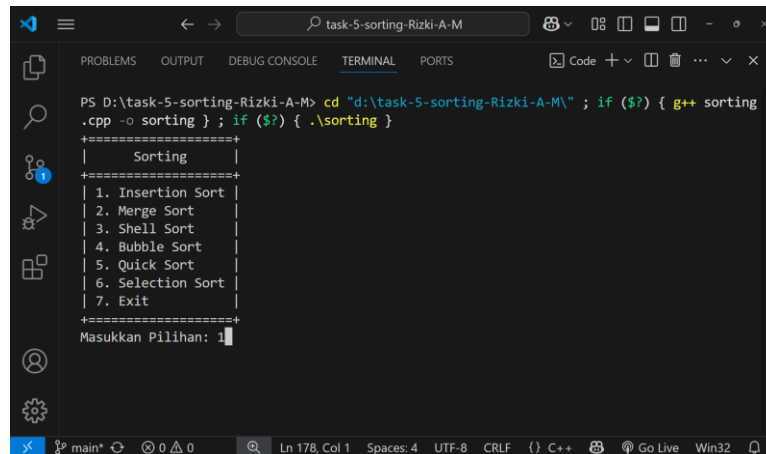
```
268         break;
269         case 7:
270             cout << "Terima Kasih" << endl;
271             cout << "This Program Was Made by
Rizki Adhitiya Maulana (2410817110014)" << endl;
272             break;
273             default:
274                 cout << "Opsi Tidak Valid. Silahkan
Coba Lagi." << endl;
275             }
276             cout << "Press any key to continue...";
277             getch();
278             system("cls");
279         }
280         while (ch != 7);
281
282         return 0;
283     }
```

B Output Program



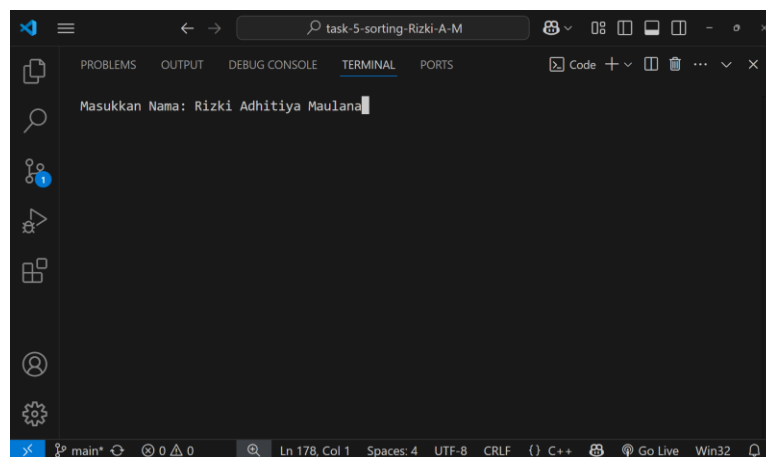
```
PS D:\task-5-sorting-Rizki-A-M> cd "d:\task-5-sorting-Rizki-A-M\" ; if ($?) { g++ sortin
g.cpp -o sorting } ; if ($?) { .\sorting }
+=====+
| Sorting |
+=====+
| 1. Insertion Sort |
| 2. Merge Sort    |
| 3. Shell Sort    |
| 4. Bubble Sort   |
| 5. Quick Sort    |
| 6. Selection Sort|
| 7. Exit          |
+=====+
Masukkan Pilihan: |
```

Gambar 2 Tampilan Menu Sorting



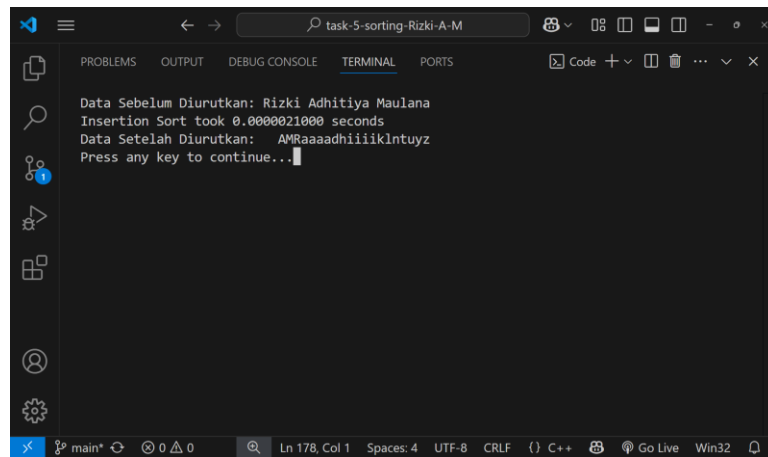
```
PS D:\task-5-sorting-Rizki-A-M> cd "d:\task-5-sorting-Rizki-A-M\" ; if ($?) { g++ sorting
.cpp -o sorting } ; if ($?) { .\sorting }
+=====+
| Sorting |
+=====+
| 1. Insertion Sort |
| 2. Merge Sort    |
| 3. Shell Sort    |
| 4. Bubble Sort   |
| 5. Quick Sort    |
| 6. Selection Sort|
| 7. Exit          |
+=====+
Masukkan Pilihan: 1|
```

Gambar 3 Pilihan 1 Pada Menu Sorting



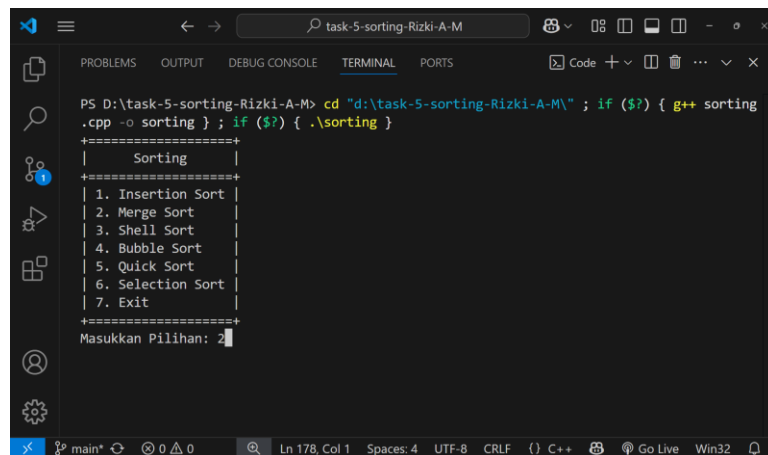
```
Masukkan Nama: Rizki Adhitiya Maulana|
```

