LAPORAN PRAKTIKUM
ALGORITMA & STRUKTUR DATA
MODUL 5

SORTING

Oleh:

Rika Fauliana Rahmi NIM. 2410817120017

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 : Rika Fauliana Rahmi
NIM : 2410817120017

Menyetujui, Asisten Praktikum	Mengetahui, Dosen Penanggung Jawab Praktikum
Muhammad Fauzan Ahsani NIM. 2310817310009	Muti'a Maulida, S.Kom., M.TI. NIP. 198810272019032013

DAFTAR ISI

DAFTAR GAMBAR

DAFTAR TABEL

SOAL PRAKTIKUM

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)

PEMBAHASAN

A. Source Code

Tabel 1 Source Code Jawaban Soal Praktikum

```
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 = "RikaFaulianaRahmi";
11  string id = "2410817120017";
12
13  void timeSort(const function<void()>& sortFunc,
    const string& sortName) {
14      auto start =
    chrono::high_resolution_clock::now();
15      sortFunc();
16      auto end =
    chrono::high_resolution_clock::now();
17      chrono::duration<double> duration = end - start;
18
19      cout << fixed << setprecision(10);
20      cout << sortName << " took " << duration.count()
    << " seconds\n";
21  }
22
```

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

```
50          if (tempL[i] <= tempR[j]) {
51              str[k] = tempL[i];
52              i++;
53          } else {
54              str[k] = tempR[j];
55              j++;
56          }
57          k++;
58      }
59
60      while (i < n1) {
61          str[k] = tempL[i];
62          i++;
63          k++;
64      }
65
66      while (j < n2) {
67          str[k] = tempR[j];
68          j++;
69          k++;
70      }
71
72      delete[] tempL;
73      delete[] tempR;
74  }
75
76  void mergeSort(string &str, int left, int right) {
77      if (left < right) {
78          int mid = left + (right - left) / 2;
79          mergeSort(str, left, mid);
```

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

```
108
109         if (!swapped) break;
110     }
111 }
112
113 int partition(string &str, int low, int high){
114     int pivot = str[high];
115     int i = (low - 1);
116
117     for (int j = low; j <= high - 1; j++) {
118         if (str[j] <= pivot) {
119             i++;
120             swap(str[i], str[j]);
121         }
122     }
123
124     swap(str[i + 1], str[high]);
125
126     return (i + 1);
127 }
128
129 void quickSort(string &str, int low, int high){
130     if (low < high) {
131         int p_idx = partition(str, low, high);
132         quickSort(str, low, p_idx - 1);
133         quickSort(str, p_idx + 1, high);
134     }
135 }
136
137 void selectionSort(string &str){
```

```
138     for (int i = 0; i < str.size() - 1; i++) {
139         int minIndex = i;
140 
141         for (int j = i + 1; j < str.size(); j++) {
142             if (str[j] < str[minIndex]) {
143                 minIndex = j;
144             }
145         }
146 
147         swap(str[i], str[minIndex]);
148     }
149 }
150 
151 
152 int main(){
153     int ch;
154     string temp;
155 
156     do {
157         cout << "+====================+" << endl;
158         cout << "| Sorting Algorithm |" << endl;
159         cout << "+====================+" << endl;
160         cout << "| 1. Insertion Sort |" << endl;
161         cout << "| 2. Merge Sort     |" << endl;
162         cout << "| 3. Shell Sort     |" << endl;
163         cout << "| 4. Bubble Sort    |" << endl;
164         cout << "| 5. Quick Sort     |" << endl;
165         cout << "| 6. Selection Sort |" << endl;
166         cout << "| 7. Exit           |" << endl;
167         cout << "+====================+" << endl;
```

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

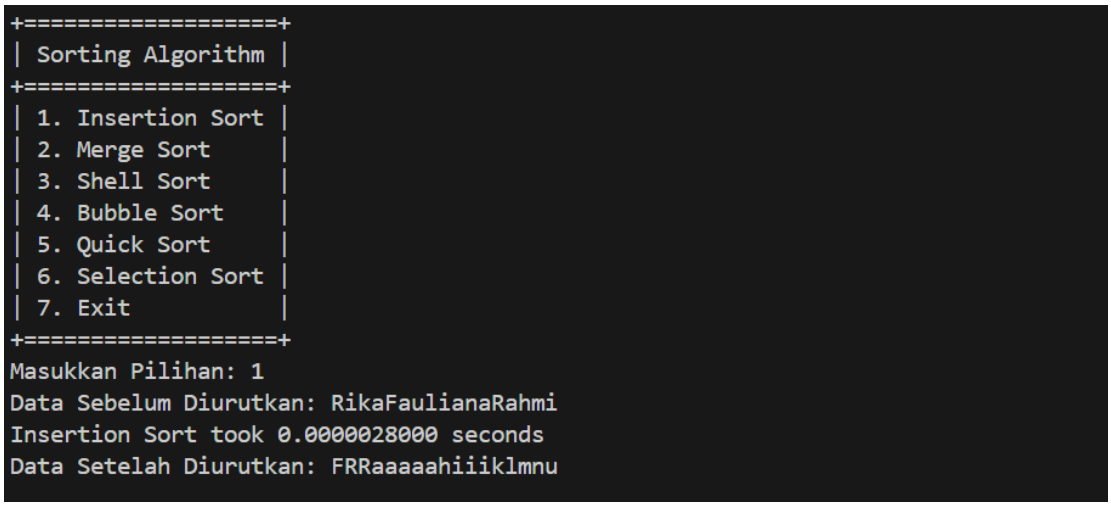
```
189             case 4:
190                 temp = id;
191                 cout << "Data Sebelum Diurutkan: "
    << temp << endl;
192                 timeSort([&]()
    {bubbleSort(temp); }, "Bubble Sort");
193                 cout << "Data Setelah Diurutkan: "
    << temp << endl;
194                 break;
195             case 5:
196                 temp = id;
197                 cout << "Data Sebelum Diurutkan: "
    << temp << endl;
198                 timeSort([&]() {quickSort(temp, 0,
    temp.size() - 1); }, "Quick Sort");
199                 cout << "Data Setelah Diurutkan: "
    << temp << endl;
200                 break;
201             case 6:
202                 temp = id;
203                 cout << "Data Sebelum Diurutkan: "
    << temp << endl;
204                 timeSort([&]()
    {selectionSort(temp); }, "Selection Sort");
205                 cout << "Data Setelah Diurutkan: "
    << temp << endl;
206                 break;
207             case 7:
208                 cout << "Terima Kasih" << endl;
```

```
209                 cout << "This Program Was Made by
    Rika Fauliana Rahmi (2410817120017)" << endl;
210                 break;
211             default:
212                 cout << "Opsi Tidak Valid. Silahkan
    Coba Lagi." << endl;
213         }
214         cout << "\nPress any key to continue..." <<
    endl;
215         getch();
216         system("cls");
217     } while (ch != 7);
218     return 0;
219 }
```

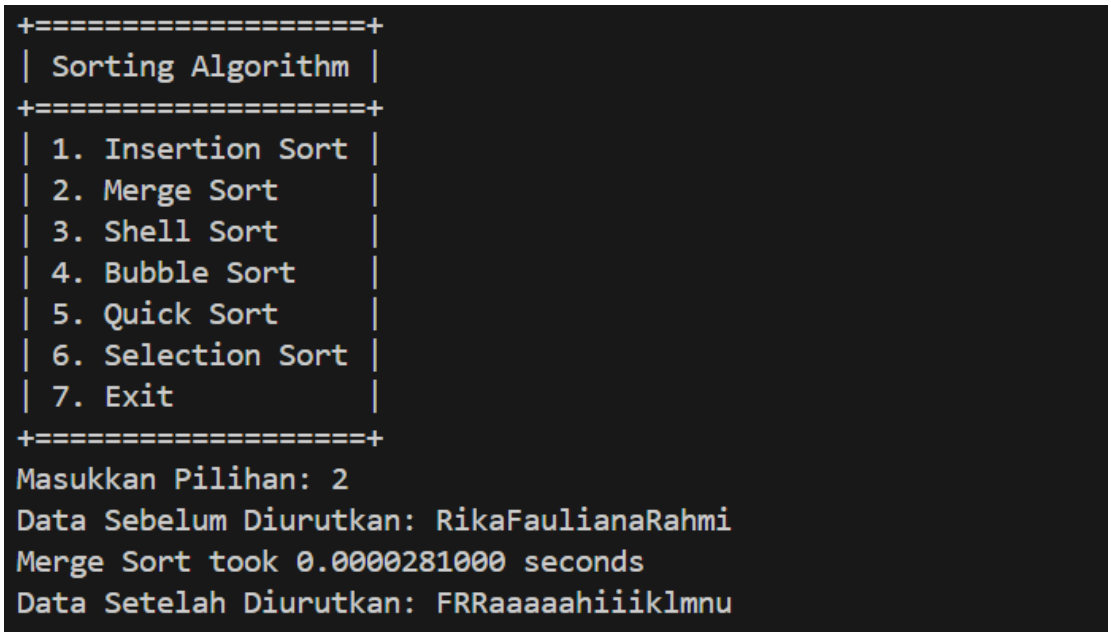
B. Output Program

```
+====================+
| Sorting Algorithm  |
+====================+
| 1. Insertion Sort  |
| 2. Merge Sort      |
| 3. Shell Sort      |
| 4. Bubble Sort     |
| 5. Quick Sort      |
| 6. Selection Sort  |
| 7. Exit            |
+====================+
Masukkan Pilihan:
```