Gambar 4 Masukkan Nama Pada Pilihan 1



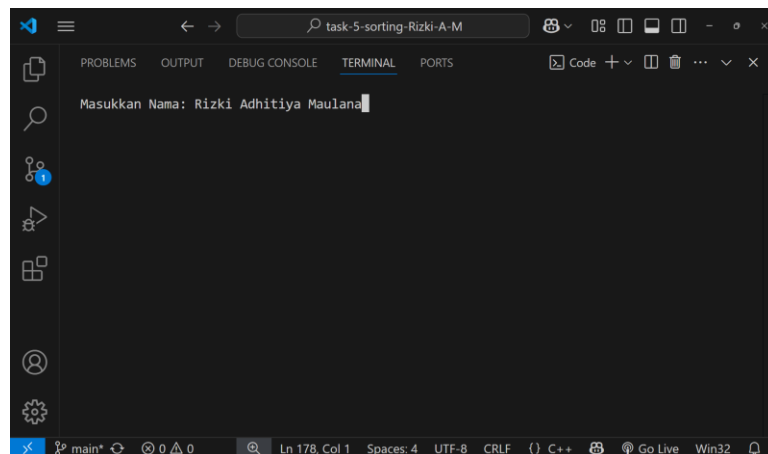
```
task-5-sorting-Rizki-A-M
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
Data Sebelum Diurutkan: Rizki Adhitiya Maulana
Insertion Sort took 0.0000021000 seconds
Data Setelah Diurutkan:  AMRaaaadhiikIntuyz
Press any key to continue...
```

Gambar 5 Tampilan Hasil Dari Insertion Sort



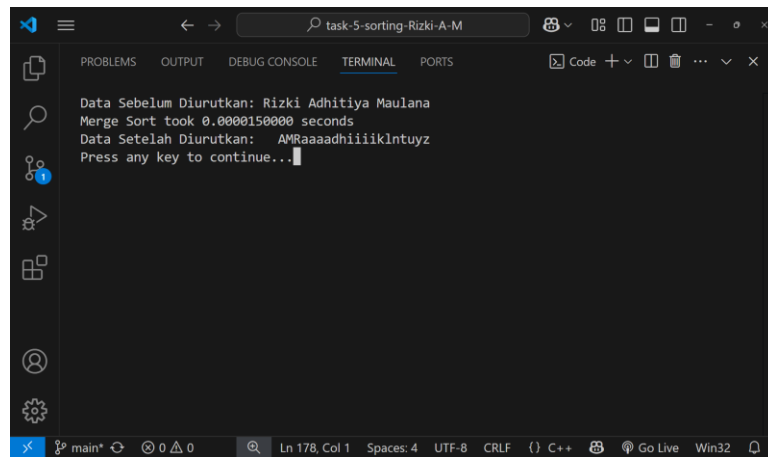
```
task-5-sorting-Rizki-A-M
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
PS D:\task-5-sorting-Rizki-A-M> cd "d:\task-5-sorting-Rizki-A-M\" ; if ($?) { g++ sorting
.cpp -o sorting } ; if ($?) { .\sorting }
+=====+
|      Sorting      |
+=====+
| 1. Insertion Sort |
| 2. Merge Sort    |
| 3. Shell Sort    |
| 4. Bubble Sort   |
| 5. Quick Sort    |
| 6. Selection Sort|
| 7. Exit          |
+=====+
Masukkan Pilihan: 2
```

Gambar 6 Pilihan 2 Pada Menu Sorting



```
task-5-sorting-Rizki-A-M
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
Masukkan Nama: Rizki Adhitiya Maulana
```

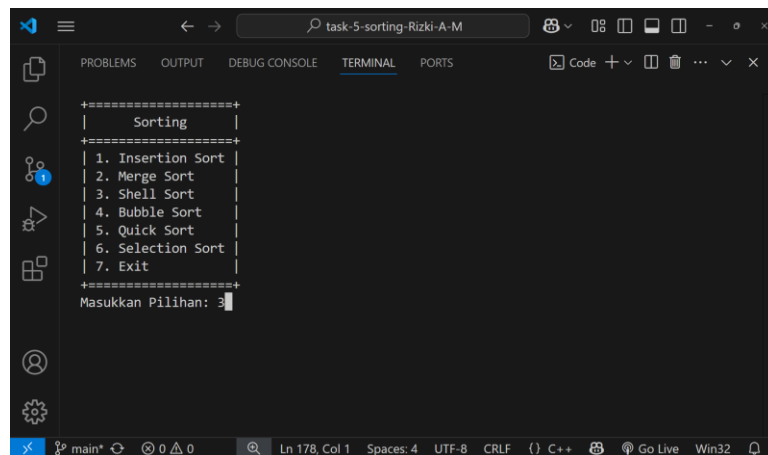
Gambar 7 Masukkan Nama Pada Pilihan 2



```
task-5-sorting-Rizki-A-M

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
Data Sebelum Diurutkan: Rizki Adhitiya Maulana
Merge Sort took 0.0000150000 seconds
Data Setelah Diurutkan:  AMRaaaadhiikIntuyz
Press any key to continue...
```

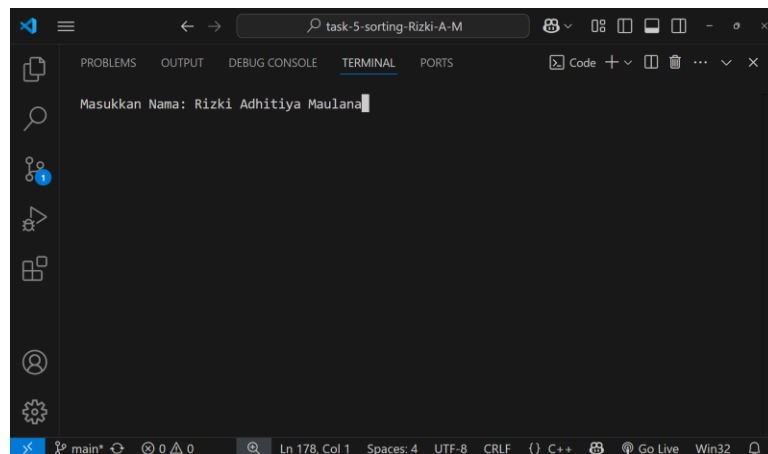
Gambar 8 Tampilan Hasil Dari Merge Sort



```
task-5-sorting-Rizki-A-M

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
+=====+
|      Sorting      |
+=====+
| 1. Insertion Sort |
| 2. Merge Sort    |
| 3. Shell Sort    |
| 4. Bubble Sort   |
| 5. Quick Sort    |
| 6. Selection Sort|
| 7. Exit          |
+=====+
Masukkan Pilihan: 3
```