Gambar 1 Screenshot tampilan awal program dijalankan



Gambar 2 Screenshot tampilan memilih opsi 1



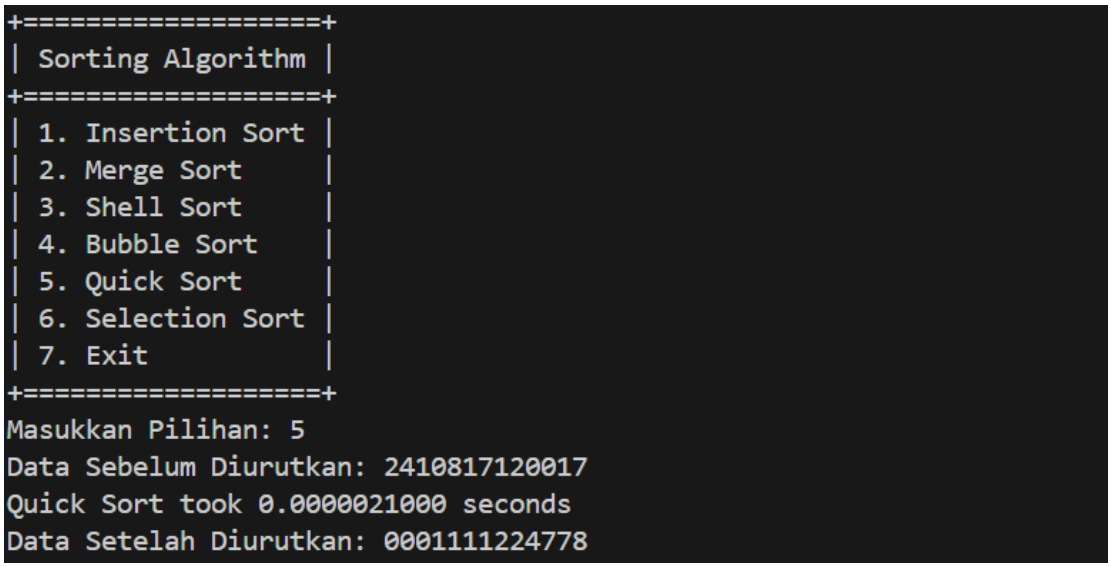
Gambar 3 Screenshot tampilan memilih opsi 2

```
 +====================+
 | 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: RikaFaulianaRahmi
 Shell Sort took 0.0000325000 seconds
 Data Setelah Diurutkan: FRRaaaaahiiiklmnu
```