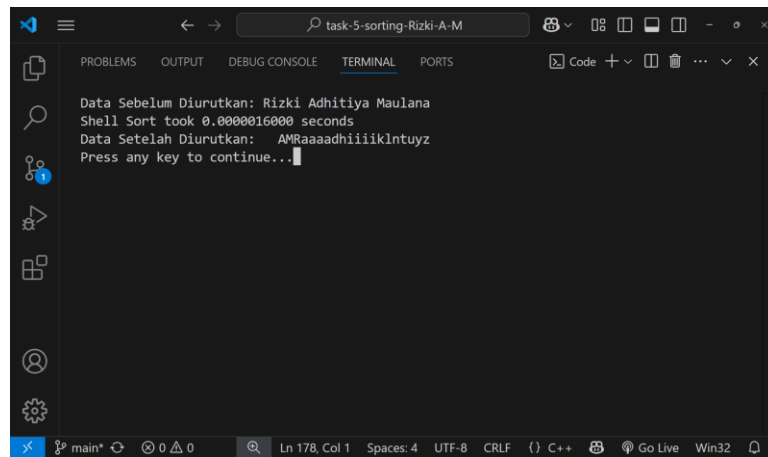
Gambar 9 Pilihan 3 Pada Menu Sorting



```
task-5-sorting-Rizki-A-M

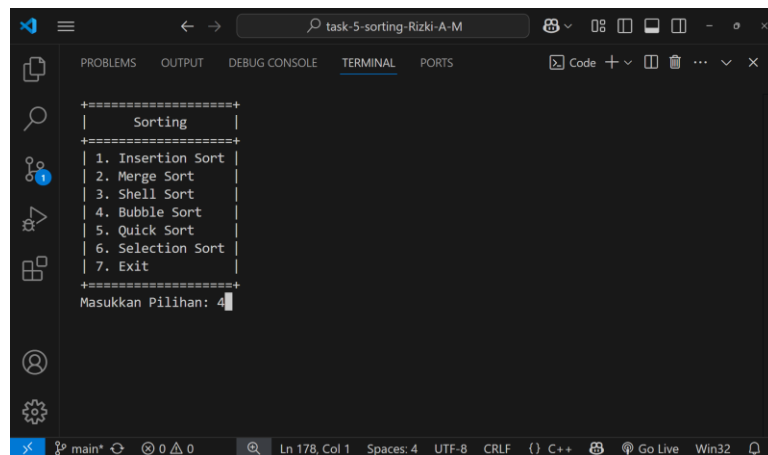
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
Masukkan Nama: Rizki Adhitiya Maulana
```

Gambar 10 Masukkan Nama Pada Pilihan 3



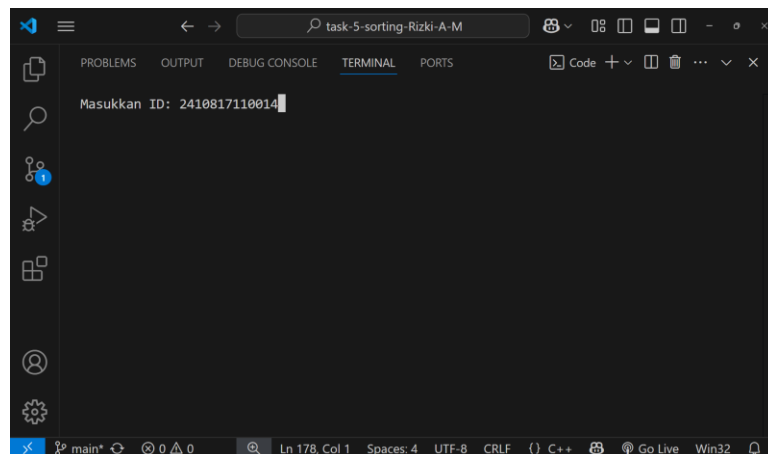
```
task-5-sorting-Rizki-A-M
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
Data Sebelum Diurutkan: Rizki Adhitiya Maulana
Shell Sort took 0.0000016000 seconds
Data Setelah Diurutkan: AMRaaaadhiikIntuyz
Press any key to continue...
```

Gambar 11 Tampilan Hasil Dari Shell Sort



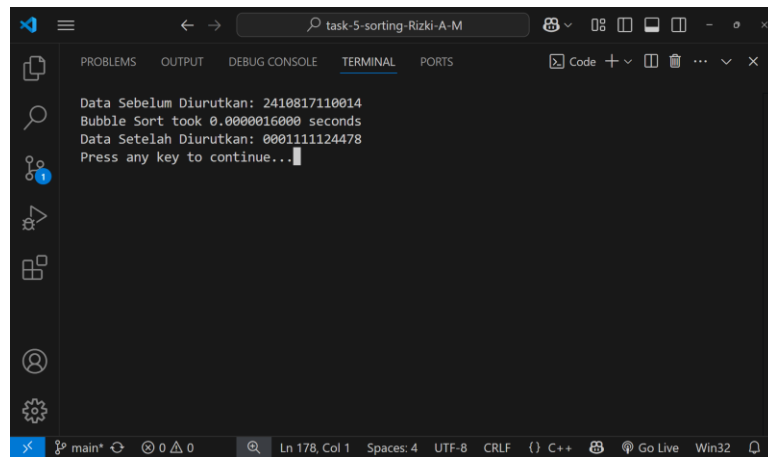
```
task-5-sorting-Rizki-A-M
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
+=====+
|      Sorting      |
+=====+
| 1. Insertion Sort |
| 2. Merge Sort    |
| 3. Shell Sort     |
| 4. Bubble Sort    |
| 5. Quick Sort     |
| 6. Selection Sort |
| 7. Exit           |
+=====+
Masukkan Pilihan: 4
```

Gambar 12 Pilihan 4 Pada Menu Sorting



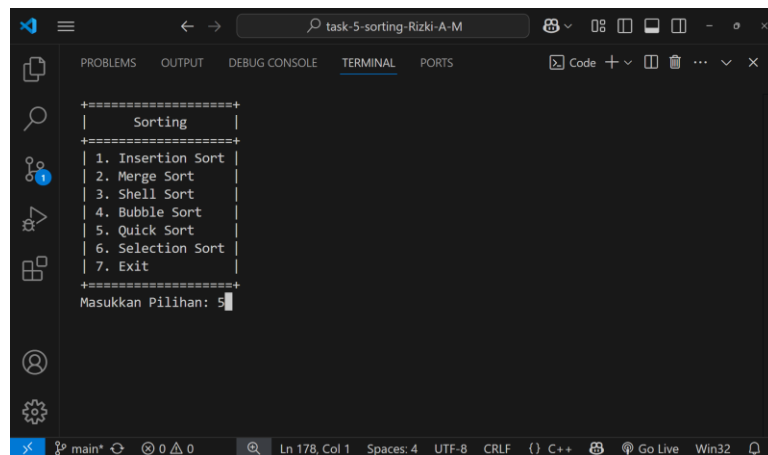
```
task-5-sorting-Rizki-A-M
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
Masukkan ID: 2410817110014
```

Gambar 13 Masukkan ID Pada Pilihan 4



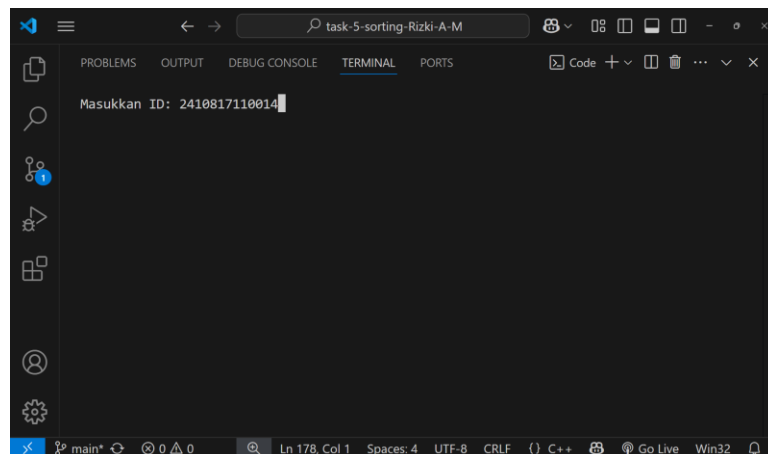
```
task-5-sorting-Rizki-A-M
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
Data Sebelum Diurutkan: 2410817110014
Bubble Sort took 0.0000016000 seconds
Data Setelah Diurutkan: 0001111124478
Press any key to continue...
```

Gambar 14 Tampilan Hasil Dari Bubble Sort



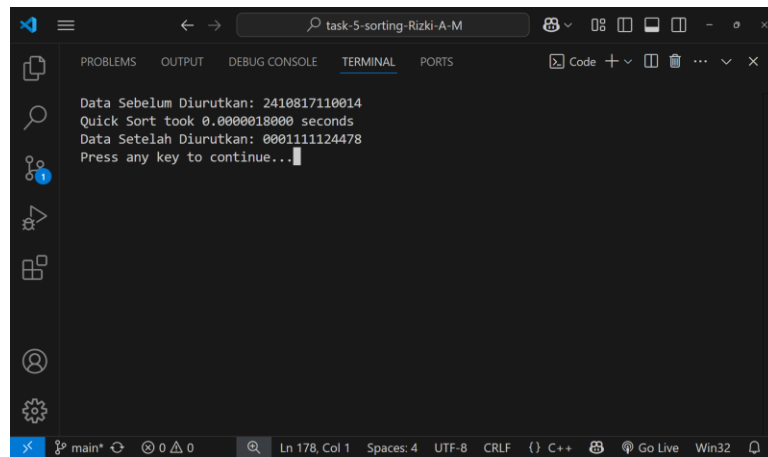
```
task-5-sorting-Rizki-A-M
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
+=====+
|      Sorting      |
+=====+
| 1. Insertion Sort |
| 2. Merge Sort    |
| 3. Shell Sort    |
| 4. Bubble Sort   |
| 5. Quick Sort    |
| 6. Selection Sort|
| 7. Exit          |
+=====+
Masukkan Pilihan: 5
```

Gambar 15 Pilihan 5 Pada Menu Sorting



```
task-5-sorting-Rizki-A-M
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
Masukkan ID: 2410817110014
```

Gambar 16 Masukkan ID Pada Pilihan 5

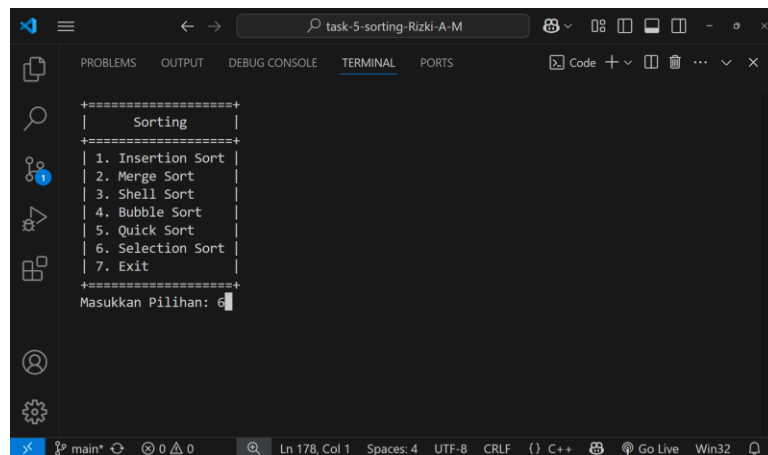


```

task-5-sorting-Rizki-A-M
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
Data Sebelum Diurutkan: 2410817110014
Quick Sort took 0.0000018000 seconds
Data Setelah Diurutkan: 0001111124478
Press any key to continue...
main* 0 0 Ln 178, Col 1 Spaces: 4 UTF-8 CRLF {} C++ Go Live Win32

```

Gambar 17 Tampilan Hasil Dari Quick Sort

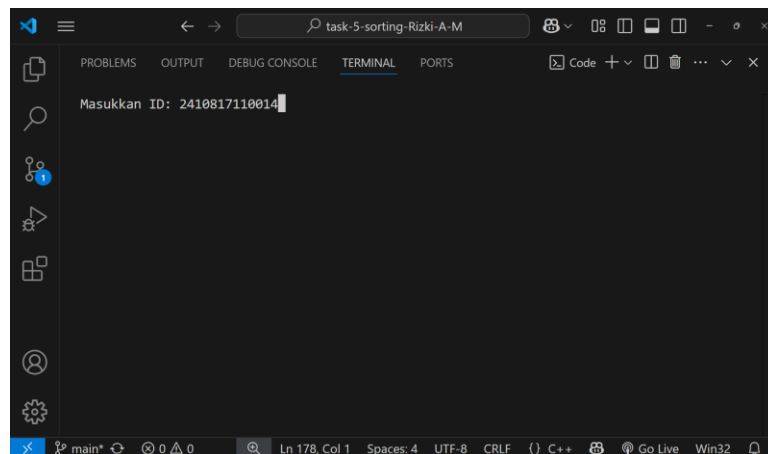


```

task-5-sorting-Rizki-A-M
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
+=====+
|      Sorting      |
+=====+
| 1. Insertion Sort |
| 2. Merge Sort    |
| 3. Shell Sort    |
| 4. Bubble Sort   |
| 5. Quick Sort    |
| 6. Selection Sort |
| 7. Exit          |
+=====+
Masukkan Pilihan: 6
main* 0 0 Ln 178, Col 1 Spaces: 4 UTF-8 CRLF {} C++ Go Live Win32