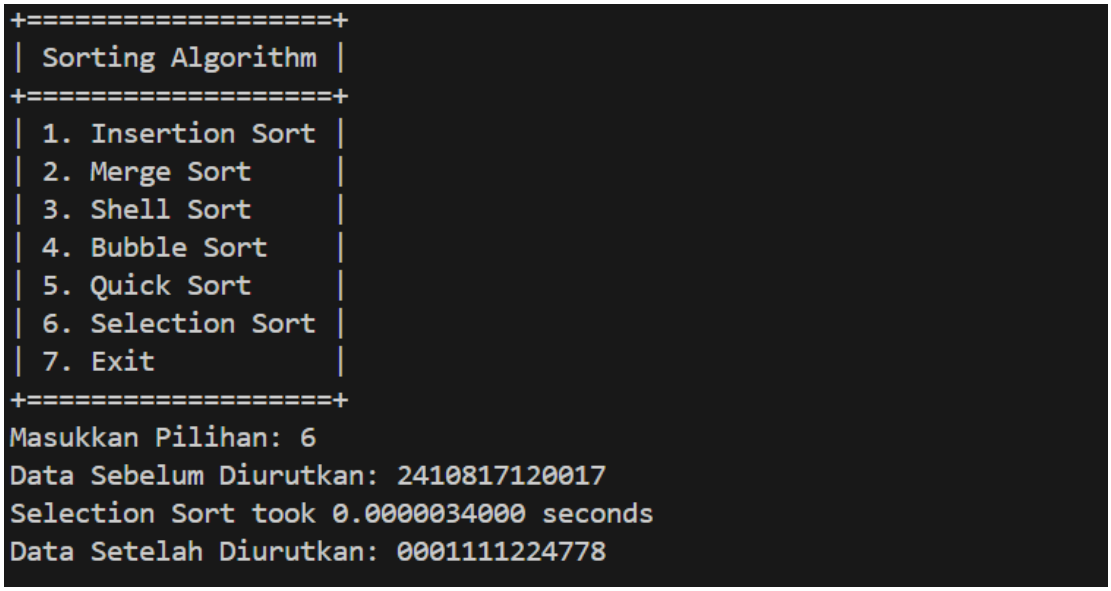
Gambar 4 Screenshot tampilan memilih opsi 3

```
+====================+
| 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: 2410817120017
Bubble Sort took 0.0000038000 seconds
Data Setelah Diurutkan: 0001111224778
```

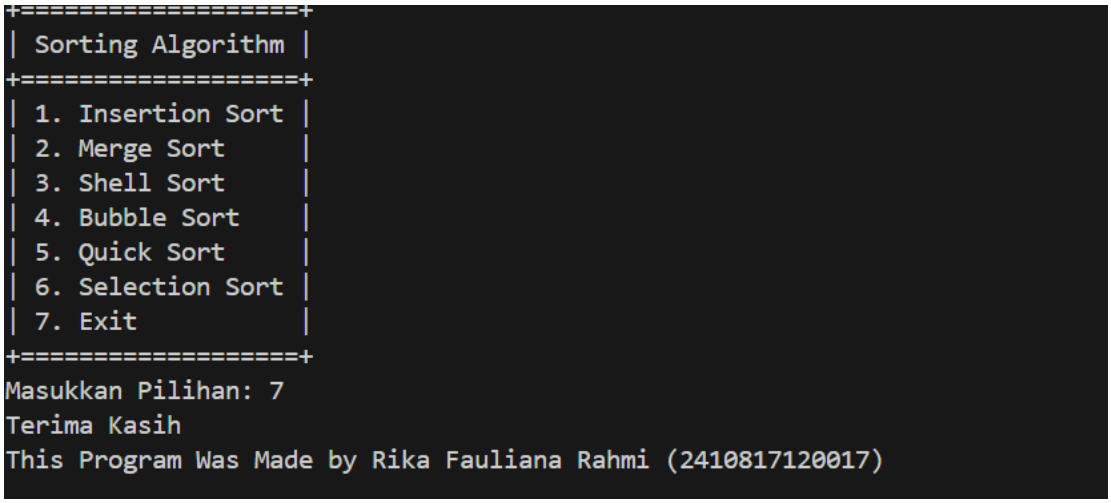
Gambar 5 Screenshot tampilan memilih opsi 4



Gambar 6 Screenshot tampilan memilih opsi 5



Gambar 7 Screenshot tampilan memilih opsi 6



Gambar 8 Screenshot tampilan memilih opsi 7

C. Pembahasan

Program ini merupakan aplikasi berbasis menu interaktif dalam bahasa C++ yang digunakan untuk mendemonstrasikan dan membandingkan performa berbagai algoritma pengurutan (sorting) terhadap data string, yaitu nama dan NIM. Program ini menampilkan hasil pengurutan serta mengukur waktu eksekusi dari setiap algoritma menggunakan fasilitas chrono.

Saat program dijalankan, pertama-tama program akan menampilkan sebuah menu utama di layar konsol yang berisi daftar enam algoritma sorting yang dapat dipilih oleh pengguna, yaitu Insertion Sort, Merge Sort, Shell Sort, Bubble Sort, Quick Sort, dan Selection Sort. Selain itu, terdapat juga opsi ke-7 untuk keluar dari program.