```

Gambar 18 Pilihan 6 Pada Menu Sorting

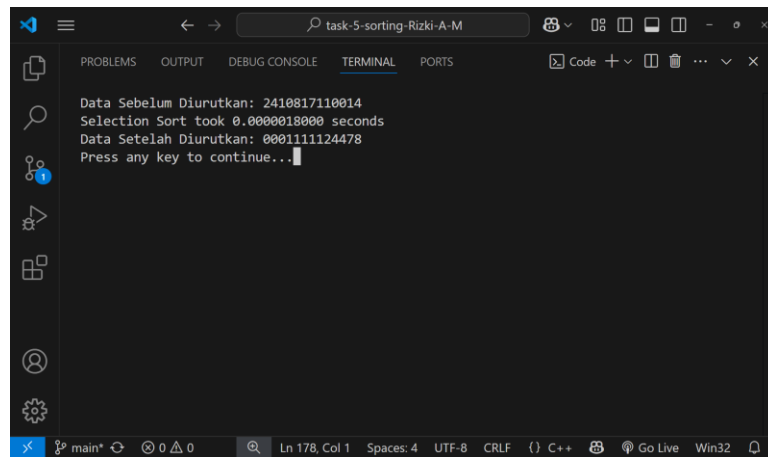


```

task-5-sorting-Rizki-A-M
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
Masukkan ID: 2410817110014
main* 0 0 Ln 178, Col 1 Spaces: 4 UTF-8 CRLF {} C++ Go Live Win32

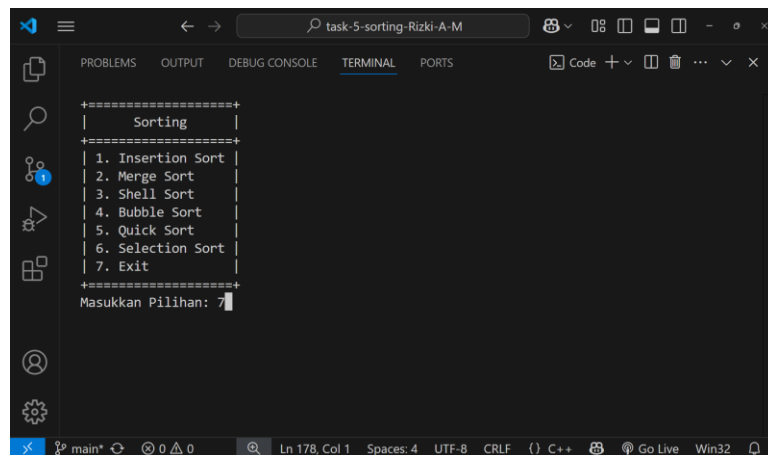
```

Gambar 19 Masukkan ID Pada Pilihan 6



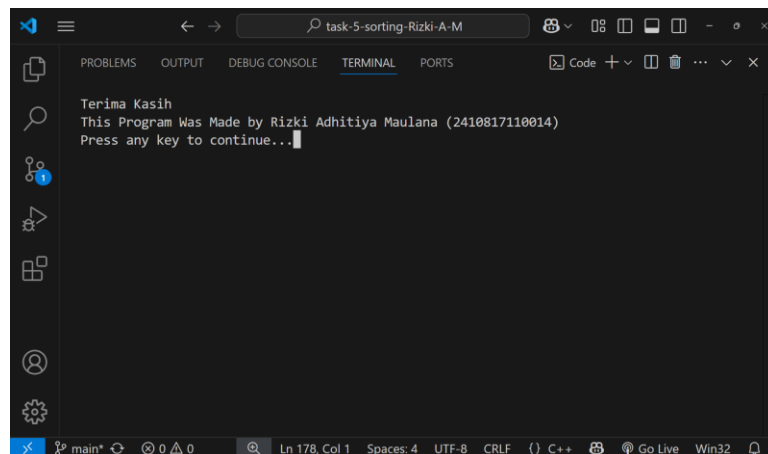
```
task-5-sorting-Rizki-A-M
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
Data Sebelum Diurutkan: 2410817110014
Selection Sort took 0.0000018000 seconds
Data Setelah Diurutkan: 0001111124478
Press any key to continue...
```

Gambar 20 Tampilan Hasil Dari Selection Sort



```
task-5-sorting-Rizki-A-M
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
+=====+
|      Sorting      |
+=====+
| 1. Insertion Sort |
| 2. Merge Sort    |
| 3. Shell Sort     |
| 4. Bubble Sort   |
| 5. Quick Sort    |
| 6. Selection Sort |
| 7. Exit          |
+=====+
Masukkan Pilihan: 7
```

Gambar 21 Pilihan 7 Pada Menu Sorting



```
task-5-sorting-Rizki-A-M
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
Terima Kasih
This Program Was Made by Rizki Adhitiya Maulana (2410817110014)
Press any key to continue...
```

Gambar 22 Tampilan Program Selesai

C Pembahasan

• Alur Program

Ketika program dimulai, akan muncul tampilan menu dengan nama “*Sorting*” yang di dalamnya terdapat tujuh opsi, diantaranya ada opsi pertama untuk “*Insetion Sort*”, opsi kedua “*Merge Sort*”, opsi ketiga “*Shell Sort*”, opsi keempat “*Bubble Sort*”, opsi kelima “*Quick Sort*”, opsi keenam “*Selection Sort*” dan opsi ketujuh “*Exit*”. Program yang ada akan selalu berjalan atau melakukan perulangan karena adanya struktur “*do-while*”, yang akan terus menampilkan menu sorting sampai pengguna memilih opsi “*Exit*”. Kemudian, setiap selesai menentukan pilihan atau melakukan input untuk menu apa yang akan diakses, fungsi atau perintah “*CLS*” akan selalu dijalankan untuk membuat tampilan dari program terlihat tidak menumpuk dan rapi.

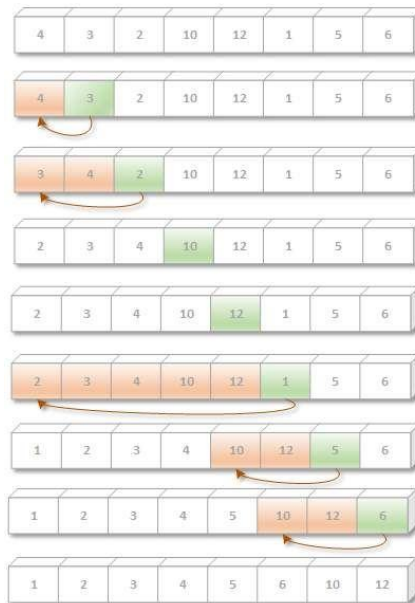
Struktur “*Switch-case*” digunakan untuk semua pilihan yang ada, mulai dari **case 1 sampai case 3**, pengguna akan diminta untuk memasukkan atau menginput “*nama*”. Sedangkan, **case 4 sampai case 6** pengguna akan diminta untuk memasukkan atau menginput “*id*”. Kecuali **case 7** yang akan menampilkan ucapan terima kasih dan nama serta nim dari pembuat program, dan **case default** yang akan memberitahukan bahwa opsi tidak valid karena pengguna memasukan pilihan di luar dari tujuh opsi yang ada. Setelah selesai melakukan input baik itu berupa “*nama*” atau “*nim*”, program akan menampilkan penampakan string sebelum dan sesudah disortir. Pada **case 1**, program akan menjalankan fungsi “*Insertion Sort*” dan menampilkan durasi eksekusinya, **case 2** menjalankan fungsi “*Merge Sort*” beserta durasinya, **case 3** menjalankan fungsi “*Shell Sort*” dengan durasinya, **case 4** menjalankan “*Bubble Sort*” disertai durasi, **case 5** menjalankan “*Quick Sort*” dengan waktu eksekusinya, dan **case 6** menjalankan “*Selection Sort*” serta menampilkan durasinya.

Setelah fungsi dari setiap pilihan dari program menyelesaikan semua tugas dan memprintkan semua hasil prosesnya, program akan menunggu input

tombol apa saja untuk melanjutkan, lalu membersihkan layar dan menampilkan tampilan menu “*Sorting*” kembali.

- **Insertion Sort**

Insertion Sort digunakan untuk melakukan sebuah pengurutan data, yang mana dengan cara membandingkan elemen yang ada pada array. Setiap elemen dalam array akan dibandingkan dengan elemen sebelumnya secara berurutan hingga semua elemen menemukan posisinya masing-masing. Proses pengurutannya di mulai dari elemen kedua, kemudian elemen kedua akan dibandingkan dengan elemen sebelumnya. Apabila elemen kedua lebih kecil dari elemen sebelumnya, maka elemen kedua akan digeser ke kiri atau disisipkan sebelum elemen yang lebih besar dari elemen kedua. Namun, apabila elemen kedua lebih besar dari elemen sebelumnya, maka tidak akan ada pergeseran posisi. Proses ini akan terus diulangi hingga elemen ketiga, keempat dan yang terakhir terurut.



Gambar 23 Ilustrasi Insertion Sort

Dalam pembahasan fungsinya, `void insertionSort(string &str)` merupakan implementasi Insertion Sort yang mengurutkan karakter-karakter dalam sebuah string secara langsung karena menggunakan pass by reference. Proses pengurutan dimulai dengan loop for dari indeks $i = 1$

(karakter kedua), di mana setiap karakter pada $\text{str}[i]$ disimpan sebagai key yang akan disisipkan ke bagian string di sebelah kirinya yang sudah terurut. Variabel j diinisialisasi dengan $i - 1$ untuk melacak posisi di bagian yang sudah terurut. Selanjutnya, loop while akan menggeser karakter $\text{str}[j]$ satu posisi ke kanan ($\text{str}[j + 1] = \text{str}[j]$) selama j masih valid ($j \geq 0$) dan $\text{str}[j]$ lebih besar dari key, sambil terus mengurangi j ($j--$) untuk mencari posisi yang tepat. Setelah loop while selesai, key ditempatkan pada posisi $\text{str}[j + 1]$, menyelesaikan proses penyisipan satu karakter dan secara bertahap mengurutkan seluruh string.

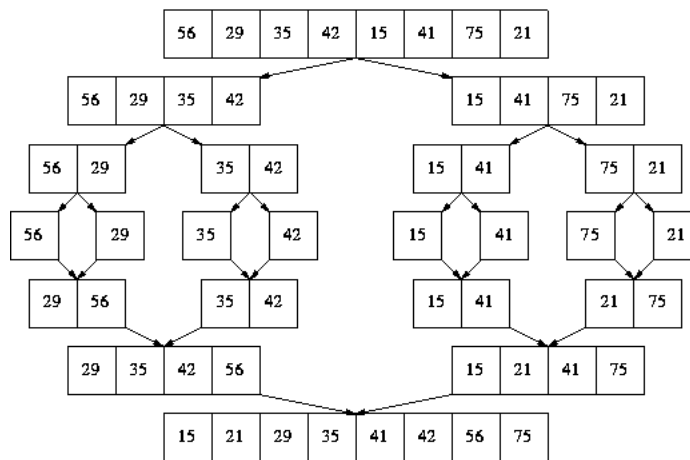
Kompleksitas waktu dari Insertion Sort tergantung pada kondisi awal data yang akan diurutkan. Pada kasus terbaik (best case) kompleksitas waktu Insertion Sort adalah $O(n)$, karena data yang ada sudah terurut, algoritma yang ada hanya akan melakukan perbandingan antar elemen atau pengecekan. Pada kasus rata-rata (average case) dan kasus terburuk (worst case) kompleksitas waktunya adalah $O(n^2)$, karena saat data terurut acak atau terbalik, algoritma yang ada harus membandingkan dan menggeser hampir semua elemen. Sementara itu, kompleksitas ruang dari Insertion Sort adalah $O(1)$ karena algoritma Insertion Sort tidak memerlukan struktur data tambahan.

Jadi kesimpulannya, Insertion Sort ini sangat cocok digunakan untuk mengurutkan data jumlah yang kecil atau hampir terurut. Kemudian juga hemat ruang penyimpanan dalam penggunaannya karena tidak perlu tambahan struktur data dalam prosesnya, hanya beberapa variabel bantu sebagai penyimpanan sementara.

- **Merge Sort**

Merge Sort digunakan untuk melakukan sebuah pengurutan data, yang mana menggunakan pendekatan divide and conquer dalam prosesnya. Cara kerja pengurutannya di mulai dengan membagi kumpulan data dalam satu array menjadi dua bagian secara terus menerus hingga menjadi bagian paling kecil yang berisi satu elemen. Setelah proses pembagian kumpulan

data telah selesai, bagian paling kecil yang ada akan digabung kembali. Dalam proses penggabungannya akan selalu melibatkan dua bagian yang telah dibagi sebelumnya, karena ketika digabungkan elemen yang ada di dua bagian tersebut akan dibandingkan dan diurutkan dari yang terkecil disebelah kiri hingga yang terbesar di sebelah kanan. Proses penggabungan akan terus dilakukan secara bertahap dengan memperhatikan urutan elemen agar tetap terurut hingga seluruh elemen yang dibagi sebelumnya kembali menjadi satu kesatuan.



Gambar 24 Ilustrasi Merge Sort

Dalam pembahasan fungsinya, `void mergeSort(string &str, int left, int right)` adalah bagian yang bekerja secara rekursif untuk mengurutkan string. Prosesnya diawali dengan kondisi dasar `if (left < right)`, yang akan terus membagi string menjadi bagian-bagian yang lebih kecil hingga hanya tersisa satu karakter (yang otomatis terurut), menghentikan rekursi. Kemudian, dua panggilan rekursif dilakukan: satu untuk mengurutkan paruh kiri (`mergeSort(str, left, mid)`) dan satu untuk mengurutkan paruh kanan (`mergeSort(str, mid + 1, right)`). Selanjutnya dilanjutkan dengan fungsi `void merge(string &str, int left, int mid, int right)` untuk menggabungkan dua sub array yang sudah terurut (yang direpresentasikan oleh bagian-bagian dari `str` dari `left` hingga `mid`, dan dari `mid + 1` hingga `right`) menjadi satu bagian yang terurut dalam string asli. Fungsi ini

pertama menghitung ukuran kedua sub-array (n_1 dan n_2), lalu mengalokasikan dua array sementara (tempL dan tempR) di heap untuk menyalin elemen-elemen dari sub-array yang bersangkutan. Proses penggabungan dilakukan dengan loop while utama yang membandingkan elemen-elemen dari tempL[i] dan tempR[j], menyalin karakter yang lebih kecil ke str[k] dan menginkrementasi indeks yang relevan (i, j, dan k). Setelah salah satu array sementara habis, dua loop while terpisah akan menyalin sisa elemen dari array sementara yang belum habis ke str. Terakhir, memori yang dialokasikan secara dinamis dibebaskan menggunakan delete[] untuk mencegah memory leak, memastikan operasi penggabungan berjalan efisien dan bersih.

Kompleksitas waktu dari Merge Sort tidak tergantung pada kondisi awal data yang akan diurutkan. Artinya baik dalam kasus baik, kasus rata-rata, ataupun kasus yang buruk kompleksitas waktu Merge Sort adalah $O(n \log n)$, karena ada setiap tahap penggabungan, seluruh elemen (n elemen) harus diproses. Sementara itu, kompleksitas ruang dari Merge Sort adalah $O(n)$ karena algoritma Merge Sort memerlukan ruang tambahan sebagai tempat penyimpanan sementara selama proses penggabungan.