Setelah menu ditampilkan, pengguna diminta untuk memasukkan angka yang sesuai dengan algoritma sorting yang ingin dijalankan. Input ini dibaca menggunakan `cin` dan disimpan dalam variabel `ch`. Berdasarkan nilai `ch`, program akan masuk ke blok `switch-case` untuk mengeksekusi fungsi sorting yang sesuai.

Jika pengguna memilih opsi 1, 2, atau 3 (Insertion, Merge, atau Shell Sort), maka data yang akan diurutkan adalah string `name`, yaitu "RikaFaulianaRahmi". Namun,

jika pengguna memilih opsi 4, 5, atau 6 (Bubble, Quick, atau Selection Sort), maka data yang diurutkan adalah string `id`, yaitu "2410817120017". Data asli tidak diubah, melainkan disalin ke dalam variabel sementara bernama `temp` agar proses pengurutan tidak mengganggu nilai awal.

Sebelum pengurutan dilakukan, program mencetak isi data `temp` sebagai "Data Sebelum Diurutkan". Kemudian, fungsi `timeSort` dipanggil untuk mengukur waktu eksekusi dari algoritma yang dipilih. Fungsi `timeSort` menerima parameter berupa lambda function yang berisi pemanggilan fungsi sorting, serta nama algoritma sorting yang digunakan. Di dalam `timeSort`, waktu mulai dan waktu selesai dieksekusi menggunakan `chrono`, lalu durasi waktunya dihitung dan ditampilkan dalam detik dengan 10 angka di belakang koma.

Setelah proses pengurutan selesai, hasil dari string yang telah diurutkan dicetak sebagai "Data Setelah Diurutkan". Lalu, program menampilkan pesan agar pengguna menekan sembarang tombol untuk melanjutkan, menggunakan fungsi `getch()` dari pustaka `<conio.h>`. Setelah itu, layar dibersihkan dengan `system("cls")`, dan program kembali ke awal dengan menampilkan menu utama lagi.

Proses ini terus berlangsung dalam sebuah perulangan `do-while` selama pengguna belum memilih opsi keluar, yaitu angka 7. Jika pengguna memilih angka 7, maka program akan mencetak ucapan terima kasih dan mencantumkan identitas pembuat program berupa nama dan NIM, kemudian keluar dari perulangan dan program pun selesai dijalankan.

1. Insertion Sort

Insertion Sort adalah algoritma pengurutan sederhana yang bekerja dengan cara membandingkan elemen satu per satu dan menyisipkannya ke posisi yang sesuai di bagian array yang sudah terurut. Algoritma ini memulai iterasi dari indeks ke-1, kemudian membandingkan elemen saat ini (disebut `key`) dengan elemen sebelumnya dalam array. Jika elemen sebelumnya lebih besar, maka elemen tersebut akan digeser

satu posisi ke kanan hingga ditemukan posisi yang tepat bagi `key`. Proses ini terus dilakukan hingga seluruh elemen berada dalam urutan yang benar.

Meskipun konsepnya mudah dipahami dan diimplementasikan, Insertion Sort memiliki performa yang kurang optimal untuk dataset berukuran besar. Waktu komputasi terburuknya adalah $O(n^2)$, terutama saat data dalam kondisi terbalik (reverse order). Namun, algoritma ini cukup efisien untuk data berukuran kecil atau data yang hampir terurut karena tidak memerlukan memori tambahan dan memiliki keunggulan dalam kesederhanaannya.

2. Merge Sort

Merge Sort adalah algoritma pengurutan berbasis paradigma *Divide and Conquer*, yang berarti membagi data menjadi dua bagian secara rekursif, mengurutkan kedua bagian tersebut, dan kemudian menggabungkannya kembali dalam urutan yang benar. Proses ini dimulai dengan membagi array menjadi dua bagian, kemudian memanggil fungsi `mergeSort` untuk mengurutkan masing-masing bagian. Setelah itu, fungsi `merge` akan menyatukan dua bagian terurut tersebut menjadi satu array yang utuh dan terurut.