Jadi kesimpulannya, Merge Sort ini cocok digunakan untuk mengurutkan data dalam jumlah yang besar karena hasil dari pengurutan datanya yang akurat dan waktu eksekusi data yang konsisten di semua kasus, namun membutuhkan ruang tambahan untuk proses di dalamnya untuk menyimpan array sementara saat prosesnya.

- **Shell Sort**

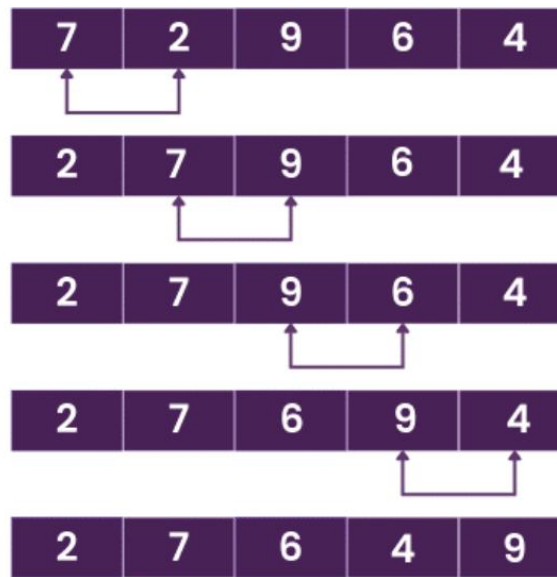
Shell Sort merupakan pengembangan dari Insertion Sort, yang mana Shell Sort digunakan untuk melakukan sebuah pengurutan data dengan cara membandingkan dan menukar elemen-elemen yang memiliki jarak terlebih dahulu. Jarak antar elemen ini disebut dengan “gap”. Cara kerja pengurutannya di mulai dengan membandingkan data yang letaknya jauh terlebih dahulu, lalu jarak yang ada tersebut dikurangi sedikit demi sedikit

sudah terurut, algoritma yang ada hanya akan melakukan perbandingan antar elemen atau pengecekan. Pada kasus rata-rata (average case) kompleksitas waktunya adalah $O(n \log n) \sim O(n^{1.25})$, karena data yang ada sudah hampir terurut hanya perlu melakukan sedikit pergeseran diakhir dengan Insertion Sort. Kasus terburuk (worst case) kompleksitas waktunya adalah $O(n^2)$, karena saat data terurut acak atau terbalik, algoritma yang ada harus membandingkan dan memindahkan dan menggeser hampir semua elemen. Sementara itu, kompleksitas ruang dari Shell Sort adalah $O(1)$ karena algoritma Insertion Sort tidak memerlukan struktur data tambahan.

Jadi kesimpulan, Shell Sort ini cocok digunakan untuk mengurutkan data dalam jumlah yang sedang atau hampir terurut. Kemudian juga hemat ruang penyimpanan dalam penggunaannya karena tidak perlu tambahan struktur data dalam prosesnya, hanya beberapa variabel bantu sebagai penyimpanan sementara.

- **Bubble Sort**

Bubble Sort digunakan untuk melakukan sebuah pengurutan data, yang mana dengan mendorong elemen terbesar yang ada di dalam kumpulan data ke kanan secara bertahap. Cara kerja pengurutannya dimulai dengan membandingkan elemen pertama dan elemen kedua. Apabila elemen pertama lebih besar dari elemen kedua maka posisi dari keduanya akan di tukar, elemen yang lebih besar akan berpindah ke kanan. Kemudian, perbandingan berlanjut antara elemen kedua dan ketiga, dan begitu seterusnya hingga elemen terakhir. Setelah satu lintasan selesai, elemen terbesar akan berada di posisi paling kanan. Proses ini diulang kembali untuk elemen-elemen yang tersisa, dengan setiap lintasan mengurangi jumlah elemen yang perlu diperiksa karena elemen-elemen terbesar sudah berada di posisi yang benar. Pengurutan akan berhenti ketika dalam satu lintasan tidak ada lagi pertukaran, yang menandakan bahwa seluruh data sudah dalam keadaan terurut.



Gambar 26 Ilustrasi Bubblu Sort

Dalam pembahasan fungsinya, void `bubbleSort(string &str)` bekerja melalui dua loop for bersarang, pada loop terluar (i) menentukan berapa banyak elemen terbesar yang sudah dipindahkan ke posisi akhirnya di bagian kanan string, sementara pada loop dalam (j) melakukan perbandingan berpasangan antara elemen yang berdekatan (`str[j]` dan `str[j+1]`). Jika `str[j]` lebih besar dari `str[j+1]`, kedua karakter tersebut akan ditukar posisinya menggunakan `swap()`, dan sebuah flag `swapped` akan diatur ke `true`. Apabila setelah satu iterasi penuh dari loop dalam tidak ada pertukaran yang terjadi (artinya `swapped` tetap `false`), maka string dianggap sudah terurut sepenuhnya, dan loop terluar akan dihentikan lebih awal untuk meningkatkan efisiensi.

Kompleksitas waktu dari Bubble Sort tergantung pada kondisi awal data yang akan diurutkan. Pada kasus terbaik (best case) kompleksitas waktu Bubble Sort adalah $O(n)$, karena data yang ada sudah terurut, algoritma yang ada hanya akan melakukan perbandingan antar elemen atau pengecekan. Pada kasus rata-rata (average case) dan kasus terburuk (worst case) kompleksitas waktunya adalah $O(n^2)$, karena saat data terurut acak atau terbalik, algoritma yang ada harus membandingkan dan menggeser

hampir semua elemen. Sementara itu, kompleksitas ruang dari Insertion Sort adalah $O(1)$ karena algoritma Insertion Sort tidak memerlukan struktur data tambahan.

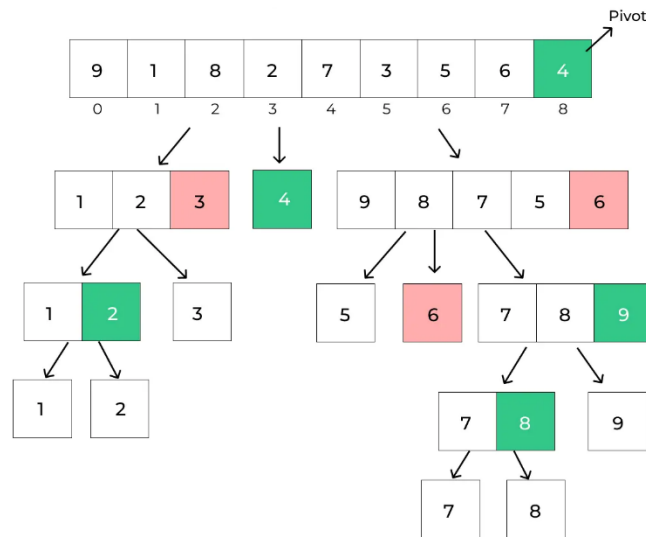
Jadi kesimpulannya, Insertion Sort ini sangat cocok digunakan untuk mengurutkan data jumlah yang kecil atau hampir terurut. Kemudian juga hemat ruang penyimpanan dalam penggunaannya karena tidak perlu tambahan struktur data dalam prosesnya, hanya beberapa variabel bantu sebagai penyimpanan sementara.

- **Quick Sort**

Quick Sort digunakan untuk melakukan sebuah pengurutan data, yang mana menggunakan pendekatan divide and conquer dalam proses. Cara kerja pengurutannya dimulai dengan memilih satu elemen sebagai pivot, biasanya elemen terakhir dalam kumpulan data. Selanjutnya, data dibagi menjadi dua bagian dimana elemen yang lebih kecil atau sama dengan pivot ditempatkan di sebelah kiri, sedangkan elemen yang lebih besar ditempatkan di sebelah kanan. Pivot kemudian diposisikan di tempat yang tepat dalam urutan akhir.

Proses pemisahan ini dilakukan oleh fungsi `partition()`, yang membandingkan setiap elemen dengan pivot. Bila ditemukan elemen yang lebih kecil atau sama dengan pivot, maka elemen itu akan ditukar ke posisi yang lebih kiri. Setelah semua elemen diperiksa, pivot akan ditukar ke posisi tengah yang sesuai, dan posisi ini akan menjadi batas pembagian data selanjutnya.

Setelah itu, fungsi `quickSort()` akan memanggil dirinya sendiri secara rekursif untuk mengurutkan bagian kiri dan kanan dari pivot. Jika masih ada bagian yang belum terurut, maka proses ini akan diulangi dengan memilih pivot baru dan membagi ulang datanya. Proses ini terus berlangsung hingga seluruh elemen berada di posisi yang tepat dan data menjadi terurut sempurna.



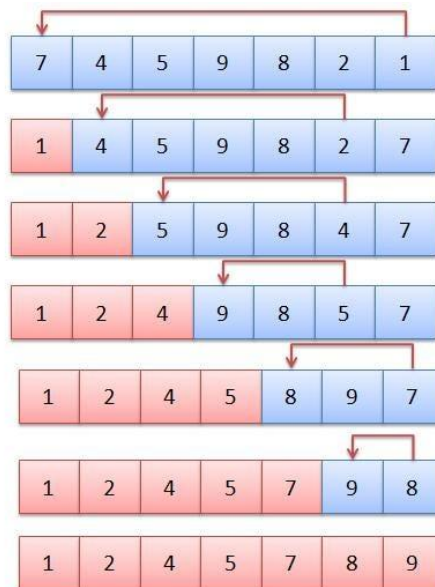
Gambar 27 Ilustrasi Quick Sort

Kompleksitas waktu dari Quick Sort tergantung pada pemilihan pivot dan kondisi awal data yang akan diurutkan. Pada kasus terbaik (best case) kompleksitas waktunya adalah $\Omega(n \log n)$, karena pembagian data menjadi dua bagian yang relatif seimbang. Pada kasus rata-rata (average case) kompleksitas waktu adalah $\theta(n \log n)$, karena pembagian data biasanya cukup merata dalam banyak kasus. Pada kasus terburuk (worst case) kompleksitas waktunya menjadi $O(n^2)$, yaitu ketika pivot yang dipilih selalu menjadi elemen terkecil atau terbesar (misalnya saat data sudah terurut atau terbalik). Sementara itu, kompleksitas ruang dari Quick Sort pada kasus terbaik dan rata-rata adalah $O(\log n)$ karena membagi dua bagian dengan seimbang atau hampir seimbang. Sedangkan, untuk kompleksitas ruang dari Quick Sort pada kasus terburuk adalah $O(n)$ karena pembagiannya yang tidak seimbang.

Jadi kesimpulannya, Quick Sort ini sangat cocok digunakan untuk mengurutkan data jumlah yang besar karena hasil dari pengurutan datanya yang akurat dan waktu eksekusi rata-ratanya yang efisien. Kemudian, penggunaan ruang penyimpanannya tergolong kecil karena tidak memerlukan struktur data tambahan.

- **Selection Sort**

Selection Sort digunakan untuk melakukan sebuah pengurutan data, yang mana akan memilih elemen terkecil dari bagian data yang belum terurut, lalu menukarnya dengan elemen di posisi paling awal dari bagian tersebut. Proses ini dilakukan berulang-ulang hingga semua elemen yang ada berada di posisi yang sudah tepat atau terurut. Cara kerja pengurutannya dimulai dengan mencari elemen terkecil dari seluruh data, lalu menempatkannya di posisi pertama. Selanjutnya, algoritma melanjutkan pencarian elemen terkecil berikutnya dari data yang tersisa dan menukarnya ke posisi kedua, dan seterusnya. Dengan pendekatan ini, data secara bertahap akan terurut dari posisi awal hingga akhir melalui proses seleksi dan pertukaran.



Gambar 28 Ilustrasi Selection Sort

Dalam pembahasan fungsinya, `void selectionSort(string &str)` mengimplementasikan algoritma Selection Sort, yang bekerja dengan cara berulang kali mencari elemen terkecil dari bagian string yang belum terurut dan menempatkannya pada posisi yang benar. Ini dilakukan melalui loop for terluar yang mengiterasi dari awal string hingga satu posisi sebelum

akhir. Dalam setiap iterasi, sebuah variabel `minIndex` diinisialisasi dengan indeks `i` saat ini, lalu loop for dalam akan mencari di sisa bagian string (dari `j = i + 1` hingga akhir) untuk menemukan indeks (`j`) dari karakter yang paling kecil. Setelah loop dalam selesai, elemen pada `str[i]` dan elemen pada `str[minIndex]` (yang merupakan elemen terkecil yang ditemukan) akan ditukar posisinya menggunakan `swap()`, sehingga elemen terkecil tersebut "dipilih" dan ditempatkan pada posisi yang benar di awal bagian yang belum terurut. Proses ini terus berlanjut hingga seluruh string terurut.

Kompleksitas waktu dari Selection Sort tidak bergantung pada kondisi awal data, baik data tersebut sudah terurut, hampir terurut, maupun acak sepenuhnya. Pada kasus terbaik (best case), kasus rata-rata (average case), dan kasus terburuk (worst case), kompleksitas waktunya tetap $O(n^2)$, karena algoritma harus mencari elemen terkecil dalam bagian yang belum terurut di setiap langkahnya, tanpa memperhatikan apakah data sudah dalam urutan tertentu. Sementara itu, kompleksitas ruang dari Selection Sort adalah $O(1)$, karena tidak perlu tambahan struktur data dalam prosesnya

TAUTAN GITHUB

[https://github.com/Rizki-A-M/Rizki-A-M-
PRAKTIKUM_ALGORITMA_DAN_STRUKTUR_DATA.git](https://github.com/Rizki-A-M/Rizki-A-M-PRAKTIKUM_ALGORITMA_DAN_STRUKTUR_DATA.git)