Keunggulan utama Merge Sort adalah kestabilannya dan konsistensi waktu komputasi O(n log n) dalam kasus terbaik, rata-rata, maupun terburuk. Meskipun demikian, algoritma ini membutuhkan ruang tambahan untuk menyimpan array sementara saat proses penggabungan, sehingga kurang efisien dalam hal penggunaan memori dibanding algoritma in-place seperti Insertion atau Quick Sort. Merge Sort sangat cocok untuk dataset besar yang membutuhkan kestabilan pengurutan.

3. Shell Sort

Shell Sort merupakan pengembangan dari Insertion Sort yang mempercepat proses pengurutan dengan cara membandingkan elemen yang berada pada jarak tertentu (gap), bukan hanya elemen yang berdekatan. Awalnya, jarak antar elemen (gap) ditetapkan besar, lalu secara bertahap dikurangi hingga menjadi 1, di mana pada akhirnya

algoritma akan berperilaku seperti Insertion Sort. Ini membuat data sebagian besar sudah terurut sebelum langkah akhir, sehingga Insertion Sort pada akhir proses menjadi jauh lebih efisien.

Keunikan Shell Sort terletak pada variasi pola pengurangan nilai gap yang digunakan, seperti Knuth sequence, Hibbard, atau Tokuda. Waktu komputasinya bisa bervariasi tergantung strategi gap yang digunakan, tetapi rata-rata berada di antara O(n log n) dan $O(n^2)$. Shell Sort tidak memerlukan memori tambahan, menjadikannya efisien secara ruang, dan lebih cepat daripada Bubble dan Insertion Sort pada dataset menengah.

4. Bubble Sort

Bubble Sort adalah algoritma pengurutan yang sangat sederhana namun tidak efisien. Algoritma ini bekerja dengan cara berulang kali membandingkan pasangan elemen yang berdekatan dan menukarnya jika urutannya salah. Setiap iterasi akan "mendorong" elemen terbesar ke posisi terakhir, seperti gelembung yang naik ke permukaan, sehingga dinamakan Bubble Sort.

Kelemahan Bubble Sort adalah kompleksitas waktu yang buruk, yaitu $O(n^2)$, bahkan dalam kasus terbaik hanya bisa mencapai O(n) jika sudah terurut dan dilengkapi dengan flag perhentian. Meskipun mudah dipahami dan diimplementasikan, algoritma ini tidak cocok untuk dataset besar karena sangat lambat dibanding algoritma lain. Namun, Bubble Sort sering diajarkan untuk pengenalan konsep dasar sorting.

5. Quick Sort

Quick Sort adalah salah satu algoritma pengurutan tercepat dalam praktik, yang juga menggunakan pendekatan *Divide and Conquer*. Algoritma ini memilih sebuah elemen sebagai pivot, kemudian membagi array menjadi dua bagian: satu berisi elemen yang lebih kecil dari pivot dan satu lagi berisi elemen yang lebih besar. Proses ini dilakukan secara rekursif pada kedua bagian hingga seluruh array terurut.

Keunggulan Quick Sort adalah efisiensinya dalam waktu rata-rata O(n log n) dan kemampuannya bekerja secara in-place tanpa membutuhkan memori tambahan seperti Merge Sort. Namun, dalam kasus terburuk (misalnya, ketika pivot dipilih sangat buruk), waktu komputasinya bisa menjadi $O(n^2)$. Untuk mencegah hal ini, strategi seperti random pivot atau median-of-three sering digunakan. Quick Sort sangat efisien untuk data besar dan sering digunakan dalam praktik.

6. Selection Sort

Selection Sort bekerja dengan cara mencari elemen terkecil dari seluruh array dan menempatkannya di posisi pertama, lalu mencari elemen terkecil dari sisa array dan menempatkannya di posisi kedua, dan seterusnya hingga array terurut seluruhnya. Dalam setiap iterasi, algoritma ini melakukan satu pertukaran posisi (swap), sehingga jumlah penukaran lebih sedikit dibanding Bubble Sort.

Namun, waktu komputasi Selection Sort tetap $O(n^2)$ karena tetap harus mencari elemen minimum dalam sisa array di setiap langkah. Selection Sort tidak stabil dan juga tidak efisien untuk data besar, tetapi karena strukturnya yang sederhana dan jumlah penukaran yang minimal, algoritma ini bisa berguna saat biaya pertukaran mahal. Meski tidak praktis untuk pengurutan nyata, algoritma ini tetap sering digunakan dalam pengajaran dasar sorting.

GITHUB

https://github.com/DSA25-ULM/task-5-sorting-rikafaulianarahmi